Adobe® Flex™ 2

ActionScript 3.0 のプログラミング

© 2006 Adobe Systems Incorporated. All rights reserved.

Flex 2 ActionScript 3.0 のプログラミング

本マニュアルが、エンドユーザー使用許諾契約付きのソフトウェアと一緒に配布される場合には、本マニュアル、および本マニュアルに記載されているソフトウェアは、ライセンスの所有者にのみ供給され、同ライセンスの条項に従っている場合にのみ使用またはコピーすることが許されます。当該エンドユーザー使用許諾契約により許可されている場合を除き、本マニュアルのいかなる部分といえども、Adobe Systems Incorporated (アドビ システムズ社) の書面による事前の許可なしに、電子的、機械的、録音、その他いかなる形式・手段であれ、複製、検索システムへの保存、または伝送を行うことはできません。本マニュアルの内容は、エンドユーザー使用許諾契約を含むソフトウェアと共に提供されていない場合であっても、著作権法により保護されていることにご留意ください。

本マニュアルに記載される内容は、あくまでも参照用としてのみ使用されること、また、なんら予告なしに変更されることを条件として、提供されるものであり、従って、当該情報が、アドビ システムズ社による確約として解釈されてはなりません。アドビ システムズ社は、本マニュアルにおけるいかなる誤りまたは不正確な記述に対しても、いかなる義務や責任を負うものではありません。

新しいアートワークを創作するためにテンプレートとして取り込もうとする既存のアートワークまたは画像は、著作権法により保護されている可能性のあるものであることをご留意ください。当該アートワークまたは画像を新しいアートワークに許可なく取り込んだ場合、著作者の権利を侵害することがあります。従って、著作権者から必要なすべての許可を必ず取得してください。

例として使用されている会社名は、実在の会社・組織を示すものではありません。

Adobe、Adobe ロゴ、Flex、Flex Builder、および Flash Player は、アドビ システムズ社の米国およびその他の国における登録商標または商標です。ActiveX and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and other countries. Macintosh is a trademark of Apple Computer, Inc., registered in the United States and other countries.All other trademarks are the property of their respective owners.

Speech compression and decompression technology licensed from Nellymoser, Inc. (www.nellymoser.com).

Sorenson™ Spark™ video compression and decompression technology licensed from Sorenson Media, Inc.

Opera ® browser Copyright © 1995-2002 Opera Software ASA and its suppliers. All rights reserved.

Adobe Systems Incorporated, 345 Park Avenue, San Jose, California 95110, USA

Notice to U.S. government end users. The software and documentation are “Commercial Items,” as that term is defined at 48 C.F.R. §2.101, consisting of “Commercial Computer Software” and “Commercial Computer Software Documentation,” as such terms are used in 48 C.F.R. §12.212 or 48 C.F.R. §227.7202, as applicable. Consistent with 48 C.F.R. §12.212 or 48 C.F.R. §§227.7202-1 through 227.7202-4, as applicable, the Commercial Computer Software and Commercial Computer Software Documentation are being licensed to U.S. Government end users (a) only as Commercial items and (b) with only those rights as are granted to all other end users pursuant to the terms and conditions herein. Unpublished-rights reserved under the copyright laws of the United States. Adobe Systems Incorporated, 345 Park Avenue, San Jose, CA 95110-2704, USA. For U.S. Government End Users, Adobe agrees to comply with all applicable equal opportunity laws including, if appropriate, the provisions of Executive Order 11246, as amended, Section 402 of the Vietnam Era Veterans Readjustment Assistance Act of 1974 (38 USC 4212), and Section 503 of the Rehabilitation Act of 1973, as amended, and the regulations at 41 CFR Parts 60-1 through 60-60, 60-250, and 60-741. The affirmative action clause and regulations contained in the preceding sentence shall be incorporated by reference.

目次

第 1 部 : ACTIONSCRIPT プログラミングの概要

本マニュアルについて 11
本マニュアルの使い方 12
ActionScript マニュアルへのアクセス 13
デベロッパーセンター 14

第 1 章 : ActionScript 3.0 の概要 15
ActionScript について 15
ActionScript 3.0 の利点 16
ActionScript 3.0 の新機能 16
旧バージョンとの互換性 20

第 2 章 : ActionScript の使用について 23
基本的な ActionScript 開発プロセス 23
コードを構成するためのオプション 24
例 : 基本的なアプリケーションの作成 26

第 3 章 : ActionScript 言語とシンタックス 35
言語の概要 35
オブジェクトとクラス 37
パッケージと名前空間 38
変数 51
データ型 55
シンタックス 71
演算子 77
条件 85
ループ 88
関数 91

第 4 章 : ActionScript のオブジェクト指向プログラミング 107
クラス 107
インターフェイス 126
継承 130
高度なテクニック 139
例 : GeometricShapes 148

第 5 章 : 表示のプログラミング 159
表示アーキテクチャについて 160
表示オブジェクトの操作 167
コア表示クラスを操作するための基礎 176
例 : SpriteArranger. 187

第 2 部 : ACTIONSCRIPT 3.0 の基本データ型およびコアクラス

第 6 章 : 日付と時刻の操作. 197
カレンダー日付と時刻の管理 197
タイマー間隔の制御 201
例 : 単純なアナログ時計. 204

第 7 章 : ストリングの操作 209
ストリングの作成 209
length プロパティ 211
ストリング内の文字の操作 211
ストリングの比較 212
他のオブジェクトのストリング表現の取得 213
ストリングの連結 213
ストリング内に含まれるサブストリングおよびパターンの検索 214
ストリングの大文字 / 小文字の変換 219
例 : ASCII アート 220

第 8 章 : 配列の操作 227
インデックス配列 227
結合配列 236
多次元配列 240
配列のクローンの作成 243
高度なテクニック 243
例 : PlayList 249

第 9 章 : エラー処理 255
エラーの種類 256
ActionScript 3.0 におけるエラー処理 258
デバッグ版の Flash Player の操作 260
アプリケーションの同期エラーの処理 261
カスタムエラークラスの作成 266
エラーイベントおよびステータスへの応答 267
Error クラスの比較 271
例 : CustomErrors アプリケーション 277

第 10 章 : 正規表現の使用 285
正規表現の概要 286
正規表現のシンタックス 287
ストリングに対して正規表現を使用するメソッド 303
例 : Wiki パーサー 304

第 11 章 : XML の操作 311
XML の基礎 312
XML 処理の E4X アプローチ 313
XML オブジェクト 315
XMLList オブジェクト 317
XML 変数の初期化 319
XML オブジェクトの構築と変形 320
XML 階層構造へのアクセス 322
XML 名前空間の使用 327
XML の型変換 328
外部 XML ドキュメントの読み込み 330
例 : インターネットから RSS データをロードする 330

第 3 部 : FLASH PLAYER API

第 12 章 : Flash Player API の概要 337
flash.accessibility パッケージ 338
flash.display パッケージ 338
flash.errors パッケージ 339
flash.events パッケージ 339
flash.external パッケージ 340
flash.filters パッケージ 340
flash.geom パッケージ 340
flash.media パッケージ 341

flash.net パッケージ. 341
flash.printing パッケージ 341
flash.profiler パッケージ 342
flash.system パッケージ. 342
flash.text パッケージ 342
flash.ui パッケージ 342
flash.utils パッケージ. 343
flash.xml パッケージ 343

第 13 章 : イベントの処理. 345

イベント処理の概要 346
ActionScript 3.0 のイベント処理と以前のバージョンにおける
イベント処理との違い 347
イベントフロー 350
イベントオブジェクト 352
イベントリスナー 357
例 : Alarm Clock 365

第 14 章 : ネットワーキングとコミュニケーション. 371

外部データの操作 372
他の Flash Player インスタンスへの接続 379
ソケット接続 385
ローカルデータの保存 390
ファイルのアップロードおよびダウンロードに関する操作. 394
例 : Telnet クライアントの構築 404
例 : ファイルのアップロードとダウンロード 408

第 15 章 : ジオメトリの操作. 417

Point オブジェクトの使用. 417
Rectangle オブジェクトの使用. 420
Matrix オブジェクトの使用 423
例 : 表示オブジェクトに対するマトリックス変換の適用 428

第 16 章 : クライアントのシステム環境 433

System クラス 433
Capabilities クラス. 435
ApplicationDomain クラス 436
IME クラス 439
例 : システム機能の検出. 444

第 17 章 : Flash Player セキュリティ 449
Flash Player セキュリティの概要. 450
アクセス許可管理の概要 452
セキュリティサンドボックス 461
ネットワーク API の制限. 464
フルスクリーンモードのセキュリティ 465
コンテンツのロード 467
クロススクリプト 470
ロードされたメディアへのデータとしてのアクセス. 474
データのロード 477
セキュリティドメインにインポートされた SWF ファイルからの
埋め込みコンテンツのロード 479
古いコンテンツの操作 480
LocalConnection 許可の設定 481
ホスト Web ページのスクリプトへのアクセス制御 481
共有オブジェクト 483
カメラ、マイク、クリップボード、マウス、キーボードアクセス 484

第 18 章 : プリント 487
ActionScript 3.0 における PrintJob クラスの新機能 488
ページのプリント 488
Flash Player タスクとシステムプリント. 489
サイズ、拡大率、用紙の向きの設定. 493
例 : 複数ページのプリント 495
例 : 拡大・縮小、トリミング、および応答 498

第 19 章 : External API の使用. 501
External API について. 502
ExternalInterface クラスの使用 503
例 : Web ページコンテナに対する External API の使用 507
例 : ActiveX コンテナに対する External API の使用. 514

第１部

ActionScript プログラミングの概要

1

ここでは、ActionScript 3.0 の基本的なプログラミング概念について説明します。

次の章が含まれます。

本マニュアルについて 11
第１章：ActionScript 3.0 の概要 15
第２章：ActionScript の使用について 23
第３章：ActionScript 言語とシンタックス 35
第４章：ActionScript のオブジェクト指向プログラミング. 107
第５章：表示のプログラミング 159

本マニュアルについて

本マニュアルでは、ActionScript 3.0 でアプリケーションを開発するための基礎について説明します。ここで説明されている概念やテクニックを理解するためには、データ型、変数、ループ、関数などの一般的なプログラミングの概念を理解している必要があります。また、クラスや継承などの基本的なオブジェクト指向プログラミングの概念についても理解していることが必要です。ActionScript 1.0 または ActionScript 2.0 の知識は役立ちますが、必要はありません。

目次

本マニュアルの使い方 12
ActionScript マニュアルへのアクセス 13
デベロッパーセンター 14

本マニュアルの使い方

本マニュアルは、以下の 3 部で構成されます。

各部のタイトル	内容
第 1 部の「ActionScript プログラミングの概要」	ActionScript 3.0 の基本的な概念について説明します。言語シンタックス、ステートメントと演算子、ECMAScript Edition 4 言語仕様案、ActionScript のオブジェクト指向プログラミング、および Adobe® Flash® Player 9 表示リストで表示オブジェクトを管理する新しい方法を解説します。
第 2 部の「ActionScript 3.0 の基本データ型およびコアクラス」	ECMAScript の草案仕様でも提案されている、ActionScript 3.0 の最上位データ型について説明します。
第 3 部の「Flash Player API」	イベント処理、ネットワーキングとコミュニケーション、ファイル入出力、外部インターフェイス、アプリケーションセキュリティモデルなど、Adobe Flash Player 9 固有のパッケージおよびクラスで実装される重要な機能について説明します。

本マニュアルには、API のアプリケーションプログラミングの概念を示すために、重要なクラスや頻繁に使用されるクラスのサンプルファイルが多数含まれています。サンプルファイルは、Adobe® Flex™ Builder 2 でロードおよび使用が簡単になるようにパッケージ化されており、ラッパーファイルが含まれている場合もあります。基本的なサンプルコードは、どの開発環境で使用できる ActionScript 3.0 です。

ActionScript 3.0 は次のような方法で記述およびコンパイルできます。

- Adobe Flex Builder 2 開発環境の使用
- テキストエディタおよび Flex Builder 2 に用意されているようなコマンドラインコンパイラの使用
- Adobe® Flash® CS3 オーサリングツールの使用

ActionScript 開発環境の詳細については、第 1 章の「ActionScript 3.0 の概要」を参照してください。

本マニュアルのサンプルコードを理解するために、Flex Builder や Flash オーサリングツールなどの ActionScript の統合開発環境の使用経験は必要ありません。ただし、ActionScript 3.0 コードを記述およびコンパイルするために、これらのツールの使用方法についてマニュアルを参照する必要があります。詳細については、13 ページの「ActionScript マニュアルへのアクセス」を参照してください。

ActionScript マニュアルへのアクセス

本マニュアルでは、豊富で強力なオブジェクト指向プログラミング言語である ActionScript 3.0 を中心に解説しているため、特定のツールまたはサーバーアーキテクチャ内のアプリケーション開発プロセスやワークフローについては詳しく説明していません。ActionScript 3.0 アプリケーションの設計、開発、テスト、デプロイについては、『ActionScript 3.0 のプログラミング』およびその他のマニュアルを参照してください。

ActionScript 3.0 マニュアル

本マニュアルでは、ActionScript 3.0 プログラミング言語の概念について理解を深めることができます。また、実装の詳細および重要な言語機能を示す例について説明します。ただし、本マニュアルは詳細なリファレンスガイドではありません。詳細なリファレンスについては、『ActionScript 3.0 リファレンスガイド』を参照してください。この言語のすべてのクラス、メソッド、プロパティ、およびイベントについて解説されています。『ActionScript 3.0 リファレンスガイド』には、コア言語、Flex MXML クラスとコンポーネント (mx パッケージ内)、および Flash Player API (flash パッケージ内) の詳細情報が記載されています。

Flex マニュアル

Flex 開発環境を使用する場合は、以下のマニュアルを参照してください。

マニュアル名	内容
Flex 2 開発ガイド	ダイナミックな Web アプリケーションの開発方法について説明します。
Flex 2 ファーストステップガイド	Flex の機能とアプリケーション開発手順の概要について説明します。
Flex 2 アプリケーションの構築と展開	Flex アプリケーションの構築およびデプロイ方法について説明します。
Flex 2 コンポーネントの作成と拡張	Flex コンポーネントの作成および拡張方法について説明します。
既存の Flex アプリケーションの Flex 2 への移行	移行プロセスの概要や、Flex と ActionScript の変更点に関する詳しい説明を記載しています。
Flex Builder 2 の使用	すべての Builder 機能に関する総合的な情報を Flex Builder ユーザーのレベルごとに示しています。
ActionScript 3.0 リファレンスガイド	Flex API の説明、シンタックス、用途、およびコード例を記載しています。

デベロッパーセンター

Adobe デベロッパーセンターでは、ActionScript の最新情報、実際のアプリケーション開発に関する記事、および新たな重要問題に関する情報を提供しています。デベロッパーセンター (http://www.adobe.com/jp/devnet/) を参照してください。

第 1 章

ActionScript 3.0 の概要

1

この章では、ActionScript の最も画期的な最新バージョンである ActionScript 3.0 の概要について説明します。

目次

ActionScript について 15
ActionScript 3.0 の利点 16
ActionScript 3.0 の新機能 16
旧バージョンとの互換性 20

ActionScript について

ActionScript は、Flash Player 実行時環境用のプログラミング言語で、Flash コンテンツおよびアプリケーションでのユーザー操作やデータ処理などを可能にします。

ActionScript は、Flash Player の一部である ActionScript 仮想マシンにより実行されます。ActionScript コードは通常、Flash オーサリングツールまたは Flex Builder に内蔵されているコンパイラや Flex SDK または Flex Data Services で使用可能なコンパイラによってバイトコード形式にコンパイルされます。バイトコードは、Flash Player 実行時環境で実行される SWF ファイルに埋め込まれます。

ActionScript 3.0 は、オブジェクト指向プログラミングの基礎知識を持つ開発者にはなじみのある堅牢なプログラミングモデルを提供します。ActionScript 3.0 には次のような機能があります。

- AVM2 という新しい ActionScript 仮想マシン。新しいバイトコード命令セットを使用し、パフォーマンスを大幅に向上させます。
- 最新のコンパイラコードベース。ECMAScript (ECMA 262) 規格に一層準拠し、旧バージョンのコンパイラより深い最適化を実行します。
- 拡張および改良されたアプリケーションプログラミングインターフェイス (API)。オブジェクトの低レベルコントロールと真のオブジェクト指向モデルを備えています。

- ECMAScript (ECMA-262) Edition 4 言語仕様提案に準拠したコア言語。
- ECMA-357 Edition 2 仕様で規定されている ECMAScript for XML (E4X) に準拠した XML API。E4X は、言語のネイティブデータ型として XML を追加する ECMAScript 言語拡張です。
- ドキュメントオブジェクトモデル (DOM) Level 3 Events 仕様に準拠したイベントモデル。

ActionScript 3.0 の利点

ActionScript 3.0 は、旧バージョンの ActionScript のスクリプト機能を上回っています。大容量のデータセットおよびオブジェクト指向の再利用可能なコードベースを使用する非常に複雑なアプリケーションを容易に作成できるように設計されています。ActionScript 3.0 は、Adobe Flash Player 9 で実行されるコンテンツには必要ありませんが、新しい仮想マシン AVM2 でのみ可能なパフォーマンスの向上を実現します。ActionScript 3.0 コードは、従来の ActionScript コードより最大で 10 倍高速に実行できます。

旧バージョンの ActionScript 仮想マシン AVM1 は、ActionScript 1.0 および ActionScript 2.0 のコードを実行します。AVM1 は、Flash Player 9 によって既存のコンテンツおよび古いコンテンツとの後方互換性がサポートされています。詳細については、20 ページの「旧バージョンとの互換性」を参照してください。

ActionScript 3.0 の新機能

ActionScript 3.0 には ActionScript のプログラマにわかりやすい多くのクラスおよび機能が含まれていますが、ActionScript 3.0 は旧バージョンの ActionScript とはアーキテクチャ上および概念上異なります。ActionScript 3.0 の強化点には、コア言語の新機能および低レベルオブジェクトの制御を高める改良された Flash Player API があります。

コア言語機能

コア言語は、ステートメント、式、条件、ループ、型などのプログラミング言語の基本的な要素を定義します。ActionScript 3.0 には、開発プロセスの効率化を促す多くの新機能が備わっています。

ランタイム例外

ActionScript 3.0 では、旧バージョンの ActionScript より多くのエラー状態が報告されます。ランタイム例外を一般的なエラー状態に使用すると、デバッグの操作性が向上し、エラーを確実に処理するアプリケーションを開発することができます。ランタイムエラーには、ソースファイルおよび行番号情報で注釈を付けたスタックトレースがあり、エラーをすばやく特定できます。

ランタイム型

ActionScript 2.0 では、型注釈は主に開発者を補助するもので、実行時にすべての値が動的に入力されました。ActionScript 3.0 では、型情報は実行時に維持され、さまざまな目的に使用されます。Flash Player 9 はランタイム型チェックを行い、システムの型安全性を向上させます。型情報はネイティブのマシン表現で変数を表すためにも使用され、パフォーマンスを向上させ、メモリの使用量を削減します。

sealed クラス

ActionScript 3.0 には、sealed クラスの概念が導入されています。sealed クラスには、コンパイル時に定義された一定のプロパティとメソッドだけが含まれ、その他のプロパティやメソッドを追加することはできません。このため、より厳密にコンパイル時のチェックを行うことができ、堅牢性の高いプログラムを作成できます。また、各オブジェクトインスタンスの内部ハッシュテーブルが不要になり、メモリの使用量を削減できます。ダイナミッククラスも `dynamic` キーワードを使用して使用可能です。ActionScript 3.0 ではすべてのクラスがデフォルトで sealed になっていますが、`dynamic` キーワードを使用すると、動的にするよう宣言できます。

メソッドクロージャ

ActionScript 3.0 では、元のオブジェクトインスタンスを自動的に記憶するメソッドクロージャが有効です。この機能はイベント処理に役立ちます。ActionScript 2.0 では、メソッドクロージャに抽出元のオブジェクトインスタンスが記憶されず、メソッドクロージャが呼び出されたときに予期しない動作が発生しました。mx.utils.Delegate クラスは一般的な方法でしたが、不要になりました。

ECMAScript for XML (E4X)

ActionScript 3.0 は、最近 ECMA-357 として標準化された ECMAScript for XML (E4X) を実装しています。E4X は、XML を操作するための自然で滑らかな言語コンストラクトを提供します。従来の XML 解析 API とは異なり、E4X の XML は、その言語のネイティブデータ型のように動作します。E4X は、必要なコード量を大幅に削減して、XML を操作するアプリケーションの開発を簡略化します。E4X の ActionScript 3.0 実装の詳細については、311 ページ、第 11 章の「XML の操作」を参照してください。

ECMA の E4X 仕様を確認するには、www.ecma-international.org/publications/files/ECMA-ST/ECMA-357.pdf を参照してください。

正規表現

ActionScript 3.0 では正規表現がネイティブにサポートされているので、ストリングをすばやく検索し、操作できます。ActionScript 3.0 は、ECMAScript Edition 3 言語仕様 (ECMA-262) で定義されている正規表現のサポートを実装しています。

名前空間

名前空間は、宣言 (`public`、`private`、`protected`) の可視性を制御するために使用する通常のアクセス指定子に似ています。名前空間は、カスタムアクセス指定子として動作し、任意の名前を指定できます。名前空間には、競合を回避するために URI (Universal Resource Identifier) が割り当てられています。また、E4X を使用するときに XML 名前空間を表すためにも使用します。

新しいプリミティブ型

ActionScript 2.0 の数値型は、倍精度浮動小数点数の Number 型のみですが、ActionScript 3.0 には、int 型と uint 型が含まれます。int 型は、ActionScript コードが CPU の高速整数演算機能を活用できる 32 ビット符号付き整数で、整数が使用されるループカウンタや変数に便利です。uint 型は符号なしの 32 ビット整数型で、RGB カラー値、バイトカウントなどに便利です。

Flash Player API の機能

ActionScript 3.0 の Flash Player API には、オブジェクトを低いレベルで制御できる多数の新しいクラスが追加されています。ActionScript 3.0 のアーキテクチャはまったく新しく、より直観的です。新しく追加されたクラスが多いため、ここでは詳しく説明しませんが、次のセクションでいくつかの大幅な変更について取り上げます。

DOM3 イベントモデル

DOM3 (Document Object Model Level 3) イベントモデルを使用すると、アプリケーション内のオブジェクトが互いにやり取りでき、その状態を保持し、変更に応答できるように、イベントメッセージを生成および処理できます。W3C (World Wide Web Consortium) DOM Level 3 Events 仕様に従って作成されたこのモデルは、旧バージョンの ActionScript で使用できるイベントシステムよりわかりやすく効率的なメカニズムを備えています。

イベントおよびエラーイベントは、flash.events パッケージにあります。Flex アプリケーションフレームワークでは、Flash Player API と同じイベントモデルを使用するため、イベントシステムが Flash プラットフォーム全体で統一されます。

表示リスト API

Flash Player 表示リスト、つまり Flash アプリケーションのビジュアルエレメントを含むツリーにアクセスするための API は、Flash のビジュアルプリミティブを操作するためのクラスで構成されます。

新しい Sprite クラスは、MovieClip によく似た軽量の構築ブロックですが、UI コンポーネントの基本クラスとして適しています。新しい Shape クラスは、未加工のベクターシェイプを表します。これらのクラスは、`new` 演算子で自然にインスタンス化でき、いつでも動的に再度親にすることができます。

深度の管理は、自動化されて Flash Player に組み込まれているため、深度番号を割り当てる必要がなくなりました。オブジェクトの z 順序を指定し管理するための新しいメソッドが用意されています。

動的なデータおよびコンテンツの処理

ActionScript 3.0 は、Flash アプリケーションでアセットおよびデータをロードし処理するための直観的で API 全体で一貫したメカニズムを備えています。新しい Loader クラスは、SWF ファイルおよびイメージアセットをロードするためのメカニズムを備え、ロードされたコンテンツに関する詳細情報にアクセスするための手段を提供します。URLLoader クラスには、データ駆動アプリケーションでテキストおよびバイナリデータをロードするための個別のメカニズムがあります。Socket クラスは、サーバーソケットに対して任意の形式でバイナリデータを読み取り、書き込む手段を提供します。

低レベルでのデータアクセス

さまざまな API で、これまで ActionScript では使用できなかったデータに低レベルでアクセスできます。ダウンロード中のデータの場合、URLLoader によって実装される URLStream クラスを使用すると、データをダウンロード中に生のバイナリデータとしてデータにアクセスできます。ByteArray クラスを使用すると、バイナリデータの読み取り、書き込み、および操作を最適化できます。新しい Sound API では、SoundChannel および SoundMixer クラスからサウンドを詳細に制御できます。セキュリティを処理する新しい API は、SWF またはロードされたコンテンツのセキュリティ権限に関する情報を提供し、セキュリティエラーをより適切に処理できるようにします。

テキストの操作

ActionScript 3.0 には、すべてのテキスト関連 API の flash.text パッケージが含まれています。TextLineMetrics クラスは、テキストフィールド内のテキストの行の詳細なメトリックを提供します。これは、ActionScript 2.0 の `TextField.getLineMetrics()` メソッドに取って代わるものです。TextField クラスには、テキストフィールド内のテキスト 1 行または 1 文字に関する情報を提供できる多数の新しい低レベルメソッドが含まれています。たとえば、文字の境界ボックスを表す矩形を返す `getCharBoundaries()`、指定されたポイントにある文字のインデックスを返す `getCharIndexAtPoint()`、段落の最初の文字のインデックスを返す `getFirstCharInParagraph()` などのメソッドがあります。行レベルのメソッドには、テキストの指定された行の文字数を返す `getLineLength()`、および指定された行のテキストを返す `getLineText()` があります。新しい Font クラスを使用すると、SWF ファイルの埋め込みフォントを管理できます。

旧バージョンとの互換性

Flash Player には、以前にパブリッシュされたコンテンツとの完全な後方互換性があります。旧バージョンの Flash Player で実行されていたコンテンツは、Flash Player 9 で実行できます。しかし、Flash Player 9 に ActionScript 3.0 が導入され、Flash Player 9 を実行する新旧コンテンツの相互運用性に次のような互換性の問題が発生しています。

- 1 つの SWF ファイルでは、ActionScript 1.0 または 2.0 のコードと ActionScript 3.0 コードを結合することはできません。
- ActionScript 3.0 コードは、ActionScript 1.0 または 2.0 で記述されている SWF ファイルをロードできますが、SWF ファイルの変数および関数にアクセスすることはできません。
- ActionScript 1.0 または 2.0 で記述されている SWF ファイルは、ActionScript 3.0 で記述されている SWF ファイルをロードすることはできません。つまり、Flash 8 または Flex Builder 1.5 以前のバージョンで作成された SWF ファイルは、ActionScript 3.0 で記述されている SWF ファイルをロードすることができません。

この規則には唯一例外があり、以前に ActionScript 2.0 SWF ファイルによってそのどのレベルにも何もロードされていなければ、ActionScript 2.0 SWF ファイル自体を ActionScript 3.0 SWF ファイルで置き換えることができることです。ActionScript 2.0 SWF ファイルは、この置き換えを `loadMovieNum()` の呼び出しによって行うことができ、そのときに値 0 を `level` パラメータに渡します。

■ 一般に、ActionScript 1.0 または 2.0 で記述されている SWF ファイルは、ActionScript 3.0 で記述されている SWF ファイルと一緒に使用する場合、移行する必要があります。たとえば、ActionScript 2.0 を使用してメディアプレーヤーを作成したとします。このメディアプレーヤーでは、ActionScript 2.0 を使用して作成されたさまざまなコンテンツをロードできます。しかし、ActionScript 3.0 で新しいコンテンツを作成して、このメディアプレーヤーにロードすることはできません。メディアプレーヤーを ActionScript 3.0 に移行する必要があります。

ただし、ActionScript 3.0 でメディアプレーヤーを作成すれば、ActionScript 2.0 で作成されたコンテンツを簡単にロードできます。

次の表に、新しいコンテンツのロードおよびコードの実行に対する旧バージョンの Flash Player の制限、および異なるバージョンの ActionScript で記述されている SWF ファイル間のクロススクリプティングの制限の概要を示します。

サポートされている機能	ランタイム環境		
	Flash Player 7	Flash Player 8	Flash Player 9
ロード可能な SWF の対象バージョン	7 以前	8 以前	9 以前
AVM の装備	AVM1	AVM1	AVM1 および AVM2
実行可能な SWF が作成された ActionScript のバージョン	1.0、2.0	1.0、2.0	1.0、2.0、3.0

サポートされている機能*	コンテンツが作成されたバージョン	
	ActionScript 1.0 および 2.0	ActionScript 3.0
ロードおよびコードの実行が可能なコンテンツが作成されたバージョン	ActionScript 1.0 および 2.0 のみ	ActionScript 1.0 および 2.0、ActionScript 3.0
クロススクリプトが可能なコンテンツが作成されたバージョン	ActionScript 1.0 および 2.0 のみ†	ActionScript 3.0‡

* Flash Player 9 以降で実行されるコンテンツです。Flash Player 8 以前で実行されるコンテンツは、ActionScript 1.0 および 2.0 のみでロード、表示、実行、およびクロススクリプトできます。

† ActionScript 3.0 (LocalConnection 経由)。

‡ ActionScript 1.0 および 2.0 (LocalConnection 経由)。

第 2 章

ActionScript の使用について

2

この章では、単純な ActionScript 3.0 アプリケーションの作成方法について段階を追って説明します。

ActionScript 3.0 プログラミング言語は、Adobe の Macromedia Flash、Adobe Flex Builder 2 を始めとするさまざまな開発環境で使用できます。この章では、いずれかのアプリケーション開発環境で、モジュール化した ActionScript コードを作成する方法を解説します。

目次

基本的な ActionScript 開発プロセス 23
基本的な ActionScript 開発プロセス 23
例 : 基本的なアプリケーションの作成 26

基本的な ActionScript 開発プロセス

ActionScript プロジェクトの大小にかかわらず、プロセスを使用してアプリケーションを設計および開発すると、作業の効率および効果が向上します。次に、ActionScript 3.0 を使用したアプリケーションを構築する基本的な開発プロセスの手順について説明します。

1. アプリケーションを設計します。

   アプリケーションの作成を開始する前に、何らかの方法でアプリケーションを記述する必要があります。

2. ActionScript 3.0 コードを構成します。

   ActionScript コードは、Flash、Flex Builder、Adobe の Macromedia Dreamweaver®、またはテキストエディタを使用して作成できます。

3. Flash または Flex アプリケーションファイルを作成して、コードを実行します。

   Flash オーサリングツールを使用する場合は、新しい FLA ファイルを作成し、パブリッシュ設定を行います。次に、アプリケーションにユーザーインターフェイスコンポーネントを追加して、ActionScript コードを参照します。Flex 開発環境で新しいアプリケーションファイルを作成するには、MXML を使用してアプリケーションを定義し、ユーザーインターフェイスコンポーネントを追加して、ActionScript コードを参照します。

4. ActionScript アプリケーションをパブリッシュし、テストします。

   Flash オーサリング環境または Flex 開発環境内からアプリケーションを実行して、意図したことがすべて実行されていることを確認します。

これらの手順を必ずしも順序どおりに実行する必要はありません。また、1 つの手順を完了する前に次の手順に移ることもできます。たとえば、アプリケーションの 1 つの画面を作成した後 (手順 1)、ActionScript コードの記述 (手順 2) やテスト (手順 4) を行う前に、グラフィックやボタンなどを作成することができます (手順 3)。あるいは、画面の一部を作成した後、ActionScript の記述と、構築後のテストを実行しながら、ボタンやインターフェイスエレメントを 1 つずつ追加していくことも可能です。開発プロセスにおけるこの 4 つのステージを覚えておくと便利ですが、通常、実際の開発では、必要に応じてステージ間を往来した方が効果的です。

コードを構成するためのオプション

ActionScript 3.0 コードを使用して、単純なグラフィックやアニメーションから複雑なクライアント / サーバートランザクション処理システムまであらゆるものを強化できます。構築するアプリケーションのタイプに応じて、ActionScript をプロジェクトにインクルードするこれらの方法のいずれか、またはその組み合わせを利用できます。

Flash タイムラインのフレームへのコードの格納

Flash オーサリング環境では、タイムラインのフレームに ActionScript コードを追加できます。このコードは、ムービーの再生中に再生ヘッドがそのフレームに入ると実行されます。

フレームに ActionScript コードを配置すると、Flash オーサリングツールに組み込まれているアプリケーションにビヘイビアを簡単に追加できます。メインタイムラインのフレームまたは MovieClip シンボルのタイムラインのフレームに、コードを追加できます。しかし、こうした柔軟性を持たせると問題がでてきます。大規模なアプリケーションを構築するときに、どのフレームにどのスクリプトが含まれているのか把握できなくなることがよくあり、時間の経過に伴ってアプリケーションのメンテナンスが困難になる場合があります。

多くの開発者は、Flash オーサリングツールで、タイムラインの先頭フレーム内のみまたは Flash ドキュメントの特定のレイヤー上のみにコードを配置して、ActionScript コードの構成を簡略化します。これにより、Flash FLA ファイルのコードが見つけやすくなり、メンテナンスも簡単になります。ただし、別の Flash または Flex プロジェクトで同じコードを使用するには、コードをコピーして新しいファイルにペーストする必要があります。

将来、別の Flash または Flex プロジェクトでこの ActionScript コードを使用できるようにするには、外部 ActionScript ファイル (.as 拡張子の付いたテキストファイル) にコードを格納します。

Flex MXML ファイルへのコードの埋め込み

Flex 開発環境では、Flex MXML ファイルの `<mx:Script>` タグ内に ActionScript コードを含めることができます。このようなインライン ActionScript コードには、Flash のフレームに配置されたコードと同じ欠点があります。つまり、コードを切り取って新しいソースの場所にペーストしないと、コードを繰り返し使用することができません。

メモ: `<mx:Script>` タグに source パラメータを指定できます。これにより、`<mx:Script>` タグ内に直接入力した場合と同じように ActionScript コードを挿入できます。ただし、使用するソースファイルでは、再利用性を制限する独自のクラスを定義することはできません。

ActionScript ファイルへのコードの格納

プロジェクトに大量の ActionScript コードが含まれている場合、コードを構成する最適な方法は、別の ActionScript ソースファイル (.as 拡張子の付いたテキストファイル) にコードを格納することです。ActionScript ファイルは、アプリケーションでの使用目的に応じて、次の 2 つの方法のいずれかで構築することができます。

- 非構造化 ActionScript コード : ActionScript コードの行。タイムラインスクリプトや MXML ファイルなどに直接入力するときと同じように記述されたステートメントや関数定義が含まれます。

  この方法で記述された ActionScript には、ActionScript の `include` ステートメント、または Flex MXML の `<mx:Script>` タグを使用してアクセスできます。ActionScript の `include` ステートメントを使用すると、外部 ActionScript ファイルの内容をスクリプト内の特定の位置および特定のスコープに、直接入力したときと同じように挿入できます。Flex の MXML 言語で、`<mx:Script>` タグを使用すると、アプリケーションのその時点でロードする外部 ActionScript ファイルを識別する source 属性を指定できます。たとえば、次のタグは "Box.as" という外部 ActionScript ファイルをロードします。

  ```
  <mx:Script source="Box.as" />
  ```

- ActionScript クラス定義 : メソッドやプロパティの定義を含む、ActionScript クラスの定義。

  クラスを定義するときに、クラスの ActionScript コードにアクセスするには、ActionScript のビルトインクラスを操作する場合と同じように、クラスのインスタンスを作成し、そのプロパティ、メソッド、イベントなどを使用します。そのためには、2 つの操作が必要です。

  - ActionScript コンパイラがクラスの検索場所を判別できるよう、`import` ステートメントを使用してクラスのフルネームを指定します。たとえば、ActionScript で MovieClip クラスを使用する場合は、最初にパッケージやクラスを含むフルネームを使ってそのクラスを読み込む必要があります。

    ```
    import flash.display.MovieClip;
    ```

    別の方法として、MovieClip クラスが含まれるパッケージを読み込むこともできます。この方法は、パッケージ内のクラスごとに個別の import ステートメントを記述するのと同じです。

    ```
    import flash.display.*;
    ```

    コード内でクラスを参照するにはそのクラスを読み込む必要がありますが、パッケージで定義されていないトップレベルのクラスだけはこの規則の例外です。

  - 特定のクラス名を参照するためのコードを記述します。通常は、そのクラスをデータ型とする変数を宣言し、変数に格納するための、そのクラスのインスタンスを作成します。ActionScript コード内の別のクラス名を参照して、コンパイラにそのクラスの定義をロードするように指示します。たとえば、Box という外部クラスがあるとすると、このステートメントによって Box クラスの新しいインスタンスが作成されます。

    ```
    var smallBox:Box = new Box(10,20);
    ```

    コンパイラが初めて Box クラスへの参照に到達すると、コンパイラは、ロードされたソースコードを検索して Box クラスの定義を見つます。

例 : 基本的なアプリケーションの作成

Flash、Flex Builder、Dreamweaver、または任意のテキストエディタを使用して、.as 拡張子を持つ外部 ActionScript ソースファイルを作成できます。

ActionScript 3.0 は、Flash オーサリングツールや Flex Builder などのさまざまなアプリケーション開発環境で使用できます。

このセクションでは、Flash オーサリングツールまたは Flex Builder 2 ツールを使用して、単純な ActionScript 3.0 アプリケーションを作成および機能拡張する手順を案内します。これから作成するアプリケーションでは、Flash および Flex アプリケーションで ActionScript 3.0 の外部クラスファイルを使用する簡単な例が示されます。その見本は、本マニュアルでの他のすべてのサンプルアプリケーションに適用されます。

ActionScript アプリケーションの設計

アプリケーションの作成を始める前に、どのようなアプリケーションにするかを考えておく必要があります。

設計は、アプリケーションの名前と目的を示す短い説明などの簡単なもので表しても、多数の UML (Unified Modeling Language) 図を含む要件書一式のような複雑なもので表しても構いません。本マニュアルでは、アプリケーションの設計方法について詳しく説明しませんが、アプリケーションの設計は ActionScript アプリケーションの開発においてきわめて重要です。

ActionScript アプリケーションの最初の例は、単純な設計を使用した、標準的な "Hello World" アプリケーションです。

- アプリケーションの名前は HelloWorld です。
- "Hello World!" というストリングが含まれたテキストフィールドが 1 つ表示されます。
- 繰り返し使用しやすいように、Greeter という名前のオブジェクト指向クラスを 1 つ使用します。このクラスは、Flash ドキュメントまたは Flex アプリケーション内から使用できます。
- 基本になるアプリケーションを作成した後、ユーザーにユーザー名を入力させ、アプリケーションにその名前をユーザーリストと照合させる新しい機能を追加します。

この簡潔な定義を使用して、アプリケーションの作成を開始します。

HelloWorld プロジェクトと Greeter クラスの作成

Hello World アプリケーションの設計説明では、再利用しやすいコードを使用するよう指定されています。この目標を考慮し、アプリケーションは Greeter という名前のオブジェクト指向クラスを 1 つ使用します。このクラスは、Flex Builder または Flash オーサリングツールで作成したアプリケーション内から使用されます。

Flex Builder で HelloWorld プロジェクトと Greeter クラスを作成するには :

1. Flex Builder で、[ファイル]-[新規]-[Flex プロジェクト] を選択します。
2. [新規 Flex プロジェクト] ダイアログボックスで Flex サーバーテクノロジーを選択するよう要求される場合は、[基本] を選択し、[次へ] をクリックします。
3. [プロジェクト名] に「HelloWorld」と入力し、[終了] をクリックします。

   新しいプロジェクトが作成され、それがナビゲータパネルに表示されます。デフォルトでは、プロジェクトは既に "HelloWorld.mxml" という名前のファイルを含み、そのファイルは [エディタ] パネルに開いています。
4. ここでカスタムの ActionScript クラスファイルを Flex Builder ツールで作成するために、[ファイル]-[新規]-[ActionScript ファイル] を選択します。

5. [新規 ActionScript ファイル] ダイアログボックスで、親フォルダとして "HelloWorld" を選択し、ファイル名に「Greeter.as」と入力して、[終了] をクリックします。

   新しい ActionScript 編集ウィンドウが表示されます。

   28 ページの「Greeter クラスへのコードの追加」に進みます。

Greeter クラスへのコードの追加

Greeter クラスは、HelloWorld アプリケーションで使用できるオブジェクト (`Greeter`) を定義します。

Greeter クラスにコードを追加するには :

1. 新しいファイルに次のコードを入力します。

```
package
{
   public class Greeter
   {
      public function sayHello():String
      {
         var greeting:String;
         greeting = "Hello World!";
         return greeting;
      }
   }
}
```

   Greeter クラスは `sayHello()` メソッドを 1 つ含み、このメソッドは指定されたユーザー名に "Hello" というストリングを返します。

2. [ファイル]-[保存] を選択して、この ActionScript ファイルを保存します。

これで、Flash または Flex アプリケーションで Greeter クラスを使用することができます。

ActionScript コードを使用したアプリケーションの作成

作成した Greeter クラスは、必要なものをすべて備えたアプリケーション機能を定義しますが、アプリケーション全部を表すわけではありません。Greeter クラスを使用するには、Flash ドキュメントまたは Flex アプリケーションを作成する必要があります。

HelloWorld アプリケーションでは、Greeter クラスの新しいインスタンスが作成されます。Greeter クラスをアプリケーションに割り当てる方法は次のとおりです。

Flex Builder を使用して ActionScript アプリケーションを作成するには :

1. "HelloWorld.mxml" ファイルを開いて、次のコードを入力します。

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
  layout="vertical"
  creationComplete = "initApp()" >

  <mx:Script>
    <![CDATA[
      private var myGreeter:Greeter = new Greeter();

      public function initApp():void
      {
        // 最初に Hello と言い、ユーザー名を尋ねる
        mainTxt.text = myGreeter.sayHello();
      }
    ]]>
  </mx:Script>

  <mx:TextArea id = "mainTxt" width="400" />

</mx:Application>
```

この Flex プロジェクトには、次の 3 つの MXML タグが含まれます。

- Application コンテナを定義する `<mx:Application>` タグ
- ActionScript コードを含む `<mx:Script>` タグ
- ユーザーにテキストメッセージを表示するフィールドを定義する `<mx:TextArea>` タグ

`<mx:Script>` タグのコードは、アプリケーションロード時に呼び出される `initApp()` メソッドを定義します。`initApp()` メソッドは、`mainTxt` TextArea のテキスト値を、記述したカスタムクラス Greeter の `sayHello()` メソッドが返す "Hello World!" というストリングに設定します。

2. [ファイル]-[保存] を選択して、アプリケーションを保存します。

30 ページの「ActionScript アプリケーションのパブリッシュとテスト」に進みます。

ActionScript アプリケーションのパブリッシュとテスト

アプリケーション開発は反復プロセスです。コードを記述し、コンパイルして、正しくコンパイルされるまでコードを編集します。コンパイルされたアプリケーションを実行してテストし、意図した設計が実現されているかどうかを確認します。実現されていない場合は、実現するまでコードを編集します。Flash および Flex Builder の開発環境には、アプリケーションをパブリッシュ、テスト、およびデバッグする多数の方法が用意されています。

次に、HelloWorld アプリケーションを各環境でテストするための基本的な手順を示します。

Flex Builder を使用して ActionScript アプリケーションをパブリッシュおよびテストするには :

1. [実行] - [実行] を選択します。[プロジェクト] フィールドに "HelloWorld" が表示され、[アプリケーションファイル] フィールドに "HelloWorld.mxml" が表示されることを確認します。
2. [実行] ダイアログボックスで、[実行] をクリックします。

   HelloWorld アプリケーションが起動します。

   - アプリケーションのテスト時に、出力ウィンドウにエラーまたは警告が表示された場合、"HelloWorld.mxml" ファイルまたは "Greeter.as" ファイルでエラーの原因を修正してから、再度アプリケーションをテストします。
   - コンパイルエラーがなければ、ブラウザウィンドウが開いて、Hello World アプリケーションが表示されます。"Hello World!" と表示されます。

これで、ActionScript 3.0 を使用した単純ながら完全なオブジェクト指向アプリケーションが作成されました。30 ページの「HelloWorld アプリケーションの機能拡張」に進みます。

HelloWorld アプリケーションの機能拡張

ここでは、アプリケーションに少し面白みを持たせるために、アプリケーションにユーザー名を要求させ、それを定義済みの名前リストと検証させます。

まず、Greeter クラスを更新して新しい機能を追加します。次に、Flex または Flash アプリケーションを更新して新しい機能を使用します。

"Greeter.as" ファイルを更新するには :

1. "Greeter.as" ファイルを開きます。

2. ファイルの内容を次のように変更します。新しい行および変更された行は太字で表示されています。

```
package
{
  public class Greeter
  {
    /**
     * 適切な挨拶を受ける側の名前を定義します。
     */
    public static var validNames:Array = ["Sammy", "Frank", "Dean"];

    /**
     * 指定された名前を使用して挨拶ストリングを作成します。
     */
    public function sayHello(userName:String = ""):String
    {
      var greeting:String;
      if (userName == "")
      {
        greeting = "Hello.Please type your user name, and then press the
  Enter key.";
      }
      else if (validName(userName))
      {
        greeting = "Hello, " + userName + ".";
      }
      else
      {
        greeting = "Sorry, " + userName + ", you are not on the list.";
      }
      return greeting;
    }

    /**
     * validNames リストに名前があるかどうかをチェックします。
     */
    public static function validName(inputName:String = ""):Boolean
    {
      if (validNames.indexOf(inputName) > -1)
      {
        return true;
      }
      else
      {
        return false;
      }
    }
  }
}
```

これで、Greeter クラスに新しい機能が追加されました。

- validNames 配列には、有効なユーザー名が一覧表示されます。この配列は、Greeter クラスのロード時に 3 つの名前のリストに初期化されます。
- これで、sayHello() メソッドは、いくつかの条件に従ってユーザー名を受け入れて挨拶を変更します。userName が空のストリング ("") の場合、greeting プロパティはユーザーに名前の入力を求めるように設定されます。ユーザー名が有効な場合、挨拶は "Hello, <ユーザー名>" になります。最後に、これら 2 つの条件のいずれかが満たされない場合は、greeting 変数は "Sorry, <ユーザー名>, you are not on the list." に設定されます。
- validName() メソッドは、inputName が validNames 配列で見つかれば true を返し、見つからなければ false を返します。ステートメント validNames.indexOf(inputName) は、validNames 配列の各ストリングを inputName ストリングと照合します。Array.indexOf() メソッドは、配列内のオブジェクトの最初のインスタンスのインデックス位置を返し、オブジェクトが配列で見つからなければ、値 -1 を返します。

次に、この ActionScript クラスを参照する Flash または Flex ファイルを編集します。

Flex Builder を使用してアプリケーションを変更するには :

1. "HelloWorld.mxml" ファイルを開きます。
2. 表示目的のみであることをユーザーに示すように、<mx:TextArea> タグを変更します。そのために次のように、背景色を淡いグレーに変更して、ユーザーによる表示テキストの編集を禁止します。

   ```
   <mx:TextArea id = "mainTxt" width="400" backgroundColor="#DDDDDD"
   editable="false" />
   ```

3. <mx:TextArea> 終了タグの直後に次の行を追加します。これらの行は、ユーザー名の値を入力することができる新しい TextInput フィールドを作成します。

   ```
   <mx:HBox width="400">
     <mx:Label text="User Name:"/>
     <mx:TextInput id="userNameTxt" width="100%" enter="mainTxt.text =
   myGreeter.sayHello(userNameTxt.text);" />
   </mx:HBox>
   ```

   enter 属性は、ユーザーが userNameTxt フィールドで Enter キーを押したときに、このフィールドのテキストが Greeter.sayHello() メソッドに渡されることを指定します。mainTxt フィールドに表示される挨拶は、それに応じて変化します。

   "HelloWorld.mxml" ファイルの最終的な内容は次のようになります。

   ```
   <?xml version="1.0" encoding="utf-8"?>
   <mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" xmlns="*"
     layout="vertical"
     creationComplete = "initApp()" >

     <mx:Script>
       <![CDATA[
   ```

```
        private var myGreeter:Greeter = new Greeter();

        public function initApp():void
        {
            // 最初に Hello と言い、ユーザー名を尋ねる
            mainTxt.text = myGreeter.sayHello();
        }

    ]]>
</mx:Script>

<mx:TextArea id = "mainTxt" width="400" backgroundColor="#DDDDDD"
editable="false" />

<mx:HBox width="400">
    <mx:Label text="User Name:"/>
    <mx:TextInput id="userNameTxt" width="100%" enter="mainTxt.text =
myGreeter.sayHello(userNameTxt.text);" />
</mx:HBox>

</mx:Application>
```

4. 編集した "HelloWorld.mxml" ファイルを保存します。[実行]-[実行] を選択してアプリケーションを実行します。

   アプリケーションを実行すると、ユーザー名の入力を求めるメッセージが表示されます。ユーザー名が有効 (Sammy、Frank、または Dean) である場合は、"Hello, ‹ユーザー名›" という確認メッセージが表示されます。

これ以降の例の実行

"Hello World" ActionScript 3.0 アプリケーションを開発し実行すると、本マニュアルに記載されている他のコード例を実行するために必要な基礎知識を習得できます。

これ以降のコード例は、この例のような順を追ったチュートリアル形式で説明されていません。各例では関連する ActionScript 3.0 コードを強調表示して説明していますが、特定の開発環境でこれらの例を実行するための手順については説明されていません。しかし、本マニュアルに付属のサンプルファイルには、任意の開発環境で簡単に例をコンパイルして実行するために必要なすべてのファイルが含まれています。

第 3 章

ActionScript 言語とシンタックス

3

ActionScript 3.0 は、コア ActionScript 言語と Flash Player アプリケーションプログラミングインターフェイス (API) の両方で構成されます。コア言語は、ECMAScript (ECMA-262), Edition 4 言語仕様案を実装する ActionScript の一部です。Flash Player API では、プログラムによって Flash Player にアクセスできます。

本章では、コア ActionScript 言語とシンタックスについて簡単に説明します。この章を読むと、データ型と変数の操作方法、適切なシンタックスの使用方法、およびプログラム内でのデータフローの制御方法についての基礎知識を習得できます。

目次

言語の概要 35
オブジェクトとクラス 37
パッケージと名前空間 38
変数 51
データ型 55
シンタックス 71
演算子 77
条件 85
ループ 88
関数 91

言語の概要

オブジェクトは ActionScript 3.0 言語の中核をなす基本的な構成要素です。宣言する変数、記述する関数、作成するクラスインスタンスはすべてオブジェクトです。ActionScript 3.0 プログラムは、タスクを実行し、イベントに応答し、相互に通信するオブジェクトのグループと考えることができます。

Java や C++ のオブジェクト指向プログラミング (OOP) に慣れているプログラマなら、オブジェクトを、メンバー変数またはプロパティに格納されているデータと、メソッドからアクセス可能なビヘイビアの 2 種類のメンバーを含むモジュールとして考えることができます。ActionScript 3.0 が準拠している ECMAScript 第 4 版案では、よく似ていますが少し異なる方法でオブジェクトが定義されています。ECMAScript の草案では、オブジェクトは単にプロパティの集まりです。プロパティは、データだけではなく関数や他のオブジェクトも保持できるコンテナです。このようにオブジェクトに関連付けられている関数をメソッドと呼びます。

ECMAScript の草案の定義は Java または C++ のプログラマには少し奇妙に思えるかもしれませんが、実際には ActionScript 3.0 クラスを使用するオブジェクト型の定義は、Java や C++ でクラスを定義する方法に非常によく似ています。オブジェクトの 2 つの定義の違いは ActionScript オブジェクトモデルおよびその他の高度なテクニックについて説明する場合には重要ですが、ほとんどの場合、" プロパティ " という用語は、メソッドとは対照的に、クラスのメンバー変数を意味します。たとえば、『ActionScript 3.0 リファレンスガイド』では、プロパティは変数または getter/setter プロパティを示すために使用されています。また、メソッドという用語は、クラスの一部である関数を示すために使用されています。

ActionScript のクラスと Java または C++ のクラスには微妙な違いがあります。それは、ActionScript ではクラスは単なる抽象エンティティではないということです。ActionScript のクラスは、クラスのプロパティおよびメソッドを格納するクラスオブジェクトによって表されます。これによって、クラスまたはパッケージの最上位にステートメントや実行可能なコードを含めるなどの、Java や C++ のプログラマにとっては考えられないテクニックが可能になります。

ActionScript のクラスと Java または C++ のクラスのもう 1 つの違いは、ActionScript の各クラスには " プロトタイプオブジェクト " と呼ばれるものがあることです。旧バージョンの ActionScript では、プロトタイプオブジェクトは " プロトタイプチェーン " に関連付けられてクラスの継承階層全体の基盤として機能しましたが、ActionScript 3.0 では、プロトタイプオブジェクトは継承システムにおいて小さな役割を果たすだけです。しかし、クラスのすべてのインスタンスでプロパティとその値を共有する場合、プロトタイプオブジェクトは静的なプロパティやメソッドの代わりとして役立ちます。

これまで、熟練した ActionScript プログラマなら、特別なビルトイン言語エレメントを使用してプロトタイプチェーンを直接操作できました。現在では、ActionScript はクラスベースのプログラミングインターフェイスのさらに成熟した実装を提供しています。`__proto__` や `__resolve` などの特別な言語エレメントの多くは ActionScript に含まれていません。また、Flash Player のパフォーマンスを大幅に向上させる内部継承メカニズムが最適化され、継承メカニズムに直接アクセスできなくなりました。

オブジェクトとクラス

ActionScript 3.0 では、各オブジェクトはクラスにより定義されます。クラスは、オブジェクト型のテンプレートまたは設計図と考えることができます。クラス定義には、データ値を保持する変数と定数、およびクラスにバインドされているビヘイビアをカプセル化する関数であるメソッドを含めることができます。プロパティに格納される値は、プリミティブ値または他のオブジェクトです。プリミティブ値とは、数値、ストリング、またはブール値です。

ActionScript には、コア言語の一部であるビルトインクラスが多数あります。Number、Boolean、String などのビルトインクラスは、ActionScript で使用可能なプリミティブ値を表します。Array、Math、XML などのその他のクラスは、ECMAScript 規格に定められている、より複雑なオブジェクトを定義します。

ビルトインおよびユーザー定義のすべてのクラスは、Object クラスから派生します。旧バージョンの ActionScript に慣れているプログラマは、他のクラスはすべて Object クラスから派生しますが、Object データ型はデフォルトのデータ型ではなくなったことに注意する必要があります。ActionScript 2.0 では、型注釈がないということは Object 型の変数になることを意味するため、次の 2 行のコードは同じでした。

```
var someObj:Object;
var someObj;
```

しかし、ActionScript 3.0 では、型指定されていない変数という概念が導入されました。これは、次の 2 つの方法で指定できます。

```
var someObj:*;
var someObj;
```

型指定されていない変数は、Object 型の変数と同じではありません。主な違いは、型指定されていない変数は特別な値 `undefined` を保持できますが、Object 型の変数はその値を保持できないことです。

`class` キーワードを使用して独自のクラスを定義することができます。クラスプロパティを宣言するには、3 つの方法があります。定数を定義するには `const` キーワード、変数を定義するには `var` キーワード、getter および setter プロパティを定義するにはメソッド宣言で `get` および `set` 属性を使用します。メソッドを宣言するには、`function` キーワードを使用します。

クラスのインスタンスを作成するには、`new` 演算子を使用します。次の例では、`myBirthday` という Date クラスのインスタンスを作成します。

```
var myBirthday:Date = new Date();
```

パッケージと名前空間

パッケージと名前空間は関連する概念です。パッケージを使用すると、コードを共有でき、名前のコンフリクトを最小限に抑えられるようにクラス定義をバンドルできます。名前空間を使用すると、プロパティ名やメソッド名などの識別子の可視性を制御でき、コードがパッケージ内部にあるのか外部にあるのかに関係なく適用できます。パッケージを使用してクラスファイルを構成でき、名前空間を使用して個々のプロパティおよびメソッドの可視性を制御できます。

パッケージ

ActionScript 3.0 では、パッケージは名前空間で実装されますが、名前空間と同義ではありません。パッケージを宣言すると、コンパイル時に必ず既知である名前空間が暗黙的に作成されます。名前空間は、明示的に作成された場合はコンパイル時に既知である必要はありません。

次の例では、package ディレクティブを使用して、クラスを 1 つ含む単純なパッケージを作成します。

```
package samples
{
   public class SampleCode
   {
      public var sampleGreeting:String;
      public function sampleFunction()
      {
         trace(sampleGreeting + " from sampleFunction()");
      }
   }
}
```

この例のクラスの名前は SampleCode です。クラスは samples パッケージ内にあるので、コンパイラはコンパイル時にクラス名を自動的に修飾して、完全修飾名の samples.SampleCode にします。また、コンパイラは、プロパティやメソッドの名前も修飾して、sampleGreeting および sampleFunction() がそれぞれ samples.SampleCode.sampleGreeting および samples.SampleCode.sampleFunction() になるようにします。

多くの開発者、特に Java 開発者は、パッケージの最上位にクラスだけを配置することを選択します。しかし、ActionScript 3.0 では、パッケージの最上位でクラスだけではなく、変数、関数、ステートメントもサポートされています。この機能の高度な使用方法として、パッケージ内のすべてのクラスで使用できるようにパッケージの最上位で名前空間を定義できます。ただし、パッケージの最上位で指定できるアクセス指定子は `public` と `internal` の 2 つだけです。ネストされたクラスをプライベートとして宣言できる Java とは異なり、ActionScript 3.0 ではネストされたクラスもプライベートクラスもサポートされていません。

しかし、その他の多くの点で、ActionScript 3.0 のパッケージは Java のパッケージと似ています。前の例を見ればわかるとおり、完全修飾パッケージ参照は、Java と同じように、ドット演算子 (.) を使用して表します。パッケージを使用すると、他のプログラマも使用できるように、直感的な階層構造にコードを構成できます。これにより、独自のパッケージを作成して他のプログラマと共有したり、他のプログラマが作成したパッケージを利用したりでき、コードの共有が容易になります。

パッケージを使用することで、使用する識別子名が一意になり、他の識別子名と競合しなくなります。実際、これがパッケージの最も重要な利点であるという人もいます。たとえば、コードを共有しようとしている 2 人のプログラマがそれぞれ SampleCode というクラスを作成したとします。パッケージを使用しなければ、名前の競合が発生し、いずれかのクラスの名前を変更するしかなくなります。しかし、パッケージを使用すると、クラスの一方、できれば両方を一意の名前のパッケージに配置することで、名前の競合は簡単に回避できます。

パッケージ名に埋め込みドットを入れて、ネストされたパッケージを作成することもできます。これにより、パッケージの階層構造を作成できます。その良い例は、Flash Player API によって提供される flash.xml パッケージです。flash.xml パッケージは、flash パッケージ内にネストされています。

flash.xml パッケージには、旧バージョンの ActionScript で使用されていた古い XML パーサーが含まれています。古い XML パーサーが flash.xml パッケージに置かれる理由の 1 つは、古い XML クラスの名前が、ActionScript 3.0 で使用可能な ECMAScript for XML 仕様の機能を実装する新しい XML クラスの名前と競合するためです。

古い XML クラスをパッケージに移動するのは最初の手順として適していますが、古い XML クラスを使用するほとんどの場合で、flash.xml パッケージが読み込まれます。この場合も、必ず古い XML クラスの完全修飾名 (flash.xml.XML) を使用しなければ、同じ名前の競合が発生します。この問題を回避するために、古い XML クラスは、次の例に示すように、XMLDocument という名前になりました。

```
package flash.xml
{
    class XMLDocument {}
    class XMLNode {}
    class XMLSocket {}
}
```

Flash Player API のほとんどは flash パッケージにあります。たとえば、flash.display パッケージには表示リスト API が含まれ、flash.events パッケージには新しいイベントモデルが含まれます。Flash Player API パッケージについては、本マニュアルの第 3 部で詳しく説明します。詳細については、335 ページの「Flash Player API」を参照してください。

パッケージの作成

ActionScript 3.0 では、パッケージ、クラス、およびソースファイルを整理する方法に大きな柔軟性を持たせています。旧バージョンの ActionScript では、ソースファイルごとにクラスが 1 つだけ許可され、ソースファイルの名前がクラスの名前と一致する必要がありました。ActionScript 3.0 では、1 つのソースファイルに複数のクラスを含めることができますが、ファイルの外部にあるコードで使用できるのは各ファイルのクラス 1 つだけです。つまり、各ファイルのクラスを 1 つだけパッケージ宣言内で宣言できます。その他のクラスはパッケージ定義の外部で定義する必要があります。これにより、追加されるクラスはソースファイル外部にあるコードに対して表示されなくなります。パッケージ定義内部で宣言されたクラスの名前は、ソースファイルの名前と一致する必要があります。

ActionScript 3.0 では、パッケージをより柔軟に宣言することができます。旧バージョンの ActionScript では、パッケージはソースファイルを配置するディレクトリを表すだけでした。また、`package` ステートメントでパッケージを宣言するのではなく、クラス宣言内の完全修飾クラス名の一部としてパッケージ名を含めていました。ActionScript 3.0 でもパッケージはディレクトリを表しますが、パッケージに含めることができるのはクラスだけではありません。ActionScript 3.0 では、`package` ステートメントを使用してパッケージを宣言します。つまり、パッケージの最上位で変数、関数、および名前空間を宣言することもできます。パッケージの最上位に実行可能ステートメントを含めることもできます。パッケージの最上位で変数、関数、または名前空間を宣言した場合は、そのレベルで使用できる属性は `public` および `internal` だけです。また、この宣言がクラス、変数、関数、または名前空間のいずれであるかに関係なく、`public` 属性を使用できるのはファイルごとにパッケージレベルの宣言 1 つだけです。

パッケージは、コードを構成したり、名前のコンフリクトを防いだりするのに便利です。パッケージの概念および関連のないクラス継承の概念を混同しないようにしてください。同じパッケージ内にある 2 つのクラスには共通する名前空間がありますが、この 2 つのクラスはその他で関連しているとは限りません。同様に、ネストされたパッケージは、親パッケージと意味的な関係がない場合があります。

パッケージの読み込み

パッケージ内部にあるクラスを使用する場合、パッケージまたは使用するクラスを読み込む必要があります。これは、クラスの読み込みがオプションであった ActionScript 2.0 とは異なります。

たとえば、この章で前述した SampleCode クラスの例を考えてみます。クラスが samples という名前のパッケージにある場合、SampleCode クラスを使用する前に次の import ステートメントのいずれかを使用する必要があります。

```
import samples.*;
```

または

```
import samples.SampleCode;
```

一般に、`import` ステートメントはできるだけ具体的で明確にする必要があります。`samples` パッケージから SampleCode クラスだけを使用する場合、属するパッケージ全体ではなく SampleCode クラスだけを読み込む必要があります。パッケージ全体を読み込むと、予期しない名前のコンフリクトが発生する可能性があります。

*クラスパス*内にパッケージまたはクラスを定義するソースコードを配置する必要もあります。クラスパスは、読み込むパッケージおよびクラスをコンパイラが検索する場所を決定するローカルディレクトリパスのユーザー定義リストです。クラスパスは " ビルドパス " または " ソースパス " と呼ばれることもあります。

クラスまたはパッケージを適切に読み込むと、クラスの完全修飾名 (samples.SampleCode) またはクラス名のみ (SampleCode) のいずれかを使用できます。

完全修飾名は同じ名前のクラス、メソッド、またはプロパティが存在してコードがあいまいになるときに役立ちますが、すべての識別子に使用すると管理が困難になる可能性があります。たとえば、SampleCode クラスのインスタンスをインスタンス化するときに完全修飾名を使用すると、コードが冗長になります。

```
var mySample:samples.SampleCode = new samples.SampleCode();
```

ネストされたパッケージのレベルが上がると、コードが読みにくくなります。あいまいな識別子が問題ではないことが確実な場合、単純な識別子を使用してコードを読みやすくすることができます。たとえば、クラス識別子のみを使用する場合、SampleCode クラスの新しいインスタンスのインスタンス化はかなり簡単になります。

```
var mySample:SampleCode = new SampleCode();
```

最初に適切なパッケージまたはクラスを読み込まずに識別子名を使用しようとすると、コンパイラはクラス定義を見つけることができません。また、パッケージまたはクラスを読み込んだ場合は、読み込んだ名前と競合する名前を定義しようとすると、エラーが生成されます。

パッケージの作成時には、そのパッケージのすべてのメンバーのデフォルトのアクセス制御子は internal です。この場合、デフォルトではパッケージメンバーは、そのパッケージの他のメンバーに対してのみ表示されます。パッケージ外部のコードでクラスを使用できるようにする場合、public でクラスを宣言する必要があります。たとえば、次のパッケージには、SampleCode と CodeFormatter の 2 つのクラスが含まれています。

```
// SampleCode.as ファイル
package samples
{
    public class SampleCode {}
}

// CodeFormatter.as ファイル
package samples
{
    class CodeFormatter {}
}
```

SampleCode クラスは、public クラスとして宣言されているため、パッケージ外部に対して表示されます。しかし、CodeFormatter クラスは、samples パッケージ内にのみ表示されます。samples パッケージ外部の CodeFormatter クラスにアクセスしようとすると、次の例に示すように、エラーが生成されます。

```
import samples.SampleCode;
import samples.CodeFormatter;
var mySample:SampleCode = new SampleCode(); // OK、public クラス
var myFormatter:CodeFormatter = new CodeFormatter(); // エラー
```

パッケージ外部で両方のクラスを使用する場合、両方のクラスを public として宣言する必要があります。パッケージ宣言に public 属性を適用することはできません。

完全修飾名は、パッケージを使用するときに発生する可能性がある名前のコンフリクトを解決する際に役立ちます。たとえば、同じ識別子でクラスを定義する 2 つのパッケージを読み込む場合などです。ここで次のパッケージについて考えてみます。このパッケージにも SampleCode という名前のクラスがあります。

```
package langref.samples
{
    public class SampleCode {}
}
```

両方のクラスを読み込んだ場合、次のように、SampleCode クラスを参照すると名前の競合が発生します。

```
import samples.SampleCode;
import langref.samples.SampleCode;
var mySample:SampleCode = new SampleCode(); // 名前の競合
```

コンパイラには、どちらの SampleCode クラスを使用するのかを知る方法はありません。この競合を解決するには、次のように各クラスの完全修飾名を使用する必要があります。

```
var sample1:samples.SampleCode = new samples.SampleCode();
var sample2:langref.samples.SampleCode = new langref.samples.SampleCode();
```

メモ C++ を使用していたプログラマは、import ステートメントと #include を混同しがちです。C++ コンパイラは一度に 1 つのファイルを処理するため、C++ では #include ディレクティブが必要で、このディレクティブはヘッダファイルが明示的に含まれていない限り、他のファイル内でクラス定義を探しません。ActionScript 3.0 には include ディレクティブがありますが、クラスやパッケージを読み込むように設計されていません。ActionScript 3.0 でクラスまたはパッケージを読み込むには、import ステートメントを使用して、パッケージを含むソースファイルをクラスパスに配置する必要があります。

名前空間

名前空間では、作成したプロパティおよびメソッドの可視性を制御することができます。public、private、protected、および internal アクセス制御指定子は、ビルトイン名前空間のようなものです。これらのあらかじめ定義されているアクセス制御指定子が要件を満たさない場合は、独自の名前空間を作成することができます。

ActionScript 実装のシンタックスおよび詳細は XML とは少し異なりますが、XML 名前空間を使い慣れていれば、ここでの説明は新しいものではないかもしれません。これまで名前空間を操作したことがない場合は、概念自体は単純ですが、その実装には特定の用語を習得する必要があります。

名前空間の動作を理解するために、プロパティまたはメソッドの名前には常に識別子と名前空間の 2 つの部分が含まれることを知っておくと役立ちます。識別子は名前のようなものと考えることができます。たとえば、次のクラス定義の識別子は、sampleGreeting と sampleFunction() です。

```
class SampleCode
{
   var sampleGreeting:String;
   function sampleFunction () {
      trace(sampleGreeting + " from sampleFunction()");
   }
}
```

定義の前に名前空間属性が付けられていない場合、その名前はデフォルトの internal 名前空間によって修飾されます。この場合、定義は同じパッケージ内の呼び出し元に対してのみ表示されます。コンパイラが strict モードに設定されている場合は、名前空間属性を変更せずに internal 名前空間がすべての識別子に適用されることを示す警告がコンパイラから出されます。識別子をどこでも使用できるようにするためには、識別子名の前に public 属性を付ける必要があります。前のコード例では、sampleGreeting および sampleFunction() の両方が名前空間値 internal を持ちます。

名前空間を使用するときは、次の 3 つの手順に従います。最初に、namespace キーワードを使用して名前空間を定義する必要があります。たとえば、次のコードは、version1 名前空間を定義します。

```
namespace version1;
```

次に、プロパティまたはメソッドの宣言にアクセス制御指定子ではなく、定義した名前空間を使用して、その名前空間を適用します。次の例では、myFunction() という関数を version1 名前空間に配置します。

```
version1 function myFunction() {}
```

3 番目に、名前空間を適用したら、use ディレクティブを使用するか、名前空間で識別子の名前を修飾して、名前空間を参照できます。次の例では、use ディレクティブから myFunction() 関数を参照します。

```
use namespace version1;
myFunction();
```

次の例に示すように、修飾名を使用して myFunction() 関数を参照することもできます。

```
version1::myFunction();
```

名前空間の定義

名前空間には、名前空間名とも呼ばれる URI (Uniform Resource Identifier) という値が含まれます。URI によって、名前空間定義を一意にすることができます。

次のいずれかの方法で名前空間定義を宣言して、名前空間を作成します。XML 名前空間を定義するように、明示的な URI で名前空間を定義することも、URI を省略することもできます。次の例では、URI を使用して名前空間を定義する方法を示します。

```
namespace flash_proxy = “http://www.adobe.com/flash/proxy”;
```

URI は、名前空間の一意な ID ストリングになります。次の例のように URI を省略した場合、コンパイラは URI の代わりに一意の内部 ID ストリングを作成します。この内部 ID ストリングにはアクセスできません。

```
namespace flash_proxy;
```

URI を使用するか、使用しないで名前空間を定義すると、その名前空間は同じスコープ内で再定義することはできません。同じスコープ内で以前に定義された名前空間を定義しようとすると、コンパイルエラーが発生します。

名前空間がパッケージまたはクラス内で定義された場合は、適切なアクセス制御指定子が使用されていなければ、名前空間はそのパッケージまたはクラスの外部にあるコードに対して表示されません。たとえば、次のコードは flash.utils パッケージ内で定義された flash_proxy 名前空間を示します。次の例では、アクセス制御指定子がないと、flash_proxy 名前空間は flash.utils パッケージ内のコードに対してのみ表示され、パッケージ外部のコードに対しては表示されないことを示します。

```
package flash.utils
{
    namespace flash_proxy;
}
```

次のコードは、public 属性を使用して、flash_proxy 名前空間をパッケージ外部のコードに対して表示されるようにします。

```
package flash.utils
{
    public namespace flash_proxy;
}
```

名前空間の適用

名前空間を適用するとは、名前空間に定義を配置することです。名前空間に配置できる定義には、関数、変数、および定数があります。カスタム名前空間にクラスを配置することはできません。

たとえば、public アクセス制御名前空間で宣言された関数があるとします。関数定義に public 属性を使用すると、関数はパブリックの名前空間に配置され、すべてのコードで利用できるようになります。名前空間を定義した後は、public 属性を使用する場合と同様に定義した名前空間を使用でき、カスタム名前空間を参照できるコードでその定義を使用できるようになります。たとえば、次の例に示すように、名前空間 example1 を定義すると、example1 を属性として使用して myFunction() というメソッドを追加できます。

```
namespace example1;
class someClass
{
    example1 myFunction() {}
}
```

名前空間 example1 を属性として使用して myFunction() メソッドを宣言すると、このメソッドは example1 名前空間に属すことになります。

名前空間を適用する際には、次の点に注意してください。

- 適用できる名前空間は宣言ごとに 1 つだけです。
- 名前空間属性を一度に複数の定義に適用することはできません。つまり、10 個の異なる関数に名前空間を適用する場合、10 個の異なる関数それぞれに名前空間を属性として追加する必要があります。
- 名前空間とアクセス制御指定子は相互に排他的であるため、名前空間を適用すると、アクセス制御指定子を指定することはできません。つまり、名前空間を適用すると、`public`、`private`、`protected`、または `internal` として関数またはプロパティを宣言することはできません。

名前空間の参照

`public`、`private`、`protected`、`internal` などのアクセス制御名前空間で宣言されたメソッドまたはプロパティを使用するとき、名前空間を明示的に参照する必要はありません。このような特別な名前空間へのアクセスはコンテキストによって制御されるからです。たとえば、`private` 名前空間に配置された定義は、自動的に同じクラス内のコードで利用できるようになります。しかし、定義した名前空間では、こうした状況依存性は存在しません。カスタム名前空間に配置したメソッドまたはプロパティを使用するには、その名前空間を参照する必要があります。

`use namespace` ディレクティブで名前空間を参照できます。また、名前修飾子 (`::`) を使用して名前空間で名前を修飾できます。`use namespace` ディレクティブで名前空間を参照すると、修飾されていない識別子に名前空間を適用できるように、名前空間が " 開き " ます。たとえば、`example1` 名前空間を定義した場合、`use namespace example1` を使用してその名前空間内の名前にアクセスできます。

```
use namespace example1;
myFunction();
```

一度に複数の名前空間を開くことができます。`use namespace` で名前空間を開くと、名前空間はそれ自体が開かれたコードブロック全体で開かれたままになります。名前空間を明示的に閉じることはできません。

しかし、複数の名前空間を開くと、名前のコンフリクトが起こりやすくなります。名前空間を開かない場合は、メソッドまたはプロパティの名前を名前空間と名前修飾子で修飾すると、`use namespace` ディレクティブを使用する必要はありません。たとえば、次のコードは、`example1` 名前空間で名前 `myFunction()` を修飾する方法を示します。

```
example1::myFunction();
```

名前空間の使用

Flash Player API の一部である flash.utils.Proxy クラスで名前のコンフリクトを防ぐために使用する名前空間の実際例を参照できます。ActionScript 2.0 の `Object.__resolve` プロパティに代わる Proxy クラスを使用すると、エラーが発生する前に、未定義のプロパティまたはメソッドへの参照を取得できます。名前の競合を防ぐために、Proxy クラスのすべてのメソッドは `flash_proxy` 名前空間内にあります。

`flash_proxy` 名前空間の使用方法についての理解を深めるために、Proxy クラスの使用方法を理解する必要があります。Proxy クラスの機能は、Proxy クラスを継承するクラスでのみ使用できます。つまり、オブジェクト上で Proxy クラスのメソッドを使用する場合、そのオブジェクトのクラス定義は Proxy クラスを拡張する必要があります。たとえば、未定義のメソッドへの呼び出しの試みを取得する場合、Proxy クラスを拡張し、Proxy クラスの `callProperty()` メソッドをオーバーライドします。

名前空間の実装には通常、名前空間の定義、適用、および参照の 3 つの手順を実行します。しかし、Proxy クラスのメソッドを明示的に呼び出していないため、`flash_proxy` 名前空間は定義および適用されるだけで、参照されません。Flash Player API では、`flash_proxy` 名前空間を定義し、Proxy クラスに適用します。コードは、`flash_proxy` 名前空間を Proxy クラスを拡張するクラスに適用するだけです。

`flash_proxy` 名前空間は、次のような方法で flash.utils パッケージで定義されます。

```
package flash.utils
{
    public namespace flash_proxy;
}
```

Proxy クラスからの次の抜粋に示すように、名前空間は Proxy クラスのメソッドに適用されます。

```
public class Proxy
{
    flash_proxy function callProperty(name:*, ... rest):*
    flash_proxy function deleteProperty(name:*):Boolean
    ...
}
```

次のコードに示すように、最初に Proxy クラスと `flash_proxy` 名前空間の両方を読み込む必要があります。次に、Proxy クラスを拡張するようにクラスを宣言する必要があります。strict モードでコンパイルしている場合は、`dynamic` 属性を追加する必要もあります。`callProperty()` メソッドをオーバーライドするときは、`flash_proxy` 名前空間を使用する必要があります。

```
package
{
```

```
  import flash.utils.Proxy;
  import flash.utils.flash_proxy;

  dynamic class MyProxy extends Proxy
  {
    flash_proxy override function callProperty(name:*, ...rest):*
    {
      trace("method call intercepted: " + name);
    }
  }
}
```

MyProxy クラスのインスタンスを作成し、次の例で呼び出されている testing() メソッドなどの未定義のメソッドを呼び出す場合は、Proxy オブジェクトは、メソッドの呼び出しを取得し、オーバーライドされた callProperty() メソッド内のステートメントを実行します。この例では、単純な trace() ステートメントを実行します。

```
var mySample:MyProxy = new MyProxy();
mySample.testing(); // method call intercepted: testing
```

flash_proxy 名前空間内に Proxy クラスのメソッドを配置すると、2 つの利点があります。1 つ目は、別の名前空間があると、Proxy クラスを拡張するクラスのパブリックインターフェイスを整理できます。Proxy クラスには、オーバーライドできるメソッドが 10 個以上ありますが、これらのメソッドは直接呼び出せるように設計されていません。これらすべてをパブリック名前空間に配置すると、混乱の元になります。2 つ目は、Proxy サブクラスに Proxy クラスのメソッドと一致する名前のインスタンスメソッドが含まれている場合、flash_proxy 名前空間を使用すると、名前のコンフリクトを回避することができます。たとえば、独自のメソッド callProperty() に名前を付けるとします。この callProperty() メソッドは別の名前空間にあるため、次のコードは有効です。

```
dynamic class MyProxy extends Proxy
{
  public function callProperty() {}
  flash_proxy override function callProperty(name:*, ...rest):*
  {
    trace("method call intercepted: " + name);
  }
}
```

名前空間は、4 つのアクセス制御指定子 (public、private、internal、および protected) ではできない方法でメソッドまたはプロパティにアクセスする場合にも便利です。たとえば、複数のパッケージに分散するいくつかのユーティリティメソッドがあるとします。これらのメソッドをパブリックにしないで、すべてのパッケージで利用できるようにするには、新しい名前空間を作成して、独自のアクセス制御指定子として使用します。

次の例では、ユーザー定義の名前空間を使用して、異なるパッケージにある 2 つの関数をグループ化します。同じ名前空間でグループ化することで、use namespace ステートメントを 1 つ使用して、クラスまたはパッケージに対してこの 2 つの関数を表示できます。

この例では、4 つのファイルを使用してその手法を示します。ファイルはすべてクラスパス内にある必要があります。1 つ目のファイル myInternal.as を使用して myInternal 名前空間を定義します。このファイルは example パッケージにあるため、example という名前のフォルダに配置する必要があります。名前空間は public とマークされているので、他のパッケージに読み込むことができます。

```
// example フォルダ内の myInternal.as
package example
{
  public namespace myInternal = "http://www.adobe.com/2006/actionscript/
  examples";
}
```

2 つ目および 3 つ目のファイル "Utility.as" と "Helper.as" は、他のパッケージで利用できるメソッドを含むクラスを定義します。Utility クラスは example.alpha パッケージ内にあります。つまり、example フォルダのサブフォルダである alpha という名前のフォルダにファイルを配置する必要があります。Helper クラスは example.beta パッケージ内にあります。つまり、example フォルダのサブフォルダでもある beta という名前のフォルダにファイルを配置する必要があります。example.alpha および example.beta パッケージは両方とも名前空間を使用する前に読み込む必要があります。

```
// example/alpha フォルダ内の Utility.as
package example.alpha
{
  import example.myInternal;
  use namespace myInternal;

  public class Utility
  {
    private static var _taskCounter:int = 0;

    public static function someTask()
    {
      _taskCounter++;
    }

    myInternal static function get taskCounter():int
    {
      return _taskCounter;
    }
  }
}

// "example/beta" フォルダ内の Helper.as
package example.beta
{
```

```
    import example.myInternal;
    use namespace myInternal;

    public class Helper
    {
        private static var _timeStamp:Date;

        public static function someTask()
        {
            _timeStamp = new Date();
        }

        myInternal static function get lastCalled():Date
        {
            return _timeStamp;
        }
    }
}
```

4 つ目のファイル "NamespaceUseCase.as" はメインアプリケーションクラスです。このファイルは "example" フォルダの兄弟である必要があります。NamespaceUseCase クラスも `myInternal` 名前空間を読み込み、この名前空間を使用して他のパッケージにある 2 つの静的メソッドを呼び出します。この例では、コードを簡略化するためにのみ静的メソッドを使用します。静的メソッドおよびインスタンスメソッドは両方とも `myInternal` 名前空間に配置することができます。

```
// NamespaceUseCase.as
package
{
    import flash.display.MovieClip;
    import example.myInternal;          // 名前空間の読み込み
    import example.alpha.Utility;       // Utility クラスの読み込み
    import example.beta.Helper;         // Helper クラスの読み込み

    use namespace myInternal;

    public class NamespaceUseCase extends MovieClip
    {
        public function NamespaceUseCase()
        {
            Utility.someTask();
            Utility.someTask();
            trace(Utility.taskCounter); // 2

            Helper.someTask();
            trace(Helper.lastCalled);   // [someTask() が最後に呼び出された時刻 ]
        }
    }
}
```

変数

変数を使用すると、プログラムで使用する値を格納できます。変数を宣言するには、変数名を使用した var ステートメントを使用する必要があります。ActionScript 2.0 では、型注釈を使用する場合にのみ var ステートメントを使用する必要があります。ActionScript 3.0 では、常に var ステートメントを使用する必要があります。たとえば、ActionScript の次の行は、i という名前の変数を宣言します。

```
var i;
```

変数を宣言するときに var ステートメントを省略した場合、strict モードではコンパイルエラーが発生し、standard モードではランタイムエラーが発生します。たとえば、次のコード行は、変数 i があらかじめ定義されていない場合はエラーになります。

```
i; // があらかじめ定義されていない場合はエラー
```

変数をデータ型に関連付ける場合は、変数を宣言するときに行います。変数の型を指定しないで変数を宣言することができますが、strict モードではコンパイラ警告が生成されます。変数の型を指定するには、変数の名前にコロン (:) と変数の型を追加します。たとえば、次のコードは int 型の変数 i を宣言します。

```
var i:int;
```

変数に値を割り当てるには、代入演算子 (=) を使用します。たとえば、次のコードは、変数 i を宣言し、その変数に値 20 を割り当てます。

```
var i:int;
i = 20;
```

次の例のように、変数の宣言と同時に変数に値を割り当てる方が便利な場合があります。

```
var i:int = 20;
```

変数の宣言時に変数に値を割り当てる方法は通常、整数やストリングなどのプリミティブ値を割り当てる場合だけではなく、配列を作成する、またはクラスのインスタンスをインスタンス化する場合にも使用されます。次の例は、コードを 1 行使用して宣言され、値が割り当てられる配列を示します。

```
var numArray:Array = ["zero", "one", "two"];
```

クラスのインスタンスを作成するには、new 演算子を使用します。次の例では、CustomClass というクラスのインスタンスを作成し、新しく作成されたクラスインスタンスへの参照を customItem という変数に割り当てます。

```
var customItem:CustomClass = new CustomClass();
```

複数の変数を宣言する場合、カンマ演算子 (,) を使用して変数を区切ることで、コード 1 行ですべての変数を宣言することができます。たとえば、次のコードは、コード 1 行で 3 つの変数を宣言します。

```
var a:int, b:int, c:int;
```

同じコード行で、各変数に値を割り当てることもできます。たとえば、次のコードは、3 つの変数 (a、b、および c) を宣言し、各変数に値を割り当てます。

```
var a:int = 10, b:int = 20, c:int = 30;
```

カンマ演算子を使用して変数宣言を 1 つのステートメントにまとめることができますが、コードが読みにくくなることがあります。

変数スコープについて

変数の " スコープ " とは、レキシカル参照によって変数にアクセスできるコードの範囲です。" グローバル " 変数は、コードのすべての範囲で定義される変数で、" ローカル " 変数は、コードの一部分でのみ定義される変数です。ActionScript 3.0 では、変数は常にその変数が宣言された関数またはクラスにスコープ設定されます。グローバル変数は、関数またはクラスの定義の外部で定義する変数です。たとえば、次のコードは、関数の外部で宣言して、グローバル変数 strGlobal を作成します。この例では、グローバル変数は関数定義の内部および外部の両方で利用できることを示します。

```
var strGlobal:String = "Global";
function scopeTest ()
{
    trace(strGlobal); // Global
}
scopeTest();
trace(strGlobal); // Global
```

ローカル変数を宣言するには、関数定義内で変数を宣言します。ローカル変数を定義できるコードの最小範囲が関数定義です。関数内で宣言されたローカル変数は、その関数内でのみ存在します。たとえば、変数 str2 を関数 localScope() の中で宣言すると、この変数は関数の外部では利用できません。

```
function localScope()
{
    var strLocal:String = "local";
}
localScope();
trace(strLocal); // strLocal はグローバルに定義されていないためエラー
```

ローカル変数に使用した変数名がグローバル変数として宣言されている場合、ローカル変数がスコープ内にある間は、ローカル定義がグローバル定義を非表示に (シャドウ) します。ただし、グローバル変数は関数の外でも存在しています。たとえば、次のコードは、グローバルストリング変数 str1 を作成した後、同じ名前のローカル変数を scopeTest() 関数内に作成します。関数内の trace ステートメントはこの変数のローカル値を出力しますが、関数の外にある trace ステートメントはこの変数のグローバル値を出力します。

```
var str1:String = "Global";
function scopeTest ()
{
    var str1:String = "Local";
    trace(str1); // Local
}
scopeTest();
trace(str1); // Global
```

C++ や Java の変数とは異なり、ActionScript の変数にはブロックレベルのスコープはありません。コードブロックは、左中括弧 ({) と右中括弧 (}) で囲まれたステートメントのグループです。C++ や Java などのプログラミング言語では、コードブロック内で宣言された変数は、コードブロック外では利用できません。このスコープ制限はブロックレベルのスコープと呼ばれますが、ActionScript にはありません。コードブロック内で変数を宣言すると、変数はそのコードブロック内だけではなく、コードブロックが属する関数の他の部分でも利用できます。たとえば、次の関数にはさまざまなブロックスコープで定義された変数が含まれています。すべての変数を関数全体で利用できます。

```
function blockTest (testArray:Array)
{
    var numElements:int = testArray.length;
    if (numElements > 0)
    {
        var elemStr:String = "Element #";
        for (var i:int = 0; i < numElements; i++)
        {
            var valueStr:String = i + ": " + testArray[i];
            trace(elemStr + valueStr);
        }
        trace(elemStr, valueStr, i);    // すべて定義済みのまま
    }
    trace(elemStr, valueStr, i); // if numElements > 0 の場合、すべて定義済み
}

blockTest(["Earth", "Moon", "Sun"]);
```

ブロックレベルのスコープがないということは、関数が終了する前に変数が宣言されていれば、宣言される前に変数の読み書きが可能であるということです。これは、" ホイスト " と呼ばれる手法によるもので、コンパイラによりすべての変数宣言が関数の最上位に移動されます。たとえば、次のコードは、num 変数が宣言される前に num 変数の初期の trace() 関数が実行されてもコンパイルされます。

```
trace(num); // NaN
var num:Number = 10;
trace(num); // 10
```

しかし、コンパイラは代入ステートメントをホイストしません。このため、num の初期の trace() は数値データ型の変数のデフォルト値である NaN (非数) になります。つまり、次の例に示すように、変数が宣言される前でも変数に値を割り当てることができます。

```
num = 5;
trace(num); // 5
var num:Number = 10;
trace(num); // 10
```

デフォルト値

" デフォルト値 " とは、値を設定する前に変数に格納されている値です。変数に初めて値を設定する場合は、変数を " 初期化 " します。変数を宣言して、値を設定しない場合、その変数は初期化されません。初期化されていない変数の値は、そのデータ型によって異なります。次の表に、データ型別の変数のデフォルト値を示します。

データ型	デフォルト値
Boolean	false
int	0
Number	NaN
Object	null
String	null
uint	0
宣言されていない (型注釈 * と同じ)	undefined
ユーザー定義クラスを含むその他すべてのクラス	null

Number 型の変数の場合、デフォルト値は NaN (非数) です。これは、IEEE-754 規格で定義されている特別な値で、数値を表さない値です。

変数を宣言して、そのデータ型を宣言しない場合、デフォルトのデータ型 * が適用されます。これは、変数の型指定がないことを示します。また、型指定されていない変数を値で初期化しない場合、デフォルト値は `undefined` です。

Boolean、Number、int、および uint 以外のデータ型の場合、初期化されていない変数のデフォルト値は `null` です。これは、Flash Player API で定義されたすべてのクラスおよび作成したカスタムクラスに適用されます。

値 `null` は、Boolean、Number、int、および uint 型の変数では有効な値ではありません。値 `null` をこれらの変数に割り当てようとすると、この値はそのデータ型のデフォルト値に変換されます。Object 型の変数の場合、値 `null` を割り当てることができます。値 `undefined` を Object 型の変数に割り当てようとすると、この値は `null` に変換されます。

Number 型の変数の場合、`isNaN()` という名前の特別なトップレベル関数があります。この関数は、変数が数値ではない場合はブール値 `true` を返し、それ以外の場合は `false` を返します。

データ型

" データ型 " は値のセットを定義します。たとえば、Boolean データ型は、`true` と `false` の 2 つの値のセットです。ActionScript 3.0 では、Boolean データ型に加えて、String、Number、Array などのよく使用されるデータ型が定義されています。独自のデータ型を定義するには、クラスまたはインターフェイスを使用して値のカスタムセットを定義します。ActionScript 3.0 のすべての値は、プリミティブ値または複合値にかかわらず、オブジェクトです。

プリミティブ値 は、Boolean、int、Number、String、および uint のデータ型のいずれかに属す値です。ActionScript では、プリミティブ値はメモリとスピードを最適化できるように特別に格納されるので、通常プリミティブ値は複合値より速く操作できます。

メモ：技術的に説明すると、ActionScript ではプリミティブ値はイミュータブルオブジェクトとして内部的に格納されます。イミュータブルオブジェクトとして格納されるということは、参照渡しが値渡しと同じように効果的であることを示しています。参照は通常、値自体よりかなり小さいので、これによりメモリ使用量が削減し、実行速度が向上します。

" 複合値 " とはプリミティブ値ではない値です。複合値のセットを定義するデータ型には、Array、Date、Error、Function、RegExp、XML、XMLList などがあります。

多くのプログラム言語では、プリミティブ値とそのラッパーオブジェクトは区別されています。たとえば、Java には int プリミティブとそれをラップする java.lang.Integer クラスがあります。Java のプリミティブはオブジェクトではありませんが、そのラッパーはオブジェクトです。このため、処理によってはプリミティブを使用する方が便利な場合とラッパーオブジェクトを使用する方が適切な場合とがあります。ActionScript 3.0 では、プリミティブ値とそのラッパーオブジェクトは実際には区別できません。プリミティブ値を含むすべての値はオブジェクトです。Flash Player では、これらのプリミティブ型は、オブジェクトのように動作しながら、オブジェクトを作成するための通常のオーバーヘッドが不要な特殊なケースとして扱われます。つまり、次の 2 行のコードは同じです。

```
var someInt:int = 3;
var someInt:int = new int(3);
```

前述のプリミティブおよび複合データ型はすべて ActionScript 3.0 コアクラスによって定義されています。コアクラスを使用すると、`new` 演算子を使用する代わりにリテラル値を使用してオブジェクトを作成できます。たとえば、次のようにリテラル値または Array クラスコンストラクタを使用して配列を作成できます。

```
var someArray:Array = [1, 2, 3]; // リテラル値
var someArray:Array = new Array(1,2,3); // Array コンストラクタ
```

型チェック

型チェックは、コンパイル時または実行時のいずれかの時点で行われます。C++ や Java のような型指定を静的に行う言語では、コンパイル時に型チェックを行います。Smalltalk や Python のような型指定を動的に行う言語では、実行時に型チェックを行います。ActionScript 3.0 は型指定を動的に行う言語なので、実行時に型チェックが行われますが、"strict モード " と呼ばれる特別なコンパイラモードでのコンパイル時型チェックもサポートされています。strict モードでは、型チェックはコンパイル時と実行時の両方で行われますが、standard モードでは、型チェックは実行時にのみ行われます。

型指定を動的に行う言語では、コードを非常に柔軟に構成できる反面、型指定エラーが実行時に明らかになるという難点があります。型指定を静的に行う言語では、コンパイル時に型指定エラーが報告されますが、コンパイル時に型情報が分かっている必要があるという難点があります。

コンパイル時の型チェック

コンパイル時の型チェックは、大規模なプロジェクトでよく利用されます。プロジェクトの規模が拡大するにつれて、通常はデータ型の柔軟性よりできるだけ早い時点で型指定エラーを把握する方が重要になるためです。Adobe Flash CS3 および Adobe Flex Builder 2 の ActionScript コンパイラがデフォルトで strict モードで実行されるように設定されているのはそのためです。Flex Builder 2 で strict モードを無効にするには、[プロジェクトのプロパティ] ダイアログボックスの ActionScript コンパイラ設定から行います。

コンパイル時に型チェックを行うためには、コンパイラがコード内の変数または式のデータ型情報を分かっている必要があります。変数のデータ型を明示的に宣言するには、変数名の接尾辞としてコロン演算子 (:) とその後ろにデータ型を追加します。データ型をパラメータに関連付けるには、コロン演算子の後ろにデータ型を使用します。たとえば、次のコードは、xParam パラメータにデータ型情報を追加し、明示的なデータ型で変数 myParam を宣言します。

```
function runtimeTest(xParam:String)
{
  trace(xParam);
}
var myParam:String = “hello”;
runtimeTest(myParam);
```

strict モードでは、ActionScript コンパイラは型の不一致をコンパイルエラーとして報告します。たとえば、次のコードは、Object 型の関数パラメータ xParam を宣言した後で、このパラメータに String 型と Number 型の値を割り当てようとします。これは、strict モードではコンパイルエラーが発生します。

```
function dynamicTest(xParam:Object)
{
  if (xParam is String)
  {
    var myStr:String = xParam; // strict モードではコンパイルエラーが発生します
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam; // strict モードではコンパイルエラーが発生します
    trace("Number: " + myNum);
  }
}
```

しかし、strict モードでも、代入ステートメントの右辺の型指定をしないでおくことで、コンパイル時の型チェックを選択しないようにすることができます。変数または式に型指定なしとマークするには、型注釈を省略するか、特別なアスタリスク (*) 型注釈を使用します。たとえば、前の例の xParam パラメータが変更されて型注釈がなくなった場合、コードは strict モードでコンパイルされます。

```
function dynamicTest(xParam)
{
  if (xParam is String)
  {
    var myStr:String = xParam;
    trace("String: " + myStr);
  }
  else if (xParam is Number)
  {
    var myNum:Number = xParam;
    trace("Number: " + myNum);
  }
}
```

```
dynamicTest(100)
dynamicTest("one hundred");
```

実行時の型チェック

ActionScript 3.0 では、実行時の型チェックは、strict モードと standard モードのいずれでコンパイルされるかに関係なく行われます。配列を必要とする関数に値 3 がパラメータとして渡されるとします。strict モードでは、値 3 は Array データ型と互換性がないため、コンパイラはエラーを生成します。strict モードを無効にし、standard モードで実行すると、コンパイラは型の不一致について報告しませんが、Flash Player による実行時の型チェックではライタイムエラーになります。

次の例は、Array パラメータを必要とする関数 `typeTest()` に値 3 が渡されることを示しています。値 3 はパラメータの宣言されたデータ型 (Array) のメンバーではないため、standard モードではランタイムエラーになります。

```
function typeTest(xParam:Array)
{
   trace(xParam);
}
var myNum:Number = 3;
typeTest(myNum);
// ActionScript 3.0 standard モードではラインタイムエラー
```

strict モードで操作しているときでも、実行時の型指定エラーが発生する場合もあります。これは strict モードを使用しているときに、型指定されていない変数を使用してコンパイル時の型チェックをしないようにしている場合に発生します。型指定されていない変数を使用すると、型チェックは省略されるのではなく、実行時まで保留されます。たとえば、前の例の `myNum` 変数に宣言されたデータ型がない場合、コンパイラは型の不一致を検出できませんが、Flash Player でランタイムエラーが生成されます。これは、Flash Player で代入ステートメントの結果 3 に設定されている `myNum` のランタイム値と Array データ型に設定されている `xParam` の型が比較されるためです。

```
function typeTest(xParam:Array)
{
   trace(xParam);
}
var myNum = 3;
typeTest(myNum);
// ActionScript 3.0 でランタイムエラー
```

実行時に型チェックを行うと、コンパイル時に型チェックを行うより継承を柔軟に使用することができます。standard モードでは、型チェックを実行時まで保留することで、" アップキャスト " する場合でもサブクラスのプロパティを参照できます。アップキャストは、基本クラスを使用してクラスインスタンスの型を宣言するときに行われますが、サブクラスはクラスインスタンスをインスタンス化します。たとえば、拡張可能な ClassBase というクラスを作成できます。ただし、final 属性を持つクラスは拡張できません。

```
class ClassBase
{
}
```

その後、次のように someString というプロパティを 1 つ持つ ClassExtender という ClassBase クラスのサブクラスを作成できます。

```
class ClassExtender extends ClassBase
{
    var someString:String;
}
```

両方のクラスを使用して、ClassBase データ型を使用して宣言され、ClassExtender コンストラクタを使用してインスタンス化されたクラスインスタンスを作成できます。基本クラスにはサブクラスにないプロパティやメソッドは含まれないため、アップキャストは安全な操作と考えられています。

```
var myClass:ClassBase = new ClassExtender();
```

しかし、サブクラスには、基本クラスには含まれないプロパティやメソッドが含まれます。たとえば、ClassExtender クラスには someString プロパティが含まれますが、このプロパティは ClassBase クラスには含まれません。ActionScript 3.0 の standard モードでは、次の例に示すように、myClass インスタンスを使用してコンパイル時エラーを生成せずにこのプロパティを参照することができます。

```
var myClass:ClassBase = new ClassExtender();
myClass.someString = "hello";
// ActionScript 3.0 standard モードではエラーなし
```

is 演算子

ActionScript 3.0 で新しく導入された is 演算子を使用すると、変数または式が特定のデータ型のメンバーであるかどうかをテストすることができます。旧バージョンの ActionScript では instanceof 演算子に同じ機能がありましたが、ActionScript 3.0 では instanceof 演算子はデータ型のメンバーシップのテストに使用しません。手動による型チェックでは、instanceof 演算子の代わりに is 演算子を使用する必要があります。これは、式 x instanceof y は、x のプロトタイプチェーンに y があるかどうかをチェックするだけであるため、また ActionScript 3.0 では、プロトタイプチェーンで継承階層全体を確認できないためです。

is 演算子は、適切な継承階層を調べます。また、オブジェクトが特定のクラスのインスタンスであるかどうかだけではなく、特定のインターフェイスを実装するクラスのインスタンスであるかどうかも調べるために使用することができます。次の例では、Sprite クラス mySprite のインスタンスを作成し、is 演算子を使用して mySprite が Sprite クラスおよび DisplayObject クラスのインスタンスであるかどうか、および IEventDispatcher インターフェイスを実装しているかどうかをテストします。

```
var mySprite:Sprite = new Sprite();
trace(mySprite is Sprite);                // true
trace(mySprite is DisplayObject);         // true
trace(mySprite is IEventDispatcher);      // true
```

is 演算子は、継承階層を調べて、mySprite が Sprite クラスおよび DisplayObject クラスと互換性があることを適切に報告します。ただし、Sprite クラスは DisplayObject クラスのサブクラスです。is 演算子は、mySprite が IEventDispatcher インターフェイスを実装するクラスを継承するかどうかも調べます。Sprite クラスは、IEventDispatcher インターフェイスを実装する EventDispatcher クラスを継承するため、is 演算子は、mySprite が同じインターフェイスを実装することを正しく報告します。

次の例は、前の例と同じテストを示していますが、is 演算子の代わりに instanceof 演算子を使用しています。instanceof 演算子は、mySprite が Sprite または DisplayObject のインスタンスであることを正しく識別しますが、mySprite が IEventDispatcher インターフェイスを実装しているかどうかの確認に使用された場合は、false を返します。

```
trace(mySprite instanceof Sprite);           // true
trace(mySprite instanceof DisplayObject);    // true
trace(mySprite instanceof IEventDispatcher); // false
```

as 演算子

ActionScript 3.0 で新しく追加された as 演算子を使用しても、式が指定されたデータ型のメンバーであるかどうかをチェックすることができます。しかし、is 演算子とは異なり、as 演算子はブール値を返しません。as 演算子は、true の代わりに式の値、false の代わりに null を返します。次の例は、Sprite インスタンスが DisplayObject、IEventDispatcher、および Number データ型のメンバーであるかどうかをチェックする場合に、is 演算子の代わりに as 演算子を使用した結果を示します。

```
var mySprite:Sprite = new Sprite();
trace(mySprite as Sprite); // [ オブジェクト Sprite]
trace(mySprite as DisplayObject); // [ オブジェクト Sprite]
trace(mySprite as IEventDispatcher); // [ オブジェクト Sprite]
trace(mySprite as Number);   // null
```

as 演算子を使用する場合、右側のオペランドはデータ型である必要があります。右側のオペランドにデータ型ではなく式を使用しようとすると、エラーになります。

ダイナミッククラス

" ダイナミック " クラスは、プロパティおよびメソッドを追加したり変更したりすることで実行時に変更可能なオブジェクトを定義します。String クラスなどの動的ではないクラスは、"sealed" クラスです。実行時に、sealed クラスにプロパティまたはメソッドを追加することはできません。

ダイナミッククラスを作成するには、クラスを宣言するときに dynamic 属性を使用します。たとえば、次のコードは Protean という名前のダイナミッククラスを作成します。

```
dynamic class Protean
{
   private var privateGreeting:String = "hi";
   public var publicGreeting:String = "hello";
   function Protean()
   {
      trace("Protean instance created");
   }
}
```

その後に Protean クラスのインスタンスをインスタンス化した場合、クラス定義の外側にあるそのインスタンスにプロパティまたはメソッドを追加できます。たとえば、次のコードは、Protean クラスのインスタンスを作成し、プロパティ aString とプロパティ aNumber をそのインスタンスに追加します。

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
trace(myProtean.aString, myProtean.aNumber); // testing 3
```

ダイナミッククラスのインスタンスに追加したプロパティは、実行時エンティティです。したがって、型チェックは実行時に行われます。この方法で追加されたプロパティに型注釈を追加することはできません。

また、myProtean インスタンスにメソッドを追加するには、関数を定義し、その関数を myProtean インスタンスのプロパティに関連付けます。次のコードは、trace ステートメントを traceProtean() というメソッドに移動します。

```
var myProtean:Protean = new Protean();
myProtean.aString = "testing";
myProtean.aNumber = 3;
myProtean.traceProtean = function ()
{
   trace(this.aString, this.aNumber);
};
myProtean.traceProtean(); // testing 3
```

しかし、この方法で作成されたメソッドは、Protean クラスのプライベートプロパティまたはメソッドにアクセスできません。また、`Protean` クラスのパブリックプロパティまたはメソッドへの参照も、`this` キーワードまたはクラス名のいずれかで修飾する必要があります。次の例は、`Protean` クラスのプライベート変数およびパブリック変数にアクセスしようとする `traceProtean()` メソッドを示します。

```
myProtean.traceProtean = function ()
{
  trace(myProtean.privateGreeting); // undefined
  trace(myProtean.publicGreeting); // hello
};
myProtean.traceProtean();
```

データ型の詳細

プリミティブデータ型には、Boolean、int、Null、Number、String、uint、および void があります。ActionScript コアクラスでは、Object、Array、Date、Error、Function、RegExp、XML、および XMLList の複合データ型も定義されています。

Boolean データ型

Boolean データ型は、`true` と `false` の 2 つの値で構成されます。Boolean 型の変数に有効な値は、この 2 つだけです。宣言されていて、初期化されていない Boolean 型変数のデフォルト値は `false` です。

int データ型

int データ型は、32 ビット整数として内部的に格納され、-2,147,483,648 (-2^{31}) 以上 2,147,483,647 ($2^{31} - 1$) 以下の整数のセットで構成されます。旧バージョンの ActionScript には、整数と浮動小数点数の両方に使用する Number データ型しかありませんでした。ActionScript 3.0 では、32 ビット符号付および符号なし整数の低レベルマシン型にアクセスすることができるようになりました。変数が浮動小数点数を使用しない場合、Number データ型ではなく int データ型を使用する方が速く、効率的です。

最小 int 値と最大 int 値の範囲外の整数値の場合は、Number データ型を使用します。Number データ型では、-9,007,199,254,740,992 ～ +9,007,199,254,740,992 の値 (53 ビット整数値) を処理できます。int データ型の変数のデフォルト値は 0 です。

Null データ型

Null データ型の値は `null` しかありません。これは、String データ型および Object クラスを含む複合データ型を定義するすべてのクラスのデフォルト値です。Boolean、Number、int、uint などの他のプリミティブデータ型には、値 `null` は含まれません。Flash Player では、Boolean、Number、int、または uint 型の変数に値 `null` を割り当てようとすると、`null` は該当するデフォルト値に変換されます。このデータ型は、型注釈として使用することはできません。

Number データ型

ActionScript 3.0 では、Number データ型は整数、符号なし整数、および浮動小数点数を表すことができます。ただし、パフォーマンスを最大化するために、32 ビット `int` および `uint` 型より大きい整数値に対してのみ、Number データ型を使用する必要があります。また、Number データ型には、浮動小数点数を格納することができます。浮動小数点数を格納するには、数値に小数点を含めます。小数点を省略すると、数値は整数として格納されます。

Number データ型では、2 進数浮動小数点計算のための IEEE 規格 (IEEE-754) で指定されている 64 ビット倍精度フォーマットを使用します。この規格では、64 ビットを使用して浮動小数点数を格納する方法が指定されています。1 ビットを使用して数値が正か負かを指定します。11 ビットを 2 を底として格納される指数に使用します。残りの 52 ビットを使用して " 仮数 " を格納します。仮数とは指数で示される累乗される数値です。

Number データ型では、ビットの一部を使用して指数を格納することで、仮数のビットすべてを使用する場合より大きな浮動小数点数を格納できます。たとえば、Number データ型で 64 ビットすべてを使用して仮数を格納した場合、2^{64} の数値を格納できます。Number データ型では、11 ビットを使用して指数を格納することで、仮数を 2^{1023} 乗できます。

Number 型で表すことができる最大値と最小値は、`Number.MAX_VALUE` および `Number.MIN_VALUE` という Number クラスの静的プロパティに格納されます。

```
Number.MAX_VALUE == 1.79769313486231e+308
Number.MIN_VALUE == 4.940656458412467e-324
```

この数値の範囲は非常に大きくなりますが、この範囲の問題は精度です。Number データ型では、仮数の格納に 52 ビットを使用します。その結果、分数 1/3 など、正確に表すためには 53 ビット以上必要な数値は近似値にしかなりません。アプリケーションで 10 進法による絶対精度が必要な場合、2 進数浮動小数点計算ではなく、10 進法浮動小数点計算を実装するアプリケーションを使用する必要があります。

Number データ型で整数値を格納する場合、仮数の 52 ビットだけが使用されます。
Number データ型はこの 52 ビットと特別な隠しビットを使用して、-9,007,199,254,740,992 (-2^{53}) ～ 9,007,199,254,740,992 (2^{53}) の整数を表します。

Flash Player では、`Number` 型の変数のデフォルト値としてだけではなく、数値を返す必要があるが返さない演算の結果としても、`NaN` 値を使用します。たとえば、負の数値の平方根を計算しようとすると、その結果は `NaN` になります。その他の特別な Number 型の値には、正の無限大および負の無限大があります。

メモ	`0` による除算の結果は、除数も `0` の場合、`NaN` だけです。`0` による除算の結果は、被除数が正の場合は `infinity`、被除数が負の場合は `-infinity` が生成されます。

String データ型

String データ型は、16 ビット文字の連続を表します。ストリングは、UTF-16 形式を使用して、Unicode 文字として内部的に格納されます。ストリングは、Java プログラミング言語の場合と同様に不変値です。String 値を操作すると、ストリングの新しいインスタンスが返されます。String データ型で宣言される変数のデフォルト値は `null` です。値 `null` と空のストリング (" ") は、両方とも文字が存在しないことを表しますが、同じものではありません。

uint データ型

uint データ型は、32 ビット符号なし整数として内部的に格納され、0 以上 4,294,967,295 ($2^{32}-1$) 以下の整数のセットで構成されます。負以外の整数が必要な特別な場合に、uint データ型を使用します。たとえば、ピクセルカラー値を表すためには uint データ型を使用する必要があります。これは、int データ型にはカラー値の処理には適していない内部符号ビットがあるためです。最大 uint 値より大きな整数値には、53 ビット整数値を処理できる Number データ型を使用します。uint データ型の変数のデフォルト値は 0 です。

void データ型

void データ型の値は `undefined` だけです。旧バージョンの ActionScript では、`undefined` は Object クラスのインスタンスのデフォルト値でした。ActionScript 3.0 では、Object インスタンスのデフォルト値は `null` です。値 `undefined` を Object クラスのインスタンスに割り当てようとすると、Flash Player ではこの値は `null` に変換されます。型指定されていない変数には、`undefined` という値のみを割り当てることができます。型指定されていない変数は、型注釈がないか、型注釈にアスタリスク記号 (*) が使用されている変数です。`void` は戻り値の型注釈としてのみ使用することができます。

Object データ型

Object データ型は Object クラスによって定義されます。Object クラスは、ActionScript のすべてのクラス定義の基本クラスです。ActionScript 3.0 の Object データ型は、次の 3 つの点で旧バージョンとは異なります。1 つ目は、Object データ型は、型注釈のない変数に割り当てられるデフォルトのデータ型ではなくなりました。2 つ目は、Object データ型には、Object インスタンスのデフォルト値であった値 `undefined` が含まれなくなりました。3 つ目は、ActionScript 3.0 では、Object クラスのインスタンスのデフォルト値は `null` です。

旧バージョンの ActionScript では、型注釈のない変数には自動的に Object データ型が割り当てられていました。ActionScript 3.0 ではこれは行われず、真に型指定されていない変数の概念が導入されました。型注釈のない変数は、型が指定されていないと見なされます。コード内で変数の型を指定しないままにしておくことを明確に示す場合、型注釈に新しくアスタリスク記号 (*) を使用することができます。これは、型注釈を省略するのと同じことを示します。次の例は、型指定されていない変数 `x` を宣言する 2 つの同じステートメントを示します。

```
var x
var x:*
```

型指定されていない変数にのみ値 `undefined` を割り当てることができます。値 `undefined` をデータ型が指定されている変数に割り当てようとすると、Flash Player は値 `undefined` をそのデータ型のデフォルト値に変換します。Object データ型のインスタンスのデフォルト値は `null` です。つまり、`undefined` を Object インスタンスに割り当てようとすると、Flash Player は値 `undefined` を `null` に変換します。

型変換

型変換は、値が別のデータ型の値に変換されることです。型変換は、" 暗黙的 " または " 明示的 " に行うことができます。暗黙的な変換は、" 強制型変換 " とも呼ばれ、実行時に Flash Player によって行われることがあります。たとえば、値 2 が Boolean データ型の変数に割り当てられると、Flash Player は値 2 をブール値 `true` に変換してから、変数に割り当てます。明示的な変換は、" キャスト " とも呼ばれ、コードであるデータ型の変数を別のデータ型に属している場合と同様に処理するようにコンパイラに指示されると実行されます。プリミティブ値が使用されている場合、キャストによって実際にあるデータ型の値が別のデータ型に変換されます。オブジェクトを異なる型にキャストするには、オブジェクト名を括弧で囲み、その前に新しい型の名前を置きます。たとえば、次のコードはブール値を整数にキャストします。

```
var myBoolean:Boolean = true;
var myINT:int = int(myBoolean);
trace(myINT); // 1
```

暗黙的な変換

暗黙的な変換は、次のような状況で実行時に行われます。

- 代入ステートメント内
- 値が関数パラメータとして渡されるとき
- 関数から値が返されるとき
- 加算 (+) 演算子などの特定の演算子を使用した式内

ユーザー定義型の場合、変換される値が変換先のクラスまたは変換先のクラスから派生するクラスのインスタンスである場合に、暗黙的な変換は成功します。暗黙的な変換が成功しなかった場合は、エラーが発生します。たとえば、次のコードには、成功する暗黙的な変換と失敗する暗黙的な変換が含まれています。

```
class A {}
class B extends A {}

var objA:A = new A();
var objB:B = new B();
var arr:Array = new Array();

objA = objB; // 変換は成功。
objB = arr; // 変換は失敗。
```

プリミティブ型の場合、暗黙的な変換は、明示的な変換関数によって呼び出される同じ内部変換アルゴリズムを呼び出して処理されます。以降のセクションでは、プリミティブ型の変換について詳しく説明します。

明示的な変換

strict モードでコンパイルする場合、型の不一致によるコンパイル時エラーを生成したくない場合があるため、明示的な変換、つまりキャストを使用すると便利です。これは、実行時に強制的に値が適切に変換されることがわかっている場合などです。たとえば、フォームから受信したデータを操作するとき、強制的にストリング値を数値に変換することができます。次のコードは、standard モードでコードは正しく実行されますが、コンパイル時エラーを生成します。

```
var quantityField:String = "3";
var quantity:int = quantityField; // strict モードではコンパイル時にエラー
```

strict モードを引き続き使用して、ストリングを整数に変換する場合、次のように明示的な変換を使用することができます。

```
var quantityField:String = "3";
var quantity:int = int(quantityField); // 明示的な変換は成功。
```

int、uint、Number 型にキャスト

任意のデータ型を int、uint、および Number の 3 つの数値型のいずれかにキャストすることができます。何らかの理由で Flash Player で数値を変換できない場合、int および uint データ型ではデフォルト値 0 が割り当てられ、Number データ型ではデフォルト値 NaN が割り当てられます。ブール値を数値に変換する場合、true は値 1 になり、false は値 0 になります。

```
var myBoolean:Boolean = true;
var myUINT:uint = uint(myBoolean);
var myINT:int = int(myBoolean);
var myNum:Number = Number(myBoolean);
trace(myUINT, myINT, myNum); // 1 1 1
myBoolean = false;
myUINT = uint(myBoolean);
myINT = int(myBoolean);
myNum = Number(myBoolean);
trace(myUINT, myINT, myNum); // 0 0 0
```

数字のみを含むストリング値は、数値型のいずれかに変換することができます。数値型では、負の数のように見えるストリングまたは 16 進数値 (たとえば、0x1A) を表すストリングも変換することができます。変換プロセスでは、ストリング値の先頭および末尾の空白は無視されます。Number() を使用して浮動小数点数のように見えるストリングをキャストすることもできます。小数点が含まれていると、uint() および int() は、小数点の後の数字が切り捨てられた整数を返します。たとえば、次のストリング値は数値にキャストすることができます。

```
trace(uint("5"));     // 5
trace(uint("-5"));    // 4294967291. It wraps around from MAX_VALUE
trace(uint(" 27 "));  // 27
trace(uint("3.7"));   // 3
trace(int("3.7"));    // 3
trace(int("0x1A"));   // 26
trace(Number("3.7")); // 3.7
```

数値以外の文字を含むストリング値は、int() または uint() でキャストすると 0 を返し、Number() でキャストすると NaN を返します。変換プロセスでは、先頭および末尾の空白は無視されますが、ストリング値に 2 つの数値を区切る空白がある場合は、0 または NaN が返されます。

```
trace(uint("5a"));    // 0
trace(uint("ten"));   // 0
trace(uint("17 63")); // 0
```

ActionScript 3.0 では、Number() 関数は 8 進数をサポートしていません。ActionScript 2.0 の Number() 関数に先頭が 0 のストリングを指定した場合、数値は 8 進数として解釈され、10 進数に変換されます。これは、ActionScript 3.0 の Number() 関数には当てはまりません。この関数では、先頭の 0 は無視されます。たとえば、次のコードは、異なるバージョンの ActionScript を使用してコンパイルされると、異なる出力を生成します。

```
trace(Number("044"));
// ActionScript 3.0 44
// ActionScript 2.0 36
```

数値型の値が別の数値型の変数に割り当てられる場合、キャストは必要ありません。strict モードでも、数値型は別の数値型に暗黙的に変換されます。これは、場合によっては、型の範囲を超えると予期しない値になることを示します。次の例では、予期しない値が生成される場合もありますが、すべて strict モードでコンパイルされます。

```
var sampleUINT:uint = -3; // int 型と Number 型の値を割り当てる。
trace(sampleUINT); // 4294967293

var sampleNum:Number = sampleUINT; // int 型と unit 型の値を割り当てる。
trace(sampleNum) // 4294967293

var sampleINT:int = uint.MAX_VALUE + 1; // Number 型の値を割り当てる。
trace(sampleINT); // 0

sampleINT = int.MAX_VALUE + 1; // uint 型と Number 型の値を割り当てる。
trace(sampleINT); // -2147483648
```

次の表に、別のデータ型から Number、int、uint データ型にキャストした結果の概要を示します。

データ型または値	Number、int、uint 型への変換結果
Boolean	値が true の場合は 1、それ以外の場合は 0
Date	Date オブジェクトの内部表現で、1970 年 1 月 1 日午前 0 時 (世界時) からのミリ秒
null	0
Object	インスタンスが null で Number 型に変換される場合は NaN、それ以外の場合は 0
ストリング	Flash Player でストリングを数値に変換できる場合は数値、それ以外の場合は Number 型に変換されると NaN、int 型または uint 型に変換されると 0
undefined	Number 型に変換される場合は NaN、int 型または uint 型に変換される場合は 0

Boolean 型へのキャスト

数値データ型 (uint、int、および Number) から Boolean 型にキャストすると、数値が 0 の場合は false、それ以外の場合は true になります。Number データ型では、値 NaN のときも false になります。次の例は、数値 -1、0、および 1 をキャストした結果を示します。

```
var myNum:Number;
for (myNum = -1; myNum<2; myNum++)
{
  trace("Boolean(" + myNum +") is " + Boolean(myNum));
}
```

この例の出力は、3 つの数値のうち 0 だけが値 false を返すことを示します。

```
Boolean(-1) is true
Boolean(0) is false
Boolean(1) is true
```

String 型から Boolean 型にキャストすると、ストリングが null または空のストリング ("") の場合は false を返します。それ以外の場合は、true を返します。

```
var str1:String;       // 初期化されていないストリングは null。
trace(Boolean(str1));  // false

var str2:String = "";  // 空のストリング
trace(Boolean(str2));  // false

var str3:String = " "; // 空白のみ
trace(Boolean(str3));  // true
```

Object クラスのインスタンスから Boolean 型にキャストすると、次の例に示すように、インスタンスが null の場合は false、それ以外の場合は true が返されます。

```
var myObj:Object;      // 初期化されていないオブジェクトは null。
trace(Boolean(myObj)); // false

myObj = new Object();  // インスタンス化
trace(Boolean(myObj)); // true
```

Boolean 型の変数は、strict モードで任意のデータ型の値をキャストしないで Boolean 型変数に割り当てることができます。すべてのデータ型から Boolean データ型への暗黙的な強制型変換は、strict モードでも行われます。つまり、ほとんどのデータ型とは異なり、strict モードのエラーを防ぐために Boolean 型へのキャストは必要ありません。次の例では、すべて strict モードでコンパイルされ、実行時に意図したとおりに動作します。

```
var myObj:Object = new Object();  // インスタンス化
var bool:Boolean = myObj;
trace(bool); // true
bool = "random string";
trace(bool); // true
bool = new Array();
trace(bool); // true
bool = NaN;
trace(bool); // false
```

次の表に、別のデータ型から Boolean データ型にキャストした結果の概要を示します。

データ型または値	Boolean 型への変換結果
ストリング	値が `null` または空のストリング ("") の場合は `false`、それ以外の場合は `true`
`null`	`false`
Number、int、または uint	値が NaN または 0 の場合は `false`、それ以外の場合は `true`
Object	インスタンスが null の場合は `false`、それ以外の場合は `true`

String 型へのキャスト

数値データ型から String データ型にキャストすると、数値のストリング表現が返されます。Boolean 型から String 型にキャストすると、値が `true` の場合はストリング “`true`”、値が `false` の場合はストリング “`false`” が返されます。

Object クラスのインスタンスから String データ型にキャストすると、インスタンスが `null` の場合はストリング “`null`” が返されます。それ以外の場合は、Object クラスから String 型にキャストすると、ストリング “`[object Object]`” が返されます。

Array クラスのインスタンスから String 型にキャストすると、すべての配列エレメントのカンマ区切りリストから構成されるストリングが返されます。たとえば、次の String データ型へのキャストは、配列の 3 つのエレメントすべてから成るストリングを 1 つ返します。

```
var myArray:Array = ["primary", "secondary", "tertiary"];
trace(String(myArray)); // primary,secondary,tertiary
```

Date クラスのインスタンスから String 型にキャストすると、インスタンスに含まれる日付のストリング表現が返されます。たとえば、次の例は Date クラスインスタンスのストリング表現を返します。この出力は、太平洋標準時の場合の結果を示します。

```
var myDate:Date = new Date(2005,6,1);
trace(String(myDate)); // 2005 年 7 月 1 日金曜日 00:00:00 GMT-0700
```

次の表に、別のデータ型から String データ型にキャストした結果の概要を示します。

データ型または値	String 型への変換結果
Array	すべての配列エレメントで構成されるストリング
Boolean	"true" または "false"
日付	Date オブジェクトのストリング表現
null	"null"
Number、int、または uint	数値のストリング表現
Object	インスタンスが null の場合は "null"、それ以外の場合は "[object Object]"。

シンタックス

言語のシンタックスは、実行可能なコードを記述するときに従う必要がある一連の規則を定義します。

大文字と小文字の区別

ActionScript 3.0 は、大文字と小文字を区別する言語です。スペルが同じで大文字か小文字かだけが異なる識別子は、異なる識別子と見なされます。たとえば、次のコードは異なる変数を 2 つ作成します。

```
var num1:int;
var Num1:int;
```

ドットシンタックス

ドット演算子 (.) は、オブジェクトのプロパティおよびメソッドにアクセスする方法を提供します。ドットシンタックスを使用すると、インスタンス名の後にドット演算子とプロパティまたはメソッドの名前を付けたものを使ってクラスのプロパティまたはメソッドを参照できます。たとえば、次のようなクラス定義があるとします。

```
class DotExample
{
    public var prop1:String;
```

```
    public function method1():void {}
}
```

ドットシンタックスを使用すると、次のコードで作成されたインスタンス名を使って prop1 プロパティおよび method1() メソッドにアクセスできます。

```
var myDotEx:DotExample = new DotExample();
myDotEx.prop1 = “hello”;
myDotEx.method1();
```

パッケージを定義するとき、ドットシンタックスを使用することができます。ドット演算子を使用してネストされたパッケージを参照します。たとえば、EventDispatcher クラスは、flash というパッケージ内でネストされている events というパッケージにあります。次の式を使用して、events パッケージを参照できます。

```
flash.events
```

この式を使用して、EventDispatcher クラスを参照することもできます。

```
flash.events.EventDispatcher
```

スラッシュシンタックス

ActionScript 3.0 では、スラッシュシンタックスはサポートされていません。旧バージョンの ActionScript では、ムービークリップや変数のパスを表すためにスラッシュシンタックスが使用されていました。

リテラル

" リテラル " は、コードに直接表示される値です。次の例はすべてリテラルです。

```
17
"hello"
-3
9.4
null
undefined
true
false
```

リテラルはグループ化して複合リテラルとすることもできます。配列リテラルは、角括弧 ([]) で囲まれ、カンマを使用して配列エレメントが区切られます。

配列リテラルを使用して配列を初期化することができます。次の例では、配列リテラルを使用して初期化される 2 つの配列を示します。new ステートメントを使用し、複合リテラルをパラメータとして Array クラスコンストラクタに渡すことができますが、Object、Array、String、Number、int、uint、XML、XMLList、Boolean の ActionScript コアクラスのインスタンスをインスタンス化すると、リテラル値を直接割り当てることもできます。

```
// 新しいステートメントを使用する
var myStrings:Array = new Array(["alpha", "beta", "gamma"]);
var myNums:Array = new Array([1,2,3,5,8]);

// 直接リテラルを割り当てる。
var myStrings:Array = ["alpha", "beta", "gamma"];
var myNums:Array = [1,2,3,5,8];
```

汎用オブジェクトを初期化するためにリテラルを使用することもできます。汎用オブジェクトは Object クラスのインスタンスです。オブジェクトリテラルは、中括弧 ({}) で囲まれ、カンマを使ってオブジェクトプロパティが区切られます。各プロパティはコロン (:) で宣言されます。コロンにより、プロパティ名とプロパティ値が区切られます。

new ステートメントを使用して汎用オブジェクトを作成し、オブジェクトリテラルをパラメータとして Object クラスコンストラクタに渡すか、宣言するインスタンスにオブジェクトリテラルを直接割り当てることができます。次の例では、新しい汎用オブジェクトを作成し、3 つのプロパティ (propA、propB、および propC) を使ってオブジェクトを初期化します。プロパティの値はそれぞれ 1、2、および 3 に設定されます。

```
// 新しいステートメントを使用する
var myObject:Object = new Object({propA:1, propB:2, propC:3});

// 直接リテラルを割り当てる。
var myObject:Object = {propA:1, propB:2, propC:3};
```

詳細については、209 ページの「ストリングの作成」、286 ページの「正規表現の概要」、および 319 ページの「XML 変数の初期化」を参照してください。

セミコロン

セミコロン (;) を使用してステートメントを終了することができます。セミコロンを省略した場合は、コンパイラはコードの各行が 1 つのステートメントであると見なします。セミコロンを使用してステートメントの終了を示すことに慣れているプログラマが多いため、セミコロンを使用してステートメントを終了するようにすると、コードが読みやすくなります。

セミコロンを使用してステートメントを終了すると、1 行に複数のステートメントを配置することができますが、こうするとコードが読みにくくなることがあります。

括弧

ActionScript 3.0 では、括弧 (( )) に 3 つの使用方法があります。1 つ目は、括弧を使用して、式内の演算の順序を変更します。括弧内にグループ化された演算は、常に最初に実行されます。たとえば、次のコードでは、括弧を使用して演算の順序を変更します。

```
trace(2 + 3 * 4);    // 14
trace( (2 + 3) * 4); // 20
```

2 つ目は、次の例に示すように、括弧にカンマ演算子 (,) を使用して、一連の式を評価し、最後に実行された式の結果を返します。

```
var a:int = 2;
var b:int = 3;
trace((a++, b++, a+b)); // 7
```

3 つ目は、次の例に示すように、括弧を使用して 1 つまたは複数のパラメータを関数またはメソッドに渡します。この例では、`trace()` 関数にストリング値を渡します。

```
trace("hello"); // hello
```

コメント

ActionScript 3.0 コードでは、単一行コメントと複数行コメントの 2 種類のコメントがサポートされています。このコメントメカニズムは、C++ や Java のコメントメカニズムに似ています。コンパイラは、コメントとしてマークされたテキストを無視します。

単一行コメントは、2 つのスラッシュ (//) で始まり、その行の最後までです。たとえば、次のコードには単一行コメントが含まれています。

```
var someNumber:Number = 3; // 単一行コメント
```

複数行コメントは、スラッシュとアスタリスク (/*) で始まり、アスタリスクとスラッシュ (*/) で終わります。

```
/* これは複数行のコードにまたがる
複数行コメント */
```

キーワードと予約語

予約語は、コードで識別子として使用することができない単語です。これは、ActionScript によって予約されているためです。予約語には、コンパイラによってプログラム名前空間から削除される " レキシカルキーワード " が含まれます。レキシカルキーワードを識別子として使用すると、コンパイラはエラーを報告します。次の表は、ActionScript 3.0 のレキシカルキーワードの一覧です。

as	break	case	catch
class	const	continue	default
delete	do	else	extends
false	finally	for	function
if	implements	import	in
instanceof	interface	internal	is
native	new	null	package
private	protected	public	return
super	switch	this	throw
to	true	try	typeof
use	var	void	while
with			

" シンタックスキーワード " と呼ばれるいくつかのキーワードがあります。このキーワードは識別子として使用できますが、コンテキストによっては特別な意味になります。次の表は、ActionScript 3.0 のシンタックスキーワードの一覧です。

each	get	set	namespace
include	dynamic	final	native
override	static		

さらに、" 将来の予約語 " と呼ばれる識別子もいくつかあります。これらの識別子は ActionScript 3.0 では予約されていませんが、その一部は、ActionScript 3.0 を組み込んだ製品でキーワードとして扱われることがあります。これらの識別子の多くはコードで使用可能ですが、使用しないことをお勧めします。以降のバージョンの言語でキーワードとして表示される可能性があるからです。

abstract	boolean	byte	cast
char	debugger	double	enum
export	float	goto	intrinsic
long	prototype	short	synchronized
throws	to	transient	type
virtual	volatile		

定数

ActionScript 3.0 では、定数を作成するために使用する `const` ステートメントがサポートされています。定数は、変更できない固定値を持つプロパティです。定数には値を 1 回だけ割り当てることができますが、定数の宣言のきわめて近くで割り当てる必要があります。たとえば、定数がクラスのメンバーとして宣言されると、宣言の一部としてのみ、またはクラスコンストラクタ内でのみ、その定数に値を割り当てることができます。

次のコードは、2 つの定数を宣言します。1 つ目の定数 MINIMUM には、宣言ステートメントの一部として値が割り当てられます。2 つ目の定数 MAXIMUM には、コンストラクタ内で値が割り当てられます。

```
class A
{
   public const MINIMUM:int = 0;
   public const MAXIMUM:int;

   public function A()
   {
      MAXIMUM = 10;
   }
}

var a:A = new A();
trace(a.MINIMUM); // 0
trace(a.MAXIMUM); // 10
```

他の方法で定数に初期値を割り当てようとすると、エラーが発生します。たとえば、クラス外部にある MAXIMUM の初期値を設定しようとすると、ランタイムエラーが発生します。

```
class A
{
  public const MINIMUM:int = 0;
  public const MAXIMUM:int;
}

var a:A = new A();
a["MAXIMUM"] = 10; // ランタイムエラー
```

Flash Player API では、さまざまな定数が定義されています。表記規則により、ActionScript の定数には大文字だけを使用し、単語はアンダースコア文字 (_) で区切ります。たとえば、MouseEvent クラス定義では、マウス入力に関連するイベントを表す定数にこの命名規則を使用します。

```
package flash.events
{
  public class MouseEvent extends Event
  {
    public static const CLICK:String              = "click";
    public static const DOUBLE_CLICK:String       = "doubleClick";
    public static const MOUSE_DOWN:String         = "mouseDown";
    public static const MOUSE_MOVE:String         = "mouseMove";
    ...
  }
}
```

演算子

演算子は、1 つまたは複数のオペランドを取り、値を返す特別な関数です。" オペランド " は、演算子が入力として使用する値で、通常はリテラル、変数、または式です。たとえば、次のコードでは、加算 (+) および乗算 (*) 演算子をリテラルオペランド (2、3、および 4）と共に使用して、値を返します。この値は、代入 (=) 演算子によって使用されて、戻り値 14 を変数 sumNumber に割り当てます。

```
var sumNumber:uint = 2 + 3 * 4; // uint = 14
```

演算子には、単項、二項、および三項があります。単項演算子は、オペランドを 1 つだけ取ります。たとえば、インクリメント (++) 演算子は、オペランドを 1 つだけ取るため、単項演算子です。二項演算子は、オペランドを 2 つ取ります。たとえば、除算 (/) 演算子はオペランドを 2 つ取ります。三項演算子はオペランドを 3 つ取ります。たとえば、条件 (?:) 演算子はオペランドを 3 つ取ります。

一部の演算子は " オーバーロード " されます。つまり、渡されたオペランドの種類または数に応じて動作が異なります。加算 (+) 演算子は、オペランドのデータ型によって動作が異なるオーバーロード演算子の例です。オペランドが両方とも数値の場合、加算演算子は値の合計値を返します。オペランドが両方ともストリングの場合、加算演算子は 2 つのオペランドの連結を返します。次のコード例は、オペランドによって演算子の動作がどう異なるのかを示します。

```
trace(5 + 5);     // 10
trace("5" + "5"); // 55
```

演算子は、指定されたオペランドの数によっても動作が異なります。除算 (-) 演算子は、単項および二項演算子です。オペランドが 1 つだけ指定されると、除算演算子はオペランドを否定し、その結果を返します。オペランドが 2 つ指定されると、除算演算子は 2 つのオペランドの差を返します。次の例では、最初に単項演算子、次に二項演算子として使用される除算演算子を示します。

```
trace(-3);  // -3
trace(7-2); // 5
```

演算子の優先順位と結合性

演算子の優先順位と結合性により、演算子を処理する順序が決まります。算術プログラミングに慣れている開発者にとって、コンパイラが乗算 (*) 演算子を加算 (+) 演算子より先に処理するのは自然なことですが、コンパイラは、どの演算子を最初に処理するかについて明示的な指示を必要とします。このような指示を、総称して " 演算子の優先順位 " と呼びます。ActionScript では、括弧 (( )) を使用して変更できる、演算子のデフォルトの優先順位が定義されています。たとえば、次のコードは、前の例のデフォルトの優先順位を変更し、コンパイラに強制的に乗算演算子の前に加算演算子を処理させます。

```
var sumNumber:uint = (2 + 3) * 4; // uint == 20
```

同じ優先順位の複数の演算子が同じ式にある状況があります。このような状況では、コンパイラは " 結合性 " のルールを使用して、最初に処理する演算子を決定します。代入演算子を除くすべての二項演算子は左結合となります。つまり、左にある演算子が右にある演算子よりも先に処理されます。代入演算子と条件 (?:) 演算子は " 右結合 " となります。つまり、右にある演算子が左にある演算子よりも先に処理されます。

たとえば、" より小さい " (<) および " より大きい " (>) 演算子について考えてみます。これらの演算子の優先順位は同じです。両方の演算子を同じ式で使用すると、左の演算子が最初に処理されます。これは、両方の演算子が左結合であるためです。つまり、次の 2 つのステートメントでは同じ出力が得られます。

```
trace(3 > 2 < 1);   // false
trace((3 > 2) < 1); // false
```

" より大きい " 演算子が最初に処理され、その結果は値 true になります。これはオペランド 3 がオペランド 2 より大きいためです。次に、値 true がオペランド 1 と共に " より小さい " 演算子に渡されます。次のコードは、この中間状態を表します。

```
trace((true) < 1);
```

" より小さい " 演算子は、値 true を数値 1 に変換し、その数値を 2 番目のオペランド 1 と比較して、値 false を返します。これは、値 1 が 1 より小さくないためです。

```
trace(1 < 1); // false
```

デフォルトの左結合を括弧 (( )) を使用して変更できます。演算子とオペランドを括弧で囲んで、" より小さい " 演算子を最初に処理するようにコンパイラに指示できます。次の例では、括弧演算子を使用して、前の例と同じ数値で異なる出力を生成します。

```
trace(3 > (2 < 1)); // true
```

" より小さい " 演算子が最初に処理され、その結果は値 false になります。これは、オペランド 2 がオペランド 1 より小さくないためです。次に、値 false がオペランド 3 と共に " より大きい " 演算子に渡されます。次のコードは、この中間状態を表します。

```
trace(3 > (false));
```

" より大きい " 演算子は、値 false を数値 0 に変換し、その数値をもう一方のオペランド 3 と比較して、true を返します。これは、値 3 は 0 より大きいためです。

```
trace(3 > 0); // true
```

次の表は、ActionScript 3.0 の演算子を優先順位の高いものから順に示します。同じ行に示されている演算子は優先順位が同じです。各行に示されている演算子はその下に表示される行の演算子より優先順位が高くなります。

グループ	演算子
基本	`[] {x:y} () f(x) new x.y x[y] <></> @ :: ..`
後置	`x++ x--`
単項	`++x --x + - ~ ! delete typeof void`
乗法	`* / %`

<table>
<tr><th>グループ</th><th>演算子</th></tr>
<tr><td>加算</td><td>+ -</td></tr>
<tr><td>ビット単位シフト</td><td><< >> >>></td></tr>
<tr><td>関係</td><td>< > <= >= as in instanceof is</td></tr>
<tr><td>等価</td><td>== != === !==</td></tr>
<tr><td>ビット単位の論理積 (AND)</td><td>&</td></tr>
<tr><td>ビット単位の排他的論理和 (XOR)</td><td>^</td></tr>
<tr><td>ビット単位の論理和 (OR)</td><td>|</td></tr>
<tr><td>論理積 (AND)</td><td>&&</td></tr>
<tr><td>論理和 (OR)</td><td>||</td></tr>
<tr><td>条件</td><td>?:</td></tr>
<tr><td>代入</td><td>= *= /= %= += -= <<= >>= >>>= &= ^= |=</td></tr>
<tr><td>カンマ</td><td>,</td></tr>
</table>

基本演算子

基本演算子は、Array リテラルおよび Object リテラルの作成、式のグループ化、関数の呼び出し、クラスインスタンスのインスタンス化、およびプロパティへのアクセスに使用する演算子です。

次の表の基本演算子の優先順位はすべて同じです。E4X 仕様に規定されている演算子には、(E4X) と表示されます。

演算子	実行される演算
[]	配列の初期化
{x:y}	オブジェクトの初期化
()	式のグループ化
f(x)	関数の呼び出し
new	コンストラクタの呼び出し
x.y x[y]	プロパティへのアクセス
<></>	XMLList オブジェクトの初期化 (E4X)
@	属性へのアクセス (E4X)
::	名前の修飾 (E4X)
..	子孫 XML エレメントへのアクセス (E4X)

後置演算子

後置演算子は、1 つの演算子を取り、演算子の値をインクリメントまたはデクリメントします。これらの演算子は単項演算子ですが、優先順位が高いことと、その特別なビヘイビアにより、他の単項演算子からは区別して分類されます。後置演算子をより大きい式の一部として使用する場合、後置演算子が処理される前に、式の値が返されます。たとえば、次のコードでは、式 xNum++ の値がインクリメントされる前に返されます。

```
var xNum:Number = 0;
trace(xNum++); // 0
trace(xNum);   // 1
```

次の表の後置演算子の優先順位はすべて同じです。

演算子	実行される演算
++	インクリメント (後置方式)
--	デクリメント (後置方式)

単項演算子

単項演算子は 1 つのオペランドを取ります。このグループのインクリメント (++) 演算子とデクリメント (--) 演算子は、" 前置演算子 " であり、式ではオペランドの前に表示されます。前置演算子が後置演算子と異なるのは、インクリメント演算またはデクリメント演算が、式全体の値が返される前に完了することです。たとえば、次のコードでは、式 ++xNum の値がインクリメントされた後に返されます。

```
var xNum:Number = 0;
trace(++xNum); // 1
trace(xNum);   // 1
```

次の表の単項演算子の優先順位はすべて同じです。

演算子	実行される演算
++	インクリメント (前置方式)
--	デクリメント (後置方式)
+	単項プラス
-	単項マイナス
!	論理否定 (NOT)
~	ビット単位の論理否定 (NOT)

演算子	実行される演算
削除	プロパティの削除
typeof	タイプ情報を返す
void	未定義の値を返す

乗法演算子

乗法演算子は 2 つのオペランドを取り、乗算、除算、または剰余計算を実行します。

次の表の乗法演算子の優先順位はすべて同じです。

演算子	実行される演算
*	乗算
/	除算
%	剰余

加算演算子

加算演算子は 2 つのオペランドを取り、加算または減算を実行します。次の表の加算演算子の優先順位はすべて同じです。

演算子	実行される演算
+	加算
-	減算

ビット単位シフト演算子

ビット単位シフト演算子は 2 つのオペランドを取り、2 番目のオペランドで指定された範囲で最初のオペランドのビットをシフトします。次の表のビット単位シフト演算子の優先順位はすべて同じです。

演算子	実行される演算
<<	ビット単位の左シフト
>>	ビット単位の右シフト
>>>	ビット単位の符号なし右シフト

関係演算子

関係演算子は 2 つのオペランドを取り、値を比較してブール値を返します。次の表の関係演算子の優先順位はすべて同じです。

演算子	実行される演算
<	より小さい
>	より大きい
<=	より小さいか等しい
>=	より大きいか等しい
as	データ型のチェック
in	オブジェクトプロパティのチェック
instanceof	プロトタイプチェーンのチェック
is	データ型のチェック

等価演算子

等価演算子は 2 つのオペランドを取り、値を比較してブール値を返します。次の表の等価演算子の優先順位はすべて同じです。

演算子	実行される演算
==	等価
!=	不等価
===	厳密な等価
!==	厳密な不等価

ビット論理演算子

ビット論理演算子は 2 つのオペランドを取り、ビットレベルの論理演算を実行します。ビット論理演算子は優先順位が異なりますが、優先順位の高いものから順に次の表に示します。

演算子	実行される演算
&	ビット単位の論理積 (AND)
^	ビット単位の排他的論理和 (XOR)
|	ビット単位の論理和 (OR)

論理演算子

論理演算子は 2 つのオペランドを取り、ブール値の結果を返します。論理演算子は優先順位が異なりますが、優先順位の高いものから順に次の表に示します。

演算子	実行される演算
`&&`	論理積 (AND)
`||`	論理和 (OR)

条件演算子

条件演算子は三項演算子であり、3 つのオペランドを取ります。条件演算子は、`if..else` 条件ステートメントを適用する簡易的な方法です。

演算子	実行される演算
`?:`	条件

代入演算子

代入演算子は 2 つのオペランドを取り、他のオペランドの値に基づいて 1 つのオペランドに値を割り当てます。次の表の代入演算子の優先順位はすべて同じです。

演算子	実行される演算
`=`	代入
`*=`	乗算後代入
`/=`	除算後代入
`%=`	剰余を代入
`+=`	加算後代入
`-=`	減算後代入
`<<=`	ビット単位での左シフト後代入
`>>=`	ビット単位での右シフト後代入
`>>>=`	ビット単位での符号なし右シフト後代入
`&=`	ビット単位の論理積 (AND) を代入
`^=`	ビット単位の論理積 (XOR) を代入
`|=`	ビット単位の論理和 (OR) を代入

条件

ActionScript 3.0 には、プログラムフローを制御するために使用する 3 つの基本的な条件ステートメントが用意されています。

if..else

`if..else` 条件ステートメントを使用すると、条件をテストし、その条件が満たされている場合はコードブロックを実行し、条件が満たされていない場合は別のコードブロックを実行することができます。たとえば、次のコードは、値 x が 20 を超えているかどうかをテストし、超えている場合は `trace()` 関数を生成し、超えていない場合は別の `trace()` 関数を生成します。

```
if (x > 20)
{
    trace("x is > 20");
}
else
{
    trace("x is <= 20");
}
```

別のコードブロックを実行しない場合は、`else` ステートメントのない `if` ステートメントを使用できます。

if..else if

if..else if 条件ステートメントを使用すると、複数の条件をテストできます。たとえば、次のコードは、x の値が 20 を超えるかどうかをテストすると共に、x が負の値かどうかをテストします。

```
if (x > 20)
{
  trace("x is > 20");
}
else if (x < 0)
{
  trace("x is negative");
}
```

if または else ステートメントに続くステートメントが 1 つのみの場合、そのステートメントを中括弧で囲む必要はありません。たとえば、次のコードは中括弧を使用していません。

```
if (x > 0)
  trace("x is positive”);
else if (x < 0)
  trace("x is negative");
else
  trace("x is 0");
```

ただし、常に中括弧を使用することをお勧めします。中括弧で囲まれない条件ステートメントに if ステートメントを後で追加したときに、予想しない動作が起きる可能性があるからです。たとえば、次のコードで positiveNums の値は、条件が true と評価されるかどうかに関係なく 1 増加します。

```
var x:int;
var positiveNums:int = 0;

if (x > 0)
  trace("x is positive");
  positiveNums++;

trace(positiveNums); // 1
```

switch

switch ステートメントは、同じ条件式に依存する実行パスが複数ある場合に便利です。一連の if..else if ステートメントに似た機能ですが、多少読みやすくなっています。switch ステートメントは、ブール値の条件をテストするのではなく、式を評価し、その結果を使用して実行するコードブロックを決定します。コードブロックは、case ステートメントで始まり、break ステートメントで終わります。たとえば、次の switch ステートメントは、Date.getDay() メソッドから返される曜日を表す数に基づいて曜日を印刷します。

```
var someDate:Date = new Date();
var dayNum:uint = someDate.getDay();
switch(dayNum)
{
    case 0:
        trace("Sunday");
        break;
    case 1:
        trace("Monday");
        break;
    case 2:
        trace("Tuesday");
        break;
    case 3:
        trace("Wednesday");
        break;
    case 4:
        trace("Thursday");
        break;
    case 5:
        trace("Friday");
        break;
    case 6:
        trace("Saturday");
        break;
    default:
        trace("Out of range");
        break;
}
```

ループ

ループステートメントを使用すると、一連の値または変数を使用して特定のコードブロックを繰り返し実行できます。コードブロックは常に中括弧 ({ }) で囲むことをお勧めします。コードブロックに含まれるステートメントが 1 つのみ場合は中括弧を省略できますが、条件の場合と同じ理由から省略することはお勧めしません。これは、後で追加したステートメントがコードブロックから誤って除外される可能性が高くなるためです。コードブロックに含めるステートメントを後で追加するときに、必要な中括弧の追加を忘れた場合、そのステートメントはループの一部としては実行されません。

for

for ループを使用すると、特定の範囲の値に対する変数の繰り返し処理を実行することができます。for ステートメントには、3 つの式を指定する必要があります。それらの式は、初期値に設定する変数、ループがいつ終了するかを指定する条件ステートメント、および各ループで変数の値を変更する式です。たとえば、次のコードは 5 回ループします。変数 i は 0 で始まり、4 で終わります。出力は 0 ～ 4 の数値となり、各数値は個別の行に出力されます。

```
var i:int;
for (i = 0; i < 5; i++)
{
   trace(i);
}
```

for..in

for..in ループは、オブジェクトのプロパティまたは配列のエレメントの繰り返し処理を実行します。たとえば、for..in ループを使用して、汎用オブジェクトのプロパティの繰り返し処理を実行できます。オブジェクトのプロパティは特定の順序では維持されないので、プロパティは予期しない順序で表示されることがあります。

```
var myObj:Object = {x:20, y:30};
for (var i:String in myObj)
{
   trace(i + ": " + myObj[i]);
}
// 出力 :
// x: 20
// y: 30
```

配列のエレメントの繰り返し処理を実行することもできます。

```
var myArray:Array = ["one", "two", "three"];
for (var i:String in myArray)
```

```
{
  trace(myArray[i]);
}
// 出力 :
// one
// two
// three
```

クラスがダイナミッククラスでない限り、オブジェクトがユーザー定義クラスのインスタンスである場合、そのプロパティの繰り返し処理を実行することはできません。ダイナミッククラスのインスタンスの場合であっても、動的に追加されるプロパティによってしか繰り返し処理を実行できません。

for each..in

`for each..in` ループは、XML または XMLList オブジェクトのタグ、オブジェクトプロパティが保持する値、または配列のエレメントのいずれかにすることができる、コレクションの項目の繰り返し処理を実行します。たとえば、次の抜粋に示すように、`for each..in` ループを使用して汎用オブジェクトのプロパティの繰り返し処理を実行できますが、`for..in` ループとは異なり、`for each..in` ループのイテレータ変数には、プロパティの名前ではなく、プロパティが保持する値が含まれます。

```
var myObj:Object = {x:20, y:30};
for each (var num in myObj)
{
  trace(num);
}
// 出力 :
// 20
// 30
```

次の例に示すように、XML または XMLList オブジェクトの繰り返し処理を実行することができます。

```
var myXML:XML = <users>
                  <fname>Jane</fname>
                  <fname>Susan</fname>
                  <fname>John</fname>
                </users>;

for each (var item in myXML.fname)
{
  trace(item);
}
/* 出力
Jane
Susan
John
*/
```

以下の例が示すように、配列のエレメントの繰り返し処理を実行することもできます。

```
var myArray:Array = ["one", "two", "three"];
for each (var item in myArray)
{
  trace(item);
}
// 出力 :
// one
// two
// three
```

オブジェクトが sealed クラスのインスタンスである場合、そのプロパティの繰り返し処理を実行することはできません。ダイナミッククラスのインスタンスの場合であっても、クラス定義の一部として定義されるプロパティである固定プロパティの繰り返し処理を実行することはできません。

while

`while` ループは、条件が `true` である場合に繰り返す `if` ステートメントに似ています。たとえば、次のコードは、`for` ループの例と同じ出力を作成します。

```
var i:int = 0;
while (i < 5)
{
  trace(i);
  i++;
}
```

`for` ループの代わりに `while` ループを使用する短所の 1 つは、`while` ループを使用すると無限ループを記述する可能性が高いことです。カウンタ変数をインクリメントする式を省略した場合、`for` ループのサンプルコードはコンパイルされませんが、`while` ループのサンプルコードはコンパイルされます。`i` をインクリメントする式がないと、ループは無限ループになります。

do..while

`do..while` ループは、コードブロックの実行後に条件がチェックされるため、コードブロックが少なくとも 1 回は実行される `while` ループです。次のコードでは、条件が満たされない場合でも出力を生成する `do..while` ループの簡単な例を示します。

```
var i:int = 5;
do
{
  trace(i);
  i++;
} while (i < 5);
// 出力 : 5
```

関数

" 関数 " は、特定のタスクを実行し、プログラム内で再利用できるコードブロックです。ActionScript 3.0 の関数には " メソッド " と " メソッドクロージャ " の 2 種類があります。関数をメソッドと呼ぶかメソッドクロージャと呼ぶかは、関数が定義されたコンテキストによって決まります。関数をクラス定義の一部として定義した場合、またはオブジェクトのインスタンスに関連付けた場合は、メソッドと呼びます。関数がその他の方法で定義された場合は、メソッドクロージャと呼びます。

ActionScript では、関数は非常に重要です。ActionScript 1.0 では、たとえば、`class` キーワードが存在しなかったので、" クラス " はコンストラクタ関数で定義されました。その後、`class` キーワードが追加されましたが、ActionScript をフルに活用するには、関数について理解しておくことが重要です。しかし、これは、ActionScript の関数が C++ や Java などの言語の関数と同じように動作することを期待するプログラマにとっては難しい場合があります。経験豊富なプログラマにとっては基本的な関数の定義や呼び出しは問題がありませんが、ActionScript の関数の高度な機能の中には説明が必要なものもあります。

基本的な関数の概念

このセクションでは、基本的な関数の定義と呼び出し方法について説明します。

呼び出し元の関数

関数を呼び出すには、識別子の後に括弧 (( )) を使用します。関数に渡す関数パラメータを括弧で囲みます。たとえば、`trace()` 関数は Flash Player API のトップレベル関数ですが、本マニュアルのさまざまな箇所で使用されています。

```
trace("Use trace to help debug your script");
```

パラメータのない関数を呼び出す場合は、空括弧を使用する必要があります。たとえば、パラメータを取らない `Math.random()` メソッドは、乱数を生成します。

```
var randomNum:Number = Math.random();
```

独自の関数の定義

ActionScript 3.0 の関数を定義するには、function ステートメントを使用する方法と関数式を使用する方法の、2 つの方法があります。どちらの方法を選択するかは、プログラミングスタイルをより静的にするか動的にするかによって決まります。静的な、つまり strict モードのプログラミングの方を好む場合は、function ステートメントで関数を定義します。そうすることに特定の必要性がある場合は、関数式で関数を定義します。関数式は、動的、つまり standard モードのプログラミングでより頻繁に使用されます。

Function ステートメント

function ステートメントは、strict モードで関数を定義するのに適しています。function ステートメントは、`function` キーワードで始まり、その後に以下が続きます。

- 関数の名前
- 括弧で囲まれたカンマ区切りリストで指定されたパラメータ
- 関数の本体、つまり関数が呼び出されると実行される、中括弧で囲まれた ActionScript コード

たとえば、次のコードは、パラメータを定義する関数を作成し、ストリング "`hello`" をパラメータ値として使用して、関数を呼び出します。

```
function traceParameter(aParam:String)
{
    trace(aParam);
}

traceParameter("hello"); // hello
```

関数式

関数を宣言する 2 つ目の方法は、代入ステートメントに関数式を使用することです。関数式は、関数リテラルまたは匿名関数とも呼ばれます。これは、旧バージョンの ActionScript で広く使用されている、より冗長になる方法です。

関数式を使用した代入ステートメントは、var キーワードで始まり、その後に以下が続きます。

- 関数の名前
- コロン演算子 (:)
- データ型を示すための Function クラス
- 代入演算子 (=)
- function キーワード
- 括弧で囲まれたカンマ区切りリストで指定されたパラメータ
- 関数の本体、つまり関数が呼び出されると実行される、中括弧で囲まれた ActionScript コード

たとえば、次のコードは関数式を使用して traceParameter 関数を宣言します。

```
var traceParameter:Function = function (aParam:String)
{
   trace(aParam);
};
traceParameter("hello"); // hello
```

function ステートメントとは異なり、関数の名前を指定していないことに注意してください。関数式が function ステートメントと異なるもう 1 つの特徴は、関数式はステートメントではなく式であることです。つまり、function ステートメントのように、関数式はそれだけでは成立しません。関数式は、通常は代入ステートメントなど、ステートメントの一部としてのみ使用することができます。次の例は、配列エレメントに代入された関数式を示します。

```
var traceArray:Array = new Array();
traceArray[0] = function (aParam:String)
{
   trace(aParam);
};
traceArray[0]("hello");
```

ステートメントと式の選択

原則として、式を使用する必要がある場合を除いて、function ステートメントを使用します。function ステートメントは、関数式より簡潔で、strict モードと standard モードで一貫した使いやすさを提供します。

function ステートメントは、関数式を含む代入ステートメントより読みやすくなります。function ステートメントを使用すると、コードが簡潔になり、var と function キーワードを両方使用する必要がある関数式よりわかりやすくなります。

function ステートメントは、2 つのコンパイラモードで一貫した使いやすさを提供します。つまり、strict モードおよび standard モードの両方でドットシンタックスを使用し、function ステートメントを使用して宣言されたメソッドを呼び出すことができます。これは、関数式を使用して宣言されたメソッドには必ずしも当てはまりません。たとえば、次のコードは、2 つのメソッドで Example というクラスを定義します。関数式で宣言される `methodExpression()` と function ステートメントで宣言される `methodStatement()` です。strict モードでは、ドットシンタックスを使用して `methodExpression()` メソッドを呼び出すことはできません。

```
class Example
{
  var methodExpression = function() {}
  function methodStatement() {}
}

var myEx:Example = new Example();
myEx.methodExpression(); // strict モードではエラー、standard モードでは OK
myEx.methodStatement(); // strict モードと standard モードで OK
```

関数式は、実行時、つまり動的なビヘイビアを中心にしたプログラミングに適しています。strict モードを使用し、関数式で宣言されるメソッドを呼び出す必要がある場合は、次の 2 つの方法のいずれかを使用することができます。1 つ目は、ドット (.) 演算子ではなく、角括弧 ([]) を使用してメソッドを呼び出す方法です。次のメソッドの呼び出しは、strict モードと standard モードの両方で成功します。

```
myExample["methodLiteral"]();
```

2 つ目は、クラス全体をダイナミッククラスとして宣言する方法です。この場合、ドット演算子を使用してメソッドを呼び出すことができますが、そのクラスのすべてのインスタンスで strict モードの一部の機能が犠牲になるという短所があります。たとえば、ダイナミッククラスのインスタンスの未定義のプロパティにアクセスしようとした場合、コンパイラはエラーを生成しません。

関数式が便利な場合があります。関数式の一般的な使用法は、1 回だけ使用された後で破棄される関数に使用することです。また、一般的な使用法ではありませんが、関数をプロトタイププロパティに関連付けるために使用します。詳細については、144 ページの「プロトタイプオブジェクト」を参照してください。

function ステートメントと関数式には、どちらを使用するかを選択する際に考慮する必要がある微妙な違いが 2 つあります。1つ目の違いは、関数式は、メモリ管理およびガベージコレクションに関してオブジェクトとして単独で存在しません。つまり、配列エレメントやオブジェクトプロパティなどの別のオブジェクトに関数式を割り当てると、コード内にその関数式への唯一の参照が作成されます。関数式が関連付けられている配列またはオブジェクトがスコープ外に移動するか、使用できなくなった場合、関数式にアクセスできなくなります。配列またはオブジェクトが削除されると、関数式が使用するメモリはガベージコレクションの対象となります。つまり、メモリは解放されて他の目的に再利用されます。

次の例では、関数式の場合、関数式が割り当てられているプロパティが削除されると、関数が利用できなくなることを示します。クラス Test は動的です。つまり、関数式を保持する `functionExp` というプロパティを追加できます。`functionExp()` 関数は、ドット演算子を使用して呼び出すことができますが、`functionExp` プロパティが削除されると、関数にアクセスできなくなります。

```
dynamic class Test {}
var myTest:Test = new Test();

// 関数式
myTest.functionExp = function () { trace("Function expression") };
myTest.functionExp();   // Function expression
delete myTest.functionExp;
myTest.functionExp();   // error
```

その一方で、関数が最初に function ステートメントで定義された場合、関数はそのオブジェクトとして存在し、関連付けられているプロパティを削除しても存在します。`delete` 演算子はオブジェクトのプロパティに対してのみ動作するため、関数 `stateFunc()` 自体を削除するための呼び出しも動作しません。

```
dynamic class Test {}
var myTest:Test = new Test();

// function ステートメント
function stateFunc() { trace("Function statement") }
myTest.statement = stateFunc;
myTest.statement(); // Function ステートメント
delete myTest.statement;
delete stateFunc;   // 無効
stateFunc();        // Function ステートメント
myTest.statement(); // エラー
```

function ステートメントと関数式の 2 つ目の違いは、function ステートメントは、関数ステートメントの前に現れるステートメント内を含む、定義されたスコープ全体で存在することです。対照的に、関数式はそれ以降のステートメントに対してのみ定義されます。たとえば、次のコードは、scopeTest() 関数が定義される前に、その関数を正常に呼び出します。

```
statementTest(); // statementTest

function statementTest():void
{
    trace("statementTest");
}
```

関数式は、定義される前に使用することはできません。したがって、次のコードはランタイムエラーになります。

```
expressionTest(); // ランタイムエラー

var expressionTest:Function = function ()
{
    trace("expressionTest");
}
```

関数からの値の戻り

関数から値を返すには、return ステートメントの後に、返す式またはリテラル値を使用します。たとえば、次のコードは、パラメータを表す式を返します。

```
function doubleNum(baseNum:int):int
{
    return (baseNum * 2);
}
```

return ステートメントは関数を終了するため、次のように、return ステートメントより下にあるステートメントは実行されません。

```
function doubleNum(baseNum:int):int {
    return (baseNum * 2);
    trace("after return"); // この trace ステートメントは実行されません。
}
```

strict モードでは、戻り値の型を指定した場合、適切な型の値を返す必要があります。たとえば、次のコードは、有効な値を返さないため、strict モードではエラーを生成します。

```
function doubleNum(baseNum:int):int
{
    trace("after return");
}
```

ネストされた関数

関数をネスト、つまり別の関数内で関数を宣言することができます。ネストされた関数は、関数への参照が外部コードに渡される場合を除いて、その親関数内でのみ使用できます。たとえば、次のコードは getNameAndVersion() 関数内でネストされた関数を 2 つ宣言します。

```
function getNameAndVersion():String
{
  function getVersion():String
  {
    return "9";
  }
  function getProductName():String
  {
    return "Flash Player";
  }
  return (getProductName() + " " + getVersion());
}
trace(getNameAndVersion()); // Flash Player 9
```

ネストされた関数が外部コードに渡される場合、メソッドクロージャとして渡されます。つまり、関数が定義されるとき、スコープ内にある定義を保持します。詳細については、104 ページの「メソッドクロージャ」を参照してください。

関数のパラメータ

ActionScript 3.0 は、ActionScript を初めて使用するプログラマには斬新な関数パラメータの機能を備えています。値または参照によってパラメータを渡すという概念は、ほとんどのプログラマにとってなじみがありますが、arguments オブジェクトおよび ... (rest) パラメータは初めて見るという人も多いかもしれません。

値渡しまたは参照渡しによるパラメータの受け渡し

多くのプログラミング言語では、値渡しと参照渡しによるパラメータの受け渡しの違いを理解しておくことが重要です。この違いは、コードの設計方法に影響します。

値渡しとは、関数内で使用するためにパラメータの値がローカル変数にコピーされることです。参照渡しとは、実際の値ではなく、パラメータへの参照のみが渡されることです。実際の値がコピーされるのではなく、パラメータとして渡される、変数への参照が作成され、関数内で使用するためにローカル変数に割り当てられます。関数の外部にある変数への参照として、ローカル変数では元の変数の値を変更できます。

ActionScript 3.0 では、値はすべてオブジェクトとして格納されているため、すべてのパラメータは参照渡しです。しかし、Boolean、Number、int、uint、String などのプリミティブデータ型に属するオブジェクトには、値渡しのように動作する特別な演算子があります。たとえば、次のコードは、xParam と yParam という 2 つの int 型パラメータを定義する passPrimitives() という関数を作成します。この 2 つのパラメータは、passPrimitives() 関数の本体内で宣言されるローカル変数に似ています。関数がパラメータ xValue および yValue で呼び出されると、パラメータ xParam および yParam は、xValue と yValue で表される int オブジェクトへの参照で初期化されます。パラメータはプリミティブなので、値渡しのように動作します。xParam および yParam は、最初は xValue および yValue オブジェクトへの参照のみを含みますが、関数の本体内の変数を変更すると、メモリに値の新しいコピーが生成されます。

```
function passPrimitives(xParam:int, yParam:int):void
{
   xParam++;
   yParam++;
   trace(xParam, yParam);
}

var xValue:int = 10;
var yValue:int = 15;
trace(xValue, yValue);          // 10 15
passPrimitives(xValue, yValue); // 11 16
trace(xValue, yValue);          // 10 15
```

passPrimitives() 関数内では、xParam および yParam の値はインクリメントされますが、最後の trace ステートメントに示されているように、これは xValue および yValue の値に影響を与えません。パラメータが変数 xValue および yValue と同じ名前である場合でも同じです。これは、関数内の xValue および yValue は関数の外部にある同じ名前の変数とは別に存在するメモリ内の新しい位置を参照するためです。

プリミティブデータ型に属していない他のすべてのオブジェクトは常に参照渡しによって渡されます。これにより、元の変数の値を変更できます。たとえば、次のコードは、2 つのプロパティ x および y を持つ objVar というオブジェクトを作成します。このオブジェクトは、passByRef() 関数にパラメータとして渡されます。オブジェクトはプリミティブ型ではないため、参照渡しで渡されるだけではなく、参照のままでもあります。つまり、関数内のパラメータを変更すると、関数の外側にあるオブジェクトのプロパティも影響を受けます。

```
function passByRef(objParam:Object):void
{
   objParam.x++;
   objParam.y++;
```

```
    trace(objParam.x, objParam.y);
}
var objVar:Object = {x:10, y:15};
trace(objVar.x, objVar.y); // 10 15
passByRef(objVar);         // 11 16
trace(objVar.x, objVar.y); // 11 16
```

objParamobjParam パラメータは、グローバル変数 objVar と同じオブジェクトを参照します。この例の trace ステートメントからもわかるように、objParam オブジェクトの x および y プロパティを変更すると、objVar オブジェクトにも反映されます。

デフォルトのパラメータ値

ActionScript 3.0 では、関数のデフォルトのパラメータ値を宣言する機能が新しく追加されました。デフォルトのパラメータ値を使用した関数の呼び出しでデフォルト値のパラメータが省略されると、そのパラメータの関数定義で指定された値が使用されます。デフォルト値のパラメータはすべてパラメータリストの末尾に配置する必要があります。デフォルト値として割り当てられた値はコンパイル時定数である必要があります。パラメータのデフォルト値が存在すると、そのパラメータは " オプションパラメータ " になります。デフォルト値のないパラメータは、必須パラメータと見なされます。

たとえば、次のコードは、3 つのパラメータで関数を作成します。このうち、2 つのパラメータにはデフォルト値があります。パラメータ 1 つだけで関数を呼び出す場合、そのパラメータのデフォルト値が使用されます。

```
function defaultValues(x:int, y:int = 3, z:int = 5):void
{
    trace(x, y, z);
}
defaultValues(1); // 1 3 5
```

arguments オブジェクト

パラメータが関数に渡されると、arguments オブジェクトを使用して関数に渡されたパラメータについての情報にアクセスできます。arguments オブジェクトには、次のような重要な特性があります。

- arguments オブジェクトは、関数に渡されるすべてのパラメータを含む配列です。
- arguments.length プロパティは、関数に渡されるパラメータの数を報告します。
- arguments.callee プロパティを使用すると、関数自体を参照することができます。これは関数式の再帰呼び出しに便利です。

メモ	パラメータに arguments という名前が付けられた場合、または ...（rest）パラメータを使用する場合、arguments オブジェクトを使用することはできません。

ActionScript 3.0 では、関数呼び出しで関数定義内で定義されているパラメータより多いパラメータを指定できますが、パラメータ数が必須パラメータ数より少ない場合は strict モードでコンパイラエラーが生成されます。arguments オブジェクトの配列を使用すると、関数に渡されるパラメータが関数定義内で定義されたかどうかに関わらず、このパラメータにアクセスできます。次の例では、arguments 配列と arguments.length プロパティを使用して、traceArgArray() 関数に渡されるすべてのパラメータをトレースします。

```
function traceArgArray(x:int):void
{
   for (var i:uint = 0; i < arguments.length; i++)
   {
      trace(arguments[i]);
   }
}

traceArgArray(1, 2, 3);

// 出力 :
// 1
// 2
// 3
```

arguments.callee プロパティは、再帰を作成する場合に匿名関数でよく使用されます。このプロパティを使用すると、コードに柔軟性を持たせることができます。開発過程で再帰関数の名前が変更された場合でも、関数名ではなく arguments.callee を使用していれば、関数本体内の再帰呼び出しの変更について考慮する必要はありません。次の関数式で arguments.callee プロパティを使用して再帰を有効にします。

```
var factorial:Function = function (x:uint)
{
   if(x == 0)
   {
      return 1;
   }
   else
   {
      return (x * arguments.callee(x - 1));
   }
}

trace(factorial(5)); // 120
```

関数宣言で ... (rest) パラメータを使用する場合は、arguments オブジェクトを使用することはできません。代わりに、宣言したパラメータ名を使用してパラメータにアクセスする必要があります。

ストリング “arguments” をパラメータ名として使用しないようにする必要があります。このストリングは arguments オブジェクトをシャドウするためです。たとえば、関数 traceArgArray() が arguments パラメータが追加されるように記述し直されると、関数本体内の arguments への参照は、arguments オブジェクトではなく、このパラメータを参照します。次のコードは出力を作成しません。

```
function traceArgArray(x:int, arguments:int):void
{
   for (var i:uint = 0; i < arguments.length; i++)
   {
      trace(arguments[i]);
   }
}

traceArgArray(1, 2, 3);

// 出力なし
```

旧バージョンの ActionScript の arguments オブジェクトには、caller という名前のプロパティも含まれていました。これは、現在の関数を呼び出した関数への参照です。ActionScript 3.0 には caller プロパティはありません。しかし、呼び出し元の関数への参照が必要な場合は、それ自体への参照である追加のパラメータを渡すように呼び出し元の関数を変更することができます。

... (rest) パラメータ

ActionScript 3.0 では、... (rest) パラメータと呼ばれる新しいパラメータ宣言が導入されました。このパラメータを使用すると、任意の数のカンマ区切りのパラメータを受け入れる配列パラメータを指定できます。パラメータには、予約語ではない名前を指定することができます。このパラメータ宣言は、指定される最後のパラメータである必要があります。このパラメータを使用すると、arguments オブジェクトは使用できなくなります。... (rest) パラメータには arguments 配列および arguments.length プロパティと同じ機能がありますが、arguments.callee のような機能はありません。... (rest) パラメータを使用する前に、arguments.callee を使用する必要はありません。

次の例では、arguments オブジェクトではなく、... (rest) パラメータを使用して traceArgArray() 関数を記述し直します。

```
function traceArgArray(... args):void
{
   for (var i:uint = 0; i < args.length; i++)
   {
      trace(args[i]);
   }
}

traceArgArray(1, 2, 3);
```

```
// 出力 :
// 1
// 2
// 3
```

... (rest) パラメータは、パラメータリストの最後のパラメータであれば、他のパラメータと使用することもできます。次の例では、traceArgArray() 関数を変更して、最初のパラメータ x を int 型にし、2 つ目のパラメータが ... (rest) パラメータを使用するようにします。最初のパラメータは ... (rest) パラメータにより作成された配列の一部ではなくなるため、最初の値は出力されません。

```
function traceArgArray(x: int, ... args)
{
  for (var i:uint = 0; i < args.length; i++)
  {
    trace(args[i]);
  }
}

traceArgArray(1, 2, 3);

// 出力 :
// 2
// 3
```

オブジェクトとしての関数

ActionScript 3.0 の関数はオブジェクトです。関数を作成すると、別の関数にパラメータとして渡すことができるだけではなく、プロパティとメソッドが関連付けられているオブジェクトが作成されます。

パラメータとして別の関数に渡される関数は、値渡しではなく、参照渡しによって渡されます。関数をパラメータとして渡す場合、識別子のみを使用し、メソッドを呼び出すために使用する括弧は使用しません。たとえば、次のコードは、addEventListener() メソッドへのパラメータとして clickListener() という関数を渡します。

```
addEventListener(MouseEvent.CLICK, clickListener);
```

Array.sort() メソッドも、関数を受け入れるパラメータを定義します。Array.sort() 関数へのパラメータとして使用される独自のソート関数の例については、231 ページの「配列のソート」を参照してください。

ActionScript を初めて使用するプログラマには奇妙に思えるかもしれませんが、他のオブジェクトと同様に関数にプロパティおよびメソッドを含めることができます。実際には、どの関数にも、その関数用に定義されたパラメータの数を格納する length という読み取り専用プロパティがあります。これは、関数に渡されたパラメータの数を報告する arguments.length プロパティとは異なります。ActionScript では、関数に渡されたパラメータの数がその関数用に定義されたパラメータの数を上回ってもかまいません。次の例では、strict モードでは渡されたパラメータの数と定義されたパラメータの数が完全に一致する必要があるので、standard モードでのみコンパイルされています。この例は 2 つのプロパティの違いを示します。

```
function traceLength (x:uint, y:uint):void
{
    trace("arguments received: " + arguments.length);
    trace("arguments expected: " + traceLength.length);
}

traceLength(3, 5, 7, 11);
/* 出力 :
arguments received: 4
arguments expected: 2 */
```

独自の関数プロパティを定義するには、関数本体の外側で定義します。関数プロパティは、関数に関連する変数の状態を保存できる準静的なプロパティになります。たとえば、特定の関数が呼び出される回数を追跡するとします。ゲームを記述していて、ユーザーが特定のコマンドを使用する回数を追跡する場合に、こうした機能は便利ですが、この場合静的クラスプロパティを使用することもできます。次のコードは、関数宣言の外側で関数プロパティを作成し、関数が呼び出されるたびにプロパティをインクリメントします。

```
someFunction.counter = 0;

function someFunction():void
{
    someFunction.counter++;
}

someFunction();
someFunction();
trace(someFunction.counter); // 2
```

関数スコープ

関数のスコープは、プログラム内の関数を呼び出すことができる場所だけではなく、関数がアクセスできる定義も決定します。変数識別子に適用される同じスコープの規則が関数識別子にも適用されます。グローバルスコープで宣言された関数は、コード全体で使用することができます。たとえば、ActionScript 3.0 には、`isNaN()`、`parseInt()` などのコード内のどこでも使用できるグローバル関数があります。ネストされた関数、つまり別の関数内で宣言された関数は、宣言された関数内のどこでも使用することができます。

スコープチェーン

関数が実行を開始するたびに、多数のオブジェクトおよびプロパティが作成されます。最初に、" アクティベーションオブジェクト " という特別なオブジェクトが作成され、そこには関数本体で宣言されるパラメータとローカル変数または関数が格納されます。アクティベーションオブジェクトは内部メカニズムであるため、そこに直接アクセスすることはできません。次に、" スコープチェーン " が作成され、そこには Flash Player によって識別子宣言の有無がチェックされるオブジェクトが列挙されたリストが含まれます。実行されるどの関数にも、内部プロパティに格納されるスコープチェーンがあります。ネストされた関数の場合は、スコープチェーンはそのアクティベーションオブジェクトから始まり、その後に親関数のアクティベーションオブジェクトが続きます。このように、スコープチェーンはグローバルオブジェクトに達するまで続きます。グローバルオブジェクトは、ActionScript プログラムが起動すると作成され、すべてのグローバル変数および関数を含みます。

メソッドクロージャ

" メソッドクロージャ " は、関数の静的なスナップショットとその " レキシカル環境 " を含むオブジェクトです。関数のレキシカル環境には、関数のスコープチェーン内のすべての変数、プロパティ、メソッド、およびオブジェクトがその値と共に含まれます。メソッドクロージャは、オブジェクトまたはクラスとは別に、関数が実行されるたびに作成されます。メソッドクロージャはそのメソッドクロージャが定義されたスコープを保持することから、関数がパラメータまたは戻り値として別のスコープに渡されると興味深い結果が生まれます。

たとえば、次のコードは 2 つの関数を作成します。`foo()` は、矩形の面積を計算する `rectArea()` というネストされた関数を返し、`bar()` は、`foo()` を呼び出し、`myProduct` という変数に返されたメソッドクロージャを格納します。`bar()` 関数が独自のローカル変数 `x` を値 2 で定義しても、メソッドクロージャ `myProduct()` が呼び出されると、関数 `foo()` 内で値 40 で定義された変数 `x` が維持されます。このため、`bar()` 関数は値 `8` ではなく、値 `160` を返します。

```
function foo():Function
{
    var x:int = 40;
```

```
    function rectArea(y:int):int // メソッドクロージャが定義された
    {
        return x * y
    }
    return rectArea;
}
function bar():void
{
    var x:int = 2;
    var y:int = 4;
    var myProduct:Function = foo();
    trace( myProduct(4) ); // メソッドクロージャが呼び出された
}
bar(); // 160
```

メソッドは、そのメソッドが作成されたレキシカル環境に関する情報も維持するように動作します。この特性が最も顕著なのは、メソッドがバインドメソッドを作成するそのインスタンスから抽出されるときです。メソッドクロージャとバインドメソッドの主な違いは、バインドメソッドの this キーワードの値は常に最初に関連付けられたインスタンスを参照しますが、メソッドクロージャでは this キーワードの値が変更可能という点です。詳細については、121 ページの「バインドメソッド」を参照してください。

第 4 章

ActionScript のオブジェクト指向プログラミング

4

この章では、抽象化、カプセル化、継承、ポリモーフィズムなどのオブジェクト指向プログラミング (OOP) の原則の基本を理解していることを前提にしています。ここでは、ActionScript 3.0 を使用してこれらの原則を適用する方法を中心に解説します。

ActionScript はスクリプト言語であることから、ActionScript 3.0 では OOP サポートを選択できます。これにより、プログラマはさまざまなスコープや複雑さのプロジェクトに対して最適な手法を選択できる柔軟性が得られます。小さなタスクの場合は、ActionScript と手続き型のプログラミングパラダイムを使用するだけで済みます。大規模なプロジェクトの場合は、OOP の原則を適用すると、コードを理解、保守、拡張しやすくすることができます。

目次

クラス. 107
インターフェイス 126
継承 130
高度なテクニック 139
例 : GeometricShapes 148

クラス

クラスは、オブジェクトの抽象表現です。クラスには、オブジェクトが保持できるデータの型およびオブジェクトが表すことができるビヘイビアに関する情報が格納されます。相互にやり取りする 2、3 個のオブジェクトのみを含む小さなスクリプトを記述する場合には、このような抽象化の利点はわかりにくいかもしれません。しかし、プログラムのスコープが拡大し、管理しなければならないオブジェクトの数が増加すると、クラスを使用することで、オブジェクトの作成方法およびオブジェクトが相互にやり取りする方法が制御しやすくなります。

ActionScript 1.0 では、Function オブジェクトを使用してクラスに似たコンストラクトを作成できました。ActionScript 2.0 では、class、extends などのキーワードによるクラスのサポートが正式に追加されました。ActionScript 3.0 では、ActionScript 2.0 で導入されたキーワードが引き続きサポートされるだけでなく、protected および internal 属性によるアクセス制御の強化、final および override キーワードによる継承の制御などの新しい機能も追加されています。

Java、C++、C# などのプログラミング言語でクラスを作成した経験があれば、ActionScript でも同じように作成できます。ActionScript では、class、extends、public など、多くの同じキーワード名および属性名を共用しています。次のセクションでは、これらについて説明します。

メモ	この章では、" プロパティ " という用語は、変数、定数、メソッドを含む、オブジェクトまたはクラスのメンバーを示します。また、" クラス " と " 静的 " という用語は同じ意味で使用されることがよくありますが、この章ではこの 2 つの用語は区別します。たとえば、" クラスプロパティ " という語句は、静的メンバーだけではなくクラスのすべてのメンバーを指します。

クラス定義

ActionScript 3.0 のクラス定義には、ActionScript 2.0 のクラス定義に使用したシンタックスと似たシンタックスを使用します。クラス定義の正しいシンタックスは、class キーワードの後にクラス名が必要です。クラス名の後には、中括弧 ({}) で囲まれたクラスの本体が続きます。たとえば、次のコードは visible という名前の変数を 1 つ含む Shape という名前のクラスを作成します。

```
public class Shape
{
   var visible:Boolean = true;
}
```

シンタックスの大きな変更の 1 つに、パッケージ内部にあるクラス定義があります。ActionScript 2.0 では、クラスがパッケージ内部にある場合は、クラス宣言にパッケージ名を含める必要があります。ActionScript 3.0 では、package ステートメントが導入されていますが、パッケージ名はクラス宣言ではなくパッケージ宣言に含める必要があります。たとえば、次のクラス宣言は、flash.display パッケージに含まれている BitmapData クラスを ActionScript 2.0 および ActionScript 3.0 で定義する方法を示します。

```
// ActionScript 2.0
class flash.display.BitmapData {}

// ActionScript 3.0
package flash.display
{
   public class BitmapData {}
}
```

クラス属性

ActionScript 3.0 では、次の 4 つの属性のいずれかを使用してクラス定義を変更できます。

属性	定義
`dynamic`	実行時にインスタンスにプロパティを追加できます。
`final`	別のクラスによって拡張することはできません。
`internal`（デフォルト）	現在のパッケージ内の参照に対して表示されます。
`public`	すべての場所にある参照に対して表示されます。

`internal` を除くこれらの属性では、属性を明示的に含めて関連付けられたビヘイビアを取得します。たとえば、クラスを定義するときに `dynamic` 属性を含めなかった場合、実行時にクラスインスタンスにプロパティを追加することはできません。次のコードに示すように、クラス定義の先頭に属性を配置して、属性を明示的に割り当てます。

```
dynamic class Shape {}
```

上記のリストには、`abstract` という属性は含まれていません。ActionScript 3.0 では抽象クラスをサポートしていないためです。リストには `private` および `protected` という属性も含まれていません。これらの属性は、クラス定義内でのみ意味を持ち、クラス自体には適用できません。パッケージの外部でクラスをパブリックに表示しない場合は、パッケージ内にクラスを配置し、`internal` 属性でクラスを宣言します。または、`internal` と `public` 属性を両方とも省略することができます。この場合は、コンパイラによって `internal` 属性が自動的に追加されます。クラスが定義されたソースファイルの外部にクラスを表示しない場合は、パッケージ定義の中括弧の下にあるソースファイルの下部にクラスを配置します。

クラスの本体

中括弧で囲まれたクラス本体は、クラスの変数、定数、およびメソッドを定義するために使用します。次の例は、Flash Player API の Accessibility クラスの宣言を示します。

```
public final class Accessibility
{
  public static function get active():Boolean;
  public static function updateProperties():void;
}
```

クラス本体内に名前空間を定義することもできます。次の例は、名前空間をクラス本体内に定義し、そのクラスのメソッドの属性として使用する方法を示します。

```
public class SampleClass
{
  public namespace sampleNamespace;
  sampleNamespace function doSomething():void;
}
```

ActionScript 3.0 では、クラス本体に定義だけでなくステートメントも含めることができます。クラス本体内部にあるが、メソッド定義の外部にあるステートメントは、クラス定義が最初に認識され、関連付けられたクラスオブジェクトが作成されるときに、1 度だけ実行されます。次の例には、外部関数 `hello()` と、クラスが定義されたときに確認メッセージを出力する `trace` ステートメントへの呼び出しが含まれています。

```
function hello():String
{
  trace ("hola");
}
class SampleClass
{
  hello();
  trace("class created");
}
// クラスが作成されると出力する
hola
class created
```

旧バージョンの ActionScript とは異なり、ActionScript 3.0 では、同じクラス本体内に同じ名前の静的プロパティとインスタンスプロパティを定義することができます。たとえば、次のコードは、`message` という名前の静的変数と同じ名前のインスタンス変数を宣言します。

```
class StaticTest
{
  static var message:String = "static variable";
  var message:String = "instance variable";
}
// スクリプト内
var myST:StaticTest = new StaticTest();
trace(StaticTest.message); // static variable
trace(myST.message);       // instance variable
```

クラスプロパティの属性

ActionScript オブジェクトモデルの説明では、" プロパティ " という用語は、変数、定数、メソッドなど、クラスのメンバーになるものを示します。しかし、『ActionScript 3.0 リファレンスガイド』では、" プロパティ " は狭い意味で使用され、変数であるか、getter または setter メソッドにより定義されるクラスメンバーのみを意味します。ActionScript 3.0 には、クラスのプロパティと共に使用できる一連の属性があります。次の表に、この属性の一覧を示します。

属性	定義
`internal` (デフォルト)	同じパッケージ内の参照に対して表示されます。
`private`	同じクラス内の参照に対して表示されます。
`protected`	同じクラスおよび派生クラス内の参照に対して表示されます。
`public`	すべての場所にある参照に対して表示されます。
`static`	プロパティが、クラスのインスタンスではなくクラスに属することを指定します。
UserDefinedNamespace	ユーザーが定義するカスタム名前空間の名前です。

アクセス制御名前空間の属性

ActionScript 3.0 には、クラス内で定義されたプロパティへのアクセスを制御する `public`、`private`、`protected`、および `internal` の 4 つの特別な属性があります。

`public` 属性は、スクリプト内の任意の場所でプロパティを表示します。たとえば、パッケージ外部のコードでメソッドを使用するには、`public` 属性でメソッドを宣言する必要があります。これは、`var`、`const`、または `function` キーワードのいずれで宣言されたかに関係なく、どのプロパティについても同じです。

`private` 属性は、プロパティが定義されたクラス内の呼び出し元に対してのみプロパティを表示します。このビヘイビアは、サブクラスがスーパークラスのプライベートプロパティにアクセスできる ActionScript 2.0 の `private` 属性とは異なります。ランタイムアクセスに関しても、ビヘイビアが大きく変更されています。ActionScript 2.0 では、`private` キーワードによりアクセスが禁止されるのはコンパイル時のみで、実行時にはアクセスが禁止されません。ActionScript 3.0 では、これは当てはまりません。`private` とマークされたプロパティは、コンパイル時も実行時も使用することができません。

たとえば、次のコードは、プライベート変数を 1 つ持つ PrivateExample という単純なクラスを作成し、クラス外部からこのプライベート変数にアクセスしようとします。ActionScript 2.0 では、コンパイル時アクセスは禁止されていましたが、コンパイル時ではなく実行時にプロパティルックアップを実行するプロパティアクセス演算子 ([ ]) を使用することでアクセスすることができました。

```
class PrivateExample
{
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);    // strict モードではコンパイル時エラー
trace(myExample["privVar"]); // ActionScript 2.0 ではアクセスを許可するが、
  ActionScript 3.0 ではこれはラインタイムエラー。
```

ActionScript 3.0 では、ドット演算子 (myExample.privVar) を使用してプライベートプロパティにアクセスしようとすると、strict モードを使用している場合にコンパイル時エラーになります。それ以外の場合は、プロパティアクセス演算子 (myExample["privVar"]) を使用するときと同じように、エラーは実行時に報告されます。

次の表に、sealed (dynamic ではない) クラスに属するプライベートプロパティにアクセスしようとした結果の概要を示します。

	strict モード	standard モード
ドット演算子 (.)	コンパイル時エラー	ランタイムエラー
角括弧演算子 ([])	ランタイムエラー	ランタイムエラー

dynamic 属性で宣言されたクラスでは、プライベート変数にアクセスしようとしても、ランタイムエラーにはなりません。変数は表示されないだけなので、Flash Player は戻り値として undefined を返します。しかし、strict モードでドット演算子を使用すると、コンパイル時エラーが発生します。次の例は、PrivateExample クラスがダイナミッククラスとして宣言されている点を除いて、前の例と同じです。

```
dynamic class PrivateExample
{
  private var privVar:String = "private variable";
}

var myExample:PrivateExample = new PrivateExample();
trace(myExample.privVar);    // strict モードではコンパイル時エラー
trace(myExample["privVar"]); // 出力 : 未定義
```

ダイナミッククラスは通常、クラス外部のコードがプライベートプロパティにアクセスしようとすると、エラーを生成するのではなく、戻り値として undefined を返します。次の表は、ドット演算子を使用して strict モードでプライベートプロパティにアクセスしようとしたときにのみエラーが生成されることを示します。

	strict モード	standard モード
ドット演算子 (.)	コンパイル時エラー	undefined
角括弧演算子 ([])	undefined	undefined

ActionScript 3.0 で新しく導入された protected 属性は、クラス内またはサブクラス内の呼び出し元に対してプロパティを表示します。つまり、protected プロパティは、そのクラス内または継承階層内でそのクラスの下の任意の場所にあるクラスで使用することができます。これは、サブクラスが同じパッケージ内にあるか別のパッケージ内にあるかに関係なく、同じです。

ActionScript 2.0 を使い慣れていれば、この機能が ActionScript 2.0 の private 属性に似ていることがわかります。ActionScript 3.0 の protected 属性は Java の protected 属性にも似ていますが、Java の属性でも同じパッケージ内の呼び出し元にアクセスできる点で異なります。protected 属性は、サブクラスで必要なメソッドまたは変数を継承チェーン外部のコードに対して非表示にするときに便利です。

ActionScript 3.0 で新しく導入された internal 属性は、パッケージ内にある呼び出し元に対してプロパティを表示します。これはパッケージ内にあるコードのデフォルトの属性であり、次のいずれの属性も持たないプロパティに適用されます。

- public
- private
- protected
- ユーザー定義の名前空間

internal 属性は、Java のデフォルトのアクセス制御に似ています。ただし、Java ではこのレベルのアクセスに明示的な名前はなく、他のアクセス修飾子が省略された場合にのみこのデフォルトのアクセス制御が使用されます。internal 属性は ActionScript 3.0 で使用可能で、パッケージ内の呼び出し元に対してのみプロパティを表示する意図を明示的に表すオプションがあります。

static 属性

static 属性は、var、const、または function キーワードで宣言されたプロパティと共に使用でき、クラスのインスタンスではなくクラスにプロパティを関連付けることができます。クラス外部にあるコードは、インスタンス名ではなくクラス名を使用して静的プロパティを呼び出す必要があります。

静的プロパティはサブクラスに継承されませんが、サブクラスのスコープチェーンの一部です。つまり、サブクラスの本体内では、静的変数またはメソッドは、定義されたクラスを参照しなくても使用できます。詳細については、136 ページの「継承されない静的プロパティ」を参照してください。

ユーザー定義の名前空間の属性

事前に定義されているアクセス制御属性の代わりに、属性として使用するカスタム名前空間を作成できます。1 つの定義に使用できる名前空間属性は 1 つだけです。また、名前空間属性をアクセス制御属性 (`public`、`private`、`protected`、`internal`) のいずれかと組み合わせて使用することはできません。名前空間を使用する方法の詳細については、43 ページの「名前空間」を参照してください。

変数

変数は、`var` または `const` のいずれかのキーワードで宣言できます。`var` キーワードで宣言した変数は、スクリプトの実行中に複数回その値を変更できます。`const` キーワードで宣言した変数は定数と呼ばれ、1 回だけ値を割り当てることができます。初期化された定数に新しい値を割り当てようとすると、エラーが発生します。詳細については、76 ページの「定数」を参照してください。

静的変数

静的変数は、`static` キーワードと、`var` または `const` ステートメントのいずれかの組み合わせを使用して宣言します。クラスのインスタンスではなく、クラスに関連付けられた静的変数は、オブジェクトのクラス全体に適用される情報の格納および共有に役立ちます。たとえば、静的変数は、クラスがインスタンス化された回数を記録する場合や可能なクラスインスタンスの最大数を格納しておく場合に適しています。

次の例では、クラスのインスタンス化の回数を追跡する `totalCount` 変数およびインスタンス化の最大回数を格納する `MAX_NUM` 定数を作成します。`totalCount` と `MAX_NUM` 変数には、特定のインスタンスではなくクラス全体に適用される値が含まれているため、この 2 つの変数は静的です。

```
class StaticVars
{
    public static var totalCount:int = 0;
    public static const MAX_NUM:uint = 16;
}
```

StaticVars クラスおよびそのサブクラスの外部にあるコードは、StaticVars クラスからのみ totalCount プロパティおよび MAX_NUM プロパティを参照できます。たとえば、次のコードは正常に動作します。

```
trace(StaticVars.totalCount); // 0
trace(StaticVars.MAX_NUM); // 16
```

クラスのインスタンスから静的変数にアクセスできないため、次のコードはエラーを返します。

```
var myStaticVars:StaticVars = new StaticVars();
trace(myStaticVars.totalCount); // エラー
trace(myStaticVars.MAX_NUM); // エラー
```

static および const の 2 つのキーワードで宣言された変数は、StaticVars クラスが MAX_NUM に対する場合と同様に、定数の宣言時に初期化する必要があります。コンストラクタまたはインスタンスメソッド内の MAX_NUM に値を割り当てることはできません。次のコードは静的定数を初期化する有効な方法でないため、エラーが生成されます。

```
// !! この方法で静的定数を初期化するとエラー
class StaticVars2
{
    public static const UNIQUESORT:uint;
    function initializeStatic():void
    {
        UNIQUESORT = 16;
    }
}
```

インスタンス変数

インスタンス変数には、var および const キーワードを使用し、static キーワードを使用せずに宣言されたプロパティが含まれます。クラス全体ではなくクラスインスタンスに関連付けられたインスタンス変数は、インスタンス固有の値の格納に便利です。たとえば、Array クラスには、この Array クラスの特定のインスタンスが保持する配列エレメントの数を格納する length というインスタンスプロパティがあります。

インスタンス変数は、var または const として宣言されたかに関係なく、サブクラスではオーバーライドできません。ただし、getter および setter メソッドをオーバーライドすることにより、変数をオーバーライドするのに似た機能を実現できます。詳細については、120 ページの「get および set アクセサメソッド」を参照してください。

メソッド

メソッドは、クラス定義の一部である関数です。クラスのインスタンスが作成されると、メソッドはそのインスタンスにバインドされます。クラス外で宣言された関数とは異なり、メソッドは関連付けられているインスタンスと別々に使用することはできません。

メソッドは、function キーワードを使用して定義します。次のような関数ステートメントを使用できます。

```
public function sampleFunction():String {}
```

または、次のように関数式を割り当てる変数を使用することもできます。

```
public var sampleFunction:Function = function () {}
```

通常は、次のような理由から関数式ではなく関数ステートメントを使用します。

- 関数ステートメントは、関数式より簡潔で読みやすくなります。
- 関数ステートメントでは、override および final キーワードを使用できます。詳細については、134 ページの「メソッドのオーバーライド」を参照してください。
- 関数ステートメントでは、識別子 (関数の名前) とメソッド本体内のコードがより強力に結合されます。変数の値は代入ステートメントで変更できるので、変数とその関数式の結合をいつでも切り離すことができます。var ではなく const で変数を宣言するとこの問題を回避できますが、コードが読みにくくなり、override および final キーワードが使用できなくなるため、最適な方法とは言えません。

関数式を使用する必要があるのは、関数をプロトタイプオブジェクトに関連付ける場合などです。詳細については、144 ページの「プロトタイプオブジェクト」を参照してください。

コンストラクタメソッド

コンストラクタメソッドは、単にコンストラクタと呼ばれることもあり、定義されたクラスと同じ名前を共有する関数です。コンストラクタメソッドに含まれるコードは、クラスのインスタンスが new キーワードで作成されるときに実行されます。たとえば、次のコードは、status というプロパティを 1 つ含む単純なクラス Example を定義します。status 変数の初期値は、コンストラクタ関数内で設定します。

```
class Example
{
    public var status:String;
    public function Example()
    {
        status = "initialized";
    }
}

var myExample:Example = new Example();
trace (myExample.status); // 出力 : initialized
```

コンストラクタメソッドにはパブリックしか指定できませんが、public 属性を使用するかどうかは任意です。コンストラクタに、private、protected、internal やその他のアクセス制御指定子を使用することはできません。また、ユーザー定義の名前空間をコンストラクタメソッドと使用することもできません。

コンストラクタは、super() ステートメントを使用して直接のスーパークラスのコンストラクタを明示的に呼び出すことができます。スーパークラスのコンストラクタを明示的に呼び出さない場合は、コンパイラによってコンストラクタ本体の最初のステートメントの前に呼び出しが自動的に挿入されます。スーパークラスへの参照として super 接頭辞を使用して、スーパークラスのメソッドを呼び出すこともできます。同じコンストラクタ本体で super() と super を使用する場合は、必ず最初に super() を呼び出します。そうしないと、super 参照は意図したとおりに動作しません。super() コンストラクタも、throw または return ステートメントの前に呼び出す必要があります。

次に、super() コンストラクタを呼び出す前に super 参照を使用しようとした場合の例を示します。新しいクラス ExampleEx は Example クラスを拡張します。ExampleEx コンストラクタは、super() を呼び出す前に、スーパークラスで定義された status 変数にアクセスしようとします。ExampleEx コンストラクタ内の trace() ステートメントは、値 null を生成します。これは、super() コンストラクタが実行されるまで status 変数を使用できないためです。

```
class ExampleEx extends Example
{
   public function ExampleEx()
   {
      trace(super.status);
      super();
   }
}

var mySample:ExampleEx = new ExampleEx(); // 出力 : null
```

コンストラクタ内で return ステートメントを使用することはできますが、値を返すことはできません。つまり、return ステートメントに式または値を関連付けることはできません。したがって、コンストラクタメソッドは値を返すことができず、戻り値の型を指定できません。

クラス内でコンストラクタメソッドを定義しない場合、コンパイラによって自動的に空のコンストラクタが作成されます。クラスが別のクラスに拡張している場合は、コンパイラによって生成されるコンストラクタ内に super() 呼び出しが含まれます。

静的メソッド

静的メソッドは、クラスメソッドとも呼ばれる、static キーワードで宣言されたメソッドです。静的メソッドは、クラスのインスタンスではなくクラスに関連付けられ、個々のインスタンスの状態以外のものに影響を与える機能のカプセル化に役立ちます。静的メソッドはクラス全体に関連付けられるため、クラスのインスタンスではなくクラスからのみアクセスできます。

静的メソッドは、クラスインスタンスの状態に影響を与えるだけではない機能のカプセル化に役立ちます。つまり、メソッドにクラスインスタンスの値に直接影響しない機能がある場合、そのメソッドは静的です。たとえば、Date クラスには、ストリングを取得して数値に変換する parse() という静的メソッドがあります。このメソッドは、クラスの個々のインスタンスに影響を与えないため静的です。parse() メソッドは、日付値を表すストリングを取得し、そのストリングを解析して、Date オブジェクトの内部表現と互換性がある形式で数値を返します。このメソッドは、Date クラスのインスタンスに適用しても意味がないため、インスタンスメソッドではありません。

静的な parse() メソッドと、getMonth() などの Date クラスのインスタンスメソッドを比較してみます。getMonth() メソッドは、Date インスタンスの特定のコンポーネント、つまり月を取得することで、インスタンスの値に対して直接実行されるため、インスタンスメソッドです。

静的メソッドは個々のインスタンスにバインドされていないため、静的メソッドの本体内で this または super キーワードを使用できません。this 参照および super 参照は、インスタンスメソッドのコンテキスト内でのみ有効です。

他のクラスベースのプログラミング言語とは異なり、ActionScript 3.0 では静的メソッドは継承されません。詳細については、136 ページの「継承されない静的プロパティ」を参照してください。

インスタンスメソッド

インスタンスメソッドは、static キーワードを使用せずに宣言されたメソッドです。インスタンスメソッドは、クラス全体ではなくクラスのインスタンスに関連付けられ、クラスの個々のインスタンスに影響を与える機能の実装に役立ちます。たとえば、Array クラスには、Array インスタンスに対して直接実行される sort() というインスタンスメソッドがあります。

インスタンスメソッドの本体内では、静的変数およびインスタンス変数はスコープ内にあります。つまり、同じクラス内で定義された変数は、単純な識別子を使用して参照できます。たとえば、次の CustomArray クラスは Array クラスを拡張します。CustomArray クラスは、クラスインスタンスの総数を追跡する静的変数 arrayCountTotal、インスタンスが作成された順序を追跡するインスタンス変数 arrayNumber、およびこれらの変数の値を返すインスタンスメソッド getPosition() を定義します。

```
public class CustomArray extends Array
{
    public static var arrayCountTotal:int = 0;
    public var arrayNumber:int;

    public function CustomArray()
    {
        arrayNumber = ++arrayCountTotal;
    }

    public function getArrayPosition():String
    {
         return ("Array " + arrayNumber + " of " + arrayCountTotal);
    }
}
```

クラス外部にあるコードは CustomArray.arrayCountTotal を使用してクラスオブジェクトから静的変数 arrayCountTotal を参照する必要がありますが、getPosition() メソッドの本体内にあるコードは静的変数 arrayCountTotal を直接参照できます。これは、スーパークラスの静的変数でも同じです。ActionScript 3.0 では静的プロパティは継承されませんが、スーパークラスの静的プロパティはスコープ内です。たとえば、Array クラスにはいくつかの静的変数があり、そのうちの 1 つは DESCENDING という定数です。Array サブクラス内にあるコードは、単純な識別子を使用して静的定数 DESCENDING を参照できます。

```
public class CustomArray extends Array
{
    public function testStatic():void
    {
        trace(DESCENDING); // 出力 : 2
    }
}
```

インスタンスメソッドの本体内にある this 参照の値は、メソッドが関連付けられているインスタンスへの参照です。次のコードは、this 参照がメソッドを含むインスタンスを参照していることを示します。

```
class ThisTest
{
    function thisValue():ThisTest
```

```
    {
        return this;
    }
}

var myTest:ThisTest = new ThisTest();
trace(myTest.thisValue() == myTest); // true
```

インスタンスメソッドの継承は、override と final のキーワードを使用して制御できます。override 属性を使用して継承されたメソッドを定義できます。また、final 属性を使用すると、サブクラスによってメソッドがオーバーライドされないようにできます。詳細については、134 ページの「メソッドのオーバーライド」を参照してください。

get および set アクセサメソッド

"getter" および "setter" とも呼ばれる get および set アクセサ関数を使用すると、作成したクラスの使いやすいプログラミングインターフェイスを提供すると同時に、情報を非表示およびカプセル化するというプログラミング原則に従うことができます。get および set 関数を使用して、クラスプロパティをクラスに対してプライベートに保持できますが、クラスのユーザーは、クラスメソッドを呼び出すのではなくクラス変数にアクセスしている場合と同じように、これらのプロパティにアクセスできます。

この方法の利点は、getPropertyName()、setPropertyName() など、従来の冗長な名前のアクセサ関数を使用しなくても済むことです。また、getter および setter には、読み取りおよび書き込みアクセスを可能にする各プロパティに対して 2 つの公開関数を持つ必要がないという利点もあります。

次の例の GetSet クラスには、privateProperty というプライベート変数へのアクセスを提供する publicAccess() という get および set アクセサ関数が含まれています。

```
class GetSet
{
    private var privateProperty:String;

    public function get publicAccess():String
    {
        return privateProperty;
    }

    public function set publicAccess(setValue:String):void
    {
        privateProperty = setValue;
    }
}
```

privateProperty プロパティに直接アクセスしようとすると、次のようなエラーが発生します。

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.privateProperty); // エラーが発生する
```

GetSet クラスを使用している場合は、publicAccess というプロパティのように見える、privateProperty というプライベートプロパティに対して動作する get と set のアクセサ関数の組み合わせを使用します。次の例では、GetSet クラスをインスタンス化し、publicAccess というパブリックアクセサを使用して privateProperty の値を設定します。

```
var myGetSet:GetSet = new GetSet();
trace(myGetSet.publicAccess); // null
myGetSet.publicAccess = "hello";
trace(myGetSet.publicAccess); // hello
```

getter および setter 関数でも、スーパークラスから継承されるプロパティをオーバーライドできますが、通常のクラスメンバー変数を使用したときにはオーバーライドできません。var キーワードで宣言されたクラスメンバー変数は、サブクラスではオーバーライドできません。一方、getter および setter 関数を使用して作成されたプロパティにはこの制限はありません。スーパークラスから継承した getter および setter 関数に override 属性を使用できます。

バインドメソッド

バインドメソッドは、メソッドクロージャとも呼ばれる、単にインスタンスから抽出されるメソッドです。バインドメソッドの例には、関数にパラメータとして渡され、関数から値として返されるメソッドがあります。ActionScript 3.0 で新しく導入されたバインドメソッドは、インスタンスから抽出されたときでもレキシカル環境が保持されるメソッドクロージャに似ています。バインドメソッドとメソッドクロージャの主な違いは、バインドメソッドの this 参照は、バインドメソッドを実装するインスタンスにリンクされたまま、つまりバインドされたままであるという点です。これは、バインドメソッドの this 参照が、常にこのメソッドを実装する元のオブジェクトを指していることを意味します。メソッドクロージャの場合、this 参照は汎用的で、呼び出されたときにこの関数が関連付けられているオブジェクトを指します。

this キーワードを使用する場合は、バインドメソッドを理解していることが重要です。this キーワードを使用すれば、メソッドの親オブジェクトを参照できます。ほとんどの ActionScript プログラマは、this キーワードが常にメソッドの定義を含むオブジェクトまたはクラスを参照すると考えますが、メソッドのバインディングがないと必ずしもそうなりません。旧バージョンの ActionScript では、たとえば、this 参照はメソッドを実装するインスタンスを常に参照するわけではありませんでした。ActionScript 2.0 でインスタンスからメソッドを抽出すると、this 参照が元のインスタンスにバインドされないだけでなく、インスタンスのクラスのメンバー変数およびメソッドも使用できません。ActionScript 3.0 では、メソッドをパラメータとして渡すとバインドメソッドが自動的に作成されるため、これは問題にはなりません。バインドメソッドにより、this キーワードは常にメソッドが定義されたオブジェクトまたはクラスを参照します。

次のコードは、ThisTest というクラスを定義します。このクラスには、バインドメソッドを定義するメソッド foo()、およびバインドメソッドを返すメソッド bar() が含まれます。クラス外部にあるコードは、ThisTest クラスのインスタンスを作成し、bar() メソッドを呼び出して、myFunc という変数に戻り値を格納します。

```
class ThisTest
{
  private var num:Number = 3;
  function foo():void // 定義されたバインドメソッド
  {
    trace("foo's this: " + this);
    trace("num: " + num);
  }
  function bar():Function
  {
    return foo; // 返されたバインドメソッド
  }
}

var myTest:ThisTest = new ThisTest();
var myFunc:Function = myTest.bar();
trace(this); // 出力 : [object global]
myFunc();
/* 出力 :
foo's this: [object ThisTest]
num: 3 */
```

コードの最後の 2 行は、バインドメソッド foo() の this 参照が、その直前の行の this 参照がグローバルオブジェクトを指していても、依然として ThisTest クラスのインスタンスを指すことを示しています。さらに、myFunc 変数に格納されているバインドメソッドは、引き続き ThisTest クラスのメンバー変数にアクセスすることができます。この同じコードを ActionScript 2.0 で実行すると、this 参照が一致し、num 変数は undefined になります。

addEventListener() メソッドでは関数またはメソッドをパラメータとして渡す必要があるため、バインドメソッドの追加が最もわかりやすいのはイベントハンドラを使用する場合です。詳細については、359 ページの「クラスメソッドとして定義したリスナー関数」を参照してください。

クラスによる列挙

" 列挙 " は、値の小さなセットをカプセル化するために作成するカスタムデータ型です。ActionScript 3.0 は、C++ の enum キーワードや Java の Enumeration インターフェイスとは異なり、特定の列挙機能をサポートしません。ただし、クラスと静的定数を使用して列挙を作成できます。たとえば、次のコードに示すように、Flash Player API の PrintJob クラスは、PrintJobOrientation という列挙を使用して "landscape" および "portrait" で構成される値のセットを格納します。

```
public final class PrintJobOrientation
{
    public static const LANDSCAPE:String = "landscape";
    public static const PORTRAIT:String = "portrait";
}
```

慣例では、列挙クラスは、拡張する必要がないので final 属性で宣言します。このクラスは静的メンバーのみで構成されます。つまり、クラスのインスタンスを作成しません。代わりに、次のコードの抜粋に示すように、クラスオブジェクトから直接列挙値にアクセスします。

```
var pj:PrintJob = new PrintJob();
if(pj.start())
{
    if(pj.orientation == PrintJobOrientation.PORTRAIT)
    {
        ...
    }
    ...
}
```

Flash Player API のすべての列挙クラスには、String 型、int 型、uint 型の変数のみが含まれます。リテラルのストリング値または数値ではなく列挙を使用する利点は、表記ミスを見つけやすいことです。列挙の名前を間違って入力した場合、ActionScript コンパイラはエラーを生成します。リテラル値を使用した場合、単語を間違って入力するか数値を間違って使用してもコンパイラに認識されません。前の例では、次のコードの抜粋に示すように、列挙定数の名前が間違っている場合にコンパイラはエラーを生成します。

```
    if(pj.orientation == PrintJobOrientation.PORTRAI) // コンパイルエラー
```

ただし、次のようにリテラルストリング値にスペルミスがある場合、コンパイラはエラーを生成しません。

```
    if(pj.orientation == "portrai") // コンパイルエラーなし
```

列挙を作成するもう 1 つの方法でも、列挙の静的プロパティで別のクラスを作成することが必要になります。ただし、この方法は、静的プロパティにストリング値または整数値ではなくクラスのインスタンスが含まれる点で異なります。たとえば、次のコードは、曜日に対する列挙クラスを作成します。

```
public final class Day
{
   public static const MONDAY:Day = new Day();
   public static const TUESDAY:Day = new Day();
   public static const WEDNESDAY:Day = new Day();
   public static const THURSDAY:Day = new Day();
   public static const FRIDAY:Day = new Day();
   public static const SATURDAY:Day = new Day();
   public static const SUNDAY:Day = new Day();
}
```

この方法は、Flash Player API によっては使用されませんが、この方法による型チェックの向上を好む多くの開発者が使用しています。たとえば、列挙値を返すメソッドは、その戻り値を列挙データ型に制限することができます。次のコードは、曜日を返す関数だけでなく、列挙型を型注釈として使用する関数呼び出しも示します。

```
function getDay():Day
{
   var date:Date = new Date();
   var retDay:Day;
   switch (date.day)
   {
      case 0:
         retDay = Day.MONDAY;
         break;
      case 1:
         retDay = Day.TUESDAY;
         break;
      case 2:
         retDay = Day.WEDNESDAY;
         break;
      case 3:
         retDay = Day.THURSDAY;
         break;
      case 4:
         retDay = Day.FRIDAY;
         break;
      case 5:
         retDay = Day.SATURDAY;
         break;
      case 6:
         retDay = Day.SUNDAY;
         break;
```

```
    }
    return retDay;
}

var dayOfWeek:Day = getDay();
```

Day クラスを機能拡張して、整数をそれぞれの曜日に関連付け、曜日のストリング表現を返す toString() メソッドを提供するようにもできます。必要に応じて、この方法で Day クラスの機能拡張を実習として試してみることができます。

アセットクラスの埋め込み

ActionScript 3.0 では、埋め込みアセットを表現するために、埋め込みアセットクラスと呼ばれる特別なクラスを使用します。" 埋め込みアセット " は、コンパイル時に SWF ファイルに含まれるサウンド、イメージ、フォントのようなアセットです。アセットを動的にロードするのではなく埋め込むと、実行時に利用可能になることが保証されますが、SWF ファイルのサイズは大きくなってしまいます。

Flex での埋め込みアセットクラスの使用

アセットを ActionScript コードに埋め込むには、[Embed] メタデータタグを使用します。プロジェクトのビルドパスにあるメインソースフォルダまたは別のフォルダ内にアセットを配置します。Embed メタデータタグを見つけると、Flex コンパイラは埋め込みアセットクラスを作成します。[Embed] メタデータタグの直後で宣言した Class データ型の変数を通してクラスにアクセスすることができます。たとえば、次のコードは sound1.mp3 という名前のサウンドを埋め込み、soundCls という名前の変数を使用して、そのサウンドに関連付けられた埋め込みアセットクラスへの参照を格納します。この例では次に、埋め込みアセットクラスのインスタンスを作成し、そのインスタンス上で play() メソッドを呼び出します。

« 下のサンプルコードは、FlexOnly+No Trans に対して二重条件となっています。FlexOnly for Flex 2.0.1 loc prodeuction を削除する必要がありました。本マニュアルは Flex 2.0.1 のものなので、Flash-MN では使用できません。»

```
package
{
    import flash.display.Sprite;
    import flash.media.SoundChannel;
    import mx.core.SoundAsset;

    public class SoundAssetExample extends Sprite
    {
        [Embed(source="sound1.mp3")]
        public var soundCls:Class;

        public function SoundAssetExample()
        {
            var mySound:SoundAsset = new soundCls() as SoundAsset;
```

```
            var sndChannel:SoundChannel = mySound.play();
        }
    }
}
```

ヒント	`[Embed]` メタデータタグを Adobe Flex Builder 2 の ActionScript プロジェクトで使用するには、Flex フレームワークから必要なクラスを読み込む必要があります。たとえば、サウンドを埋め込むには、mx.core.SoundAsset クラスを読み込む必要があります。Flex フレームワークを使用するには、"framework.swc" ファイルを ActionScript ビルドパスに含めます。このため、SWF ファイルのサイズは大きくなります。

代わりの方法として、MXML タグ定義の `@Embed()` ディレクティブでアセットを埋め込むことができます。詳細については、『Flex 2 開発ガイド』の「「アセットの埋め込みについて」」を参照してください。

インターフェイス

インターフェイスは、関連しないオブジェクトを相互に通信可能にするメソッド宣言の集まりです。たとえば、Flash Player API では、クラスでイベントオブジェクトを処理するために使用できるメソッド宣言を含む IEventDispatcher インターフェイスを定義します。IEventDispatcher インターフェイスは、オブジェクトが相互にイベントオブジェクトを渡し合うための標準的な手段を確立します。次のコードは、IEventDispatcher インターフェイスの定義を示します。

```
public interface IEventDispatcher
{
    function addEventListener(type:String, listener:Function,
            useCapture:Boolean=false, priority:int=0,
            useWeakReference:Boolean = false):void;
    function removeEventListener(type:String, listener:Function,
            useCapture:Boolean=false):void;
    function dispatchEvent(event:Event):Boolean;
    function hasEventListener(type:String):Boolean;
    function willTrigger(type:String):Boolean;
}
```

インターフェイスは、メソッドのインターフェイスとその実装間の違いを基にしています。メソッドのインターフェイスには、メソッドの名前、すべてのパラメータ、および戻り値の型など、そのメソッドを呼び出すために必要なすべての情報が含まれます。メソッドの実装には、インターフェイス情報だけではなく、メソッドのビヘイビアを実行する実行可能ステートメントも含まれます。インターフェイス定義には、メソッドインターフェイスのみが含まれ、インターフェイスを実装するクラスはメソッドの実装を定義します。

Flash Player API では、EventDispatcher クラスは、IEventDispatcher インターフェイスのメソッドすべてを定義し、各メソッドにメソッドの本体を追加して、IEventDispatcher インターフェイスを実装します。次のコードは、EventDispatcher クラスの定義からの抜粋です。

```
public class EventDispatcher implements IEventDispatcher
{
   function dispatchEvent(event:Event):Boolean
   {
      /* 実装ステートメント */
   }

   ...
}
```

IEventDispatcher インターフェイスは、EventDispatcher インスタンスがイベントオブジェクトを処理し、IEventDispatcher インターフェイスを実装する他のオブジェクトに渡すために使用するプロトコルとして機能します。

インターフェイスは、クラスのようにデータ型を定義すると説明することもできます。したがって、インターフェイスはクラスのように型注釈として使用できます。インターフェイスは、データ型として is および as 演算子などのデータ型を必要とする演算子と使用することもできます。しかし、クラスとは異なり、インターフェイスをインスタンス化することはできません。この違いから、多くのプログラマはインターフェイスを抽象データ型、クラスを具象データ型と捉えています。

インターフェイスの定義

インターフェイス定義の構造は、クラス定義の構造に似ていますが、インターフェイスにはメソッド本体のないメソッドしか含めることができません。インターフェイスに変数や定数を含めることはできませんが、getters および setter は含めることができます。インターフェイスを定義するには、interface キーワードを使用します。たとえば、次の IExternalizable インターフェイスは、Flash Player API の flash.utils パッケージに含まれています。IExternalizable インターフェイスは、オブジェクトを直列化する、つまりオブジェクトをデバイスでの保存用またはネットワーク上の伝達用に適した形式に変換するためのプロトコルを定義します。

```
public interface IExternalizable
{
   function writeExternal(output:IDataOutput):void;
   function readExternal(input:IDataInput):void;
}
```

IExternalizable インターフェイスは、public アクセス制御修飾子で宣言します。インターフェイス定義は、public および internal アクセス制御指定子でのみ変更できます。インターフェイス定義内のメソッド宣言には、アクセス制御指定子を使用できません。

Flash Player API では、インターフェイス名は大文字の I で始まるという表記規則に従いますが、インターフェイス名には有効な任意の識別子を使用できます。インターフェイス定義は、通常パッケージの最上位に配置されます。クラス定義内または別のインターフェイス定義内に、インターフェイス定義を配置することはできません。

インターフェイスは、他のインターフェイスを拡張できます。たとえば、次の IExample インターフェイスは IExternalizable インターフェイスを拡張します。

```
public interface IExample extends IExternalizable
{
    function extra():void;
}
```

IExample インターフェイスを実装するクラスは、`extra()` メソッドの実装だけでなく、IExternalizable インターフェイスから継承した `writeExternal()` および `readExternal()` メソッドの実装も含む必要があります。

クラス内でのインターフェイスの実装

クラスは、インターフェイスを実装できる唯一の ActionScript 3.0 言語エレメントです。クラス宣言内で `implements` キーワードを使用して、1 つまたは複数のインターフェイスを実装します。次の例では、IAlpha および IBeta の 2 つのインターフェイスと、これら両方のインターフェイスを実装する Alpha クラスを定義します。

```
interface IAlpha
{
    function foo(str:String):String;
}

interface IBeta
{
    function bar():void;
}

class Alpha implements IAlpha, IBeta
{
    public function foo(param:String):String {}
    public function bar():void {}
}
```

インターフェイスを実装するクラスでは、実装されるメソッドは以下を行う必要があります。

- `public` アクセス制御識別子を使用する。
- インターフェイスメソッドと同じ名前を使用する。
- 同じ数のパラメータを含む。各パラメータのデータ型は、インターフェイスメソッドパラメータのデータ型と一致する必要があります。
- 同じ戻り値の型を使用する。

実装するメソッドのパラメータには、ある程度自由に名前を付けることができます。実装されるメソッドとインターフェイスメソッドのパラメータ数および各パラメータのデータ型は一致する必要がありますが、パラメータ名を一致させる必要はありません。たとえば、前の例では Alpha.foo() メソッドのパラメータの名前は param です。

```
public function foo(param:String):String {}
```

しかし、IAlpha.foo() インターフェイスメソッドのパラメータの名前は str です。

```
function foo(str:String):String;
```

デフォルトのパラメータ値で柔軟性も得られます。インターフェイス定義は、デフォルトのパラメータ値を備えた関数宣言を含むことができます。このような関数宣言を実装するメソッドは、インターフェイス定義に指定されている値と同じデータ型のメンバーであるデフォルトのパラメータ値を持つ必要がありますが、実際の値が一致する必要はありません。たとえば、次のコードは、デフォルトのパラメータ値 3 を持つメソッドを含むインターフェイスを定義します。

```
interface Igamma
{
    function doSomething(param:int = 3):void;
}
```

次のクラス定義は、Igamma インターフェイスを実装しますが、異なるデフォルトパラメータ値を使用します。

```
class Gamma implements Igamma
{
    public function doSomething(param:int = 4):void {}
}
```

この柔軟性の理由は、インターフェイスを実装する規則がデータ型の互換性を確保するように特別に設計されているからで、同一のパラメータ名およびデフォルトパラメータ値を要求することはその目的の達成のために必要ありません。

継承

継承は、コード再利用の一形式であり、プログラマは既存のクラスを基に新しいクラスを開発することができます。既存のクラスは " 基本クラス " または " スーパークラス " と呼ばれますが、新しいクラスは通常 " サブクラス " と呼ばれます。継承の主な利点は、既存のコードをそのまま維持しながら、基本クラスのコードを再利用できることです。さらに、継承では他のクラスが基本クラスとやり取りする方法を変更する必要がありません。入念にテストされたか、既に使用されている既存クラスを変更するのではなく、継承を使用することでそのクラスを追加プロパティまたはメソッドで拡張できる統合モジュールとして使用できます。このため、extends キーワードを使用して、クラスが別のクラスを継承することを示すことができます。

継承を使用すると、コード内で " ポリモーフィズム " を利用することもできます。ポリモーフィズムを使用すると、異なるデータ型に適用された場合に異なる動作を実行するメソッドに単一のメソッド名を使用できます。単純な例として、2 つのサブクラス Circle と Square を持つ Shape という名前の基本クラスがあります。Shape クラスは、形状の面積を返す area() というメソッドを定義します。ポリモーフィズムが実装されている場合、Circle 型および Square 型のオブジェクトで area() メソッドを呼び出して、正しい計算を行うことができます。継承では、サブクラスが基本クラスからメソッドを継承し再定義できるようにする、つまり " オーバーライド " を許可することで、ポリモーフィズムが有効になります。次の例では、area() メソッドを Circle クラスと Square クラスで再定義します。

```
class Shape
{
    public function area():Number
    {
        return NaN;
    }
}

class Circle extends Shape
{
    private var radius:Number = 1;
    override public function area():Number
    {
        return (Math.PI * (radius * radius));
    }
}

class Square extends Shape
{
    private var side:Number = 1;
    override public function area():Number
    {
        return (side * side);
    }
}
```

```
var cir:Circle = new Circle();
trace(cir.area()); // 3.141592653589793
var sq:Square = new Square();
trace(sq.area()); // 1
```

各クラスはデータ型を定義するため、継承を使用すると、基本クラスと基本クラスを拡張するクラスの特別な関係が作成されます。サブクラスは、基本クラスのすべてのプロパティを保有します。これは、サブクラスのインスタンスは常に基本クラスのインスタンスの代わりに使用できることを意味します。たとえば、メソッドが Shape 型のパラメータを定義する場合、Circle 型は Shape 型を拡張するので Circle 型のパラメータを渡すことができます。次に例を示します。

```
function draw(shapeToDraw:Shape) {}

var myCircle:Circle = new Circle();
draw(myCircle);
```

インスタンスプロパティと継承

インスタンスプロパティは、`function`、`var`、または `const` キーワードのいずれで定義されたかに関係なく、プロパティが基本クラスの `private` 属性で宣言されていない限り、すべてのサブクラスに継承されます。たとえば、Flash Player API の Event クラスには、すべてのイベントオブジェクトに共通するプロパティを継承する多数のサブクラスがあります。

一部の型のイベントでは、Event クラスにイベントを定義するために必要なすべてのプロパティが含まれます。これらの型のイベントでは、Event クラスで定義されたもの以外のインスタンスプロパティは必要ありません。こうしたイベントの例として、データが正常にロードされたときに発生する `complete` イベント、ネットワーク接続が確立されたときに発生する `connect` イベントなどがあります。

次の例は、サブクラスに継承されるプロパティおよびメソッドの一部を示す Event クラスからの抜粋です。プロパティが継承されるため、サブクラスのインスタンスはこれらのプロパティにアクセスできます。

```
public class Event
{
  public function get type():String;
  public function get bubbles():Boolean;
  ...

  public function stopPropagation():void {}
  public function stopImmediatePropagation():void {}
  public function preventDefault():void {}
  public function isDefaultPrevented():Boolean {}
  ...
}
```

その他の型のイベントには、Event クラスで使用できない固有のプロパティが必要です。これらのイベントは、Event クラスで定義されたプロパティに新しいプロパティを追加できるように、Event クラスのサブクラスを使用して定義されます。こうしたサブクラスには、mouseMove イベント、click イベントなど、マウス操作またはマウスクリックに関連付けられているイベントに固有のプロパティを追加する MouseEvent クラスなどがあります。次の例は、サブクラスにあるが基本クラスにはないプロパティの定義を示す MouseEvent クラスからの抜粋です。

```
public class MouseEvent extends Event
{
    public static const CLICK:String      = "click";
    public static const MOUSE_MOVE:String = "mouseMove";
    ...

    public function get stageX():Number {}
    public function get stageY():Number {}
    ...
}
```

アクセス制御指定子と継承

プロパティが public キーワードで宣言された場合、このプロパティは任意の場所のコードから参照できます。つまり、public キーワードは、private、protected、および internal キーワードとは異なり、プロパティの継承に制限を加えません。

プロパティが private キーワードで宣言された場合、このプロパティは定義されたクラス内でのみ参照でき、サブクラスに継承されません。このビヘイビアは、旧バージョンの ActionScript とは異なります。旧バージョンでは、private キーワードは ActionScript 3.0 の protected キーワードのように動作します。

protected キーワードは、プロパティが、定義されたクラス内だけでなくすべてのサブクラスからも参照できることを示します。Java の protected キーワードとは異なり、ActionScript 3.0 の protected キーワードでは同じパッケージ内の他のすべてのクラスからプロパティを参照できるわけではありません。ActionScript 3.0 では、サブクラスのみが protected キーワードで宣言されたプロパティにアクセスできます。また、protected プロパティは、サブクラスが基本クラスと同じパッケージ内にあるか別のパッケージ内にあるかに関係なく、サブクラスから参照できます。

プロパティが定義されているパッケージからのプロパティへの参照を制限するには、internal キーワードを使用するか、またはアクセス制御指定子を一切使用しないようにします。internal アクセス制御指定子は、アクセス制御指定子が指定されていないときに適用されるデフォルトのアクセス制御指定子です。internal とマークされたプロパティは、同じパッケージ内にあるサブクラスにのみ継承されます。

次の例を使用して、各アクセス制御指定子がパッケージ境界を越えてどのように継承に影響するかを確認できます。次のコードは、AccessControl というメインアプリケーションクラスと Base および Extender というその他のクラス 2 つを定義します。Base クラスは foo パッケージ内にあり、Base クラスのサブクラスである Extender クラスは bar パッケージ内にあります。AccessControl クラスは Extender クラスのみを読み込み、Base クラスで定義されている `str` という変数にアクセスしようとする Extender のインスタンスを作成します。コードがコンパイルおよび実行されるように、次の抜粋に示すように、`str` 変数は `public` として宣言されます。

```
// "foo" フォルダ内の "Base.as"
package foo
{
    public class Base
    {
        public var str:String = "hello"; // この行の public を変更する
    }
}

// "bar" フォルダ内の "Extender.as"
package bar
{
    import foo.Base;
    public class Extender extends Base
    {
        public function getString():String {
            return str;
        }
    }
}

// "ProtectedExample.as" ファイル内のメインアプリケーションクラス
    import flash.display.MovieClip;
    import bar.Extender;
    public class AccessControl extends MovieClip
    {
        public function AccessControl()
        {
            var myExt:Extender = new Extender();
            trace (myExt.testString);  // str が public でない場合はエラー
            trace (myExt.getString()); // str が private または internal の場合はエラー
        }
    }
}
```

前の例のコンパイルおよび実行にその他のアクセス制御指定子がどのように影響を与えるかを確認するには、AccessControl クラスの次の行を削除またはコメントアウトした後で、str 変数のアクセス制御指定子を private、protected、または internal に変更します。

```
trace (myExt.testString);  // str が public でない場合はエラー
```

変数のオーバーライド禁止

var または const キーワードで宣言されたプロパティは継承されますが、オーバーライドすることはできません。プロパティをオーバーライドすると、サブクラスでプロパティが再定義されます。オーバーライドできるプロパティの型は、メソッド、つまり function キーワードで宣言されたプロパティだけです。インスタンス変数をオーバーライドすることはできませんが、インスタンス変数の getter および setter メソッドを作成し、これらのメソッドをオーバーライドすることで、同じ機能を実現できます。詳細については、135 ページの「getter および setter のオーバーライド」を参照してください。

メソッドのオーバーライド

メソッドをオーバーライドすると、継承されたメソッドの動作が再定義されます。静的メソッドは継承されず、オーバーライドすることはできません。しかし、次の 2 つの条件が満たされている場合、インスタンスメソッドはサブクラスに継承され、オーバーライドできます。

- インスタンスメソッドが基本クラス内で final キーワードで宣言されていない。final キーワードがインスタンスメソッドで使用された場合、サブクラスによってメソッドがオーバーライドされないようにプログラマが意図的に指定したことを示します。
- インスタンスメソッドが基本クラス内で private アクセス制御指定子で宣言されていない。基本クラス内でメソッドが private としてマークされている場合、基本クラスのメソッドはサブクラスから参照できないため、サブクラス内で同じ名前のメソッドを定義するときに override キーワードを使用する必要はありません。

上記の条件を満たすインスタンスメソッドをオーバーライドするには、サブクラス内のメソッドの定義は override キーワードを使用し、次の方法でメソッドのスーパークラスバージョンと一致する必要があります。

- オーバーライドメソッドには、基本クラスのメソッドと同じレベルのアクセス制御が必要です。internal とマークされたメソッドには、アクセス制御指定子を持たないメソッドと同じレベルのアクセス制御が必要です。
- オーバーライドメソッドには、基本クラスのメソッドと同じ数のパラメータが必要です。
- オーバーライドメソッドのパラメータには、基本クラスのメソッドのパラメータと同じデータ型注釈が必要です。
- オーバーライドメソッドには、基本クラスのメソッドと同じ戻り値の型が必要です。

ただし、オーバーライドメソッドのパラメータの名前は、パラメータの数および各パラメータのデータ型が一致していれば、基本クラスのパラメータの名前と一致する必要はありません。

super ステートメント

メソッドをオーバーライドするとき、多くのプログラマが、動作を完全に置き換えるのではなく、オーバーライドするスーパークラスメソッドの動作に追加したいと考えるでしょう。これには、サブクラス内のメソッドがそれ自体のスーパークラスバージョンを呼び出すメカニズムが必要です。super ステートメントは、直接のスーパークラスへの参照が含まれるそのようなメカニズムを提供します。次の例では、thanks() というメソッドを含む Base クラスと、thanks() メソッドをオーバーライドする Extender という Base クラスのサブクラスを定義します。Extender.thanks() メソッドは、super ステートメントを使用して Base.thanks() を呼び出します。

```
package {
    import flash.display.MovieClip;
    public class SuperExample extends MovieClip
    {
        public function SuperExample()
        {
            var myExt:Extender = new Extender()
            trace(myExt.thanks()); // 出力 : Mahalo nui loa
        }
    }
}

class Base {
    public function thanks():String
    {
        return "Mahalo";
    }
}

class Extender extends Base
{
    override public function thanks():String
    {
        return super.thanks() + " nui loa";
    }
}
```

getter および setter のオーバーライド

スーパークラスで定義された変数をオーバーライドすることはできませんが、getter および setter をオーバーライドすることができます。たとえば、次のコードは、Flash Player API の MovieClip クラスで定義された currentLabel という名前の getter をオーバーライドします。

```
package
{
   import flash.display.MovieClip;
   public class OverrideExample extends MovieClip
   {
      public function OverrideExample()
      {
         trace(currentLabel)
      }
      override public function get currentLabel():String
      {
         var str:String = "Override: ";
         str += super.currentLabel;
         return str;
      }
   }
}
```

OverrideExample クラスコンストラクタの `trace()` ステートメントの出力は `Override: null` で、この例では継承された `currentLabel` プロパティをオーバーライドできたことを示します。

継承されない静的プロパティ

静的プロパティはサブクラスに継承されません。つまり、サブクラスのインスタンスから静的プロパティにアクセスすることはできません。静的プロパティは、そのプロパティが定義されたクラスオブジェクトからのみアクセス可能です。たとえば、次のコードは、Base という基本クラスと、Extender という Base クラスを拡張するサブクラスを定義します。静的変数 `test` は Base クラスで定義されています。次の抜粋のコードは、strict モードではコンパイルされず、standard モードではランタイムエラーを生成します。

```
package {
   import flash.display.MovieClip;
   public class StaticExample extends MovieClip
   {
      public function StaticExample()
      {
         var myExt:Extender = new Extender();
         trace(myExt.test);  // エラー
      }
   }
}

class Base {
   public static var test:String = "static";
}

class Extender extends Base { }
```

静的変数 `test` にアクセスするには、次のコードに示すように、クラスオブジェクトからアクセスする必要があります。

```
Base.test;
```

静的プロパティと同じ名前を使用してインスタンスプロパティを定義することができます。このインスタンスプロパティは、静的プロパティと同じクラス内またはサブクラス内で定義できます。たとえば、前述の例の Base クラスでは、`test` というインスタンスプロパティを定義できます。次のコードはコンパイルおよび実行されます。これは、インスタンスプロパティが Extender クラスに継承されるからです。test インスタンス変数の定義が Extender クラスにコピーされるのではなく、移動された場合も、コードはコンパイルされ、実行されます。

```
package
{
	import flash.display.MovieClip;
	public class StaticExample extends MovieClip
	{
		public function StaticExample()
		{
			var myExt:Extender = new Extender();
			trace(myExt.test);  // 出力 : instance
		}
	}
}

class Base
{
	public static var test:String = "static";
	public var test:String = "instance";
}

class Extender extends Base {}
```

静的プロパティとスコープチェーン

静的プロパティは継承されませんが、そのプロパティが定義されたクラスおよびそのクラスのサブクラスのスコープチェーン内にあります。このため、静的プロパティは、定義されたクラスおよびサブクラスの " スコープ内 " にあると言います。つまり、静的プロパティが定義されたクラスおよびそのサブクラスの本体内から直接静的プロパティにアクセスできます。

次の例では、前述の例で定義したクラスを変更して、Base クラスで定義された静的変数 `test` が Extender クラスのスコープ内にあることを示します。つまり、Extender クラスは、`test` を定義したクラスの名前を接頭辞として変数に付けなくても静的変数 `test` にアクセスできます。

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
  }
}

class Base {
  public static var test:String = "static";
}

class Extender extends Base
{
  public function Extender()
  {
    trace(test); // 出力 : static
  }

}
```

インスタンスプロパティが、同じクラスまたはスーパークラス内で静的プロパティと同じ名前を使用するように定義されている場合、インスタンスプロパティはスコープチェーン内で優先順位が高くなります。インスタンスプロパティは、静的プロパティを " シャドウする " と言います。これは、静的プロパティの値ではなくインスタンスプロパティの値が使用されることを意味します。たとえば、次のコードは、Extender クラスが test というインスタンス変数を定義すると、`trace()` ステートメントは、静的変数の値ではなくインスタンス変数の値を使用することを示します。

```
package
{
  import flash.display.MovieClip;
  public class StaticExample extends MovieClip
  {
    public function StaticExample()
    {
      var myExt:Extender = new Extender();
    }
  }
}

class Base
{
  public static var test:String = "static";
}

class Extender extends Base
{
  public var test:String = "instance";
```

```
    public function Extender()
    {
        trace(test); // 出力 : instance
    }

}
```

高度なテクニック

このセクションでは、最初に ActionScript と OOP の歴史について簡単に説明します。次に、ActionScript 3.0 オブジェクトモデル、およびそれによって新しい ActionScript 仮想マシン (AVM2) を古い ActionScript 仮想マシン (AVM1) を使用する旧バージョンの Flash Player に対して大幅に高速に実行可能にする方法について解説します。

ActionScript の OOP サポートの歴史

ActionScript 3.0 は旧バージョンの ActionScript をベースに構築されているため、ActionScript オブジェクトモデルがどのように進化してきたかを理解しておくと役立ちます。ActionScript は、Flash オーサリングツールの初期バージョン用の単純なスクリプトメカニズムとして生まれました。その後、プログラマは ActionScript を使用してより複雑なアプリケーションを作成し始めました。プログラマのニーズに応えるために、その後の各リリースには複雑なアプリケーションの作成を容易にする言語機能が追加されてきました。

ActionScript 1.0

ActionScript 1.0 は、Flash Player 6 以前で使用されていた言語のバージョンです。この開発初期段階でも、ActionScript オブジェクトモデルは基本的なデータ型としてのオブジェクトの概念をベースにしていました。ActionScript オブジェクトは、" プロパティ " のグループを持つ複合データ型です。オブジェクトモデルについて説明するとき、" プロパティ " という用語には、変数、関数、メソッドなど、オブジェクトに関連付けられるあらゆるものが含まれます。

第 1 世代の ActionScript では `class` キーワードによるクラスの定義はサポートされませんが、プロトタイプオブジェクトと呼ばれる特殊な種類のオブジェクトを使用してクラスを定義することができます。`class` キーワードを使用して具象オブジェクトにインスタンス化する抽象クラスの定義を作成するのではなく、Java や C++ などのクラスベース言語の場合と同様に、ActionScript 1.0 のようなプロトタイプベース言語では、既存オブジェクトを他のオブジェクトのモデル (またはプロトタイプ) として使用します。クラスベース言語のオブジェクトはテンプレートになるクラスを示す場合がありますが、プロトタイプベース言語のオブジェクトはテンプレートになる別のオブジェクト、つまりプロトタイプを示します。

ActionScript 1.0 でクラスを作成するには、クラスのコンストラクタ関数を定義します。ActionScript の関数は、単に抽象的な定義ではなく実際のオブジェクトです。作成したコンストラクタ関数は、そのクラスのインスタンスのプロトタイプ的なオブジェクトになります。次のコードは、Shape というクラスを作成し、デフォルトで `true` に設定される `visible` というプロパティを 1 つ定義します。

```
// 基本クラス
function Shape() {}
// visible というプロパティを作成する。
Shape.prototype.visible = true;
```

このコンストラクタ関数は、次のように `new` 演算子でインスタンス化できる Shape クラスを定義します。

```
myShape = new Shape();
```

Shape コンストラクタ関数オブジェクトが Shape クラスのインスタンスのプロトタイプになるように、これは Shape のサブクラスのプロトタイプ、つまり Shape クラスを拡張するその他のクラスにもなります。

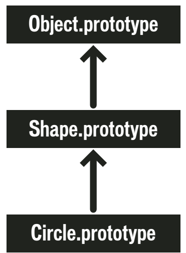
Shape クラスのサブクラスであるクラスを作成するには、次の 2 つの手順を実行します。最初に、次のようにクラスのコンストラクタ関数を定義して、クラスを作成します。

```
// 子クラス
function Circle(id, radius)
{
    this.id = id;
    this.radius = radius;
}
```

次に、`new` 演算子を使用して、Shape クラスが Circle クラスのプロトタイプであると宣言します。デフォルトでは、作成したクラスはそのプロトタイプとして Object クラスを使用します。つまり、`Circle.prototype` には現在汎用オブジェクト (Object クラスのインスタンス) が含まれています。Circle のプロトタイプに Object ではなく Shape を指定するには、汎用オブジェクトではなく Shape オブジェクトが含まれるように、次のコードを使用して `Circle.prototype` の値を変更します。

```
// Circle を Shape のサブクラスにする
Circle.prototype = new Shape();
```

これで、Shape クラスと Circle クラスは " プロトタイプチェーン " と呼ばれる継承関係内でリンクされました。次の図は、プロトタイプチェーン内の関係を示します。



各プロトタイプチェーンの最後にある基本クラスは Object クラスです。Object クラスには、ActionScript 1.0 で作成されたすべてのオブジェクトの基本プロトタイプオブジェクトを指す `Object.prototype` という静的プロパティが含まれます。このプロトタイプチェーンの次のオブジェクトは Shape オブジェクトです。これは、`Shape.prototype` プロパティは明示的に設定されていないため、依然として汎用オブジェクト (Object クラスのインスタンス) を保持しているからです。プロトタイプチェーンの最後のリンクは Circle クラスで、プロトタイプの Shape クラスにリンクされています。`Circle.prototype` プロパティは Shape オブジェクトを保持します。

次の例に示すように、Circle クラスのインスタンスを作成すると、インスタンスは Circle クラスのプロトタイプチェーンを継承します。

```
// Circle クラスのインスタンスを作成する
myCircle = new Circle();
```

Shape クラスのメンバーとして `visible` というプロパティを既に作成しました。例では、`visible` プロパティは、`myCircle` オブジェクトの一部としてではなく Shape オブジェクトのメンバーとしてのみ存在しますが、次のコード行では `true` が出力されます。

```
trace(myCircle.visible); // true
```

Flash Player では、プロトタイプチェーン内を移動することにより、`myCircle` オブジェクトが `visible` プロパティを継承することを確認できます。このコードを実行すると、Flash Player は最初に `myCircle` オブジェクトのプロパティから `visible` というプロパティを検索しますが、このプロパティは見つかりません。Flash Player は次に `Circle.prototype` オブジェクトを検索しますが、`visible` というプロパティは見つかりません。Flash Player はプロトタイプチェーン内を引き続き検索し、最後に `Shape.prototype` オブジェクトで定義された `visible` プロパティを見つけ、そのプロパティの値を出力します。

簡潔にするために、このセクションではプロトタイプチェーンの詳細および複雑さに関する説明の多くを省略し、代わりに ActionScript 3.0 オブジェクトモデルの理解に役立つ情報を提供します。

ActionScript 2.0

ActionScript 2.0 では、Java や C++ などのクラスベース言語の経験があればなじみのある方法でクラスを定義できる class、extends、public、および private などの新しいキーワードが導入されました。ActionScript 1.0 と ActionScript 2.0 では基になる継承メカニズムが変更されていないことを理解しておくことが重要です。ActionScript 2.0 では、クラスを定義するための新しいシンタックスが追加されただけです。プロトタイプチェーンは、この 2 つのバージョンで同じように機能します。

次の抜粋に示すように ActionScript 2.0 で導入された新しいシンタックスでは、より直観的な方法でクラスを定義できます。

```
// 基本クラス
class Shape
{
    var visible:Boolean = true;
}
```

ActionScript 2.0 では、コンパイル時の型チェックで使用するための型注釈も導入されました。これにより、前述の例の visible プロパティにはブール値だけが含まれることを宣言できます。また、新しい extends キーワードにより、サブクラスを作成するプロセスが簡略化されます。次の例では、ActionScript 1.0 では 2 つの手順が必要なプロセスが、extends キーワードにより 1 つの手順で実行されています。

```
// 子クラス
class Circle extends Shape
{
   var id:Number;
   var radius:Number;
   function Circle(id, radius)
   {
      this.id = id;
      this.radius = radius;
   }
}
```

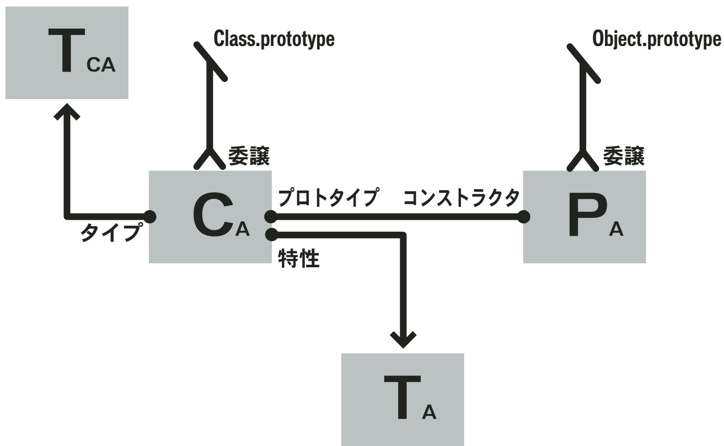
コンストラクタは、クラス定義の一部として宣言されています。クラスプロパティ id と radius も明示的に宣言する必要があります。

ActionScript 2.0 ではインターフェイス定義のサポートも追加され、オブジェクト間の通信用に正式に定義されたプロトコルでオブジェクト指向プログラムをさらに改良できるようになりました。

ActionScript 3.0 のクラスオブジェクト

通常 Java や C++ に関連付けられる一般的なオブジェクト指向プログラミングパラダイムでは、クラスを使用してオブジェクトの型を定義します。このパラダイムを採り入れたプログラミング言語も、クラスにより定義されるデータ型のインスタンスを作成するためにクラスを使用する傾向にあります。ActionScript では、この両方の目的でクラスを使用しますが、プロトタイプベース言語であることから興味深い特徴が付け加えられています。ActionScript ではクラス定義ごとに、動作と状態の両方を共有できる特別なクラスオブジェクトを作成します。とは言え、コーディングする上でこの違いが実質的な影響があると感じる ActionScript プログラマはほとんどいないでしょう。ActionScript 3.0 は、この特別なオブジェクトを使用しなくても、さらには理解していなくても、高度なオブジェクト指向 ActionScript アプリケーションを作成できるように設計されています。このセクションでは、クラスオブジェクトを最大限に活用したい熟練プログラマを対象に、この問題を掘り下げて説明します。

次の図は、`class A {}` ステートメントで定義された A という名前の単純なクラスを表すクラスオブジェクトの構造を示します。



図中の四角形はオブジェクトを表します。図中のオブジェクトには、クラス A に属していることを表す下付き文字 A が付いています。クラスオブジェクト (C_A) には、その他の重要なオブジェクトへの参照が含まれます。インスタンス特性オブジェクト (T_A) には、クラス定義内で定義されたインスタンスプロパティが格納されます。クラス特性オブジェクト (T_{CA}) は、クラスの内部型を表し、そのクラスにより定義された静的プロパティを格納します。下付き文字 C は " クラス " を表します。プロトタイプオブジェクト (P_A) は、`constructor` プロパティで最初に関連付けられたクラスオブジェクトを常に参照します。

特性オブジェクト

ActionScript 3.0 で新しく導入された特性オブジェクトは、パフォーマンスを考慮して実装されました。旧バージョンの ActionScript では、名前のルックアップは、Flash Player がプロトタイプチェーン内を移動するため時間のかかるプロセスでした。ActionScript 3.0 では、継承されたプロパティがスーパークラスからサブクラスの特性オブジェクトにコピーされるので、名前のルックアップは効率的になり時間が短縮されました。

特性オブジェクトはプログラマコードに直接アクセスできませんが、パフォーマンスが向上しメモリ使用量が削減することからオブジェクトの存在がわかります。特性オブジェクトは、AVM2 にクラスのレイアウトと内容に関する詳細情報を提供します。この情報があれば、AVM2 は直接マシン命令を生成して時間のかかる名前のルックアップを行わずにプロパティにアクセスしたり、メソッドを直接呼び出したりすることができるため、実行時間を大幅に短縮できます。

特性オブジェクトのおかげで、オブジェクトのメモリフットプリントは、旧バージョンの ActionScript の同様のオブジェクトと比較してかなり小さくすることができます。たとえば、クラスが sealed の場合 (つまり、クラスが dynamic と宣言されていない場合)、クラスのインスタンスは動的に追加するプロパティにハッシュテーブルを必要とせず、このクラスで定義された固定プロパティの特性オブジェクトおよびスロットへのポインタを保持するだけです。その結果、ActionScript 2.0 で 100 バイトのメモリが必要だったオブジェクトが、ActionScript 3.0 では 20 バイトで済みます。

メモ	特性オブジェクトは、内部実装の詳細です。ActionScript の今後のバージョンで変更されない、またはなくならないとは限りません。

プロトタイプオブジェクト

ActionScript のクラスオブジェクトには、クラスのプロトタイプオブジェクトを参照する `prototype` プロパティがあります。プロトタイプオブジェクトは、ActionScript のプロトタイプベース言語としてのルーツのレガシーです。詳細については、139 ページの「ActionScript 1.0」を参照してください。

`prototype` プロパティは読み取り専用で、別のオブジェクトを指すように変更することはできません。これは、旧バージョンの ActionScript のクラスの `prototype` プロパティとは異なります。旧バージョンのプロパティでは、別のクラスを指すようにプロトタイプを再割り当てできました。`prototype` プロパティは読み取り専用ですが、参照されるプロトタイプオブジェクトは読み取り専用ではありません。つまり、プロトタイプオブジェクトに新しいプロパティを追加することができます。プロトタイプオブジェクトに追加されたプロパティは、クラスのすべてのインスタンス間で共有されます。

旧バージョンの ActionScript では唯一の継承メカニズムであったプロトタイプチェーンは、ActionScript 3.0 では二次的な役割のみを果たします。主要継承メカニズムである固定プロパティの継承は、特性オブジェクトによって内部的に処理されます。固定プロパティは、クラス定義の一部として定義される変数またはメソッドです。固定プロパティの継承は、class、extends、override などのキーワードで関連付けられる継承メカニズムであるため、クラス継承とも呼ばれます。

プロトタイプチェーンは、固定プロパティの継承より動的な代替継承メカニズムになります。プロパティは、クラス定義の一部としてだけでなく、実行時にクラスオブジェクトの prototype プロパティからもクラスのプロトタイプオブジェクトに追加することができます。ただし、コンパイラを strict モードに設定した場合は、クラスを dynamic キーワードで宣言しない限りプロトタイプオブジェクトに追加されたプロパティにアクセスできない場合があります。

いくつものプロパティがプロトタイプオブジェクトに関連付けられているクラスの好例として、Object クラスがあります。Object クラスの toString() と valueOf() メソッドは、実際には Object クラスのプロトタイプオブジェクトのプロパティに割り当てられた関数です。次の例は、これらのメソッドの宣言が理論的にどのように見えるかを示します。ただし、実装の詳細により実際の実装はやや異なります。

```
public dynamic class Object
{
    prototype.toString = function()
    {
        // ステートメント
    };
    prototype.valueOf = function()
    {
        // ステートメント
    };
}
```

前述したように、プロパティは、クラス定義外部でクラスのプロトタイプオブジェクトに関連付けることができます。たとえば、toString() メソッドは、次のように Object クラスの定義外部で定義することもできます。

```
Object.prototype.toString = function()
{
    // ステートメント
};
```

しかし、固定プロパティの継承とは異なり、サブクラスでメソッドを再定義する場合、プロトタイプの継承では override キーワードが必要ありません。たとえば、Object クラスのサブクラスで valueOf() メソッドを再定義する場合、次の 3 つのオプションがあります。1 つ目は、クラス定義内部のサブクラスのプロトタイプオブジェクトで valueOf() メソッドを定義します。次のコードは、Foo という Object のサブクラスを作成し、Foo のプロトタイプオブジェクトでクラス定義の一部として valueOf() メソッドを再定義します。どのクラスも Object を継承するため、extends キーワードを使用する必要はありません。

```
dynamic class Foo
{
  prototype.valueOf = function()
  {
    return "Instance of Foo";
  };
}
```

2 つ目は、次のコードに示すように、クラス定義外にある Foo のプロトタイプオブジェクトで valueOf() メソッドを定義します。

```
Foo.prototype.valueOf = function()
{
  return "Instance of Foo";
};
```

3 つ目は、Foo クラスの一部として valueOf() という固定プロパティを定義します。この方法は、固定プロパティの継承とプロトタイプの継承が混在するという点で他の 2 つとは異なります。valueOf() を再定義する Foo のサブクラスでは、override キーワードを使用する必要があります。次のコードは、Foo で固定プロパティとして定義された valueOf() を示します。

```
class Foo
{
  function valueOf() {
    return "Instance of Foo";
  }
}
```

AS3 名前空間

固定プロパティの継承とプロトタイプの継承という 2 つの別々の継承メカニズムが存在すると、コアクラスのプロパティおよびメソッドに関して、興味深い互換性の問題が生じます。ECMAScript Edition 4 言語仕様案との互換性からは、プロトタイプの継承を使用する必要があります。つまり、コアクラスのプロパティおよびメソッドは、そのクラスのプロトタイプオブジェクトで定義されます。一方、Flash Player API との互換性では、固定プロパティの継承を使用することが要求されます。つまり、コアクラスのプロパティおよびメソッドは、`const`、`var`、および `function` のキーワードを使用してクラス定義で定義されます。さらに、プロトタイプバージョンの代わりに固定プロパティを使用することで、ランタイムパフォーマンスを著しく向上させることが可能です。

ActionScript 3.0 では、この問題を解決するために、コアクラスにプロトタイプの継承と固定プロパティの継承の両方を使用しています。各コアクラスに、2 セットのプロパティおよびメソッドが含まれます。1 セットは、ECMAScript 仕様との互換性のためにプロトタイプオブジェクトで定義され、残りの 1 セットは、Flash Player API との互換性のために固定プロパティおよび AS3 名前空間で定義されます。

AS3 名前空間は、この 2 セットのプロパティおよびメソッド間の選択のために便利なメカニズムを提供します。AS3 名前空間を使用しない場合、コアクラスのインスタンスは、コアクラスのプロトタイプオブジェクトに定義されているプロパティおよびメソッドを継承します。AS3 名前空間を使用することに決めた場合は、固定プロパティがプロトタイププロパティよりも常に優先されるため、コアクラスのインスタンスは AS3 バージョンを継承します。つまり、固定プロパティが利用可能な場合には、同じ名前のプロトタイププロパティではなく、その固定プロパティが必ず使用されます。

AS3 名前空間で修飾することによって、AS3 名前空間バージョンのプロパティまたはメソッドを選択的に使用することができます。たとえば、次のコードでは、AS3 バージョンの `Array.pop()` メソッドを使用しています。

```
var nums:Array = new Array(1, 2, 3);
nums.AS3::pop();
trace(nums); // output: 1,2
```

代わりに、`use namespace` ディレクティブを使用してコードブロック内のすべての定義に対して AS3 名前空間を開くことができます。たとえば、次のコードでは、`use namespace` ディレクティブを使用して `pop()` メソッドと `push()` メソッドの両方に対して AS3 名前空間を開いています。

```
use namespace AS3;

var nums:Array = new Array(1, 2, 3);
nums.pop();
nums.push(5);
trace(nums) // output: 1,2,5
```

ActionScript 3.0 では、AS3 名前空間をプログラム全体に適用できるように、プロパティセットごとにコンパイラオプションも用意されています。-as3 コンパイラオプションは AS3 名前空間を表し、-es コンパイラオプション (es は ECMAScript の略) はプロトタイプ継承オプションを表します。プログラム全体に対して AS3 名前空間を開くには、-as3 コンパイラオプションを true に、-es コンパイラオプションを false に設定します。プロトタイプバージョンを使用するには、両方のコンパイラオプションを反対の値に設定します。Adobe Flex Builder 2 および Adobe Flash CS3 Professional のデフォルトのコンパイラ設定は、-as3 = true および -es = false です。

いずれかのコアクラスを拡張してメソッドのいずれかをオーバーライドする計画がある場合には、オーバーライドメソッドの宣言方法に AS3 名前空間がどのように影響する可能性があるかを理解しておく必要があります。AS3 名前空間を使用する場合は、コアクラスメソッドのいずれのメソッドオーバーライドも、override 属性で AS3 名前空間を使用する必要があります。AS3 名前空間を使用せずにコアクラスメソッドをサブクラスで再定義する場合は、AS3 名前空間も override キーワードも使用しないでください。

例 : GeometricShapes

サンプルアプリケーション GeometricShapes では、ActionScript 3.0 を使用して次のようにオブジェクト志向の概念および機能をいろいろと適用できることがわかります。

- クラスの定義
- クラスの拡張
- ポリモーフィズムと override キーワード
- インターフェイスの定義、拡張、実装

さらに、クラスインスタンスを作成する " ファクトリメソッド " も含まれ、インターフェイスのインスタンスとして戻り値を宣言し、戻されたオブジェクトを汎用的な方法で使用する方法が示されます。

GeometricShapes アプリケーションのファイルは、"Examples/GeometricShapes" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
GeometricShapes.mxml	MXML で記述された Flex 用メインアプリケーションファイル
com/example/programmingas3/geometricshapes/IGeometricShape.as	すべての GeometricShapes アプリケーションクラスによって実装されるメソッドを定義する基本インターフェイス
com/example/programmingas3/geometricshapes/IPolygon.as	複数の辺を持つ GeometricShapes アプリケーションクラスによって実装されるメソッドを定義するインターフェイス

ファイル	説明
com/example/programmingas3/geometricshapes/RegularPolygon.as	図形の中心をめぐって対称の位置に長さの等しい複数の辺を持つ図形のタイプ。
com/example/programmingas3/geometricshapes/Circle.as	円を定義する一種の幾何学図形
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	すべての辺が同じ長さの三角形を定義する RegularPolygon のサブクラス
com/example/programmingas3/geometricshapes/Square.as	四辺すべてが同じ長さの四角形を定義する RegularPolygon のサブクラス
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	指定されたシェイプ種類およびサイズでシェイプを作成するためのファクトリメソッドを含むクラス

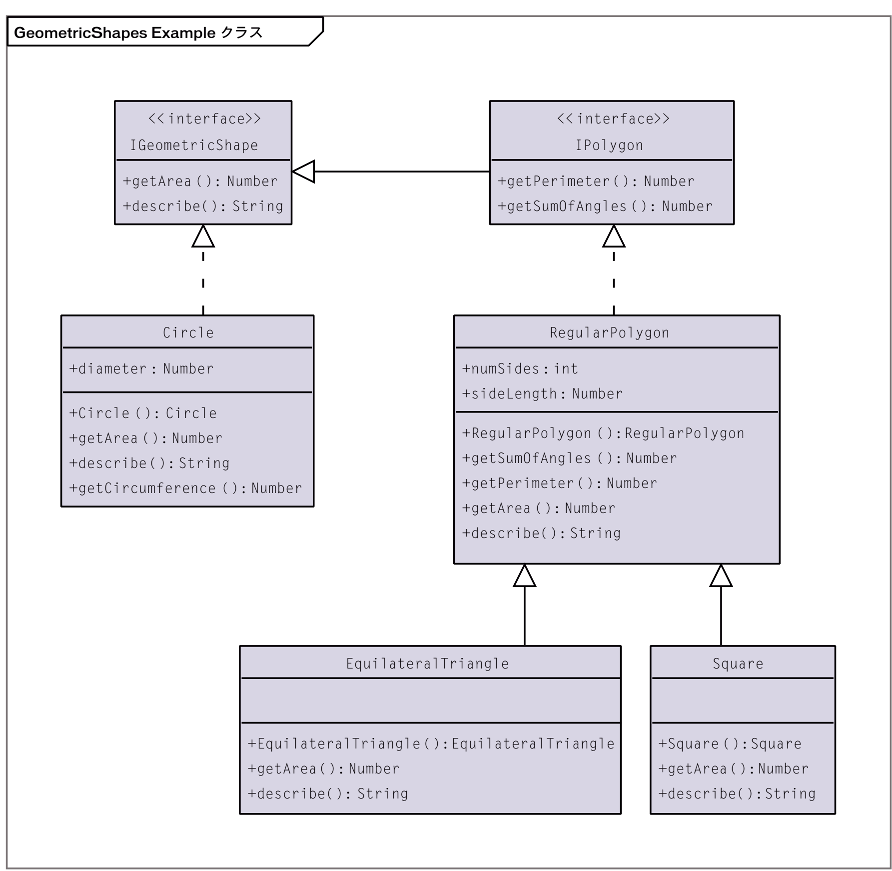
GeometricShapes クラスの定義

GeometricShapes アプリケーションを使用するユーザーは、幾何学図形の種類とサイズを指定することができます。それに対しアプリケーションは、シェイプの説明、その面積、および周辺の長さで応答します。

アプリケーションユーザーインターフェイスは、シェイプの種類を選択、サイズを設定、説明を表示するための小数のコントロールを含み、ごく普通です。このアプリケーションで最も興味深い部分は、内部にありそれはクラスおよびインターフェイス自体の構造に関係します。

このアプリケーションは幾何学図形を扱いますが、それを図では表示しません。後の章の 187 ページの「例 : SpriteArranger」で再利用する、クラスおよびインターフェイスの小規模なライブラリを提供します。Sprite Arranger の例ではシェイプが図で表示され、GeometricShapes アプリケーションによってここで提供されるクラスフレームワークに基づいて、ユーザーはシェイプを操作することができます。

次の図に、この例で幾何学図形を定義するクラスとインターフェイスを UML (Unified Modeling Language) 表記で示します。



インターフェイスでの一般的な動作の定義

この GeometricShapes アプリケーションでは、円、正方形、および等辺三角形の 3 種類のシェイプを扱います。GeometricShapes クラス構造は、非常に単純なインターフェイスである、3 種類すべてのシェイプに共通するメソッドをリストする IGeometricShape で開始します。

```
package com.example.programmingas3.geometricshapes
{
   public interface IGeometricShape
   {
      function getArea():Number;
      function describe():String;
   }
}
```

このインターフェイスでは getArea() メソッドと describe() メソッドの 2 つのメソッドを定義し、前者のメソッドはシェイプの面積を計算して返し、後者のメソッドはシェイプのプロパティのテキスト説明を組み立てます。

各シェイプの周囲の長さを知りたいこともあります。ただし、円の周囲は円周と呼ばれ、その計算方法は固有であるため、三角形または正方形とは動作が異なります。それでも、三角形、正方形、およびその他の多角形の間には十分に類似点があるため、それらに新しいインターフェイスクラスの IPolygon を定義することは意味があります。IPolygon インターフェイスも、次に示すように、どちらかと言えば単純です。

```
package com.example.programmingas3.geometricshapes
{
   public interface IPolygon extends IGeometricShape
   {
      function getPerimeter():Number;
      function getSumOfAngles():Number;
   }
}
```

このインターフェイスはすべての多角形に対して、すべての辺の長さを合算して計測する getPerimeter() メソッドと、すべての内角の総和を出す getSumOfAngles() メソッドの 2 つのメソッドを定義します。

IPolygon インターフェイスは IGeometricShape インターフェイスを拡張します。IPolygon インターフェイスを実装する任意のクラスは、4 つのすべてのメソッド、つまり IGeometricShape インターフェイスから 2 つと IPolygon インターフェイスから 2 つのメソッドを宣言する必要があることを意味します。

シェイプクラスの定義

各種類のシェイプに共通するメソッドに関する考えがまとまったら、シェイプクラス自体を定義することができます。実装する必要のあるメソッドの数で見た場合、最も単純なシェイプはここに示すように、Circle クラスです。

```
package com.example.programmingas3.geometricshapes
{
   public class Circle implements IGeometricShape
   {
      public var diameter:Number;

      public function Circle(diam:Number = 100):void
      {
         this.diameter = diam;
      }

      public function getArea():Number
      {
```

```
        // The formula is Pi * radius^2.
        return Math.PI * ((diameter / 2)^2);
      }

      public function getCircumference():Number
      {
        // The formula is Pi * radius * 2.
        return Math.PI * diameter;
      }

      public function describe():String
      {
        var desc:String = "This shape is a Circle.\n";
        desc += "Its diameter is " + diameter + " pixels.\n";
        desc += "Its area is " + getArea() + ".\n";
        desc += "Its circumference is " + getCircumference() + ".\n";
        return desc;
      }
    }
}
```

Circle クラスは IGeometricShape インターフェイスを実装するので、`getArea()` メソッドおよび `describe()` メソッドの両方のコードを提供する必要があります。さらに、Circle クラスに固有である `getCircumference()` メソッドを定義します。Circle クラスでは、他の多角形クラスでは見られない `diameter` プロパティの宣言もあります。

残りの 2 つの種類のシェイプの正方形および等辺三角形には、それぞれ等しい長さの辺を有するので、どちらも共通の式を使用して周辺と内角の総和を計算できるという共通点があります。実際、これらの共通の式は、今後定義する必要がある他の正多角形にも適用されます。

RegularPolygon クラスは、Square クラスと EquilateralTriangle クラスの両方のスーパークラスです。スーパークラスを使用すると、共通メソッドを 1 か所で定義することできるため、サブクラスごとに別々に共通メソッドを定義する必要がありません。RegularPolygon クラスのコードを次に示します。

```
package com.example.programmingas3.geometricshapes
{
  public class RegularPolygon implements IPolygon
  {
    public var numSides:int;
    public var sideLength:Number;

    public function RegularPolygon(len:Number = 100, sides:int = 3):void
    {
      this.sideLength = len;
      this.numSides = sides;
    }
```

```
    public function getArea():Number
    {
      // このメソッドはサブクラスでオーバーライドが必要
      return 0;
    }

    public function getPerimeter():Number
    {
      return sideLength * numSides;
    }

    public function getSumOfAngles():Number
    {
      if (numSides >= 3)
      {
        return ((numSides - 2) * 180);
      }
      else
      {
        return 0;
      }
    }

    public function describe():String
    {
      var desc:String = "Each side is " + sideLength + " pixels long.\n";
      desc += "Its area is " + getArea() + " pixels square.\n";
      desc += "Its perimeter is " + getPerimeter() + " pixels long.\n";
      desc += "The sum of all interior angles in this shape is " +
  getSumOfAngles() + " degrees.\n";
      return desc;
    }
  }
}
```

最初に、RegularPolygon クラスは、すべての正多角形に共通する 2 つのプロパティ、各辺の長さ sideLength プロパティと辺の数の numSides プロパティを宣言します。

RegularPolygon クラスは、IPolygon インターフェイスを実装し、IPolygon インターフェイスメソッドのすべての 4 つを宣言します。これらのうちの 2 つ、つまり getPerimeter() メソッドと getSumOfAngles() メソッドは、共通の式を使用して実装します。

getArea() メソッドの式はシェイプによって異なるため、基本クラスバージョンのメソッドは、サブクラスメソッドで継承可能な共通の論理を含むことができません。代わりに、面積を計算しなかったことを示すには、単にデフォルト値 0 を返します。各シェイプの面積を正しく計算するには、RegularPolygon クラスのサブクラスは getArea() メソッド自体をオーバーライドする必要があります。

EquilateralTriangle クラスの次のコードは、getArea() メソッドをどのようにオーバーライドするかを示しています。

```
package com.example.programmingas3.geometricshapes
{
  public class EquilateralTriangle extends RegularPolygon
  {
    public function EquilateralTriangle(len:Number = 100):void
    {
      super(len, 3);
    }

    public override function getArea():Number
    {
        // 式は ((sideLength の 2 乗 ) * (3 の平方根 )) / 4
        return ( (this.sideLength * this.sideLength) * Math.sqrt(3) ) / 4;
    }

    public override function describe():String
    {
         /* シェイプの名前で開始し、残りの説明を
         RegularPolygon スーパークラスに委任する */
        var desc:String = "This shape is an equilateral Triangle.\n";
        desc += super.describe();
        return desc;
    }
  }
}
```

override キーワードは、RegularPolygon スーパークラスから EquilateralTriangle.getArea() メソッドが getArea() メソッドを意図的にオーバーライドすることを示します。EquilateralTriangle.getArea() メソッドが呼び出されると、前記のコードの式を使用して面積が計算されます。RegularPolygon.getArea() メソッドのコードが実行されることは決してありません。

対照的に、EquilateralTriangle クラスは独自のバージョンの getPerimeter() メソッドを定義しません。EquilateralTriangle.getPerimeter() メソッドが呼び出されると、継承チェーンに進み、RegularPolygon サブクラスの getPerimeter() メソッドのコードが実行されます。

EquilateralTriangle() コンストラクタは super() ステートメントを使用して、そのスーパークラスの RegularPolygon() コンストラクタを明示的に呼び出します。両方のコンストラクタのパラメータセットが同じ場合は、EquilateralTriangle コンストラクタを完全に省略しておくこともできますが、代わりに RegularPolygon() コンストラクタが実行されます。ただし、RegularPolygon() コンストラクタは追加パラメータの numSides を必要とします。そのため EquilateralTriangle() コンストラクタは、三角形は 3 辺であることを示すために len 入力パラメータと値 3 を渡す super(len, 3) を呼び出します。

describe() メソッドも super() ステートメントを使用しますが、方法は異なり、RegularPolygon スーパークラスバージョンの describe() メソッドを呼び出します。EquilateralTriangle.describe() メソッドは最初に、desc ストリング変数をシェイプの種類に関する記述に設定します。次に、super.describe() を呼び出すことによって RegularPolygon.describe() メソッドの結果を取得し、その結果を desc ストリングに追加します。

ここでは Square クラスについて詳細は説明しません。しかし、このクラスは、EquilateralTriangle クラスと類似点があり、コンストラクタと getArea() および describe() メソッドの独自の実装を提供します。

ポリモーフィズムとファクトリメソッド

インターフェイスと継承を有効に利用するクラスセットは、いろいろな興味深い方法で使用することができます。たとえば、これまでに説明したすべてのシェイプクラスは、IGeometricShape インターフェイスを実装するか、またはそのようにスーパークラスを拡張します。したがって、IGeometricShape のインスタンスであるように変数を定義する場合は、describe() メソッドを呼び出すために、実際にインスタンスが Circle のインスタンスであるか Square クラスのインスタンスであるかを知っている必要はありません。

次のコードは、この働きを示します。

```
var myShape:IGeometricShape = new Circle(100);
trace(myShape.describe());
```

myShape.describe() が呼び出された場合、変数が IGeometricShape インターフェイスのインスタンスと定義されていても Circle が基礎のなるクラスであるため、Circle.describe() メソッドが実行されます。

この例は、ポリモーフィズムの動作原理を示しています。つまり、正確に同じメソッド呼び出しの場合も、メソッドが呼び出されるオブジェクトのクラスに応じて、実行されるコードは異なることになります。

GeometricShapes アプリケーションでは、この種類のインターフェイスベースのポリモーフィズムを適用するために、ファクトリメソッドとして知られている簡略化バージョンの設計パターンを使用します。" ファクトリメソッド " という用語は、基礎になるデータ型または内容が状況によって変わる可能性のあるオブジェクトを返す関数を指します。

ここに示す GeometricShapeFactory クラスは、createShape() というファクトリメソッドを次のように定義します。

```
package com.example.programmingas3.geometricshapes
{
    public class GeometricShapeFactory
    {
        public static var currentShape:IGeometricShape;

        public static function createShape(shapeName:String,
                                            len:Number):IGeometricShape
        {
```

```
        switch (shapeName)
        {
            case "Triangle":
                return new EquilateralTriangle(len);

            case "Square":
                return new Square(len);

            case "Circle":
                return new Circle(len);
        }
        return null;
    }

    public static function describeShape(shapeType:String,
  shapeSize:Number):String
    {
        GeometricShapeFactory.currentShape =
            GeometricShapeFactory.createShape(shapeType, shapeSize);
        return GeometricShapeFactory.currentShape.describe();
    }
  }
}
```

createShape() ファクトリメソッドを使用すると、作成するインスタンスの詳細をシェイプクラスのコンストラクタで定義することができます。もっと一般的な方法でアプリケーションが扱えるように、IGeometricShape インスタンスとして新しいオブジェクトを返します。

前述の例の describeShape() メソッドは、もっと具体的なオブジェクトへの汎用的な参照を取得するために、アプリケーションでファクトリメソッドをどのように使用することができるかを示しています。次のように、新しく作成された Circle オブジェクトの説明を取得することができます。

```
GeometricShapeFactory.describeShape(“Circle”, 100);
```

続いて、describeShape() メソッドは同じパラメータ付きで createShape() ファクトリメソッドを呼び出しますが、IGeometricShape オブジェクトとして型指定された currentShape という静的変数には、新しい Circle オブジェクトが格納されています。次に、currentShape オブジェクトで describe() メソッドが呼び出され、その呼び出しは自動的に解決されて Circle.describe() を実行し、円の詳細な説明を返します。

サンプルアプリケーションの機能拡張

インターフェイスおよび継承の実際の威力は、アプリケーションを機能拡張または変更したときに明らかになります。

新しいシェイプとして五角形をこのサンプルアプリケーションに追加するとします。RegularPolygon クラスを拡張して独自のバージョンの `getArea()` メソッドおよび `describe()` メソッドを定義する新しい Pentagon クラスを作成します。次に、アプリケーションのユーザーインターフェイスのコンボボックスに新しい Pentagon オプションを追加します。しかし、これで全部です。Pentagon クラスは、継承によって RegularPolygon クラスから `getPerimeter()` メソッドおよび `getSumOfAngles()` メソッドの機能を自動的に取得します。IGeometricShape インターフェイスを実装するクラスから継承するため、Pentagon インスタンスは IGeometricShape インスタンスとしても扱うことができます。つまり、GeometricShapeFactory クラスのいずれのメソッドも変更する必要がないので、新しい種類のシェイプを必要なときに追加することが非常に簡単になります。

実習として Pentagon クラスを Geometric Shapes の例に追加してみると、アプリケーションに新機能を追加する負荷がインターフェイスおよび継承によってどの程度軽減できるかがわかります。

第 5 章

表示のプログラミング

5

ActionScript 3.0 の表示のプログラミングでは、Adobe Flash Player 9 のステージ上に表示されるエレメントを操作できます。この章では、画面上のエレメントを操作するための基本概念について説明します。ビジュアルエレメントのプログラムによる構成の詳細および表示オブジェクトのカスタムクラスの作成について学習します。

目次

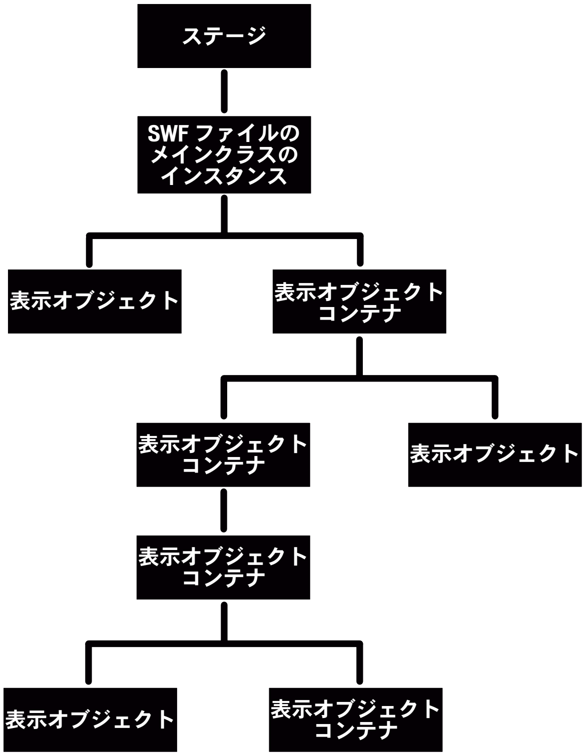
表示アーキテクチャについて 160
コア表示クラスを操作するための基礎 176
表示オブジェクトの操作 167
例 : SpriteArranger 187

表示アーキテクチャについて

ActionScript 3.0 で作成されたアプリケーションには、" 表示リスト " と呼ばれる表示されるオブジェクトの階層があります。表示リストには、アプリケーション内で表示されるエレメントがすべて含まれます。表示エレメントは、次の 1 つまたは複数のグループに分類されます。

■ ステージ

ステージは、表示オブジェクトの基本コンテナです。アプリケーションには、画面の表示オブジェクトすべてを含む Stage オブジェクトが 1 つあります。ステージは、最上位コンテナで、表示リスト階層の最上位に位置します。



それぞれの SWF ファイルには、関連する ActionScript クラスがあります。これは、SWF ファイルのメインクラスと呼ばれます。SWF ファイルを HTML ページに埋め込むと、Flash Player はそのクラスのコンストラクタ関数を呼び出し、作成されたインスタンス (これは常に一種の表示オブジェクト) が Stage オブジェクトの子として追加されます。SWF ファイルのメインクラスは、常に Sprite クラスを拡張します。詳細については、162 ページの「コア表示クラス」を参照してください。

DisplayObject インスタンスの stage プロパティからステージにアクセスできます。詳細については、173 ページの「Stage プロパティの設定」を参照してください。

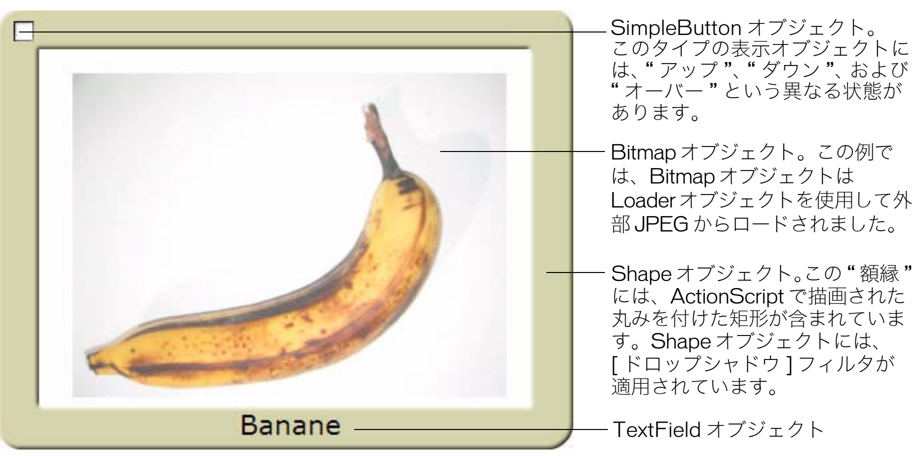
- 表示オブジェクト

  ActionScript 3.0 では、アプリケーション内で画面上に表示されるすべてのエレメントのタイプは、" 表示オブジェクト " です。flash.display パッケージには、他の多くのクラスにより拡張される基本クラスである DisplayObject クラスが含まれます。これらのさまざまなクラスは、ベクターシェイプ、ムービークリップ、テキストフィールドなどの各種の表示オブジェクトを表します。これらのクラスの概要については、162 ページの「コア表示クラス」を参照してください。

- 表示オブジェクトコンテナ

  表示オブジェクトコンテナは、表示オブジェクトである子オブジェクトを含めることができる特殊なタイプの表示オブジェクトです。

  DisplayObjectContainer クラスは、DisplayObject クラスのサブクラスです。DisplayObjectContainer オブジェクトには、子リスト内の複数の表示リストを含めることができます。たとえば、次の例は、さまざまな表示オブジェクトを格納する Sprite として知られている DisplayObjectContainer オブジェクトのタイプを示しています。



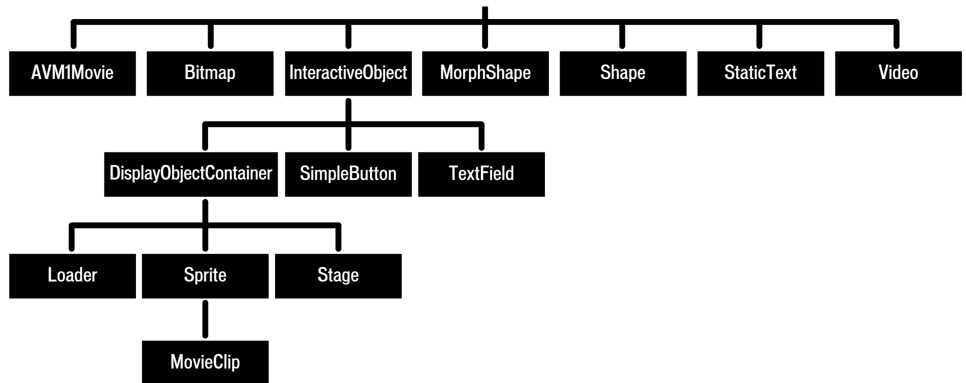
表示オブジェクトの説明では、DisplayObjectContainer オブジェクトは " 表示オブジェクトコンテナ "、または単に " コンテナ " と呼ばれます。

表示可能な表示オブジェクトはすべて DisplayObject クラスを継承しますが、そのタイプは DisplayObject クラスの特定のサブクラスです。たとえば、Shape クラスまたは Video クラスにはコンストラクタ関数がありますが、DisplayObject クラスにはコンストラクタ関数はありません。

前に説明したように、ステージは表示オブジェクトコンテナです。

コア表示クラス

ActionScript 3.0 の flash.display パッケージには、Flash Player に表示できるビジュアルオブジェクトのクラスが含まれています。次の図は、このコア表示オブジェクトクラスのサブクラスの関係を示します。



この図は、表示オブジェクトクラスのクラス継承を示します。StaticText、TextField、および Video などの一部のクラスは、flash.display パッケージに含まれていませんが、DisplayObject クラスを継承します。

DisplayObject クラスを拡張するすべてのクラスはそのメソッドとプロパティを継承します。詳細については、167 ページの「DisplayObject クラスのプロパティとメソッド」を参照してください。

flash.display パッケージに含まれる次のクラスのオブジェクトをインスタンス化することができます。

- Bitmap - Bitmap クラスを使用して、外部ファイルからロードするか、ActionScript でレンダリングするビットマップオブジェクトを定義します。Loader クラスを使用して外部ファイルからビットマップをロードできます。ロードできるのは、GIF、JPG、PNG ファイルです。カスタムデータを使用して BitmapData オブジェクトを作成した後、そのデータを使用する Bitmap オブジェクトを作成することもできます。ビットマップがロードされたか ActionScript で作成されたかに関係なく、BitmapData クラスのメソッドを使用してビットマップを変更できます。詳細については、179 ページの「Loader クラス」、および184 ページの「ビットマップの作成と操作」を参照してください。
- Loader - Loader クラスを使用して外部アセット (SWF ファイルまたはグラフィック) をロードします。詳細については、179 ページの「コンテンツの動的ロード」を参照してください。
- Shape - Shape クラスを使用して、矩形、線、円などのベクターグラフィックを作成します。詳細については、176 ページの「ベクターグラフィックの描画」を参照してください。
- SimpleButton - SimpleButton オブジェクトには、アップ、ダウン、オーバーの 3 つのボタンの状態があります。詳細については、184 ページの「SimpleButton オブジェクトの操作」を参照してください。

- Sprite - Sprite オブジェクトには、独自のグラフィックおよび子表示オブジェクトを含めることができます。Sprite クラスは DisplayObjectContainer クラスを拡張します。詳細については、168 ページの「表示オブジェクトコンテナの操作」、および 176 ページの「ベクターグラフィックの描画」を参照してください。
- MovieClip - MovieClip オブジェクトは Sprite オブジェクトによく似ていますが、タイムラインもある点が異なります。詳細については、183 ページの「ActionScript 3.0 のムービークリップの制御」を参照してください。

flash.display パッケージ内にない次のクラスは、DisplayObject クラスのサブクラスです。

- flash.text パッケージ内に含まれる TextField クラス - TextField オブジェクトは、テキストの表示と入力用の表示オブジェクトです。詳細については、178 ページの「テキストの操作」を参照してください。
- flash.media パッケージ内に含まれる Video クラス - 186 ページの「ビデオの操作」を参照してください。

flash.display パッケージ内に含まれる次のクラスのインスタンスをインスタンス化することはできませんが、次のクラスは DisplayObject クラスを拡張します。

- AVM1Movie - AVM1Movie クラスを使用して、ActionScript 1.0 または 2.0 で作成されたロードされた SWF ファイルを表します。
- DisplayObjectContainer - Loader、Stage、Sprite、および MovieClip クラスは DisplayObjectContainer クラスを拡張します。詳細については、168 ページの「表示オブジェクトコンテナの操作」を参照してください。
- InteractiveObject - InteractiveObject は、マウスとキーボードで操作するために使用されるすべてのオブジェクトの基本クラスです。SimpleButton、TextField、Video、Loader、Sprite、Stage、および MovieClip オブジェクトはすべて InteractiveObject クラスのサブクラスです。
- MorphShape - シェイプトゥイーンを適用したときに作成されるオブジェクトです。このオブジェクトは ActionScript を使用してインスタンス化することはできません。MorphShape を作成できるのは、Flash オーサリングツールだけです。
- Stage - Stage クラスは DisplayObjectContainer クラスを拡張します。アプリケーションには Stage インスタンスが 1 つあります。Stage インスタンスは表示リスト階層の最上位にあります。詳細については、173 ページの「Stage プロパティの設定」を参照してください。

また、flash.text パッケージ内にある StaticText クラスは DisplayObject クラスを拡張しますが、DisplayObject クラスのインスタンスをインスタンス化することはできません。静止テキストフィールドを作成できるのは Flash オーサリングツールだけです。

ActionScript 3.0 の表示リストを使用するアプローチの利点

ActionScript 3.0 では、各種の表示オブジェクトに対して別個のクラスがあります。ActionScript 1.0 および 2.0 では、同じタイプの多数のオブジェクトが、すべて単一のクラス、つまり MovieClip クラスに含まれます。

こうしたクラスの個別化および表示リストの階層構造には、次のような利点があります。

- レンダリングの効率化とメモリ使用量の削減
- 深度管理の向上
- 表示リスト内の自由な移動
- リスト外表示オブジェクト
- 表示オブジェクトの簡単なサブクラス化

上記の利点については、次のセクションで説明します。

レンダリングの効率化とファイルサイズの削減

ActionScript 1.0 および 2.0 では、MovieClip オブジェクトでのみシェイプを描画できました。ActionScript 3.0 には、シェイプを描画できる簡単な表示オブジェクトクラスがあります。ActionScript 3.0 表示オブジェクトクラスには、MovieClip オブジェクトに含まれるメソッドとプロパティがすべて含まれないため、メモリおよびプロセッサリソースへの負担が軽減されます。

たとえば、各 MovieClip オブジェクトにはムービークリップのタイムラインのプロパティが含まれますが、Shape オブジェクトには含まれません。タイムラインを制御するためのプロパティは、大量のメモリおよびプロセッサリソースを使用する場合があります。ActionScript 3.0 では、Shape オブジェクトを使用することでパフォーマンスが向上します。Shape オブジェクトは、複雑な MovieClip オブジェクトよりオーバーヘッドが少なくなります。Flash Player では、使用されていない MovieClip プロパティを管理する必要はないため、スピードが向上し、オブジェクトが使用するメモリフットプリントが削減されます。

深度管理の向上

ActionScript 1.0 および 2.0 では、深度は直線的な深度管理スキームおよび `getNextHighestDepth()` などのメソッドを使用して管理されていました。

ActionScript 3.0 には、表示オブジェクトの深度を管理するために便利なメソッドおよびプロパティが含まれる DisplayObjectContainer クラスがあります。

ActionScript 3.0 では、表示オブジェクトを DisplayObjectContainer インスタンスの子リスト内の新しい位置に移動すると、表示オブジェクトコンテナ内の他の子は、自動的に位置が変更され、表示オブジェクトコンテナ内の適切な子インデックス位置が割り当てられます。

ActionScript 3.0 では、常に表示オブジェクトコンテナの子オブジェクトをすべて見つけることができます。どの DisplayObjectContainer インスタンスにも、表示オブジェクトコンテナの子の数を一覧表示する `numChildren` プロパティがあります。また、表示オブジェクトコンテナの子リストは常にインデックスリストであるため、リスト内のインデックス位置 0 から最後のインデックス位置 (`numChildren - 1`) までのすべてのオブジェクトを調べることができます。これは、ActionScript 1.0 および 2.0 の MovieClip オブジェクトのメソッドとプロパティでは不可能でした。

ActionScript 3.0 では、表示リスト内を順に移動するのは簡単です。表示オブジェクトコンテナの子リストのインデックス番号に欠落番号がないためです。表示リスト内の移動とオブジェクトの深度管理は、ActionScript 1.0 および 2.0 より簡単になりました。ActionScript 1.0 および 2.0 では、ムービークリップに、深度順に不連続な欠落があるオブジェクトが含まれていたため、オブジェクトのリスト内を移動するのが困難でした。ActionScript 3.0 では、表示オブジェクトコンテナのそれぞれの子リストが 1 つの配列として内部にキャッシュされるので、インデックスによるルックアップが非常に高速に実行されます。また、表示オブジェクトコンテナのすべての子を非常に高速にループすることもできます。

ActionScript 3.0 では、DisplayObjectContainer クラスの `getChildByName()` メソッドを使用して表示オブジェクトコンテナの子にアクセスすることもできます。

表示リスト内の自由な移動

ActionScript 1.0 および 2.0 では、ベクターシェイプなど、Flash オーサリングツールで描画されたオブジェクトにはアクセスできませんでした。ActionScript 3.0 では、表示リスト上のすべてのオブジェクト、つまり ActionScript を使用して作成されたオブジェクトおよび Flash オーサリングツールで作成されたすべての表示オブジェクトにアクセスできます。詳細については、171 ページの「表示リスト内の移動」を参照してください。

リスト外表示オブジェクト

ActionScript 3.0 では、表示可能な表示リストにない表示オブジェクトを作成することができます。これらのオブジェクトを " リスト外 " 表示オブジェクトと呼びます。表示オブジェクトは、既に表示リストに追加されている DisplayObjectContainer インスタンスの `addChild()` または `addChildAt()` メソッドを呼び出したときにのみ、表示可能な表示リストに追加されます。

リスト外表示オブジェクトを使用すると、複数の表示オブジェクトが含まれる複数の表示オブジェクトコンテナを持つ表示オブジェクトなど、複雑な表示オブジェクトを構成できます。表示オブジェクトをリスト外にしておくことで、これらの表示オブジェクトをレンダリングする処理時間をかけずに、複雑なオブジェクトを構成できます。これにより、必要に応じてリスト外表示オブジェクトを表示リストに追加できます。また、表示オブジェクトコンテナの子を表示リスト内および外に、または表示リスト内の必要な場所に任意に移動できます。

表示オブジェクトの簡単なサブクラス化

ActionScript 1.0 および 2.0 では、通常は SWF ファイルに新しい MovieClip オブジェクトを追加して、基本シェイプを作成するか、またはビットマップを表示します。ActionScript 3.0 では、DisplayObject クラスに Shape、Bitmap などの多くのビルトインサブクラスが含まれています。ActionScript 3.0 のクラスは特定のタイプのオブジェクト用に特化されているので、ビルトインクラスの基本サブクラスを簡単に作成できます。

たとえば、ActionScript 2.0 で円を描画するには、カスタムクラスのオブジェクトをインスタンス化したときに MovieClip クラスを拡張する CustomCircle クラスを作成できました。しかし、このクラスには、クラスに適用されない MovieClip クラスの多数のプロパティやメソッド (`totalFrames` など) も含まれてしまいます。ActionScript 3.0 では、Shape オブジェクトを拡張する CustomCircle クラスを作成できるため、MovieClip クラスに含まれている関係のないプロパティやメソッドが含まれることはありません。次のコードは、CustomCircle クラスの例です。

```
import flash.display.*;

private class CustomCircle extends Shape
{
    var xPos:Number;
    var yPos:Number;
    var radius:Number;
    var color:uint;
    public function CustomCircle(xInput:Number,
                                yInput:Number,
                                rInput:Number,
                                colorInput:uint)
    {
        xPos = xInput;
        yPos = yInput;
        radius = rInput;
        color = colorInput;
        this.graphics.beginFill(color);
        this.graphics.drawCircle(xPos, yPos, radius);
    }
}
```

表示オブジェクトの操作

ステージ、表示オブジェクト、表示オブジェクトコンテナ、および表示リストの基本概念を理解できたので、このセクションでは、ActionScript 3.0 での表示オブジェクトの操作について詳しく説明します。

DisplayObject クラスのプロパティとメソッド

表示オブジェクトはすべて DisplayObject クラスのサブクラスです。したがって、表示オブジェクトは DisplayObject クラスのプロパティおよびメソッドを継承します。継承したプロパティは、すべての表示オブジェクトに適用される基本プロパティです。たとえば、表示オブジェクトには、表示オブジェクトコンテナ内のオブジェクトの位置を指定する x プロパティと y プロパティがあります。

DisplayObject クラスにはコンストラクタ関数はありません。Sprite などの別のタイプのオブジェクト (DisplayObject 型のサブクラスであるオブジェクト) を作成し、new コンストラクタでオブジェクトをインスタンス化する必要があります。また、カスタム表示オブジェクトクラスを作成する場合は、たとえば、Shape クラスや Sprite クラスなど、使用可能なコンストラクタ関数を持ついずれかの表示オブジェクトサブクラスのサブクラスを作成する必要があります。

表示リストへの表示オブジェクトの追加

表示オブジェクトをインスタンス化すると、表示リスト上にある表示オブジェクトコンテナに表示オブジェクトインスタンスを追加するまで、表示オブジェクトは画面上 (ステージ上) に表示されません。たとえば、次のコードでは、コードの最後の行を省略すると myText TextField オブジェクトは表示されません。コードの最後の行で、this キーワードは、既に表示リストに追加されている表示オブジェクトコンテナを参照する必要があります。

```
import flash.display.*;
import flash.text.TextField;
var myText:TextField = new TextField();
myText.text = "Buenos dias.";
this.addChild(myText);
```

ステージにビジュアルエレメントを追加すると、このエレメントは Stage オブジェクトの " 子 " になります。アプリケーション内にロードされた最初の SWF ファイル (HTML ページに埋め込む SWF ファイルなど) は、自動的に Stage オブジェクトの子として追加されます。これは、Sprite クラスを拡張する型のオブジェクトです。

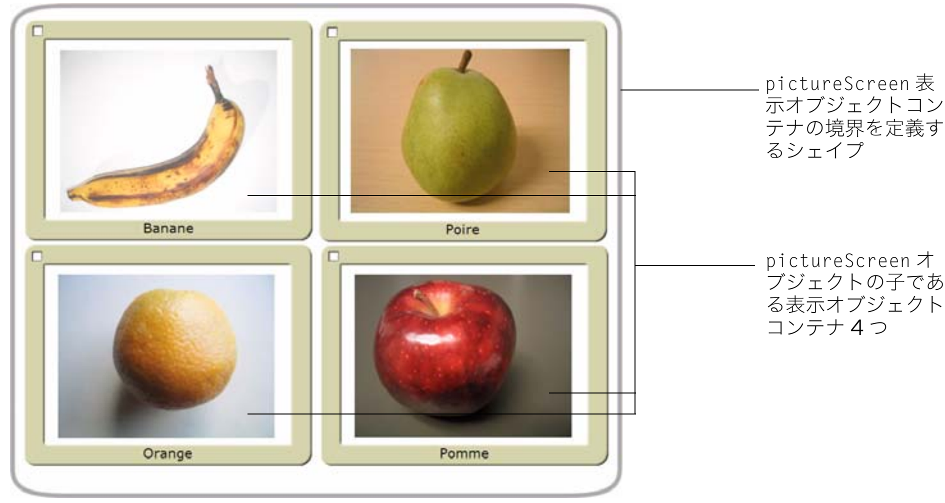
Adobe Flex Builder 2 で MXML タグを追加したり、Flash で描画ツールを使用したりするなど、ActionScript を使用しないで作成された表示オブジェクトは、表示リストに追加されます。これらの表示オブジェクトは ActionScript を使用して追加されていませんが、ActionScript からアクセスできます。たとえば、次のコードは、ActionScript ではなくオーサリングツールで追加された button1 というオブジェクトの幅を調整します。

```
button1.width = 200;
```

表示オブジェクトコンテナの操作

DisplayObjectContainer オブジェクトが表示リストから削除された場合、またはその他の方法で移動されたか変形された場合、DisplayObjectContainer 内の各表示オブジェクトも削除、移動、または変形されます。

表示オブジェクトコンテナは一種の表示オブジェクトで、別の表示オブジェクトコンテナに追加できます。たとえば、次の図は、アウトラインシェイプ 1 つと (PictureFrame 型の) 表示オブジェクトコンテナ 4 つを含む表示オブジェクトコンテナ pictureScreen を示します。



表示リストに表示オブジェクトを表示するには、表示リスト上にある表示オブジェクトコンテナに表示オブジェクトを追加する必要があります。これには、コンテナオブジェクトの addChild() メソッドまたは addChildAt() メソッドを使用します。たとえば、次のコードの最後の行がなければ、myTextField オブジェクトは表示されません。

```
var myTextField:TextField = new TextField();
```

```
myTextField.text = "hello";
this.root.addChild(myTextField);
```

このサンプルコードでは、this.root にこのコードを含む MovieClip 表示オブジェクトコンテナを指定します。実際のコードでは、別のコンテナを指定できます。

表示オブジェクトコンテナの子リスト内の特定の位置に子を追加するには、addChildAt() メソッドを使用します。子リスト内の 0 から始まるインデックス位置は、表示オブジェクトのレイヤー (前から後ろへの順) に対応しています。たとえば、次のような 3 つの表示オブジェクトがあるとします。各オブジェクトは、Ball と呼ばれるカスタムクラスから作成されました。

コンテナ内のこれらの表示オブジェクトのレイヤーは、addChildAt() メソッドを使用して調整できます。たとえば、次のようなコードがあるとします。

```
ball_A = new Ball(0xFFCC00, "a");
ball_A.name = "ball_A";
ball_A.x = 20;
ball_A.y = 20;
container.addChild(ball_A);

ball_B = new Ball(0xFFCC00, "b");
ball_B.name = "ball_B";
ball_B.x = 70;
ball_B.y = 20;
container.addChild(ball_B);

ball_C = new Ball(0xFFCC00, "c");
ball_C.name = "ball_C";
ball_C.x = 40;
ball_C.y = 60;
container.addChildAt(ball_C, 1);
```

このコードを実行した後、表示オブジェクトは次のように container DisplayObjectContainer オブジェクト内に配置されます。オブジェクトのレイヤーに注意してください。

表示オブジェクトのレイヤーの順序を確認するには、getChildAt() メソッドを使用します。getChildAt() メソッドは、メソッドに渡すインデックス番号に基づいてコンテナの子オブジェクトを返します。たとえば、次のコードは、container DisplayObjectContainer オブジェクトの子リスト内の異なる位置にある表示オブジェクトの名前を表示します。

```
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_C
trace(container.getChildAt(2).name); // ball_B
```

親コンテナの子リストから表示オブジェクトを削除すると、リスト上でそれより上位にあるエレメントの子インデックス内の位置がそれぞれ下がります。たとえば、引き続き前のコードを使用すると、次のコードは、子リスト内の下位にある表示オブジェクトが削除された場合、container DisplayObjectContainer 内の位置 2 にあった表示オブジェクトが位置 1 に移動することを示します。

```
container.removeChild(ball_C);
trace(container.getChildAt(0).name); // ball_A
trace(container.getChildAt(1).name); // ball_B
```

removeChild() および removeChildAt() メソッドは、表示オブジェクトインスタンスを完全に削除しません。コンテナの子リストから削除するだけです。インスタンスは、別の変数で参照できます オブジェクトを完全に削除するには、delete 演算子を使用します。

表示オブジェクトには親コンテナが 1 つだけあるので、その 1 つだけの表示オブジェクトコンテナに表示オブジェクトのインスタンスを追加できます。たとえば、次のコードは、表示オブジェクト tf1 が 1 つだけのコンテナ (この例では、DisplayObjectContainer クラスを拡張する Sprite) 内にあることを示します。

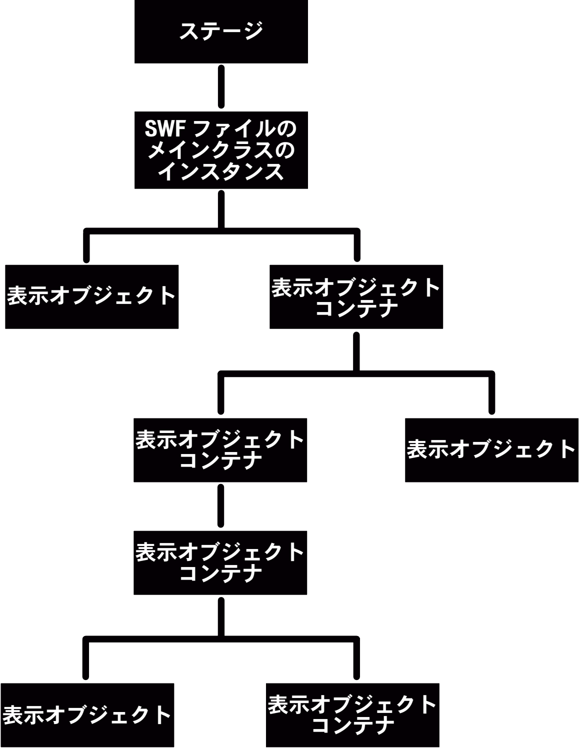
```
tf1:TextField = new TextField();
tf2:TextField = new TextField();
tf1.name = "text 1";
tf2.name = "text 2";

container1:Sprite = new Sprite();
container2:Sprite = new Sprite();

container1.addChild(tf1);
container1.addChild(tf2);
container2.addChild(tf1);

trace(container1.numChildren); // 1
trace(container1.getChildAt(0).name); // text 2
trace(container2.numChildren); // 1
trace(container2.getChildAt(0).name); // text 1
```

表示オブジェクトコンテナ内に含まれる表示オブジェクトを別の表示オブジェクトコンテナに追加すると、表示オブジェクトは最初の表示オブジェクトコンテナの子リストから削除されます。

表示リスト外にある表示オブジェクト、つまり Stage オブジェクトの子である表示オブジェクトコンテナ内に含まれていない表示オブジェクトは、" リスト外 " 表示オブジェクトといいます。

表示リスト内の移動

既に見たように、表示リストはツリー構造です。ツリーの一番上には、複数の表示オブジェクトを含めることができるステージがあります。これらの表示オブジェクトは、それ自体が表示オブジェクトコンテナであり、別の表示オブジェクト、つまり表示オブジェクトコンテナを含めることができます。



DisplayObjectContainer クラスには、表示オブジェクトコンテナの子リストを使用して表示リスト内を移動するためのプロパティとメソッドが含まれています。たとえば、次のようなコードがあるとします。このコードは、2 つの表示オブジェクト `title` と `pict` を `container` オブジェクト (Sprite、Sprite クラスは DisplayObjectContainer クラスを拡張します) に追加します。

```
var container:Sprite = new Sprite();
var title:TextField = new TextField();
title.text = "Hello";
var pict:Loader = new Loader();
```

```
var url:URLRequest = new URLRequest("banana.jpg");
pict.load(url);
pict.name = "banana loader";
container.addChild(title);
container.addChild(pict);
```

getChildAt() メソッドは、特定のインデックス位置にある表示リストの子を返します。

```
trace(container.getChildAt(0) is TextField); // true
```

子オブジェクトに名前でアクセスすることもできます。各表示オブジェクトには name プロパティがあります。name プロパティを割り当てなかった場合は、Flash Player では "instance1" などのデフォルト値が割り当てられます。たとえば、次のコードは、getChildByName() メソッドを使用して "banana loader" という名前で子表示オブジェクトにアクセスする方法を示します。

```
trace(container.getChildByName("banana loader") is Loader); // true
```

getChildByName() メソッドを使用すると、getChildAt() メソッドを使用する場合よりパフォーマンスが低下します。

表示オブジェクトコンテナには表示リストの子オブジェクトとして別の表示オブジェクトコンテナを含めることができるため、ツリーのようにアプリケーションの表示リスト内を移動できます。たとえば、前述のコードの抜粋では、pict Loader オブジェクトのロード操作が完了すると、pict オブジェクトにビットマップである子表示オブジェクトがロードされます。このビットマップ表示オブジェクトにアクセスするために、pict.getChildAt(0) を記述することができます。また、container.getChildAt(0) == pict であるため、container.getChildAt(0).getChildAt(0) を記述することもできます。

次の関数は、表示オブジェクトコンテナから表示リストのインデントされた trace() を出力します。

```
function traceDisplayList(container:DisplayObjectContainer,
                            indentString:String = ""):void
{
  var child:DisplayObject;
  for (var i:uint=0; i < container.numChildren; i++)
  {
    child = container.getChildAt(i);
    trace (indentString, child, child.name);
    if (container.getChildAt(i) is DisplayObjectContainer)
    {
      traceDisplayList(DisplayObjectContainer(child), indentString + "")
    }
  }
}
```

Flex を使用している場合、Flex は多くのコンポーネント表示オブジェクトクラスを定義し、これらのクラスは DisplayObjectContainer クラスの表示リストアクセスメソッドをオーバーライドすることを知っておく必要があります。たとえば、mx.core パッケージの Container クラスは、(Container クラスによって拡張される) DisplayObjectContainer クラスの addChild() メソッドおよびその他のメソッドをオーバーライドします。addChild() メソッドの場合、Container クラスは、Flex で Container インスタンスにすべてのタイプの表示オブジェクトを追加できないように、このメソッドをオーバーライドします。この場合、オーバーライドされたメソッドでは、追加する子オブジェクトが mx.core.UIComponent オブジェクトの一種である必要があります。

Stage プロパティの設定

Stage クラスは、DisplayObject クラスのほとんどのプロパティとメソッドをオーバーライドします。オーバーライドされたプロパティまたはメソッドのいずれかを呼び出した場合、Flash Player は例外をスローします。たとえば、Stage オブジェクトは、アプリケーションのメインコンテナとして位置が固定されているため、x プロパティや y プロパティを持ちません。x プロパティおよび y プロパティは、コンテナを基準として表示オブジェクトの位置を参照します。Stage オブジェクトは別の表示オブジェクトコンテナに含まれていないため、この 2 つのプロパティは適用されません。

メモ	Stage クラスの一部のプロパティおよびメソッドは、最初にロードされた SWF ファイルと同じセキュリティサンドボックス内にないオブジェクトを表示するために使用することはできません。詳細については、461 ページの「セキュリティサンドボックス」を参照してください。

再生フレームレートの制御

Stage クラスの framerate プロパティは、アプリケーションにロードされたすべての SWF ファイルのフレームレートを設定するために使用します。詳細については、『ActionScript 3.0 リファレンスガイド』を参照してください。

フルスクリーンモードの操作

フルスクリーンモードでは、境界線、メニューバーなどを表示せずに、ビューアのモニタ全体に SWF を表示できます。Stage クラスの displayState プロパティは、SWF のフルスクリーンモードのオンとオフの切り替えに使用されます。displayState プロパティは、flash.display.StageDisplayState クラス内の定数で定義された値のいずれかに設定できます。フルスクリーンモードをオンにするには、displayState を StageDisplayState.FULL_SCREEN に設定します。

```
// mySprite は表示リストに既に追加されている Sprite インスタンスです。
mySprite.stage.displayState = StageDisplayState.FULL_SCREEN;
```

フルスクリーンモードを終了するには、displayState プロパティを StageDisplayState.NORMAL に設定します。

```
mySprite.stage.displayState = StageDisplayState.NORMAL;
```

フルスクリーンモードは別のウィンドウにフォーカスを切り替える、またはキーの組み合わせのいずれか (Esc キー (すべてのプラットフォーム)、Ctrl + W キー (Windows)、Cmd + w キー (Mac)、または Alt + F4 (Windows)) を押しても終了できます。

フルスクリーンモードのステージの拡大・縮小の動作は、通常モードのものと同じです。
拡大・縮小は、Stage クラスの scaleMode プロパティで制御されます。scaleMode プロパティが StageScaleMode.NO_SCALE に設定されると、ステージの stageWidth と stageHeight プロパティは SWF によって専有される画面のサイズに合わせて変更されます (この場合は画面全体)。

Stage クラスの fullScreen イベントを使用して、フルスクリーンモードのオン・オフを検出して、対応することができます。たとえば、この例のように、フルスクリーンモードにする、またはフルスクリーンモードを終了する際にアイテムの位置を変更する、画面にアイテムを追加する、または画面からアイテムを削除することができます。

```
import flash.events.FullScreenEvent;

function fullScreenRedraw(event:FullScreenEvent):void
{
  if (event.fullScreen)
  {
    // 入力テキストフィールドを削除する
    // フルスクリーンモードを閉じるボタンを追加する
  }
  else
  {
    // 入力テキストフィールドを再追加する
    // フルスクリーンモードを閉じるボタンを削除する
  }
}

mySprite.stage.addEventListener(FullScreenEvent.FULL_SCREEN, fullScreenRedraw);
```

このコードに示されているように、fullScreen イベントのイベントオブジェクトは、flash.events.FullScreenEvent クラスのインスタンスであり、フルスクリーンモードが有効 (true) であるか無効 (false) であるかを示す fullScreen プロパティが含まれています。

ActionScript のフルスクリーンモードを操作する際には、以下の注意事項を考慮してください。

- フルスクリーンモードは、マウスクリック (右クリックを含む) またはキー押下げに対する応答として ActionScript を介してのみ開始できます。
- ユーザーが複数のモニタを使用している場合、SWF のコンテンツは 1 台のモニタ画面でのみ拡大されます。Flash Player ではメトリックを使用して、SWF の領域を最も多く含むモニタが判断され、そのモニタがフルスクリーンモードで使用されます。

- HTML ページに組み込まれた SWF ファイルの場合、Flash Player に組み込む HTML コードには、次のように <param> タグと <embed> 属性 (名前 allowFullScreen、値 true) が含まれている必要があります。

```
< オブジェクト >
    ...
    <param name="allowFullScreen" value="true" />
    <embed ... allowfullscreen="true" />
</object>
```

  SWF 組み込みタグでを生成するために Web ページで JavaScript を使用する場合、allowFullScreen パラメータタグ/属性を追加するために JavaScript を変更する必要があります。たとえば、HTML ページで AC_FL_RunContent() 関数 (Flex Builder 生成 HTML ページと Flash 生成 HTML ページで使用される関数) を使用する場合、次のように、この関数に allowFullScreen パラメータを追加する必要があります。

```
AC_FL_RunContent(
    ...
    'allowFullScreen','true',
    ...
    ); //end AC code
```

  これはスタンドアロン Flash Player で実行されている SWF ファイルには当てはまりません。

- キーボードイベントや TextFields 内のテキストエントリなどの、すべてのキーボード関連 ActionScript は無効になります。フルスクリーンモードを閉じるキーボードショートカットは例外です。

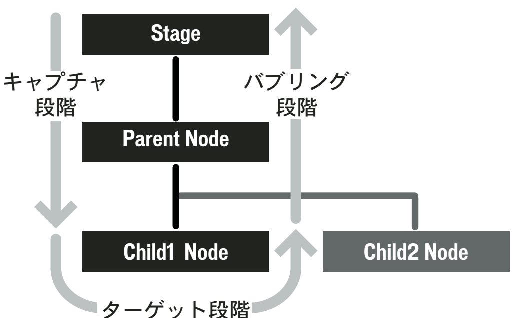
この他にも、理解する必要があるセキュリティ関連の制限がいくつかあります。これらについては、465 ページの「フルスクリーンモードのセキュリティ」で説明します。

表示オブジェクトのイベント処理

DisplayObject クラスは EventDispatcher クラスを継承します。つまり、どの表示オブジェクトでもイベントモデルに含めることができます (345 ページ、第 13 章の「イベントの処理」参照)。表示オブジェクトは、EventDispatcher クラスから継承された独自の addEventListener() メソッドを使用して、特定のイベントを受け取ることができます。ただし、これは、リスナーオブジェクトがそのイベントのイベントフローの一部である場合のみです。

Flash Player によってイベントオブジェクトが送出されると、そのイベントオブジェクトはステージからイベントが発生した表示オブジェクトへと往復します。たとえば、ユーザーが child1 という名前の表示オブジェクトをクリックすると、Flash Player により、ステージから表示リスト階層を下がって child1 表示オブジェクトまでイベントオブジェクトが送出されます。

次の図に示されているように、概念上イベントフローは 3 段階に分けられます。



詳細については、「345 ページ、第 13 章の「イベントの処理」」を参照してください。

コア表示クラスを操作するための基礎

以降のセクションでは、ActionScript 3.0 でコア表示クラスを操作するための基礎について説明します。以降のセクションでは、各トピックの概要について説明します。詳細については、『ActionScript 3.0 リファレンスガイド』の該当するクラスを説明するセクションを参照してください。

ベクターグラフィックの描画

それぞれの Shape、Sprite、および MovieClip オブジェクトは、graphics プロパティを備えています。各オブジェクトの graphics プロパティは Graphics オブジェクトで、Graphics クラスには、線、塗り、シェイプを描画し操作するためのプロパティとメソッドが含まれます。

たとえば、次のコードは Shape オブジェクトでオレンジの円を描画します。

```
import flash.display.*;
var circle:Shape = new Shape()
var xPos:Number = 100;
var yPos:Number = 100;
var radius:Number = 50;
circle.graphics.beginFill(0xFF8800);
circle.graphics.drawCircle(xPos, yPos, radius);
this.addChild(circle);
```

Graphics クラスには、単純なシェイプを簡単に描画するための、drawCircle()、drawEllipse()、drawRect()、drawRoundRect()、および drawRoundRectComplex() メソッドが含まれます。描画メソッドを呼び出す前に、linestyle()、lineGradientStyle()、beginFill()、beginGradientFill()、または beginBitmapFill() メソッドを呼び出して、線スタイル、塗り、またはその両方を定義します。

(他の表示オブジェクトを含む) 表示オブジェクトコンテナでもありながら、タイムラインを必要としないグラフィカルオブジェクトを作成する場合は、Sprite クラスを使用します。

たとえば、次の Sprite オブジェクトには、graphics プロパティで描画した円があり、子リスト内に TextField オブジェクトがあります。

```
var mySprite:Sprite = new Sprite();
mySprite.graphics.beginFill(0xFFCC00);
mySprite.graphics.drawCircle(30, 30, 30);
var label:TextField = new TextField();
label.text = "hello";
label.x = 20;
label.y = 20;
mySprite.addChild(label);
this.addChild(mySprite);
```

Sprite または MovieClip オブジェクトのグラフィックレイヤーは、常に Sprite または MovieClip の子表示オブジェクトの背後に表示されます。また、グラフィックレイヤーは、Sprite または MovieClip の子リストには表示されません。

テキストの操作

flash.text パッケージ内にある TextField クラスを使用すると、ダイナミックテキストフィールドおよびテキスト入力フィールドを操作できます。flash.text パッケージには、StaticText クラスもあります。ただし、StaticText オブジェクトは Flash オーサリングツールで作成されているため、ActionScript でインスタンス化することはできません。

次の例に示すように、TextFormat オブジェクトを使用して、TextField 全体または任意の範囲のテキストのフォーマットを設定できます。

```
var tf:TextField = new TextField();
tf.text = "Hello Hello";

var format1:TextFormat = new TextFormat();
format1.color = 0xFF0000;

var format2:TextFormat = new TextFormat();
format2.font = "Courier";

tf.setTextFormat(format1);
var startRange:uint = 6;
tf.setTextFormat(format2, startRange);

addChild(tf);
```

Text フィールドには、プレーンテキストまたは HTML 形式のテキストのいずれかを含めることができます。プレーンテキストはインスタンスの `text` プロパティに格納され、HTML テキストは `htmlText` プロパティに格納されます。

StyleSheet オブジェクトを使用して CSS スタイルシートを HTML テキストに適用できます。次に例を示します。

```
var style:StyleSheet = new StyleSheet();

var styleObj:Object = new Object();
styleObj.fontSize = "bold";
styleObj.color = "#FF0000";
style.setStyle(".darkRed", styleObj);
delete styleObj;

var tf:TextField = new TextField();
tf.styleSheet = style;
tf.htmlText = "<span class = 'darkRed'>Red</span> apple";

addChild(tf);
```

詳細については、『ActionScript 3.0 リファレンスガイド』の TextField クラスの説明を参照してください。

コンテンツの動的ロード

次の外部表示アセットを ActionScript 3.0 アプリケーションにロードできます。

- ActionScript 3.0 で作成された SWF ファイル - このファイルは、Sprite、MovieClip、または Sprite を拡張するクラスです。
- イメージファイル - これには、JPG、PNG、および GIF ファイルが含まれます。
- AVM1 SWF ファイル - これは、ActionScript 1.0 または 2.0 で記述された SWF ファイルです。

これらのアセットをロードするには、Loader クラスを使用します。

Loader クラス

Loader オブジェクトを使用して、アプリケーションに SWF ファイルおよびグラフィックファイルをロードします。Loader クラスは、DisplayObjectContainer クラスのサブクラスです。Loader オブジェクトには、表示リスト内の子表示オブジェクト、つまりロードされる SWF ファイルまたはグラフィックファイルを表す表示オブジェクトを 1 つだけ含むことができます。表示リストに Loader オブジェクトを追加するときは、次のコードのように、子表示オブジェクトのロードが完了したら、ロードされた子表示オブジェクトも表示リストに追加します。

```
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
this.addChild(pictLdr);
```

SWF ファイルまたはイメージがロードされると、ロードされた表示オブジェクトを、この例の `container` DisplayObjectContainer オブジェクトなどの別の表示オブジェクトコンテナに移動できます。

```
import flash.display.*;
import flash.net.URLRequest;
import flash.events.Event;
var container:Sprite = new Sprite();
addChild(container);
var pictLdr:Loader = new Loader();
var pictURL:String = "banana.jpg"
var pictURLReq:URLRequest = new URLRequest(pictURL);
pictLdr.load(pictURLReq);
pictLdr.contentLoaderInfo.addEventListener(Event.COMPLETE, imgLoaded);
function imgLoaded(e:Event):void
{
  container.addChild(pictLdr.content);
}
```

LoaderInfo クラス

ファイルがロードされると、LoaderInfo オブジェクトが作成されます。このオブジェクトは、Loader オブジェクトおよびロードされた表示オブジェクト両方のプロパティです。つまり、LoaderInfo オブジェクトは、Loader オブジェクトの contentLoaderInfo プロパティ経由の Loader オブジェクトのプロパティであり、表示オブジェクトの loaderInfo プロパティでロードされた表示オブジェクトのプロパティです。ロードされた表示オブジェクトの loaderInfo プロパティは、Loader オブジェクトの contentLoaderInfo プロパティと同じ LoaderInfo オブジェクトを参照します。つまり、LoaderInfo オブジェクトは、ロードされたオブジェクトとロードした Loader オブジェクト間 (ロードされる側とロードする側) で共有されます。

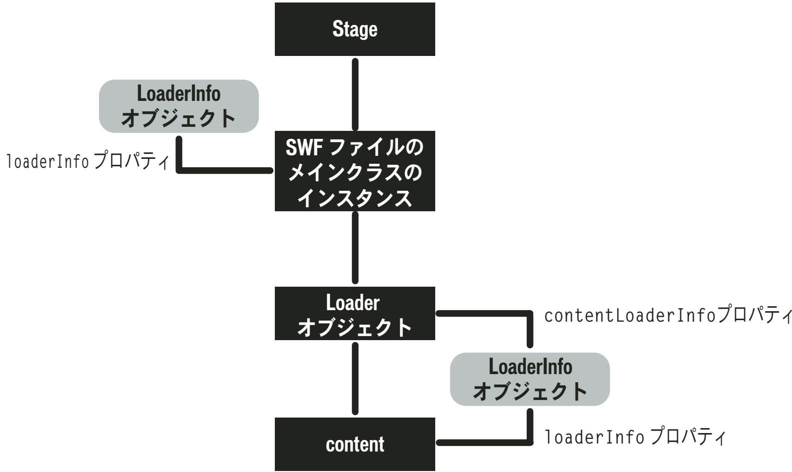
LoaderInfo クラスは、ロードの進行状況、ロードする側とロードされる側の URL、メディアの総バイト数、メディアの規格高さと幅などの情報を提供します。LoaderInfo オブジェクトは、ロードの進行状況を監視するためのイベントも送出します。

ロードされたコンテンツのプロパティにアクセスするには、次のコードのように、LoaderInfo オブジェクトにイベントリスナーを追加します。

```
import flash.display.Loader;
import flash.display.Sprite;
import flash.events.Event;

var ldr:Loader = new Loader();
var urlReq:URLRequest = new URLRequest("Circle.swf");
ldr.load(urlReq);
ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, loaded);
addChild(ldr);

function loaded(event:Event):void
{
  var content:Sprite = event.target.content;
  content.scaleX = 2;
}
```

詳細については、「345 ページ、第 13 章の「イベントの処理」」を参照してください。

次の図は、LoaderInfo オブジェクトのさまざまな使用方法を示しています。このオブジェクトは、SWF ファイルのメインクラスのインスタンスや Loader オブジェクトに使用されるほか、Loader オブジェクトによってロードされたオブジェクトにも使用されます。



LoaderContext クラス

Loader クラスの `load()` または `loadBytes()` メソッドを使用して Flash Player に外部ファイルをロードする場合、`context` パラメータを指定することもできます。このパラメータは LoaderContext オブジェクトです。

LoaderContext クラスには、ロードされたコンテンツの使用方法のコンテキストを定義できる次の 3 つのプロパティが含まれています。

- `checkPolicyFile`—SWF ファイルではなく、イメージファイルをロードする場合にのみ、このプロパティを使用します。このプロパティを `true` に設定すると、Loader によってオリジンサーバーでクロスドメインポリシーファイルの有無がチェックされます (456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」参照)。これは、Loader オブジェクトを含む SWF ファイルのドメイン以外のドメインにあるコンテンツにのみ必要です。サーバーから Loader ドメインにアクセス許可が与えられている場合、Loader ドメイン内の SWF ファイルの ActionScript はロードされたイメージ内のデータにアクセスできます。つまり、`BitmapData.draw()` コマンドを使用して、ロードされたイメージ内のデータにアクセスできます。

  Loader オブジェクトのドメイン以外の他のドメインにある SWF ファイルは、`Security.allowDomain()` を呼び出して特定のドメインを許可できます。

- securityDomain— イメージではなく、SWF ファイルをロードする場合にのみ、このプロパティを使用します。Loader オブジェクトを含むファイルのドメイン以外のドメインの SWF ファイルに対してこのプロパティを指定します。このオプションを指定すると、Flash Player はクロスドメインポリシーファイルが存在するかどうかをチェックします。ファイルが存在する場合は、クロスドメインポリシーファイルで許可されているドメインの SWF ファイルはロードされた SWF コンテンツをクロススクリプトできます。このパラメータとして flash.system.SecurityDomain.currentDomain を指定できます。
- applicationDomain—ActionScript 1.0 または 2.0 で記述されたイメージまたは SWF ファイルではなく、ActionScript 3.0 で記述された SWF ファイルをロードする場合にのみ、このプロパティを使用します。ファイルをロードするとき、applicationDomain パラメータを flash.system.ApplicationDomain.currentDomain に設定することで、ファイルが Loader オブジェクトと同じアプリケーションドメインに含まれるように指定できます。ロードされた SWF ファイルを同じアプリケーションドメインに置くと、そのクラスに直接アクセスできます。これは、関連付けられたクラス名を使用してアクセスできる埋め込みメディアを含む SWF ファイルをロードする場合に便利です。詳細については、436 ページの「ApplicationDomain クラス」を参照してください。

次に、別のドメインからビットマップをロードするときに、クロスドメインポリシーファイルの有無をチェックする例を示します。

```
var context:LoaderContext = new LoaderContext();
context.checkPolicyFile = true;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  photo11.jpg");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

次に、SWF ファイルを Loader オブジェクトと同じセキュリティサンドボックスに置くために、別のドメインから SWF ファイルをロードするときに、クロスドメインポリシーファイルの有無をチェックする例を示します。また、このコードは、ロードされた SWF ファイルのクラスを Loader オブジェクトと同じアプリケーションドメインに追加します。

```
var context:LoaderContext = new LoaderContext();
context.securityDomain = SecurityDomain.currentDomain;
context.applicationDomain = ApplicationDomain.currentDomain;
var urlReq:URLRequest = new URLRequest("http://www.[your_domain_here].com/
  library.swf");
var ldr:Loader = new Loader();
ldr.load(urlReq, context);
```

詳細については、『ActionScript 3.0 リファレンスガイド』の LoaderContext クラスの説明を参照してください。

ActionScript 3.0 のムービークリップの制御

MovieClip オブジェクトにはタイムラインがあり、再生ヘッドを制御し、フレームおよびシーンを操作するためのプロパティとメソッドも含まれます。

ActionScript 1.0 および 2.0 の MovieClip クラスのプロパティおよびメソッドの多くは、ActionScript 3.0 では MovieClip クラスのプロパティおよびメソッドとして提供されています。その一部はスーパークラスから継承されます。たとえば、DisplayObject クラスの x、y、および blendMode などのプロパティは、MovieClip クラスで有効です。これは、MovieClip クラスは DisplayObject クラスを拡張するためです。_x、_y、_root などのアンダースコア文字で始まる名前のプロパティは、ActionScript 3.0 ではアンダースコア文字のない名前に変更されました。また、_xmouse、_xscale、_ymouse、および _yscale プロパティの名前は、mouseX、scaleX、mouseY、および scaleY に変更されました。

MovieClip クラスには、ムービークリップを制御するためのメソッドとプロパティが含まれます。次のプロパティを使用すると、ムービークリップのフレームとシーンの総数、および現在の再生ヘッド位置にあるフレームとシーンに関する情報を監視できます。currentFrame、currentLabel、currentLabels、currentScene、scenes、および totalFrames。次のメソッドを使用すると、ムービークリップの再生ヘッドを制御できます。gotoAndPlay()、gotoAndStop()、nextFrame()、nextScene()、play()、prevFrame()、prevScene()、および stop()。

たとえば、次のコードは、myMovie MovieClip オブジェクトの現在のシーンの先頭フレームに再生ヘッドを移動します。

```
myMovie.gotoAndPlay(0);
```

上記を含むプロパティおよびメソッドの詳細については、『ActionScript 3.0 リファレンスガイド』の MovieClip のセクションを参照してください。

Stage オブジェクトの frameRate プロパティを使用して、アプリケーション内のすべてのムービークリップのフレームレートを設定できます。

ActionScript 1.0 および 2.0 とは異なり、ActionScript 3.0 では、ロードされた SWF ファイルは、MovieClip オブジェクト、AVM1Movie オブジェクト、または Sprite オブジェクトにすることができます。しかし、ActionScript 3.0 では MovieClip オブジェクトのタイムラインのみを制御できます。Sprite および AVM1Movie オブジェクトには、タイムライン関連のメソッドおよびプロパティは含まれません。

ビットマップの作成と操作

BitmapData クラスを使用すると、Bitmap オブジェクトのピクセルを操作できます。操作できるのは、ファイルからロードしたビットマップや BitmapData メソッドのいずれかだけを使用して描画したビットマップです。Bitmap オブジェクトには、BitmapData オブジェクトである `bitmapData` プロパティがあります。

BitmapData オブジェクトは、ピクセルの矩形配列を表します。新しいビットマップをプログラム処理によって作成できます。次の例に示すように、BitmapData コンストラクタを使用して、指定した色で矩形を作成でき、その BitmapData オブジェクトを新しい Bitmap オブジェクトに割り当てることができます。

```
var bdWidth:Number = 100;
var bdHeight:Number = 100;
var bdTransparent:Boolean = true;
var bdFillColorARGB:uint = 0xFF007090;
var myBitmapData:BitmapData = new BitmapData(bdWidth,
            bdHeight,
            bdTransparent,
            bdFillColorARGB);
var myBitmap:Bitmap = new Bitmap(myBitmapData);
addChild(myBitmap)
```

しかし、通常はロードされたイメージファイルのビットマップデータを操作します。いずれの場合も、BitmapData クラスには、BitmapData を操作し変更するためのメソッドが含まれます。たとえば、`setPixel()` メソッドを使用して、ピクセルを特定の RGB 値に設定できます。Bitmap および BitmapData クラスのプロパティとメソッドの詳細については、『ActionScript 3.0 リファレンスガイド』を参照してください。

複数の Bitmap オブジェクトの `bitmapData` が同じ BitmapData オブジェクトを参照でき、それぞれの Bitmap オブジェクトに異なるエフェクトや変形を適用できます。

SimpleButton オブジェクトの操作

ActionScript 3.0 では、SimpleButton クラスを使用してボタンの動作を定義できます。SimpleButton には、`upState`、`downState`、および `overState` の 3 つの状態があります。これらは SimpleButton オブジェクトのプロパティで、それぞれが DisplayObject オブジェクトです。たとえば、次のクラスは単純なテキストボタンを定義します。

```
import flash.display.*;
import flash.events.*;

public class TextButton extends SimpleButton
{
   public var selected:Boolean = false;
   public function TextButton(txt:String)
```

```
  {
    upState = new TextButtonState(0xFFFFFF, txt);
    downState = new TextButtonState(0xCCCCCC, txt);
    hitTestState = upState;
    overState = upState;
    addEventListener(MouseEvent.CLICK, buttonClicked);
  }
  public function buttonClicked(e:Event)
  {
    trace("Button clicked.");
  }
}
```

SimpleButton オブジェクトの `hitTestState` プロパティは、ボタンのマウスイベントに応答する DisplayObject インスタンスです。この例では、`hitTestState` プロパティ (および `overState` プロパティ) を `upState` プロパティと同じ DisplayObject インスタンスになるように設定します。

この例の TextButton クラスは、ボタンの状態 (アップ、ダウン、またはオーバー) に使用する DisplayObject を定義する次のクラスを参照します。

```
import flash.display.*;
import flash.text.TextFormat;
import flash.text.TextField;

class TextButtonState extends Sprite
{
  public var label:TextField;
  public function TextButtonState(color:uint, labelText:String)
  {
    label = new TextField();
    label.text = labelText;
    label.x = 2;
    var format:TextFormat = new TextFormat("Verdana");
    label.setTextFormat(format);
    var buttonWidth:Number = label.textWidth + 10;
    var background:Shape = new Shape();
    background.graphics.beginFill(color);
    background.graphics.lineStyle(1, 0x000000);
    background.graphics.drawRoundRect(0, 0, buttonWidth, 18, 4);
    addChild(background);
    addChild(label);
  }
}
```

ビデオの操作

Video クラスは、flash.display パッケージ内にありませんが、DisplayObject クラスのサブクラスです。ビデオを Video オブジェクトに関連付けるには、`attachNetStream()` メソッドまたは `attachCamera()` メソッドを使用する必要があります。

次に、ネットストリームをビデオに関連付けて、ビデオを Sprite 表示オブジェクトコンテナに追加する単純な例を示します。

```
import flash.display.Sprite;
import flash.net.*;
import flash.media.Video;

public class VideoTest extends Sprite
{
  private var videoUrl:String = "http://example.com/test.flv";

  public function VideoTest()
  {
    var connection:NetConnection = new NetConnection();
    connection.connect(null);

    var stream:NetStream = new NetStream(connection);

    var myVideo:Video = new Video(360, 240);
    myVideo.attachNetStream(stream);
    stream.play(videoUrl);

    addChild(myVideo);
  }
}
```

詳細については、『ActionScript 3.0 リファレンスガイド』の Video クラスの説明を参照してください。

例 : SpriteArranger

サンプルの SpriteArranger アプリケーションは、第 4 章で説明した図形の例 (148 ページの「例 : GeometricShapes」参照) の上に構築されます。

サンプルの SpriteArranger アプリケーションは、表示オブジェクトの操作に関する次のような概念を示しています。

- 表示オブジェクトクラスの拡張
- 表示リストへのオブジェクトの追加
- 表示オブジェクトのレイヤーと表示オブジェクトコンテナの操作
- 表示オブジェクトイベントへの応答
- 表示オブジェクトのプロパティとメソッドの使用

SpriteArranger アプリケーションのファイルは、"Examples/SpriteArranger" フォルダに入っています。アプリケーションは、次のファイルで構成されています。

ファイル	説明
SpriteArranger.mxml	MXML で記述された Flex 用メインアプリケーションファイル
com/example/programmingas3/SpriteArranger/CircleSprite.as	画面に円をレンダリングするタイプの Sprite オブジェクトを定義するクラス。
com/example/programmingas3/SpriteArranger/DrawingCanvas.as	キャンバスを定義するクラス。キャンバスは、GeometricSprite オブジェクトを格納する表示オブジェクトコンテナです。
com/example/programmingas3/SpriteArranger/SquareSprite.as	画面に矩形をレンダリングするタイプの Sprite オブジェクトを定義するクラス。
com/example/programmingas3/SpriteArranger/TriangleSprite.as	画面に三角形をレンダリングするタイプの Sprite オブジェクトを定義するクラス。
com/example/programmingas3/SpriteArranger/GeometricSprite.as	Sprite オブジェクトを拡張するクラス。画面上のシェイプを定義するために使用されます。CircleSprite、SquareSprite、および TriangleSprite は、それぞれこのクラスを拡張します。
com/example/programmingas3/geometricshapes/IGeometricShape.as	すべての図形クラスによって実装されるメソッドを定義する基本インターフェイス。
com/example/programmingas3/geometricshapes/IPolygon.as	複数の辺を持つ図形クラスによって実装されるメソッドを定義するインターフェイス。
com/example/programmingas3/geometricshapes/RegularPolygon.as	図形の中心をめぐって対称の位置に長さの等しい複数の辺を持つ図形のタイプ。
com/example/programmingas3/geometricshapes/Circle.as	円を定義する図形のタイプ。

ファイル	説明
com/example/programmingas3/geometricshapes/EquilateralTriangle.as	正三角形を定義する、RegularPolygon のサブクラス。
com/example/programmingas3/geometricshapes/Square.as	正四角形を定義する、RegularPolygon のサブクラス。
com/example/programmingas3/geometricshapes/GeometricShapeFactory.as	与えられたシェイプのタイプとサイズで、そのシェイプを作成する " ファクトリメソッド " を格納するクラス。

SpriteArranger クラスの定義

SpriteArranger アプリケーションを使用すると、さまざまな表示オブジェクトを画面上の " キャンバス " に追加できます。

DrawingCanvas クラスは、ユーザーが画面上のシェイプを追加できる描画領域、つまり、一種の表示オブジェクトコンテナを定義します。それらの画面上のシェイプは、GeometricSprite クラスのいずれかのサブクラスのインスタンスです。

DrawingCanvas クラス

Flex では、コンテナオブジェクトに追加されるすべての子表示オブジェクトは、mx.core.UIComponent クラスから生成されたクラスのオブジェクトである必要があります。このアプリケーションは、SpriteArranger.mxml ファイル内の MXML コードで定義されているように、DrawingCanvas クラスのインスタンスを mx.containers.VBox オブジェクトの子として追加します。この継承は、DrawingCanvas クラス宣言の中で次のように定義されています。

```
public class DrawingCanvas extends UIComponent
```

UIComponent クラスは、DisplayObject、DisplayObjectContainer、Sprite の各クラスを継承し、DrawingCanvas クラス内のコードは、それらのクラスのメソッドとプロパティを使用します。

```
public class DrawingCanvas extends Sprite
```

`DrawingCanvas()` コンストラクタメソッドは、Rectangle オブジェクトを `bounds` に設定します。これは、後でキャンバスのアウトラインを描画する際に使用されるプロパティです。その後、コンストラクタメソッドは次のようにして `initCanvas()` メソッドを呼び出します。

```
this.bounds = new Rectangle(0, 0, w, h);
initCanvas(fillColor, lineColor);
```

次の例が示すように、initCanvas() メソッドは DrawingCanvas オブジェクトの各種プロパティを定義します。これらのプロパティは、コンストラクタ関数へ引数として渡されたものです。

```
this.lineColor = lineColor;
this.fillColor = fillColor;
this.width = 500;
this.height = 200;
```

その後、initCanvas() メソッドは drawBounds() メソッドを呼び出し、このメソッドにより、DrawingCanvas クラスの graphics プロパティを使用してキャンバスが描画されます。graphics プロパティは、Shape クラスから継承されます。

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
this.graphics.drawRect(bounds.left - 1,
              bounds.top - 1,
              bounds.width + 2,
              bounds.height + 2);
this.graphics.endFill();
```

次に示す DrawingCanvas クラスの追加メソッドは、アプリケーションに対するユーザーの操作に基づいて呼び出されます。

- addShape() および describeChildren() メソッド。これらについては、191 ページの「キャンバスへの表示オブジェクトの追加」で説明します。
- moveToBack()、moveDown()、moveToFront()、および moveUp() メソッド。これらについては、194 ページの「表示オブジェクトのレイヤーの再配置」で説明します。
- onMouseUp() メソッド。これについては、192 ページの「表示オブジェクトのクリックとドラッグ」で説明します。

GeometricSprite クラスとそのサブクラス

ユーザーがキャンバスに追加できる表示オブジェクトは、GeometricSprite クラスの次のいずれかのサブクラスのインスタンスです。

- CircleSprite
- SquareSprite
- TriangleSprite

GeometricSprite クラスは、flash.display.Sprite クラスを拡張します。

```
public class GeometricSprite extends Sprite
```

GeometricSprite クラスは、すべての GeometricSprite オブジェクトに共通するいくつかのプロパティを含んでいます。それらは、関数に渡されたパラメータに基づいてコンストラクタ関数の中で設定されます。次に例を示します。

```
this.size = size;
this.lineColor = lColor;
this.fillColor = fColor;
```

GeometricSprite クラスの `geometricShape` プロパティは、IGeometricShape インターフェイスを定義し、このインターフェイスはシェイプの数学プロパティを定義しますが、ビジュアルプロパティは定義しません。IGeometricShape インターフェイスを実装するクラスは、第 4 章のアプリケーションの例で定義されています (148 ページの「例 : GeometricShapes」参照)。

GeometricSprite クラスは `drawShape()` メソッドを定義し、このメソッドは GeometricSprite の各サブクラス内のオーバーライド定義の中でさらに改良されます。詳細については、この後に示す「キャンバスへの表示オブジェクトの追加」のセクションを参照してください。

GeometricSprite クラスは、次のメソッドも備えています。

- `onMouseDown()` および `onMouseUp()` メソッド。これらについては、192 ページの「表示オブジェクトのクリックとドラッグ」で説明します。
- `showSelected()` および `hideSelected()` メソッド。これらについては、192 ページの「表示オブジェクトのクリックとドラッグ」で説明します。

キャンバスへの表示オブジェクトの追加

ユーザーが Add Shape ボタンをクリックすると、アプリケーションは DrawingCanvas クラスの addShape() メソッドを呼び出します。アプリケーションは、次の例に示すように、新しい GeometricSprite のインスタンス化するために、いずれかの GeometricSprite サブクラスの適切なコンストラクタ関数を呼び出します。

```
public function addShape(shapeName:String, len:Number):void
{
    var newShape:GeometricSprite;
    switch (shapeName)
    {
        case "Triangle":
            newShape = new TriangleSprite(len);
            break;
        case "Square":
            newShape = new SquareSprite(len);
            break;
        case "Circle":
            newShape = new CircleSprite(len);
            break;
    }
    newShape.alpha = 0.8;
    this.addChild(newShape);
}
```

各コンストラクタメソッドは drawShape() メソッドを呼び出し、このメソッドは、Sprite クラスから継承したクラスの graphics プロパティを使用して適切なベクターグラフィックを描画します。たとえば、CircleSprite クラスの drawShape() メソッドには、次のコードが含まれています。

```
this.graphics.clear();
this.graphics.lineStyle(1.0, this.lineColor, 1.0);
this.graphics.beginFill(this.fillColor, 1.0);
var radius:Number = this.size / 2;
this.graphics.drawCircle(radius, radius, radius);
```

addShape() 関数の最後から 2 番目の行は、DisplayObject クラスから継承した表示オブジェクトの alpha プロパティを設定します。これにより、キャンバスに追加される表示オブジェクトはわずかながら透明になり、ユーザーはその背後にあるオブジェクトを見ることができます。

addChild() メソッドの最後の行は、既に表示リストに載っている新しい表示オブジェクトを DrawingCanvas クラスのインスタンスの子リストに追加します。これにより、新しい表示オブジェクトがステージに表示されます。

アプリケーションのインターフェイスには、selectedSpriteTxt と outputTxt の 2 つのテキストフィールドが含まれています。これらのテキストフィールドのテキストプロパティは、キャンバスに追加されるかユーザーによって選択された GeometricSprite オブジェクトに関する情報で更新されます。GeometricSprite クラスは、次のように toString() メソッドをオーバーライドすることにより、この情報報告タスクを処理します。

```
public override function toString():String
{
  return this.shapeType + “ of size “ + this.size + " at " + this.x + ", " +
  this.y;
}
```

shapeType プロパティは、各 GeometricSprite サブクラスのコンストラクタメソッドの中で、適切な値に設定されます。たとえば、toString() メソッドは、DrawingCanvas インスタンスに新たに追加された CircleSprite インスタンスの次の値を返す場合があります。

```
Circle of size 50 at 0, 0
```

DrawingCanvas クラスの describeChildren() メソッドは、DisplayObjectContainer クラスから継承した numChildren プロパティを使用して for ループの限度を設定し、キャンバスの子リストをループします。このメソッドは、次のように、それぞれの子をリストしたストリングを生成します。

```
var desc:String = "";
var child:DisplayObject;
for (var i:int = 0; i < this.numChildren; i++)
{
    child = this.getChildAt(i);
    desc += i + ": " + child + '\n';
}
```

結果のストリングは、outputTxt テキストフィールドの text プロパティを設定するために使用されます。

表示オブジェクトのクリックとドラッグ

ユーザーが GeometricSprite インスタンスをクリックすると、アプリケーションは onMouseDown() イベントハンドラを呼び出します。次に示すように、このイベントハンドラは GeometricSprite クラスのマウスダウンイベントを受け取るように設定されています。

```
this.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
```

次に、onMouseDown() メソッドは GeometricSprite オブジェクトの showSelected() メソッドを呼び出します。このオブジェクトに対してこのメソッドが最初に呼び出された場合は、selectionIndicator という名前の新しい Shape オブジェクトが作成され、次のように、その Shape オブジェクトの graphics プロパティを使用して赤いハイライトの矩形が描画されます。

```
this.selectionIndicator = new Shape();
this.selectionIndicator.graphics.lineStyle(1.0, 0xFF0000, 1.0);
```

```
this.selectionIndicator.graphics.drawRect(-1, -1, this.size + 1,
  this.size + 1);
this.addChild(this.selectionIndicator);
```

onMouseDown() メソッドの呼び出しが初めてでない場合、このメソッドは単に、DisplayObject クラスから継承した selectionIndicator シェイプの visible プロパティを次のように設定します。

```
this.selectionIndicator.visible = true;
```

hideSelected() メソッドは、前に選択されたオブジェクトの selectionIndicator シェイプを、その visible プロパティを false に設定することによって非表示にします。

また、onMouseDown() イベントハンドラは、次のコードを含んでいる startDrag() メソッドも呼び出します。このメソッドは、Sprite クラスから継承されたものです。

```
var boundsRect:Rectangle = this.parent.getRect(this.parent);
boundsRect.width -= this.size;
boundsRect.height -= this.size;
this.startDrag(false, boundsRect);
```

これにより、ユーザーは選択したオブジェクトを、キャンバスの boundsRect 矩形によって設定された境界内でドラッグできます。

ユーザーがマウスボタンを離すと、mouseUp イベントが送出されます。DrawingCanvas のコンストラクタメソッドは、次のイベントリスナーを設定します。

```
this.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
```

このイベントリスナーは、個々の GeometricSprite オブジェクト用でなく、DrawingCanvas オブジェクト用に設定されます。なぜなら、GeometricSprite オブジェクトは、ドラッグされた場合、マウスボタンが離されたときに、別の表示オブジェクト (別の GeometricSprite オブジェクト) の背後で終了する可能性があるからです。前景の表示オブジェクトは、マウスアップイベントを受け取りますが、ユーザーがドラッグしている表示オブジェクトは、それを受け取りません。DrawingCanvas オブジェクトにリスナーを追加すると、そのイベントが常に確実に処理されます。

onMouseUp() メソッドは GeometricSprite オブジェクトの onMouseUp() メソッドを呼び出し、後者のメソッドは GeometricSprite オブジェクトの stopDrag() メソッドを呼び出します。

表示オブジェクトのレイヤーの再配置

アプリケーションのユーザーインターフェイスには、Move Back、Move Down、Move Up、および Move to Front というラベルの付いたボタンが含まれます。ユーザーがこれらのボタンのいずれかをクリックすると、アプリケーションは DrawingCanvas クラスの対応するメソッド、moveToBack()、moveDown()、moveUp()、または moveToFront() を呼び出します。たとえば、moveToBack() メソッドには次のコードが含まれています。

```
public function moveToBack(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, 0);
    }
}
```

このメソッドは、DisplayObjectContainer クラスから継承された setChildIndex() メソッドを使用して、表示オブジェクトを DrawingCanvas インスタンス (this) の子リスト内のインデックス位置 0 に位置付けます。

moveDown() メソッドもほとんど同じ働きをしますが、次のように、DrawingCanvas インスタンスの子リスト内で表示オブジェクトのインデックス位置を 1 ずつデクリメントする点が異なります。

```
public function moveDown(shape:GeometricSprite):void
{
    var index:int = this.getChildIndex(shape);
    if (index > 0)
    {
        this.setChildIndex(shape, index - 1);
    }
}
```

moveUp() および moveToFront() メソッドは、moveToBack() および moveDown() メソッドと同様の働きをします。

第２部

ActionScript 3.0 の基本データ型およびコアクラス

2

ここでは、主要な ActionScript 3.0 クラスおよびデータ型の実装と、それらを使用する際の方針について説明します。

次の章が含まれます。

第 6 章：日付と時刻の操作 197
第 7 章：ストリングの操作 209
第 8 章：配列の操作 227
第 9 章：エラー処理 255
第 10 章：正規表現の使用 285
第 11 章：XML の操作 311

第 6 章

日付と時刻の操作

6

時間を扱う処理は常に必要なわけではありませんが、多くのソフトウェアアプリケーションにとって重要な要素の 1 つです。ActionScript 3.0 には、カレンダーの日付、時刻、および時間間隔を扱う強力な機能が用意されています。時間に関連したそれらの機能のほとんどは、2 つのメインクラス、つまり Data クラス、および、flash.utils パッケージに含まれる新しい Timer クラスにより提供されます。

目次

カレンダー日付と時刻の管理 197
タイマー間隔の制御 201
例 : 単純なアナログ時計 204

カレンダー日付と時刻の管理

ActionScript 3.0 では、カレンダー日付と時刻の管理に関するすべての機能がトップレベルの Date クラスにまとめられています。Date クラスには、世界標準時 (UTC) または何らかのタイムゾーンに基づくローカル時間において日付と時刻を処理するメソッドおよびプロパティがあります。UTC は時刻の基準となるもので、グリニッジ標準時 (GMT) と基本的には同じです。

Date オブジェクトの作成

Date クラスは、すべてのコアクラスの中で最も多用途に使用できるコンストラクタメソッドを備えています。このコンストラクタは次のように 4 つの方法で使用できます。

(1) パラメータを指定しない場合、`Date()` コンストラクタでは、動作環境のタイムゾーンにおけるローカル時間で現在の日付と時刻を格納した Date オブジェクトを返します。その例を次に示します。

```
var now:Date = new Date();
```

(2) 1 個の数値パラメータを指定した場合、Date() コンストラクタではその値を 1970 年 1 月 1 日からの経過ミリ秒数と解釈し、それに基づいた Date オブジェクトを返します。パラメータは UTC の 1970 年 1 月 1 日を起点とする経過ミリ秒数として解釈されますが、返される Date オブジェクトが示す値は、(UTC 用メソッドを使用して取得および表示しない限り) ローカルのタイムゾーンで表示されます。ミリ秒数を表す 1 個のパラメータを指定して Date オブジェクトを新規作成する際は、ローカル時間と UTC の間のタイムゾーンの時差に注意してください。次のステートメントでは、UTC の 1970 年 1 月 1 日 0 時を表す Date オブジェクトを作成します。

```
var millisecondsPerDay:int = 1000 * 60 * 60 * 24;
// 起点 1970/1/1 から 1 日後の Date を取得する
var startTime:Date = new Date(millisecondsPerDay);
```

(3) 複数個の数値パラメータを Date() コンストラクタに指定することもできます。その場合、コンストラクタでは各パラメータを順に年、月、日、時、分、秒、ミリ秒の値と解釈し、それらに基づいた Date オブジェクトを返します。パラメータの値は UTC ではなくローカル時間を表すものと見なされます。次のステートメントでは、ローカル時間の 2000 年 1 月 1 日午前 0 時を表す Date オブジェクトを作成します。

```
var millenium:Date = new Date(2000, 0, 1, 0, 0, 0, 0);
```

(4) 1 個のストリングパラメータを Date() コンストラクタに指定した場合、コンストラクタではそのストリングを解析して日付と時刻の要素を抽出しようと試み、それらに基づいた Date オブジェクトを返します。この方法で使用する場合は、Date() コンストラクタを try..catch ブロックで囲み、解析時のエラーをトラップできるようにしておくことをお勧めします。Date() コンストラクタが受け付けるストリング形式は多数あり、その一覧が『ActionScript 3.0 リファレンスガイド』に示されています。次のステートメントでは、新しい Date オブジェクトの初期化にストリング値を使用しています。

```
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");
```

Date() コンストラクタがストリングパラメータの解析に失敗した場合は、例外が発生するのではなく、作成した Date オブジェクトに無効な日付値が格納されます。

時間単位の値の取得

Date オブジェクトに格納された各種の時間単位の値は、Date クラスのプロパティとメソッドを使用して取得できます。Date オブジェクトに格納されている時間単位の値を取得するには、次の各プロパティを使用します。

- fullYear プロパティ (年)
- month プロパティ (月): 1 〜 12 月を 0 〜 11 で表す数値形式
- date プロパティ (日): 月の暦日を 1 〜 31 で表す数値

- day プロパティ (曜日): 日曜〜土曜を 0 〜 6 で表す数値形式
- hours プロパティ (時): 0 〜 23
- minutes プロパティ (分)
- seconds プロパティ (秒)
- milliseconds プロパティ (ミリ秒)

また、Date クラスにはそれぞれの時間単位の値を取得する方法が複数用意されています。たとえば、Date オブジェクトに格納された月の値は次の 4 とおりの方法で取得できます。

- month プロパティ
- getMonth() メソッド
- monthUTC プロパティ
- getMonthUTC() メソッド

4 つの方法は効率面ではいずれも同等なので、必要に応じて便利な方法を使用してください。

上記のプロパティはそれぞれ、日付の値全体の一部を構成する要素を表しています。たとえば、milliseconds プロパティの値が 999 を超えることはありません。1000 に到達すると seconds プロパティの値が 1 加算され、milliseconds プロパティの値は 0 になります。

Date オブジェクトの値を 1970 年 1 月 1 日 (UTC) からの経過ミリ秒数として取得する必要がある場合は、getTime() メソッドを使用してください。また、このメソッドと対照の関係にある setTime() メソッドを使用すれば、1970 年 1 月 1 日 (UTC) からの経過ミリ秒数を指定して既存の Date オブジェクトに値を設定できます。

日付と時刻の加算および減算の実行

Date クラスでは、日付と時刻に対して加算および減算を実行できます。日付値は、内部的にはミリ秒換算で維持されているので、Date オブジェクトとの間で加算または減算を行う前に、その他の値をミリ秒に変換する必要があります。

日付と時刻の加算および減算を数多く実行するアプリケーションの場合は、次のように、共通の時刻単位値をミリ秒換算で保持する定数を作成すると便利です。

```
public static const millisecondsPerMinute:int = 1000 * 60;
public static const millisecondsPerHour:int = 1000 * 60 * 60;
public static const millisecondsPerDay:int = 1000 * 60 * 60 * 24;
```

これで標準の時刻単位を使用して日付の加算および減算を実行することが簡単になります。次のコードでは、getTime() および setTime() メソッドを使用して、日付値を現在の時刻から 1 時間後に設定します。

```
var oneHourFromNow:Date = new Date();
oneHourFromNow.setTime(oneHourFromNow.getTime() + millisecondsPerHour);
```

日付値を設定するもう 1 つの方法は、ミリ秒を表す 1 個のパラメータを使用して新しい Date オブジェクトを作成することです。たとえば、次のコードでは、ある日付に日数を 30 加えた日付を計算します。

```
// 今日を請求書の日付として設定する
var invoiceDate:Date = new Date();

// 30 日を加算して支払日を取得する
var dueDate:Date = new Date(invoiceDate.getTime() + (30 * millisecondsPerDay));
```

次に、30 日の時間を表すために millisecondsPerDay 定数に 30 を掛け、その結果を invoiceDate 値に加えて dueDate 値の設定に使用します。

タイムゾーン間の変換

日付と時刻の演算処理は、あるタイムゾーンを別のタイムゾーンに換算する必要がある場合に便利です。また、その際には、Date オブジェクトのタイムゾーンと UTC との時差 (分) を返す getTimezoneOffset() メソッドも役立ちます。戻り値が分単位なのは、タイムゾーンによっては 1 時間未満の単位で時差が設定されており、隣接するタイムゾーンとの差が 30 分という場合があるためです。

次の例では、タイムゾーンのオフセットを使用してローカル時間の日付を UTC に変換します。まずミリ秒単位のタイムゾーン値を算出し、それに基づいて Date オブジェクトの値を調整するという方法で処理を実行しています。

```
// ローカル時間の Date を作成する
var nextDay:Date = new Date("Mon May 1 2006 11:30:00 AM");

// タイムゾーンのオフセットを加算または減算して、Date を UTC に変換する
var offsetMilliseconds:Number = nextDay.getTimezoneOffset() * 60 * 1000;
nextDay.setTime(nextDay.getTime() + offsetMilliseconds);
```

タイマー間隔の制御

Flash オーサリングツールによるアプリケーション開発の場合は、タイムラインを使用してアプリケーションの進行をフレーム単位で確実に制御できますが、ActionScript だけを使用するプロジェクトの場合は、別のタイミング制御メカニズムを使用する必要があります。

ループとタイマーの違い

プログラミング言語によっては、`for` や `do..while` などのループステートメントを使ってタイミングの制御方法を独自に工夫しなければならないことがあります。

一般にループステートメントは、動作環境となるローカルマシンの処理速度に応じてなるべく高速に実行されるため、使用するマシンによって速度に違いが生じます。アプリケーションを一定のタイミングで動作させるには、何らかのカレンダーやクロックによって生成されるタイミングを利用する必要があります。ゲーム、アニメーション、およびリアルタイム制御など、多くのアプリケーションは、どのマシンでも一貫して利用できる、安定した時間ベースのタイミングメカニズムを必要とします。

ActionScript 3.0 の Timer クラスには、こうしたニーズを満たす強力な機能があります。Timer クラスでは ActionScript 3.0 のイベントモデルを使用し、指定された時間間隔が経過するごとにタイマーイベントを送出します。

Timer クラス

タイミング制御機能の扱いについて、ActionScript 3.0 では Timer クラス (flash.utils.Timer) を使用する方法が推奨されています。このクラスを使用すると、所定の時間間隔が経過するたびにイベントを送出することができます。

タイマーを開始するには、まず Timer クラスのインスタンスを作成してから、そのインスタンスに対し、タイマーイベントを生成する頻度および生成終了までの送出回数を指定します。

たとえば、次のコードでは、1 秒ごとのイベント送出を 60 秒間続ける Timer インスタンスを作成します。

```
var oneMinuteTimer:Timer = new Timer(1000, 60);
```

Timer オブジェクトでは、所定の時間が経過するたびに TimerEvent オブジェクトを送出します。TimerEvent オブジェクトのイベント型は、`timer` (定数 `TimerEvent.TIMER` によって定義される) です。TimerEvent オブジェクトには、標準の Event オブジェクトと同じプロパティが格納されます。

また、Timer インスタンスで時間間隔の数が固定されている場合、最後の時間間隔に到達したときに `timerComplete` イベント (定数 `TimerEvent.TIMER_COMPLETE` によって定義される) も送出します。

Timer クラスの実際の動作を確認できる短いサンプルアプリケーションを次に示します。

```
package
{
    import flash.display.Sprite;
    import flash.events.TimerEvent;
    import flash.utils.Timer;

    public class ShortTimer extends Sprite
    {
        public function ShortTimer()
        {
            // 5 秒の Timer を新規作成する
            var minuteTimer:Timer = new Timer(1000, 5);

            // 時間間隔イベントおよび完了イベント用のリスナーを指定する
            minuteTimer.addEventListener(TimerEvent.TIMER, onTick);
            minuteTimer.addEventListener(TimerEvent.TIMER_COMPLETE,
    onTimerComplete);

            // タイマーのカウントを開始する
            minuteTimer.start();
        }

        public function onTick(evt:TimerEvent):void
        {
            // 現在のカウントを表示する
            // このイベントの target は、該当する Timer インスタンス自体を示す
            trace("tick " + evt.target.currentCount);
        }

        public function onTimerComplete(evt:TimerEvent):void
        {
            trace("Time's Up!");
        }
    }
}
```

この例では、ShortTimer クラスが作成されたとき、1 秒ごとのカウントを 5 秒間続ける Timer インスタンスを作成します。次に、このタイマーに 2 つのリスナーを追加します。1 つはカウント各回のイベントを受け取るリスナーで、もう 1 つは `timerComplete` イベントを受け取るリスナーです。

その後、タイマーのカウントを開始すると、以後は 1 秒間隔で `onTick()` メソッドが繰り返し呼び出されます。

`onTick()` メソッドでは、単に現在のカウントを表示しています。5 秒が経過すると `onTimerComplete()` メソッドが呼び出され、カウントが終了したことを表示します。

このサンプルを実行すると、コンソールまたはトレースウィンドウに 1 秒ごとに次の行が表示されます。

```
tick 1
tick 2
tick 3
tick 4
tick 5
Time's Up!
```

flash.utils パッケージに含まれるタイマー制御関数

ActionScript 3.0 には、ActionScript 2.0 と同様のタイマー制御関数が多数用意されています。それらはパッケージレベル関数として flash.utils パッケージに含まれており、ActionScript 2.0 の場合と同じように動作します。

関数	説明
`clearInterval(id:uint):void`	指定した `setInterval()` 呼び出しをキャンセルします。
`clearTimeout(id:uint):void`	指定した `setTimeout()` 呼び出しをキャンセルします。
`getTimer():int`	Flash Player が初期化された時点からの経過ミリ秒数を返します。
`setInterval(closure:Function, delay:Number, ... arguments):uint`	ミリ秒単位で指定した間隔ごとに関数を実行します。
`setTimeout(closure:Function, delay:Number, ... arguments):uint`	ミリ秒単位で指定した遅延時間の経過後に、指定した関数を実行します。

これらの関数は、後方互換性を維持する目的で ActionScript 3.0 にも残されているものです。新しい ActionScript 3.0 アプリケーションで使用することはお勧めしません。一般に、Timer クラスを使用するほうが容易さの点でも効率の点でも有利です。

例 : 単純なアナログ時計

ここでは、単純なアナログ時計の例を使って、この章で述べた日付と時刻に関する内容のうち次の 2 つについて示します。

- 現在の日付と時刻を取得し、時、分、秒の値を抽出する方法
- Timer を使用してアプリケーションの進行ペースを制御する方法

SimpleClock アプリケーションのファイルは、"Samples/SimpleClock" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
SimpleClockApp.mxml	MXML で記述された Flex 用メインアプリケーションファイル
SimpleClockApp.fla	FLA ファイル形式の Flash オーサリングツール用メインアプリケーションファイル
com/example/programmingas3/simpleclock/SimpleClock.as	メインアプリケーションファイル
com/example/programmingas3/simpleclock/AnalogClockFace.as	時計の丸い文字盤と時間に合わせた時針、分針、秒針を描画するコード

SimpleClock クラスの定義

この時計のように単純なアプリケーションであっても、構造をわかりやすく整理しておけば、後で容易に拡張できて便利です。そこで、この SimpleClock アプリケーションの場合は SimpleClock クラスで起動時とタイマー制御の処理を扱い、時刻を実際に表示する処理には別の AnalogClockFace というクラスを使用しています。

次のコードでは、SimpleClock クラスの定義と初期化を行います。

```
public class SimpleClock extends UIComponent
{
   /**
    * 時刻表示コンポーネント
    */
   private var face:AnalogClockFace;

   /**
    * このアプリケーションにとって心臓の鼓動のような役割を持つ Timer
    */
   private var ticker:Timer;
```

このクラスには 2 つの重要なプロパティがあります。

- `face` プロパティ : AnalogClockFace クラスのインスタンス
- `ticker` プロパティ : Timer クラスのインスタンス

SimpleClock クラスではデフォルトのコンストラクタを使用します。`initClock()` メソッドでは実際のセットアップ処理を担当し、文字盤を作成して Timer インスタンスの動作を開始します。

文字盤の作成

次のコードでは、時刻の表示に使用する文字盤を作成します。

```
/**
 * SimpleClock インスタンスをセットアップする
 */
public function initClock(faceSize:Number = 200)
{
    // 文字盤を作成して表示リストに追加する
    face = new AnalogClockFace(Math.max(20, faceSize));
    face.init();
    addChild(face);

    // 時計の初期状態を表示する
    face.draw();
```

`initClock()` メソッドに文字盤のサイズを指定できます。`faceSize` の値を指定しない場合、デフォルト値の 200 ピクセルが使用されます。

次に、このアプリケーションでは文字盤を初期化し、DisplayObject クラスから継承した `addChild()` メソッドを使用して表示リストに文字盤を追加します。さらに、いったん現在の時刻で盤面を表示するために `AnalogClockFace.draw()` メソッドを呼び出します。

タイマーの開始

文字盤を作成した後は、initClock() メソッドでタイマーをセットアップします。

```
// 1 秒ごとにイベントを送出する Timer を作成する
ticker = new Timer(1000);

// Timer イベントを処理する onTick() メソッドを指定する
ticker.addEventListener(TimerEvent.TIMER, onTick);

// 時間の計測を開始する
ticker.start();
```

このメソッドでは、まず、1 秒 (1000 ミリ秒) ごとにイベントを送出する Timer インスタンスを作成します。第 2 パラメータの repeatCount を Timer() コンストラクタに渡していないので、この Timer はイベントの送出を無期限に繰り返します。

SimpleClock.onTick() メソッドは、timer イベントを受け取って 1 秒ごとに実行されます。

```
public function onTick(evt:TimerEvent):void
{
    // 時計の表示を更新する
    face.draw();
}
```

AnalogClockFace.draw() メソッドでは単に時計の文字盤と針を表示します。

現在時刻の表示

AnalogClockFace クラスのコードのほとんどは、時計の文字盤を構成する表示要素のセットアップに関するものです。AnalogClockFace が初期化される際には、丸い輪郭を描き、個々の定時目盛りの位置にテキストラベルで数字を配置し、さらに、時針、分針、秒針を表す 3 つの Shape オブジェクトを作成します。

SimpleClock アプリケーションが動作し始めた後は、次のように AnalogClockFace.draw() メソッドが 1 秒ごとに呼び出されます。

```
/**
* 表示を描画する際に親コンテナから呼び出される
*/
public override function draw():void
{
    // 現在の日付と時刻をインスタンス変数に格納する
    currentTime = new Date();
    showTime(currentTime);
}
```

このメソッドでは、描画の最中に時刻が変化するのを避けるため、現在時刻を変数に格納しています。それから、次のようにshowTime()メソッドを呼び出して針を表示します。

```
/**
* 指定された日付 / 時刻を、懐かしのアナログ時計スタイルで表示する
*/
public function showTime(time:Date):void
{
    // 時刻を構成する値を取得する
    var seconds:uint = time.getSeconds();
    var minutes:uint = time.getMinutes();
    var hours:uint = time.getHours();

    // 6 を掛けて角度を求める
    this.secondHand.rotation = 180 + (seconds * 6);
    this.minuteHand.rotation = 180 + (minutes * 6);

    // 30 を掛けて基本の角度を求め、さらに
    // 最大 29.5 度 (59×0.5) までの角度を
    // 分の値に応じて加える
    this.hourHand.rotation = 180 + (hours * 30) + (minutes * 0.5);
}
```

このメソッドでは、まず現在時刻の時、分、秒それぞれの値を抽出し、それらに基づいて3本の針の角度を計算します。秒針は60秒で1周するので、1秒につき6度(360/60)だけ進むことになります。分針も1分につきこれと同じ角度だけ進みます。

分針が進むにつれて、時針も少しずつ動きます。時針は1時間につき30度(360/12)だけ進みますが、その間も1分につき0.5度(30度/60分)ずつ変化していきます。

第 7 章

ストリングの操作

7

String クラスには、テキストストリングを操作するメソッドが含まれています。ストリングの扱いは、さまざまなオブジェクトを使用する場合において重要となります。この章で説明する各種のメソッドは、TextField、StaticText、XML、ContextMenu、FileReference などのオブジェクトで使用されるストリングを操作する際に役立ちます。

ストリングは文字の連続によって構成されます。ActionScript 3.0 では ASCII と Unicode の文字がサポートされます。

目次

ストリングの作成 209
length プロパティ 211
ストリング内の文字の操作 211
ストリングの比較 212
他のオブジェクトのストリング表現の取得 213
ストリングの連結 213
ストリング内に含まれるサブストリングおよびパターンの検索 214
ストリングの大文字 / 小文字の変換 219
例 : ASCII アート 220

ストリングの作成

ActionScript 3.0 では、String クラスを使用してストリング (テキスト) データを表現します。ActionScript ストリングでは、ASCII 文字と Unicode の文字がサポートされています。ストリングを作成する最も簡単な方法は、ストリングリテラルを使用する方法です。ストリングリテラルを宣言するには、二重引用符 (") または一重引用符 (') で囲みます。たとえば、次の 2 つのストリングは同等です。

```
var str1:String = "hello";
var str2:String = 'hello';
```

また、次のように new 演算子を使用してストリングを宣言することもできます。

```
var str1:String = new String("hello");
```

```
var str2:String = new String(str1);
var str3:String = new String();       // str3 == null
```

次の 2 つのストリングは同等です。

```
var str1:String = "hello";
var str2:String = new String("hello");
```

区切り文字が一重引用符 (') で記述されているストリングリテラル内で一重引用符 (') を使用するには、エスケープ文字として円記号 (\) を付けます。同様に、区切り文字が二重引用符 (") で記述されているストリングリテラル内で二重引用符 (") を使用するには、エスケープ文字として円記号 (\) を付けます。次の 2 つのストリングは同等です。

```
var str1:String = "That's \"A-OK\"";
var str2:String = 'That\'s "A-OK"';
```

一重引用符または二重引用符の使用は、次に示すように、ストリングリテラルに存在するのが一重引用符であるか二重引用符であるかに基づいて選択することもできます。

```
var str1:String = "ActionScript <span class='heavy'>3.0</span>";
var str2:String = '<item id="155">banana</item>';
```

ActionScript では、垂直の一重引用符 (') と右向きまたは左向きの引用符 (‘ または ’) は区別されるので注意してください。この点は二重引用符についても同様です。ストリングリテラルを囲む場合は垂直の引用符を使用する必要があります。別の場所からコピーしたテキストを ActionScript にペーストする際は、正しい文字になっているか確認してください。

次の表に示すとおり、エスケープ文字の円記号 (\) を使用すると、その他の文字をストリングリテラル内に記述することもできます。

エスケープシーケンス	文字
`\b`	バックスペース文字
`\f`	改ページ
`\n`	改行
`\r`	復帰文字
`\t`	タブ文字
`\u`*nnnn*	16 進数値 nnnn で指定した文字コードの Unicode 文字。例 : `\u263a` はスマイル記号
`\x`*nn*	16 進数値 nn で指定した文字コードの ASCII 文字
`\'`	一重引用符
`\"`	二重引用符
`\\`	単一の円記号

length プロパティ

各ストリングには、そのストリング内に含まれる文字の個数を示す `length` プロパティがあります。

```
var str:String = "macromedia";
trace(str.length);            // 10
```

次のとおり、空のストリングと null ストリングの length はいずれも 0 です。

```
var str1:String = new String();
trace(str1.length);           // 0

str2:String = '';
trace(str2.length);           // 0
```

ストリング内の文字の操作

ストリング内のすべての文字が、ストリング内のインデックス位置を持っています (整数)。先頭の文字を指すインデックス位置は 0 です。たとえば、次のストリング内で `y` という文字は 0 の位置にあり、`w` という文字は 5 の位置にあります。

```
"yellow"
```

次の例のように、`charAt()` メソッドおよび `charCodeAt()` メソッドを使用して、ストリング内のさまざまな位置にある個々の文字を調べることができます。

```
var str:String = "hello world!";
for (var:i = 0; i < str.length; i++)
{
  trace(str.charAt(i) + " - " + str.charCodeAt(i));
}
```

このコードを実行すると次の出力が表示されます。

```
h - 104
e - 101
l - 108
l - 108
o - 111
  - 32
w - 119
o - 111
r - 114
l - 108
d - 100
! - 33
```

また、次の例のように、fromCharCode() メソッドを使用して、文字コードでストリングを定義することもできます。

```
var myStr:String =
   String.fromCharCode(104,101,108,108,111,32,119,111,114,108,100,33);
      // myStr を "hello world!" に設定する
```

ストリングの比較

ストリングを比較するには、演算子 <、<=、!=、==、=> および > を使用することができます。これらの演算子は、次の例のように if や while などの条件ステートメントで使用できます。

```
var str1:String = "Apple";
var str2:String = "apple";
if (str1 < str2)
{
   trace("A < a, B < b, C < c, ...");
}
```

これらの演算子をストリングで使用した場合、次のように、ActionScript では左から右に文字を比較しながら、ストリングの各文字の文字コード値を検討します。

```
trace("A" < "B"); // true
trace("A" < "a"); // true
trace("Ab" < "az"); // true
trace("abc" < "abza"); // true
```

ストリング同士を比較する場合、およびストリングを別の型のオブジェクトと比較する場合は、次の例のように == 演算子と != 演算子を使用します。

```
var str1:String = "1";
var str1b:String = "1";
var str2:String = "2";
trace(str1==str1b); // true
trace(str1==str2); // false
var total:uint = 1;
trace(str1 == total); // true
```

次のように、比較する対象のオブジェクトが両方とも同じ型で、同じ内容であるかどうかを検証する場合は、=== および !== 演算子を使用します。

```
var str1:String = "4";
var str2:String = "4";
var total:uint = 4;
trace(str1 === str2); // true
trace(str1 === total); // false
```

他のオブジェクトのストリング表現の取得

任意の種類のオブジェクトのストリング表現を取得することができます。この変換を行うために、すべてのオブジェクトに toString() メソッドがあります。

```
var n:Number = 99.47;
var str:String = n.toString();
  // str == "99.47"
```

String オブジェクトとストリング以外のオブジェクトの組み合わせに対して + 連結演算子を使用する場合、toString() メソッドは必要ありません。連結の詳細については、次のセクションを参照してください。

String() グローバル関数は、指定されたオブジェクトに対し、toString() メソッドを呼び出すオブジェクトが返す値と同じ値を返します。

ストリングの連結

ストリングの連結とは、2 つのストリングを前後に結合して 1 つのストリングを作成することです。たとえば、次のように + 演算子を使用すると 2 つのストリングを連結できます。

```
var str1:String = "green";
var str2:String = "ish";
var str3:String = str1 + str2;
    // str3 == "greenish"
```

また、次の例に示すように += 演算子を使用しても同じ結果が得られます。

```
var str:String = "green";
str += "ish";
    // str == "greenish"
```

さらに、次のように String クラスの concat() メソッドを使用して連結を実行する方法もあります。

```
var str1:String = "Bonjour";
var str2:String = "from";
var str3:String = "Paris";
var str4:String = concat(str1, " ", str2, " ", str3)
    // str4 == "Bonjour from Paris"
```

+ 演算子 (または += 演算子) を使用して、String オブジェクトとストリング以外のオブジェクトを連結する場合、ActionScript では、次のようにストリングでないオブジェクトが String オブジェクトに自動変換され、それから式が評価されます。

```
var str:String = "Area = ";
var area:Number = Math.PI * Math.pow(3, 2);
str = str + area;
      // str == "Area = 28.274333882308138"
```

ただし、次のように括弧をグループ化に使用すれば、+ 演算子のコンテキストを提供することができます。

```
trace("Total: $" + 4.55 + 1.45);
     //"Total: $4.551.45"
trace("Total: $" + (4.55 + 1.45));
     //"Total: $6"
```

ストリング内に含まれるサブストリングおよびパターンの検索

サブストリングとは、ストリング内の一部を構成する連続した文字です。たとえば、ストリング "abc" にサブストリング ""、"a"、"ab"、"abc"、"b"、"bc"、"c" があるとします。ストリング内のサブストリングは ActionScript のメソッドを使用して抽出できます。

パターンは、ActionScript ではストリングまたは正規表現によって定義されます。たとえば、次の正規表現では、文字 A、B、C の後に数字 1 文字が続くという特定のパターンを定義しています (スラッシュは正規表現の区切り文字です)。

```
/ABC\d/
```

ActionScript には、ストリング内でパターンによって検索するメソッドと、検出された一致パターンを置換サブストリングに置き換えるメソッドが用意されています。これらの方法については、次のセクションで説明します。

正規表現を使用すると複雑なパターンを定義できます。詳細については、285 ページ、第 10 章の「正規表現の使用」を参照してください。

文字の位置によるサブストリングの検索

substr() メソッドと substring() メソッドは似ています。いずれもストリング内のサブストリングを戻り値として返すメソッドです。また、指定するパラメータは 2 つであり、どちらのメソッドでも、最初のパラメータは、与えられたストリングの開始文字の位置です。しかし、substr() メソッドの場合はサブストリングの " 長さ " を第 2 パラメータで指定するのに対し、substring() メソッドの場合はサブストリングの " 終了位置 " を第 2 パラメータで指定します (終了位置にある文字そのものは戻り値に含まれません)。2 つのメソッドの違いを確認できる例を次に示します。

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.substr(11,15)); // Paris, Texas!!!
trace(str.substring(11,15)); // output: Pari
```

slice() メソッド関数は、substring() メソッドに似ています。パラメータに負でない整数を指定した場合の動作はまったく同じです。しかし、slice() メソッドではパラメータに負の整数を指定することもでき、その場合は、ストリングの末尾を基準として文字の位置を指定したことになります。例を次に示します。

```
var str:String = "Hello from Paris, Texas!!!";
trace(str.slice(11,15)); // Pari
trace(str.slice(-3,-1)); // !!
trace(str.slice(-3,26)); // !!!
trace(str.slice(-3,str.length)); // !!!
trace(str.slice(-8,-3)); // Texas
```

slice() メソッドのパラメータには、負でない整数と負の整数を組み合わせて指定することもできます。

パターンに一致するサブストリングが存在する文字位置の検索

indexOf() メソッドと lastIndexOf() メソッドを使用すると、次のように、パターンに一致するサブストリングがストリング内に出現する位置を検索できます。

```
var str:String = "The moon, the stars, the sea, the land";
trace(str.indexOf("the")); // 10
```

indexOf() メソッドでは大文字と小文字が区別されます。

次のように第 2 パラメータを指定すると、ストリング内で検索を開始する位置を示すことができます。

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.indexOf("the", 11)); // 21
```

lastIndexOf() メソッドは、パターンに一致するサブストリングがストリング内に最後に出現する位置を検索します。

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the")); // 30
```

lastIndexOf() メソッドで次のように第 2 パラメータを指定すると、ストリング内の指定したインデックス位置から先頭に向かって (右から左へ) 検索が実行されます。

```
var str:String = "The moon, the stars, the sea, the land"
trace(str.lastIndexOf("the", 29)); // 21
```

区切り文字で分割したサブストリングからなる配列の作成

`split()` メソッドを使用すると､ストリングを区切り文字で分割することによってサブストリングの配列を作成できます。たとえば、カンマ区切りやタブ区切りのデータを含んだストリングを複数のストリングに分割できます。

次の例では、アンパサンド (&) を区切り文字としてストリングを分割し、サブストリングの配列を作成します。

```
var queryStr:String = "first=joe&last=cheng&title=manager&StartDate=3/6/65";
var params:Array = queryStr.split("&", 2);
     //     params == ["first=joe","last=cheng"]
```

`split()` メソッドの第 2 パラメータはオプションで、返される配列の最大サイズを指定します。

区切り文字には、次のように正規表現を指定することもできます。

```
var str:String = "Give me\t5."
var a:Array = str.split(/\s+/);
  // a == ["Give","me","5."]
```

詳細については、285 ページ、第 10 章の「正規表現の使用」および ActionScript 3.0 リファレンスガイドを参照してください。

ストリング内のパターンの検索およびサブストリングの置換

String クラスには、ストリング内のパターンを扱う次のメソッドがあります。

- `match()` および `search()` メソッドでは、パターンに一致するサブストリングを検索できます。
- `replace()` メソッドでは､パターンに一致するサブストリングを検索し､該当する部分を指定のサブストリングに置換できます。

これらの方法については、次のセクションで説明します。

これらのメソッドで使用するパターンは、ストリングまたは正規表現によって定義できます。正規表現の詳細については、285 ページ、第 10 章の「正規表現の使用」を参照してください。

パターンに一致するサブストリングの検索

search() メソッドでは、次の例のように、指定したパターンに一致するサブストリングが最初に出現するインデックス位置を返します。

```
var str:String = "The more the merrier.";
trace(str.search("the"));
  // output: 9
  // ( この検索では大文字と小文字が区別される )
```

検索するパターンは、次の例に示すように、正規表現を使用して定義することもできます。

```
var pattern:RegExp = /the/i;
var str:String = "The more the merrier.";
trace(str.search(pattern)); // 0
```

ストリングの先頭にある文字のインデックス位置は 0 なので、この場合は trace() メソッドで 0 が出力されます。正規表現の i フラグを設定することで、大文字と小文字を区別せずに検索を実行しています。

search() メソッドは、一致するサブストリングを 1 つだけ検索してその開始位置のインデックスを返します。たとえ g (グローバル) フラグを設定しても、複数の一致箇所を検索することはありません。

次の例では、より複雑な正規表現を指定することで、二重引用符に囲まれたストリングを検索します。

```
var pattern:RegExp = /"[^"]*"/;
var str:String = "The \"more\" the merrier.";
trace(str.search(pattern));
  // output: 4

str = "The \"more the merrier.";
trace(str.search(pattern));
  // output: -1
  // ( 一致なしを表す。二重引用符が閉じていないため一致しない )
```

match() メソッドの機能も search() メソッドと似ており、パターンに一致するサブストリングを検索する点は同じです。ただし、次のように正規表現パターンでグローバルフラグを指定すると、match() の場合は一致したサブストリングの配列を返します。

```
var str:String = "bob@example.com, omar@example.org";
var pattern:RegExp = /\w*@\w*\.[org|com]+/g;
var results:Array = str.match(pattern);
```

results 配列は次のように設定されています。

```
["bob@example.com","omar@example.org"]
```

正規表現の詳細については、285 ページ、第 10 章の「正規表現の使用」を参照してください。

パターンに一致するサブストリングの置換

replace() メソッドを使用すると、次のように、パターンに一致するサブストリングがストリング内に出現する位置を検索し、指定の置換ストリングに置換できます。

```
var str:String = "She sells seashells by the seashore.";
var pattern:RegExp = /sh/gi;
trace(str.replace(pattern, "sch"));
  //sche sells seaschells by the seaschore.
```

この例で注意する点は、正規表現の i (ignoreCase) フラグの設定により、一致したストリングの大文字と小文字が区別されていないことと、g (global) フラグの設定により複数の一致箇所が置換されていることです。詳細については、285 ページ、第 10 章の「正規表現の使用」を参照してください。

置換ストリングには、次の $ 置換コードを含めることができます。$ 置換コードを記述した位置には、次の表に示す置換テキストが挿入されます。

$ コード	置換テキスト
$$	$
$&	パターンに一致したサブストリング
$`	ストリングのうち、パターンに一致したサブストリングより前にある部分。このコードで使用する記号は、垂直の左側一重引用符 (`) です。垂直の一重引用符 (') や、曲がった左側一重引用符 (‘) ではありません。
$'	ストリングのうち、パターンに一致したサブストリングより後にある部分。このコードで使用する記号は、垂直の一重引用符 (') です。
$*n*	パターン内のグループ化括弧によってキャプチャされた n 番目のサブストリング。*n* は 1 桁の数字 (1 ～ 9) であり、$*n* の後に 10 進数字は続きません。
$*nn*	パターン内のグループ化括弧によってキャプチャされた nn 番目のサブストリング。nn は 2 桁の数字 (01 ～ 99) です。nn 番目のキャプチャが未定義の場合、置換テキストは空のストリングになります。

たとえば、一致する 1 番目と 2 番目のキャプチャグループを表す $2 および $1 置換コードを使用した場合を次に示します。

```
var str:String = "flip-flop";
var pattern:RegExp = /(\w+)-(\w+)/g;
trace(str.replace(pattern, "$2-$1")); // flop-flip
```

replace() メソッドの第 2 パラメータには関数を指定することもできます。その場合は、一致したテキストが、指定した関数の戻り値によって置換されます。

```
var str:String = "Now only $9.95!";
var price:RegExp = /\$([\d,]+.\d+)+/i;
trace(str.replace(price, usdToEuro));
```

```
function usdToEuro(matchedSubstring:String,
                    capturedMatch1:String,
                    index:int,
                    str:String):String
{
   var usd:String = capturedMatch1;
   usd = usd.replace(",", "");
   var exchangeRate:Number = 0.853690;
   var euro:Number = usd * exchangeRate;
   const euroSymbol:String = String.fromCharCode(8364);
   return euro.toFixed(2) + " " + euroSymbol;
}
```

replace() メソッドの第 2 パラメータとして関数を使用した場合は、次のパラメータが関数に渡されます。

- ストリング内の一致する部分。
- グループ化括弧によってキャプチャされる一致。この方法で渡される引数の数は、括弧内のパターンとの一致数によって異なります。括弧内のパターンとの一致数は、関数コード内の arguments.length - 3 を確認して特定することができます。
- ストリング内で一致部分が始まる場所のインデックス位置。
- ストリング全体。

ストリングの大文字 / 小文字の変換

次の例で示すように、toLowerCase() メソッドと toUpperCase() メソッドは、ストリング内のアルファベット文字をそれぞれ小文字と大文字に変換します。

```
var str:String = "Dr. Bob Roberts, #9."
trace(str.toLowerCase());
   // dr. bob roberts, #9.
trace(str.toUpperCase());
   // DR. BOB ROBERTS, #9.
```

これらのメソッドを実行した後も、元のストリングの内容は変化しません。元のストリング自体を変換するには、次のコードを使用します。

```
str = str.toUpperCase();
```

これらのメソッドは、a ～ z および A ～ Z だけでなく、拡張文字を操作します。

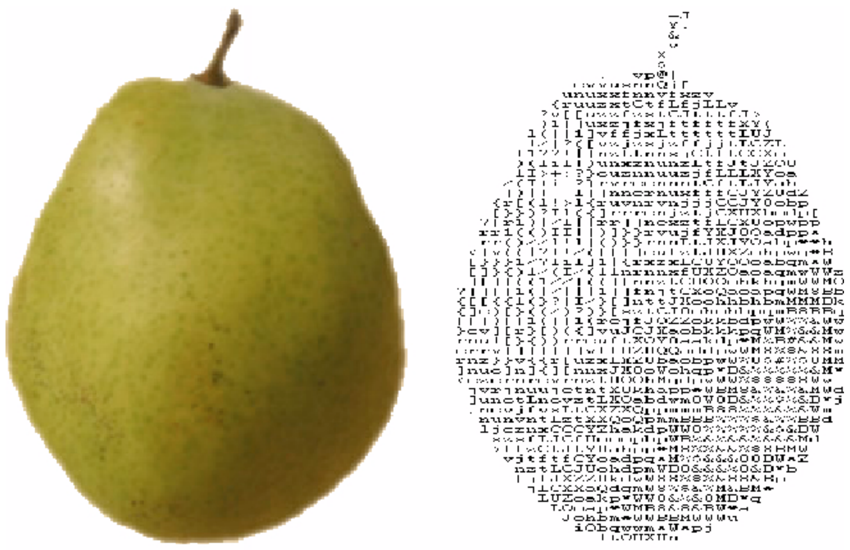
```
var str:String = "José Barça";
trace(str.toUpperCase(), str.toLowerCase()); // JOSÉ BARÇA josé barça
```

例 : ASCII アート

ここでは ASCII アートの例を使って、ActionScript 3.0 の String クラスの操作について次のような機能を示します。

- String クラスの `split()` メソッドを使用して、文字区切りのストリング (タブ区切りテキストファイルのイメージ情報) から値を抽出します。
- `split()`、連結、および `substring()` と `substr()` の使用によるストリング部分の抽出など、何とおりものストリング操作テクニックを使用して、イメージタイトルの各単語の先頭文字を大文字にします。
- `getCharAt()` メソッドを使用して、ストリングから 1 文字を取得します (グレースケールビットマップ値に対応する ASCII 文字を決定します)。
- ストリングの連結を使用して、イメージの ASCII アート表現を一度に 1 文字ずつ構築します。

"ASCII アート " という用語はイメージのテキスト表現を指し、イメージは Courier New 文字のような等幅フォント文字のグリッドで描画されます。次のイメージは、アプリケーションが生成した ASCII アートの例です。



ASCII アートのグラフィックを右側に示してあります。

ASCIIArt アプリケーションのファイルは、"Samples/AsciiArt" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
AsciiArtApp.mxml	MXML で記述された Flex 用アプリケーションのユーザーインターフェイス
com/example/programmingas3/asciiArt/AsciiArtBuilder.as	テキストファイルからのイメージメタデータの抽出、イメージのロード、イメージからテキストへの変換プロセスの管理など、アプリケーションのメイン機能を提供するクラス
com/example/programmingas3/asciiArt/BitmapToAsciiConverter.as	イメージデータをストリングに変換するための parseBitmapData() メソッドを提供するクラス
com/example/programmingas3/asciiArt/Image.as	ロードしたビットマップイメージを表すクラス
com/example/programmingas3/asciiArt/ImageInfo.as	タイトル、イメージファイル URL など、ASCII アートイメージのメタデータを表すクラス
image/	アプリケーションが使用するイメージを含むフォルダ
txt/ImageData.txt	アプリケーションがロードするイメージに関する情報を含む、タブ区切りのテキストファイル

タブ区切り値の抽出

この例では、アプリケーションデータをアプリケーション自体とは別に格納する一般的な慣習に従っています。そのようにしておくと、たとえば、別のイメージが追加またはイメージのタイトルが変更されたときなどデータが変更された場合に、SWF ファイルを再作成する必要がありません。この場合、イメージタイトル、実際のイメージファイルの URL、およびイメージの操作に使用する一部の値を含むイメージメタデータは、テキストファイル (プロジェクトの txt/ImageData.txt ファイル) に格納されています。実際のテキストファイルの中身は次のようになっています。

```
FILENAMETITLEWHITE_THRESHHOLDBLACK_THRESHHOLD
FruitBasket.jpgPear, apple, orange, and bananad810
Banana.jpgA picture of a bananaC820
Orange.jpgorangeFF20
Apple.jpgpicture of an apple6E10
```

このファイルは特定のタブ区切り形式を使用しています。最初の行はヘッダー行です。残りの行は、ロードするビットマップごとに次のデータを含みます。

- ビットマップのファイル名
- ビットマップの表示名
- ビットマップの白しきい値および黒しきい値。これらは 16 進値で、この値を超えた場合および下回った場合にピクセルは完全に白または黒と見なされます。

アプリケーションが起動するとすぐに、AsciiArtBuilder クラスは、AsciiArtBuilder クラスの `parseImageInfo()` メソッドから次のコードを使用して、表示するイメージの " スタック " を作成するためにテキストファイルの内容をロードして解析します。

```
var lines:Array = _imageInfoLoader.data.split("\n");
var numLines:uint = lines.length;
for (var i:uint = 1; i < numLines; i++)
{
    var imageInfoRaw:String = lines[i];
    ...
    if (imageInfoRaw.length > 0)
    {
        // 新しい image info レコードを作成し、それを image info の配列に追加する
        var imageInfo:ImageInfo = new ImageInfo();

        // 現在の行を ( タブ文字 (\t) で区切って ) 値に分割し、
        // 個々のプロパティを抽出する :
        var imageProperties:Array = imageInfoRaw.split("\t");
        imageInfo.fileName = imageProperties[0];
        imageInfo.title = normalizeTitle(imageProperties[1]);
        imageInfo.whiteThreshold = parseInt(imageProperties[2], 16);
        imageInfo.blackThreshold = parseInt(imageProperties[3], 16);
        result.push(imageInfo);
    }
}
```

テキストファイルの内容全体は、単一の String インスタンスの `_imageInfoLoader.data` プロパティに含まれます。改行 (`"\n"`) をパラメータとして `split()` メソッドを使用すると、String インスタンスは、要素がテキストファイルの各行になる Array (`lines`) に分割されます。次に、ループを使用して各行の操作を行います (実際の内容ではなくヘッダーのみを含む先頭行は除く)。ループ内で、`split()` メソッドをもう一度使用して、単一行の内容を値セット (`imageProperties` という Array オブジェクト) に分割します。各行の値はタブ文字で区切られるため、この場合の `split()` メソッドで使用するパラメータはタブ文字 (`"\t"`) です。

String メソッドを使用したイメージタイトルの正規化

このアプリケーションの設計に関する決定事項の 1 つは、すべてのイメージタイトルを標準形式で表示し、そのとき各単語の先頭文字を大文字にします (英語タイトルで先頭大文字にしないのが一般的ないくつかの単語は除く)。適切に整形済みのタイトルがテキストファイルに含まれると見なすのではなく、テキストファイルからの抽出時にタイトルを整形します。

以前のコードリストでは、個々のイメージメタデータ値を抽出する一部として、次のコード行が使用されています。

```
imageInfo.title = normalizeTitle(imageProperties[1]);
```

このコードでは、テキストファイルからのイメージのタイトルは、normalizeTitle() メソッドを使用して渡した後で ImageInfo オブジェクトに格納されます。

```
private function normalizeTitle(title:String):String
{
  var words:Array = title.split(" ");
  var len:uint = words.length;
  for(var i:uint; i < len; i++)
  {
    words[i] = capitalizeFirstLetter(words[i]);
  }

  return words.join(" ");
}
```

このメソッドは、split() メソッドを使用してタイトルを個々の単語に (空白文字で区切って) 分割し、capitalizeFirstLetter() メソッドを通じて各単語を渡してから、Array クラスの join() メソッドを使用して単語を結合し単一のストリングに再び戻します。

名前が示すとおり、capitalizeFirstLetter() メソッドは実際に各単語の先頭文字を大文字にする働きがあります。

```
/**
 * 以下に示す単語のいずれかでなければ、その先頭文字を大文字にする
 * 除かれるのは、通常は英語で先頭大文字にしない単語である
 */
private function capitalizeFirstLetter(word:String):String
{
  switch (word)
  {
    case "and":
    case "the":
    case "in":
    case "an":
    case "or":
    case "at":
    case "of":
    case "a":
      // これらの単語に対しては何も行わない
      break;
    default:
      // 他の単語の場合は、先頭文字を大文字にする
      var firstLetter:String = word.substr(0, 1);
      firstLetter = firstLetter.toUpperCase();
      var otherLetters:String = word.substring(1);
      word = firstLetter + otherLetters;
  }
  return word;
}
```

英語では、タイトルの各単語の先頭文字が "and"、"the"、"in"、"an"、"or"、"at"、"of" または "a" のいずれかの場合、その文字は大文字にしません。これは簡易版の規則です。このロジックを実行するために、最初に switch ステートメントを使用して、先頭を大文字にしない単語のいずれかであるかどうかを確認します。先頭大文字にしない単語の場合は、switch ステートメントからジャンプします。一方、先頭大文字にする単語の場合は、次のように何ステップかでそれを行います。

1. substr(0, 1) を使用して、単語の先頭文字を抽出します。この抽出では、第 1 パラメータ 0 が指定するストリングの先頭文字、つまりインデックス 0 の文字から開始するサブストリングを抽出します。サブストリングは、第 2 パラメータ 1 が指定する 1 文字長になります。
2. toUpperCase() メソッドを使用して、その文字を大文字にします。

3. `substring(1)` を使用して、元の単語の残りの文字を抽出します。この抽出では、インデックス 1 (第 2 文字) から開始して、ストリングの末尾 (`substring()` メソッドの第 2 パラメータの省略が示す) までのサブストリングを抽出します。
4. 新しく大文字にした先頭文字と残りの文字を結合することによって、最終的な単語を作成します。使用するストリングの連結は、`firstLetter + otherLetters` です。

ASCII アートテキストの生成

BitmapToAsciiConverter クラスは、ビットマップイメージを ASCII テキスト表現に変換する機能を提供します。このプロセスは、次に一部を示してある `parseBitmapData()` メソッドが実行します。

```
var result:String = "";

// 先頭から末尾までピクセル行のループ処理をする
for (var y:uint = 0; y < _data.height; y += verticalResolution)
{
    // 各行内で、左から右までピクセルのループ処理をする
    for (var x:uint = 0; x < _data.width; x += horizontalResolution)
    {
        ...

        // 0 ～ 255 の範囲にあるグレー値を、
        // 0 ～ 64 の範囲の値に変換する ( 利用可能な文字セットにおける
        // グレーシェードの数であるため )
        index = Math.floor(grayVal / 4);
        result += palette.charAt(index);
    }
    result += "\n";
}
return result;
```

このコードでは最初に、ASCII アート版のビットマップイメージの構築に使用する `result` という String インスタンスを定義します。次に、ソースビットマップイメージについて個々のピクセルのループ処理をします。何とおりかのカラー操作テクニックを使用して (ここでは簡略化のために省略)、個々のピクセルの赤、緑、青のカラー値を単一のグレースケール値 (0 から 255 までの数値) に変換します。続いて例に示してあるように、その値を 4 で割り、0 〜 63 スケールの値に変換します。値は、変数 `index` に格納されます。0 〜 63 スケールを使用するのは、このアプリケーションで使用する利用可能な ASCII 文字の " パレット " に格納される値の数が 64 のためです。文字パレットは、BitmapToAsciiConverter クラスの String インスタンスとして定義されています。

```
// 文字は最も濃いほうから最も薄いほうの順に並ぶ
// ストリングの位置 ( インデックス ) を相対カラー値に対応させるためである
// (0 = 黒 )
private static const palette:String =
   "@#$%&8BMW*mwqpdbkhaoQ0OZXYUJCLtfjzxnuvcr[]{}1()|/?Il!i><+_~-;,. ";
```

`index` 変数は、パレットのどの ASCII 文字がビットマップイメージの現在のピクセルに対応するかを定義するため、その文字は `charAt()` メソッドを使用して `palette` String から取得されます。続いて、連結代入演算子 (+=) を使用して、`result` String インスタンスに追加されます。さらに、各行のピクセルの最後で `result` String の最後に改行文字を連結して、文字 "ピクセル" の新しい行を作成するために行の折り返しを強制します。

第 8 章

配列の操作

8

配列を使用すると、1つのデータ構造の中に複数の値を格納できます。配列にはインデックス配列と結合配列の 2 種類があります。値の格納場所を指定する方法として、インデックス配列では単純な整数のインデックスを使用し、結合配列では任意のキーを使用します。また、配列の個々の要素そのものが配列であるような多次元の配列も作成できます。この章では、さまざまな種類の配列を作成および操作する方法について説明します。

目次

インデックス配列 227
結合配列 236
多次元配列 240
配列のクローンの作成 243
高度なテクニック 243
例 : PlayList 249

インデックス配列

インデックス配列では、格納した一連の値に、符号なし整数値によるインデックスを指定してアクセスします。配列内の先頭の要素は常にインデックス 0 で示され、それに続く個々の要素は順に 1 ずつ大きい数のインデックスで示されます。次のコードに示すように、インデックス配列を作成するには Array クラスのコンストラクタを呼び出すか、配列リテラルを使用して配列を初期化します。

```
// Array コンストラクタを使用する
var myArray:Array = new Array();
myArray.push("one");
myArray.push("two");
myArray.push("three");
trace(myArray); // 出力 : one,two,three

// Array リテラルを使用する
var myArray:Array = ["one", "two", "three"];
trace(myArray); // 出力 : one,two,three
```

また、Array クラスのプロパティとメソッドを使用すると、インデックス配列に変更を加えることができます。それらのプロパティとメソッドはほとんどがインデックス配列専用であり、結合配列には使用できません。

インデックス配列のインデックスに使用する数値は、符号なし 32 ビット整数です。インデックス配列の最大サイズは 2^{32}-1(4,294,967,295) です。この最大サイズより大きい配列を作成しようとするとランタイムエラーが発生します。

配列の要素には、あらゆるデータ型の値を格納できます。ActionScript 3.0 では " 型付き配列 " の概念がサポートされていないため、配列内のすべての要素が特定の同じデータ型を持つことを指定することはできません。

このセクションでは、Array クラスを使用してインデックス配列を作成および変更する方法について、まず配列の作成から順に説明します。配列に変更を加えるメソッドは、要素の挿入に関するもの、要素の削除に関するもの、配列のソートに関するものの 3 つのグループに大別されます。その他に、既存の配列を読み取り専用として扱うメソッドがあり、そのようなメソッドでは配列に対してクエリを実行するだけです。クエリ関連のメソッドは、既存の配列を変更することなく、新しく作成した配列を返します。このセクションの最後には、Array クラスの拡張に関する話題を取り上げます。

配列の作成

Array コンストラクタ関数には 3 つの使用方法があります。第 1 に、パラメータを指定せずに呼び出した場合、コンストラクタは空の配列を返します。Array クラスの `length` プロパティを使用すると、配列の要素が 1 つもないことを確認できます。たとえば、次のコードでは Array コンストラクタをパラメータなしで呼び出しています。

```
var names:Array = new Array();
trace(names.length); // 出力 : 0
```

第 2 に、1 個の数値パラメータだけを指定した場合、Array コンストラクタではそのパラメータで指定された長さの配列を作成し、各要素の値を `undefined` に設定します。指定するパラメータは、0 ～ 4,294,967,295 の値を持つ符号なし整数である必要があります。たとえば、次のコードでは Array コンストラクタに 1 個の数値パラメータを指定して呼び出します。

```
var names:Array = new Array(3);
trace(names.length); // 出力 : 3
trace(names[0]); // 出力 : undefined
trace(names[1]); // 出力 : undefined
trace(names[2]); // 出力 : undefined
```

第 3 に、パラメータとして要素リストを指定した場合、コンストラクタでは、各パラメータの値を個々の要素に設定した配列を作成します。次のコードでは、Array コンストラクタに対し 3 個のパラメータを指定しています。

```
var names:Array = new Array("John", "Jane", "David");
trace(names.length); // 出力 : 3
trace(names[0]); // 出力 : John
trace(names[1]); // 出力 : Jane
trace(names[2]); // 出力 : David
```

また、配列リテラルまたはオブジェクトリテラルから配列を作成することもできます。配列リテラルは、次の例に示すように配列変数に直接割り当てることができます。

```
var names:Array = ["John", "Jane", "David"];
```

配列要素の挿入

配列に要素を挿入する場合は、Array クラスにある push()、unshift()、splice() の 3 つのメソッドを使用できます。push() メソッドでは、配列の末尾に1つまたは複数の要素を追加します。したがって、push() メソッドで挿入した最後の要素には最も大きいインデックス番号が付きます。unshift() メソッドでは、配列の先頭に1つまたは複数の要素を挿入します。したがって、挿入した要素にはインデックス番号 0 が付きます。splice() メソッドでは、インデックスで指定した配列内の任意の場所に任意の数のアイテムを挿入します。

これら 3 つのメソッドについて使用例を次に示します。この例では、planets という名前の配列を作成し、惑星の名前を太陽から近い順に並ぶように格納します。まず push() メソッドを呼び出し、最初の要素として Mars を挿入します。次に、配列の先頭に位置する要素 Mercury を unshift() メソッドで挿入します。最後に splice() メソッドで、Venus および Earth を Mercury より後、Mars より前に挿入します。splice() の第1パラメータとして渡している整数1は、挿入先をインデックス1の場所にするという指定です。また、splice() 第 2 パラメータの整数 0 は、既存の要素を1つも削除しないという指定です。さらに、splice() の第 3 および第 4 パラメータで指定している Venus と Earth は挿入する要素のリストです。

```
var planets:Array = new Array();
planets.push("Mars");        // 配列の内容 : Mars
planets.unshift("Mercury"); // 配列の内容 : Mercury,Mars
planets.splice(1, 0, "Venus", "Earth");
trace(planets);              // 配列の内容 : Mercury,Venus,Earth,Mars
```

push() および unshift() メソッドはいずれも、変更後の配列の長さを示す符号なし整数値を返します。splice() メソッドは、要素の挿入に使用した場合、空の配列を返します。これは奇異な動作のようにも見えますが、splice() メソッドの幅広い用途を考えると理にかなっています。このメソッドは配列に要素を挿入する場合だけでなく、配列から要素を削除する場合にも使用でき、削除の場合は、元の配列から削除した要素を格納した配列を返します。

配列要素の削除

配列から要素を削除する場合は、Array クラスにある pop()、shift()、splice() の 3 つのメソッドを使用できます。pop() メソッドでは、配列の末尾にある 1 個の要素、つまり最も大きいインデックス番号の付いた要素を削除します。shift() メソッドは、配列の先頭から要素を削除します。つまり、常にインデックス番号 0 の要素が削除されます。splice() メソッドは、要素の挿入にも使用できますが、このメソッドに渡された最初のパラメータで指定されたインデックス番号から数えて、任意の数の要素を削除します。

これら 3 つのメソッドを使用して配列の要素を削除する例を次に示します。この例では、広い海の名前を格納するために oceans という名前の配列を作成していますが、海ではなく湖の名前がいくつか含まれているため、それらを削除します。

まず、splice() メソッドを使用して Aral および Superior の要素を削除し、Atlantic および Indian の要素を挿入します。splice() の第 1 パラメータとして渡している整数 2 は、3 番目の要素 (インデックス 2) から始まる範囲を操作対象にするという指定です。また、第 2 パラメータの整数 2 は、既存の要素を 2 つ削除するという指定です。残りのパラメータで指定している Atlantic と Indian はインデックス 2 の位置に挿入する要素のリストです。

次に、配列の末尾に位置する要素 Huron を pop() メソッドで削除します。最後に shift() メソッドで、配列の先頭に位置する要素 Victoria を削除します。

```
var oceans:Array = ["Victoria", "Pacific", "Aral", "Superior", "Indian",
  "Huron"];
oceans.splice(2, 2, "Arctic", "Atlantic"); // Aral と Superior を置き換える
oceans.pop();   // Huron を削除する
oceans.shift(); // Victoria を削除する
trace(oceans);  // 出力 : Pacific,Arctic,Atlantic,Indian
```

pop() および shift() メソッドはいずれも、削除した要素を返します。配列にはあらゆるデータ型の値を格納できるため、戻り値のデータ型は Object になっています。splice() は、削除した要素を格納した配列を返します。この海の例を次のように改変すると、splice() の呼び出しで返される戻り値を新しい配列変数に割り当てることができます。

```
var lakes:Array = oceans.splice(2, 2, "Arctic", "Atlantic");
trace(lakes); // 出力 : Aral,Superior
```

配列の要素に対して delete 演算子を使用しているコードもありますが、delete 演算子はその要素の値を undefined に設定するだけであり、配列から要素を削除するわけではありません。たとえば、次のコードでは oceans 配列の 3 番目の要素に対して delete 演算子を使用していますが、配列の長さはその後も変わらず 5 のままです。

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Indian", "Atlantic"];
delete oceans[2];
trace(oceans);        // 出力 : Arctic,Pacific,,Indian,Atlantic
trace(oceans[2]);     // 出力 : undefined
trace(oceans.length); // 出力 : 5
```

配列の長さを切り詰めるには、length プロパティを使用します。length プロパティをその配列の現状の長さより小さく設定すると、配列を切り詰めることになり、変更後の length の値から 1 を引いた値よりも大きいインデックス番号を持つ要素はすべて削除されます。たとえば、oceans 配列の要素を並べ替え、残したい要素をすべて配列の先頭側に集めたとすると、次のように length プロパティを使用することで末尾側の不要な要素を削除できます。

```
var oceans:Array = ["Arctic", "Pacific", "Victoria", "Aral", "Superior"];
oceans.length = 2;
trace(oceans); // 出力 : Arctic,Pacific
```

配列のソート

配列要素の並び順を変更する場合は、ソートまたは順序の逆転を行う reverse()、sort()、sortOn() の 3 つのメソッドを使用できます。いずれのメソッドでも既存の配列が変化します。reverse() メソッドでは、配列要素の並び順を逆転します。つまり、末尾にあった要素は先頭に、末尾の直前にあった要素は 2 番目に移動します。sort() メソッドでは、さまざまな定義済みの方法で配列要素をソートできる他、独自のソートアルゴリズムを作成して使用することもできます。sortOn() メソッドでは、1 つまたは複数の共通プロパティを持つオブジェクトをソートキーとして使用して、インデックス配列をソートできます。

reverse() メソッドは、パラメータも戻り値もありませんが、既存の配列について現在の状態と、その並び順を逆転した状態とを切り替えるのに使用できます。次の例では、oceans 配列に格納した海の名前を元とは逆の順序に並べ替えます。

```
var oceans:Array = ["Arctic", "Atlantic", "Indian", "Pacific"];
oceans.reverse();
trace(oceans); // 出力 : Pacific,Indian,Atlantic,Arctic
```

sort() メソッドは、デフォルトのソート順に基づいて配列要素を並べ替えます。このデフォルトのソート順とは、次の規則に従うものです。

- 大文字と小文字は区別され、大文字のほうが小文字より先に並びます。たとえば、D は b よりも先に並びます。
- ソート方向は正順です。つまり、文字コードの値の小さい文字 (例 : A) のほうが、文字コードの値の大きい文字 (例 : B) よりも先に並びます。
- 値が同等と見なされる要素は互いに隣接して並びますが、それらの間における並び順は決まっていません。
- ソートはストリングベースで実行されます。つまり、要素は文字列に変換してから比較されます。たとえば、数値の 10 は 3 よりも先に並びます。これは、ストリングの "1" が持つ文字コードのほうがストリングの "3" が持つ文字コードよりも小さいためです。

大文字と小文字を区別しないソート、逆順のソート、またはアルファベット順でなく数値の大きさ順によるソートなどが必要な場合のために、sort() メソッドには、デフォルトのソート順に定められた各種の規則を変更する options パラメータがあります。それらのオプションは、次に示す Array クラスの静的定数セットによって定義されます。

- Array.CASEINSENSITIVE: ソートで大文字と小文字を区別しないようにします。これにより、たとえば小文字の b が大文字の D よりも先に並ぶようになります。
- Array.DESCENDING: デフォルトの昇順ではなく降順でソートします。たとえば、B が A よりも先に並ぶようになります。
- Array.UNIQUESORT: 同じ値を持つ要素が 2 つ見つかった場合にソートを中止するようにします。
- Array.NUMERIC: 数値順でソートします。たとえば、3 が 10 よりも先に並ぶようにします。

これらのオプションの一部を使用した例を次に示します。poets という名前の配列を作成し、いろいろなオプションを使用してソートを実行しています。

```
var poets:Array = ["Blake", "cummings", "Angelou", "Dante"];
poets.sort(); // デフォルトのソート
trace(poets); // 出力 : Angelou,Blake,Dante,cummings

poets.sort(Array.CASEINSENSITIVE);
trace(poets); // 出力 : Angelou,Blake,cummings,Dante

poets.sort(Array.DESCENDING);
trace(poets); // 出力 : cummings,Dante,Blake,Angelou

poets.sort(Array.DESCENDING | Array.CASEINSENSITIVE); // 2 つのオプションを使用する
trace(poets); // 出力 : Dante,cummings,Blake,Angelou
```

また、独自のソート関数を作成して sort() メソッドのパラメータに指定することもできます。たとえば、各人のフルネームを要素として格納した人名リストがあるとします。このリストを姓を基準にしてソートする場合は、各要素を解析して姓の部分を比較するソート関数を独自に作成する必要があります。これを実現するコードの例を次に示します。独自の関数を使用することを Array.sort() メソッドのパラメータで指定しています。

```
var names:Array = new Array("John Q. Smith", "Jane Doe", "Mike Jones");
function orderLastName(a, b):int
{
  var lastName:RegExp = /\b\S+$/;
  var name1 = a.match(lastName);
  var name2 = b.match(lastName);
  if (name1 < name2)
  {
    return -1;
  }
  else if (name1 > name2)
```

```
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

```
trace(names); // 出力 : John Q. Smith,Jane Doe,Mike Jones
names.sort(orderLastName);
trace(names); // 出力 : Jane Doe,Mike Jones,John Q. Smith
```

独自のソート関数 orderLastName() では、正規表現を使用して各要素から姓の部分を抽出し、それを比較演算に使用します。names 配列に対する sort() メソッド呼び出しのパラメータとして、この関数の識別子である orderLastName だけを指定しています。ソート関数に a および b の 2 つのパラメータが渡されるのは、一度に 2 つの配列要素が処理の対象となるからです。ソート関数の戻り値は 2 つの要素のソート順を示すものであり、次の意味があります。

- 戻り値が -1 の場合、第 1 パラメータの a を第 2 パラメータの b よりも先に並べることを示します。
- 戻り値が 1 の場合、第 2 パラメータの b を第 1 パラメータの a よりも先に並べることを示します。
- 戻り値が 0 の場合、2 つの要素がソートにおいて同じ順位を持つことを示します。

sortOn() メソッドは、要素がオブジェクトであるインデックス配列のためにあります。すべての要素のオブジェクトには、ソート用キーとなる共通のプロパティが少なくとも 1 つ含まれている必要があります。これらの条件に合わない配列に対して sortOn() メソッドを使用した場合の結果は予測不能です。

次の例は、poets 配列をストリング要素ではなくオブジェクト要素の配列に改変したものです。各オブジェクトには、詩人の姓と生誕年が格納されています。

```
var poets:Array = new Array();
poets.push({name:"Angelou", born:"1928"});
poets.push({name:"Blake", born:"1757"});
poets.push({name:"cummings", born:"1894"});
poets.push({name:"Dante", born:"1265"});
poets.push({name:"Wang", born:"701"});
```

この配列では、born プロパティを基準として sortOn() メソッドによるソートを実行できます。sortOn() メソッドには、fieldName および options の 2 つのパラメータがあります。fieldName にはストリングを指定する必要があります。次の例では、sortOn() にパラメータとして "born" および Array.NUMERIC の 2 つを渡しています。Array.NUMERIC オプションを指定しているのは、アルファベット順ではなく数値順で確実にソートを実行するためです。たとえ各要素に格納されている数値の桁数がすべて同じであっても、必ずこのオプションを指定するようにしておけば、後で桁数の異なる要素が追加されたとしても正常に動作します。

```
poets.sortOn("born", Array.NUMERIC);
for (var i:int = 0; i < poets.length; ++i)
```

```
{
   trace(poets[i].name, poets[i].born);
}
/* 出力 :
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

一般に、sort() および sortOn() メソッドを使用すると元の配列が変化します。既存の配列を変更せずにソートを実行する場合は、options パラメータに Array.RETURNINDEXEDARRAY 定数を含めます。このオプションを指定した場合、メソッドの戻り値はソート結果を示す新しい配列となり、元の配列の内容はそのまま維持されます。返される配列はソート後の並び順を示すインデックス番号だけを格納した単純なものであり、元の配列内の要素を格納した配列ではありません。たとえば、poets 配列について元の状態を維持したまま生誕年でのソートを実行するには、options パラメータに Array.RETURNINDEXEDARRAY 定数を含めます。

次の例では、戻り値のインデックス情報を indices という配列に格納し、この indices 配列と元のままの poets 配列を使用して詩人のリストを生誕年順に表示します。

```
var indices:Array;
indices = poets.sortOn("born", Array.NUMERIC | Array.RETURNINDEXEDARRAY);
for (var i:int = 0; i < indices.length; ++i)
{
   var index:int = indices[i];
   trace(poets[index].name, poets[index].born);
}
/* 出力 :
Wang 701
Dante 1265
Blake 1757
cummings 1894
Angelou 1928
*/
```

配列のクエリ

以上で説明した他に、Array クラスには concat()、join()、slice()、toString() の 4 つのメソッドがあります。これらはいずれもクエリによって配列から情報を取得するものであり、配列に変更を加えることはありません。concat() および slice() メソッドは新しい配列を返し、join() および toString() メソッドはストリングを返します。concat() メソッドのパラメータには配列または要素リストを指定します。既存の配列と、パラメータで指定した別の配列または要素とを連結し、新しい配列を作成して返します。slice() メソッドには、startIndex および endIndex という 2 つのパラメータがあります。元の配列から、これらのパラメータで指定したインデックスの範囲を切り出し、その部分のコピーを格納した新しい配列を返します。ただし、返される範囲は startIndex が指す要素から、endIndex が指す要素の直前までであり、戻り値に endIndex の位置にある要素自体は含まれません。

次の例では、既存の配列から concat() および slice() を使用して新しい配列を作成します。

```
var array1:Array = ["alpha", "beta"];
var array2:Array = array1.concat("gamma", "delta");
trace(array2); // 出力 : alpha,beta,gamma,delta

var array3:Array = array1.concat(array2);
trace(array3); // 出力 : alpha,beta,alpha,beta,gamma,delta

var array4:Array = array3.slice(2,5);
trace(array4); // 出力 : alpha,beta,gamma
```

join() および toString() メソッドでは、配列の内容をストリングとして取得できます。join() メソッドのパラメータを指定しない場合、これら 2 つのメソッドはいずれも、すべての配列要素をカンマで区切った内容のストリングを返します。join() メソッドには、toString() メソッドにない delimiter というパラメータがあり、戻り値のストリングで要素間に使用する区切り文字を指定できます。

次の例では rivers という配列を作成し、join() および toString() を呼び出すことで、配列の内容をストリングとして取得します。toString() メソッドはカンマ区切りの値 (riverCSV) を得るため、また、join() メソッドは + で区切った値を得るために使用しています。

```
var rivers:Array = ["Nile", "Amazon", "Yangtze", "Mississippi"];
var riverCSV:String = rivers.toString();
trace(riverCSV); // 出力 : Nile,Amazon,Yangtze,Mississippi
var riverPSV:String = rivers.join("+");
trace(riverPSV); // 出力 : Nile+Amazon+Yangtze+Mississippi
```

ネストされた配列に対して join() メソッドを使用する場合は注意が必要です。区切り文字の指定はメインの配列要素の間にしか適用されず、次の例に示されるように、下位の要素の間はすべてカンマで区切られます。

```
var nested:Array = ["b","c","d"];
var letters:Array = ["a",nested,"e"];
var joined:String = letters.join("+");
trace(joined); // 出力 : a+b,c,d+e
```

結合配列

結合配列は " ハッシュ または " マップ " とも呼ばれ、値の格納場所を数値のインデックスではなく " キー " によって指定する配列です。結合配列に使用する個々のキーは、それぞれに対応する特定の格納値にアクセスするための一意のストリングです。結合配列は Object クラスのインスタンスであり、したがって、各キーはそれぞれ 1 つのプロパティ名に対応します。結合配列は、キー / 値ペアの集合からなる、ソートされていないコレクションです。結合配列内のキーに何らかの決まった順序があるかのような前提でコードを記述してはなりません。

ActionScript 3.0 では、新たに " ディクショナリ " と呼ばれる高機能な結合配列を使用できるようになりました。ディクショナリは flash.utils パッケージに属する Dictionary クラスのインスタンスであり、任意のデータ型 (通常は Object クラスのインスタンス) をキーとして使用できます。つまり、ディクショナリのキーは String 型の値である必要はありません。

このセクションでは、ストリングのキーを使用した結合配列の作成方法、および Dictionary クラスの使用方法について説明します。

ストリングのキーを使用した結合配列

ActionScript 3.0 の結合配列を作成するには 2 つの方法があります。1 つは Object コンストラクタを使用する方法で、これには、オブジェクトリテラルを使用して配列を初期化できるというメリットがあります。Object クラスのインスタンス (汎用オブジェクトとも呼ばれる) は、機能的には結合配列と同じです。汎用オブジェクトの個々のプロパティ名が、格納した個々の値にアクセスするためのキーとして機能します。

次の例では、monitorInfo という結合配列を作成し、2 つのキー / 値ペアをオブジェクトリテラルで指定することにより配列を初期化します。

```
var monitorInfo:Object = {type:"Flat Panel", resolution:"1600 x 1200"};
trace (monitorInfo["type"], monitorInfo["resolution"]);
// 出力 : Flat Panel 1600 x 1200
```

宣言時に配列を初期化する必要がある場合は、次のように、Object コンストラクタを使って配列を作成できます。

```
var monitorInfo:Object = new Object();
```

オブジェクトリテラルまたは Object クラスのコンストラクタで配列を作成した後は、角括弧演算子 ([]) またはドット演算子 (.) を使用して新しい値を追加できます。次の例では、monitorArray に新しい値を 2 つ追加します。

```
monitorInfo["aspect ratio"] = "16:10"; // 誤った形式。空白は使用しないこと
monitorInfo.colors = "16.7 million";
trace (monitorInfo["aspect ratio"], monitorInfo.colors);
// 出力 : 16:10 16.7 million
```

この例で、aspect ratio というキーの名前には空白文字が含まれています。角括弧演算子ではこのような名前も使用できますが、ドット演算子で使用するとエラーになります。空白文字を含んだキー名を使用することはお勧めしません。

結合配列を作成するもう 1 つの方法は、Array コンストラクタを使用し、それから角括弧演算子 ([]) またはドット演算子 (.) で配列にキー / 値ペアを追加する方法です。結合配列を Array 型として宣言した場合、オブジェクトリテラルで配列を初期化することはできません。次の例では、monitorInfo という名前の結合配列を Array コンストラクタで作成し、type および resolution という 2 つのキーとそれぞれに対応する値を追加します。

```
var monitorInfo:Array = new Array();
monitorInfo["type"] = "Flat Panel";
monitorInfo["resolution"] = "1600 x 1200";
trace(monitorInfo["type"], monitorInfo["resolution"]);
// 出力 : Flat Panel 1600 x 1200
```

結合配列の作成に Array コンストラクタを使用するメリットはありません。たとえ Array コンストラクタや Array データ型を使用して宣言した結合配列であっても、Array.length プロパティや Array クラスのメソッドはまったく使用できないからです。Array コンストラクタはインデックス配列を作成する場合に限って使用するようお勧めします。

オブジェクトのキーを使用した結合配列

Dictionary クラスを使用すると、ストリングではなくオブジェクトをキーとする結合配列を作成できます。そのような配列は、ディクショナリ、ハッシュ、またはマップと呼ばれることがあります。たとえば、特定のコンテナとの位置関係によって Sprite オブジェクトの位置が決まるようなアプリケーションでは、個々の Sprite オブジェクトとコンテナとの対応付けを保持しておくために Dictionary オブジェクトを使用できます。

次のコードでは、Dictionary オブジェクトのキーとして使用する Sprite クラスのインスタンスを 3 つ作成します。各キーに対しては、GroupA または GroupB いずれかの値を格納します。格納する値には任意のデータ型を使用できますが、この例では GroupA、GroupB のいずれも Object クラスのインスタンスです。それから、プロパティアクセス演算子 ([]) を使用して、各キーに対応する値にアクセスします。次にこのコードを示します。

```
import flash.display.Sprite;
import flash.utils.Dictionary;

var groupMap:Dictionary = new Dictionary();

// キーとして使用するオブジェクト
var spr1:Sprite = new Sprite();
var spr2:Sprite = new Sprite();
var spr3:Sprite = new Sprite();

// 値として使用するオブジェクト
var groupA:Object = new Object();
var object:Object = new Object();

// 新しいキー / 値ペアをディクショナリ内に作成する
groupMap[spr1] = groupA;
groupMap[spr2] = groupB;
groupMap[spr3] = groupB;

if (groupMap[spr1] == groupA)
{
  trace("spr1 is in groupA");
}
if (groupMap[spr2] == groupB)
{
  trace("spr2 is in groupB");
}
if (groupMap[spr3] == groupB)
{
  trace("spr3 is in groupB");
}
```

オブジェクトのキーを使用した反復処理

Dictionary オブジェクトの内容に対して反復処理を実行する場合は、for..in ループまたは for each..in ループを使用できます。for..in ループはキーに基づいた反復処理に使用し、for each..in ループは各キーに対応する値に基づいた反復処理に使用します。

for..in ループの場合、Dictionary オブジェクト内のオブジェクトキーに直接アクセスできます。また、プロパティアクセス演算子 ([]) を使用すれば Dictionary オブジェクト内の値にアクセスできます。次のコードでは、前の例と同じ groupMap ディクショナリを使用して、for..in ループで Dictionary オブジェクトの内容を反復処理する方法を示します。

```
for (var key:Object in groupMap)
{
    trace(key, groupMap[key]);
}
/* 出力 :
[object Sprite] [object Object]
[object Sprite] [object Object]
[object Sprite] [object Object]
*/
```

for each..in ループの場合、Dictionary オブジェクト内の値に直接アクセスできます。次のコードでもやはり groupMap ディクショナリを使用して、for each..in ループで Dictionary オブジェクトの内容を反復処理する方法を示します。

```
for each (var item:Object in groupMap)
{
    trace(item);
}
/* 出力 :
[object Object]
[object Object]
[object Object]
*/
```

オブジェクトキーとメモリ管理

Flash Player では、使用されなくなったメモリはガベージコレクションのシステムによって開放されます。オブジェクトはどこからも参照されなくなった時点でガベージコレクションの対象とされ、次回ガベージコレクションが実行されるとき、そのオブジェクトに使用されていたメモリが開放されます。たとえば、次のコードでは新しいオブジェクトを作成し、そのオブジェクトへの参照を myObject という変数に割り当てます。

```
var myObject:Object = new Object();
```

オブジェクトに対して何らかの参照が存在する間は、そのオブジェクトのメモリがガベージコレクションのシステムによって開放されることはありません。myObject の値が別のオブジェクトを参照するように変更されるか、または null に設定されると、それまで参照先となっていたオブジェクトはガベージコレクションの対象とされます (そのオブジェクトに対する参照が他に存在しない場合)。

この myObject を Dictionary オブジェクトのキーとして使用すると、参照先オブジェクトに対する参照がもう 1 つ作成されます。たとえば、次のコードでは 1 つのオブジェクトに対して 2 つの参照、つまり myObject 変数と、myMap オブジェクトのキーを作成しています。

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary();
myMap[myObject] = "foo";
```

myObject で参照しているオブジェクトをガベージコレクションの対象とするには、そのオブジェクトに対する参照をすべて削除する必要があります。この場合、そのためには myObject の値を変更するのに加え、myMap から myObject のキーを削除します。

```
myObject = null;
delete myMap[myObject];
```

別の方法としては、Dictionary コンストラクタに対して useWeakReference パラメータを指定し、すべてのディクショナリキーが "弱参照" となるようにすることもできます。弱参照はガベージコレクションのシステムで無視されるため、弱参照による参照しか受けていないオブジェクトはガベージコレクションの対象となります。たとえば、次のコードでは myMap 内の myObject キーを削除しなくても、該当するオブジェクトをガベージコレクションの対象とすることができます。

```
import flash.utils.Dictionary;

var myObject:Object = new Object();
var myMap:Dictionary = new Dictionary(true);
myMap[myObject] = "foo";
myObject = null; // オブジェクトをガベージコレクションの対象とする
```

多次元配列

多次元配列には、配列の要素としてさらに配列が格納されています。たとえば、予定リストをストリングのインデックス配列として格納する場合を考えます。

```
var tasks:Array = ["wash dishes", "take out trash"];
```

曜日ごとにタスクのリストを個別に格納する場合は、各曜日を要素として持つ多次元配列を作成します。それぞれの要素として、予定リストを表す tasks 配列と同様のインデックス配列を格納します。多次元配列内では、インデックス配列と結合配列を任意に組み合わせて使用できます。以降のセクションでは、2 つのインデックス配列を使用する例と、結合配列およびインデックス配列を 1 つずつ使用する例を示します。また、実習として、考えられる残りの組み合わせについても作成してみてください。

2 つのインデックス配列を使用する例

インデックス配列を 2 つ使用した場合の結果は、テーブルまたはスプレッドシートのような図で表現できます。第1の配列に含まれる要素がテーブルの行を表し、第 2 の配列に含まれる要素が列を表します。

たとえば、次の多次元配列には、曜日ごとの予定リストを 2 つのインデックス配列として格納しています。第1の配列 masterTaskList は、Array クラスのコンストラクタを使用して作成します。この配列の各要素は曜日に対応し、インデックス 0 が月曜、インデックス 6 が日曜を表します。それらの要素は、テーブルの行と見なすことができます。各曜日の予定リストは、masterTaskList 配列内に作成する 7 つの要素それぞれに配列リテラルを割り当てることで作成できます。これらの配列リテラルは、テーブルの列を表します。

```
var masterTaskList:Array = new Array();
masterTaskList[0] = ["wash dishes", "take out trash"];
masterTaskList[1] = ["wash dishes", "pay bills"];
masterTaskList[2] = ["wash dishes", "dentist", "wash dog"];
masterTaskList[3] = ["wash dishes"];
masterTaskList[4] = ["wash dishes", "clean house"];
masterTaskList[5] = ["wash dishes", "wash car", "pay rent"];
masterTaskList[6] = ["mow lawn", "fix chair"];
```

いずれかの予定リストに含まれる個別の予定項目にアクセスするには、角括弧表記法を使用します。最初の角括弧で曜日を指定し、2 番目の角括弧でその日の予定リスト内の項目を指定します。たとえば、水曜日の予定リストから 2 番目の予定を取得するには、水曜日を表すインデックス 2 と、予定リストの 2 番目の項目を表すインデックス 1 を指定します。

```
trace(masterTaskList[2][1]); // 出力 : dentist
```

日曜日の予定リストから 1 番目の予定を取得するには、日曜日を表すインデックス 6 と、予定リストの 1 番目の項目を表すインデックス 0 を指定します。

```
trace(masterTaskList[6][0]); // 出力 : mow lawn
```

結合配列の下位にインデックス配列を組み合わせる例

曜日について結合配列を使用し、予定リストについてインデックス配列を使用すると、個別の配列に対するアクセスが容易になります。結合配列では、特定の曜日を参照するときにドットシンタックスを使用することができますが、結合配列の各要素にアクセスするために、実行時の処理時間が余分に犠牲になります。次の例では、曜日ごとにキー / 値ペアを指定し、予定リストの基礎として結合配列を使用しています。

```
var masterTaskList:Object = new Object();
masterTaskList["Monday"] = ["wash dishes", "take out trash"];
masterTaskList["Tuesday"] = ["wash dishes", "pay bills"];
masterTaskList["Wednesday"] = ["wash dishes", "dentist", "wash dog"];
masterTaskList["Thursday"] = ["wash dishes"];
masterTaskList["Friday"] = ["wash dishes", "clean house"];
masterTaskList["Saturday"] = ["wash dishes", "wash car", "pay rent"];
masterTaskList["Sunday"] = ["mow lawn", "fix chair"];
```

ドットシンタックスを使用すると、角括弧の繰り返しが避けられ、コードの読みやすさが向上します。

```
trace(masterTaskList.Wednesday[1]); // 出力 : dentist
trace(masterTaskList.Sunday[0]);    // 出力 : mow lawn
```

for..in ループを使用して予定リストの反復処理を実行できますが、各キーに対応する値にアクセスするために、ドットシンタックスではなく角括弧表記法を使用する必要があります。masterTaskList は結合配列であるため、次の例に示すように、必ずしも期待する順序で要素を取得する必要はありません。

```
for (var day:String in masterTaskList)
{
  trace(day + ": " + masterTaskList[day])
}
/* 出力 :
Sunday: mow lawn,fix chair
Wednesday: wash dishes,dentist,wash dog
Friday: wash dishes,clean house
Thursday: wash dishes
Monday: wash dishes,take out trash
Saturday: wash dishes,wash car,pay rent
Tuesday: wash dishes,pay bills
*/
```

配列のクローンの作成

Array クラスには、配列のコピーを作成する組み込みメソッドは用意されていません。配列の " 浅いコピー " は、concat() メソッドまたは slice() メソッドをパラメータなしで呼び出すことで作成できます。浅いコピーでは、オブジェクトである要素が元の配列に格納されている場合、コピーした配列内の要素は元の要素に格納されているオブジェクトへの参照となり、参照先オブジェクトはコピーされません。したがって、元の配列とコピーした配列が同一のオブジェクトを参照することになり、一方からオブジェクトに変更を加えると、もう一方にもその変更が反映されます。

" 深いコピー " では、元の配列内で参照されているオブジェクトもすべてコピーされるので、元の配列とコピーした配列が同一のオブジェクトを参照する結果にはなりません。深いコピーを実行するには複数行のコードを記述する必要があり、通常、何らかの関数を作成することになります。この用途の関数は、グローバルなユーティリティ関数として実装するか、または Array のサブクラスにメソッドとして実装することが考えられます。

次の例では、深いコピーを実行するための clone() という関数を定義しています。採用したアルゴリズムは、Java プログラミングで一般的に使用されているテクニックです。まず元の配列を直列化して ByteArray クラスのインスタンスを作成し、それを新しい配列に読み込むことでコピーを実行しています。コードに示されているとおり、この関数はオブジェクトを受け付けるため、インデックス配列と結合配列の両方に対応できます。

```
import flash.utils.ByteArray;

function clone(source:Object):*
{
   var myBA:ByteArray = new ByteArray();
   myBA.writeObject(source);
   myBA.position = 0;
   return(myBA.readObject());
}
```

高度なテクニック

Array クラスの拡張

Array クラスはコアクラスとしては珍しく final 指定されていないため、Array のサブクラスを独自に定義できます。このセクションでは、Array のサブクラスを作成する方法について例を示し、その際に発生すると考えられる問題について説明します。

前述したとおり ActionScript の配列には型がありませんが、Array のサブクラスを定義すれば、特定のデータ型の要素だけを格納する配列を作成できるようになります。以下のセクションの例では、Array のサブクラスとして、第 1 パラメータに指定されているデータ型の要素だけを格納する TypedArray というクラスを定義します。TypedArray クラスは、Array クラスの拡張方法の例として示したにすぎず、いくつかの理由で運用目的には適さない場合があります。第 1 に、コンパイル時ではなく、実行時に型チェックが行われます。第 2 に、メソッドは簡単な変更によって例外をスローする可能性がありますが、TypedArray メソッドで不一致が見つかった場合に、その不一致は無視されて例外がスローされます。第 3 に、このクラスでは、配列に任意の型の要素を挿入する配列アクセス演算子の使用を防止できません。第 4 に、コーディングスタイルでは、パフォーマンスの最適化よりも単純さのほうが優先されます。

サブクラスの宣言

定義するクラスが Array のサブクラスであることを示すには、`extends` キーワードを使用します。Array のサブクラスでは、Array と同様に `dynamic` 属性を使用します。これに従わないとサブクラスが正常に機能しません。

次のコードに示す TypedArray クラスの定義には、データ型を保持する定数、コンストラクタメソッド、配列要素の追加に使用できる 4 つのメソッドがあります。ここでは各メソッドのコードを省略していますが、それらの詳細については以降のセクションで説明していきます。

```
public dynamic class TypedArray extends Array
{
   private const dataType:Class;

   public function TypedArray(...args) {}

   AS3 override function concat(...args):Array {}

   AS3 override function push(...args):uint {}

   AS3 override function splice(...args) {}

   AS3 override function unshift(...args):uint {}
}
```

4 つのオーバーライドしたメソッドはすべてが AS3 名前空間を使用し、public 属性は使用していません。この例では、コンパイラオプション -as3 の設定は true、コンパイラオプション -es の設定は false と見なしているからです。これらは、Adobe Flex Builder 2 のデフォルト設定であり、Adobe Flash CS3 Professional。詳細については、147 ページの「AS3 名前空間」を参照してください。

ヒント	上級開発者でプロトタイプ継承の使用を優先したい場合は、TypedArray クラスに 2 つのマイナー変更を加え、コンパイラオプション -es を true に設定してコンパイルします。最初に、すべての出現箇所の override 属性を削除し、AS3 名前空間を public 属性で置換します。次に、出現するすべての 4 つの super の代わりに Array.prototype を使用します。

TypedArray コンストラクタ

このコンストラクタを定義するにあたっては興味深い問題が 1 つあります。それは、任意の長さを持つリストをパラメータとして受け付け、さらに、配列を作成するためにパラメータをスーパーコンストラクタに引き渡す必要があるということです。パラメータのリストを配列として渡せば、スーパーコンストラクタには Array 型のパラメータ 1 個として扱われてしまい、結果として配列の要素数は常に 1 になります。パラメータリストをそのまま引き渡す場合の伝統的な処理方法として Function.apply() メソッドがあります。パラメータ配列をこのメソッドの第 2 パラメータに指定すると、その要素を個々のパラメータに展開して関数を実行させることができます。しかし、Function.apply() メソッドはコンストラクタ内では使用できません。

残る唯一の選択肢は、TypedArray コンストラクタ内に Array コンストラクタと同様のロジックを再現することです。Array クラスのコンストラクタでは次のコードに示すアルゴリズムが使用されているので、これを Array のサブクラスのコンストラクタに流用することにします。

```
public dynamic class Array
{
   public function Array(...args)
   {
      var n:uint = args.length
      if (n == 1 && (args[0] is Number))
      {
         var dlen:Number = args[0];
         var ulen:uint = dlen;
         if (ulen != dlen)
         {
            throw new RangeError("Array index is not a 32-bit unsigned integer
   ("+dlen+")");
         }
         length = ulen;
      }
      else
      {
         length = n;
         for (var i:int=0; i < n; i++)
```

```
            {
               this[i] = args[i]
            }
         }
      }
   }
```

TypedArray コンストラクタはほとんどが Array コンストラクタから流用したコードであり、4 つの変更点がコードにあるだけです。1つ目は、パラメータリストに Class 型の新しい必須パラメータを含め、配列のデータ型を指定できるようにしたことです。2 つ目は、コンストラクタに渡されるデータ型を dataType 変数に割り当てたことです。3 つ目は、else ステートメントにおいて、length プロパティの値は for ループの実行後に設定し、適切な型のパラメータだけが length に含まれるようにしたことです。4 つ目は、オーバーライドした push() メソッドを for ループの本体の中で使用し、適切なデータ型のパラメータだけを配列に追加するようにしたことです。TypedArray コンストラクタ関数を次に示します。

```
public dynamic class TypedArray extends Array
{
   private var dataType:Class;
   public function TypedArray(typeParam:Class, ...args)
   {
      dataType = typeParam;
      var n:uint = args.length
      if (n == 1 && (args[0] is Number))
      {
         var dlen:Number = args[0];
         var ulen:uint = dlen
         if (ulen != dlen)
         {
            throw new RangeError("Array index is not a 32-bit unsigned integer
   ("+dlen+")")
         }
         length = ulen;
      }
      else
      {
         for (var i:int=0; i < n; i++)
         {
            // 型チェックは push() により実行される
            this.push(args[i])
         }
         length = this.length;
      }
   }
}
```

TypedArray でオーバーライドしたメソッド

TypedArray クラスでは、Array クラスのメソッドのうち、配列要素の追加に使用できる 4 つのメソッドをオーバーライドしています。いずれのメソッドでも、適切なデータ型以外の要素が追加されるのを防ぐために型チェックを実行してから、スーパークラスにある同じメソッドを呼び出します。

push() メソッドでは、パラメータリストに含まれる個々の要素の型を for..in ループで順にチェックします。不適切な型のパラメータがあれば、splice() メソッドで args 配列から削除します。この for..in ループを終了した後は、args には dataType 型の要素しか含まれていないことになります。それから、次のようにスーパークラスの push() に改変後の args 配列を渡して呼び出します。

```
AS3 override function push(...args):uint
{
   for (var i:* in args)
   {
      if (!(args[i] is dataType))
      {
         args.splice(i,1);
      }
   }
   return (super.push.apply(this, args));
}
```

concat() メソッドでは、型チェックに適合したパラメータを格納するために、passArgs という名前で一時的な TypedArray を作成します。これは、push() メソッド内の型チェックのコードを再利用するためです。for..in ループで、args 配列内の個々の要素について push() を呼び出します。passArgs の型は TypedArray なので、実行される push() メソッドは TypedArray クラスのものです。それから、次のようにスーパークラスの concat() メソッドを呼び出します。

```
AS3 override function concat(...args):Array
{
   var passArgs:TypedArray = new TypedArray(dataType);
   for (var i:* in args)
   {
      // 型チェックは push() により実行される
      passArgs.push(args[i]);
   }
   return (super.concat.apply(this, passArgs));
}
```

splice() メソッドには任意の数のパラメータを指定できますが、最初の 2 つは必ず、インデックス番号と削除する要素の個数を表しています。このため、オーバーライドした splice() メソッドでは、args のインデックス番号が 2 以降の配列要素に対してのみ型チェックを実行します。このコードでは、for ループ内で splice() を再帰的に呼び出しているように見えますが、実際には再帰ではありません。args の型は TypedArray ではなく Array なので、args.splice() の呼び出しで実行されるのはスーパークラスのメソッドです。この for..in ループを終了した後は、args にはインデックス番号が 2 以上の適切な型の要素しか含まれていません。それから、次のようにスーパークラスの splice() を呼び出します。

```
AS3 override function splice(...args):*
{
    if (args.length > 2)
    {
        for (var i:int=2; i< args.length; i++)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
    }
    return (super.splice.apply(this, args));
}
```

配列の先頭に要素を挿入する unshift() も、任意のパラメータリストを受け付けるメソッドです。オーバーライドした unshift() メソッドでは、次のコードに示すとおり、push() メソッドと非常によく似たアルゴリズムを使用しています。

```
    AS3 override function unshift(...args):uint
    {
        for (var i:* in args)
        {
            if (!(args[i] is dataType))
            {
                args.splice(i,1);
            }
        }
        return (super.unshift.apply(this, args));
    }
}
```

例 : PlayList

ここでは PlayList の例を使って、配列を操作するテクニックについて、曲のリストを管理する音楽プレイリストアプリケーションのコンテキストで示します。扱うテクニックは次のとおりです。

- インデックス配列の作成
- インデックス配列への項目の追加
- 各種ソートオプションを使用したオブジェクト配列のプロパティ別ソート
- 文字で区切られたストリングへの配列の変換

PlayList アプリケーションのファイルは、"Samples/PlayList" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
PlayListApp.mxml	MXML で記述された Flex 用メインアプリケーションファイル。
com/example/programmingas3/playlist/PlayList.as	曲の追加やソートなど、プレイリスト機能をアプリケーションに提供します。
com/example/programmingas3/playlist/Song.as	単一の曲に関する情報を表す値オブジェクト。PlayList クラスが管理する項目は Song インスタンスです。
com/example/programmingas3/playlist/SortProperty.as	利用可能な値が Song クラスのプロパティを表し、このプロパティで Song オブジェクトのリストをソート可能な擬似列挙。

PlayList クラスの概要

PlayList クラスは Song オブジェクトセットを管理するクラスで、曲をプレイリストに追加するための addSong() メソッドや、リストの曲をソートするための sortList() メソッドなど、さまざまな機能のパブリックメソッドを備えています。このクラスには、読み取り専用アクセサプロパティの songList もあり、これによってプレイリストの実際の曲セットにアクセスできます。内部的には、PlayList クラスは、Array プライベート変数を使用して曲を追跡し続けます。

```
public class PlayList
{
    private var _songs:Array;
    private var _currentSort:SortProperty = null;
    private var _needToSort:Boolean = false;
    ...
}
```

PlayList クラスが曲のリストを追跡し続けるために使用する _songs という Array 変数の他に、2 つのプライベート変数 (_needToSort と _currentSort) があり、前者はリストのソートが必要であるかどうかを、後者は指定の時刻にどのプロパティで曲のリストをソートするかを追跡し続けます。

すべてのオブジェクトの場合と同様に、Array インスタンスを宣言するだけでは Array の作成作業は完了しません。Array インスタンスのプロパティまたはメソッドにアクセスする前に、このインスタンスを PlayList クラスのコンストラクタでインスタンス化しておく必要があります。

```
public function PlayList()
{
    this._songs = new Array();
    // 最初のソートを設定する
    this.sortList(SortProperty.TITLE);
}
```

コンストラクタの 1 行目は、_songs 変数をインスタンス化して使用できる状態にします。さらに、sortList() メソッドを呼び出し、最初のソート基準プロパティを設定します。

リストへの曲の追加

ユーザーが新しい曲をアプリケーションに登録すると、データ入力フォームのコードが PlayList クラスの addSong() メソッドを呼び出します。

```
/**
 * 曲をプレイリストに追加する
 */
public function addSong(song:Song):void
{
    this._songs.push(song);
    this._needToSort = true;
}
```

addSong() 内では _songs 配列の push() メソッドを呼び出し、その配列の新しい要素として addSong() に渡された Song オブジェクトを追加します。push() メソッドでは、以前に適用していたソートとは関係なく、新しい要素を配列の末尾に追加します。つまり、push() メソッドを呼び出した後では、曲のリストはソート順が正しくなっていない可能性が高いため、_needToSort 変数は true に設定されます。理論上は、sortList() メソッドをすぐに呼び出すため、指定の時刻にリストをソートするかどうかを追跡し続けることは不要になります。しかし、実際には、取得する直前まで曲のリストをソートする必要はありません。たとえば、取得前に何曲かリストに追加したときなど、ソート操作を遅らせると、アプリケーションは不要なソートを実行しなくなります。

曲のリストのソート

プレイリストが管理する Song インスタンスは複雑なオブジェクトであるため、そのアプリケーションのユーザーは、曲のタイトルまたは発表の年など、さまざまなプロパティに従ってプレイリストをソートしたいと考えることがあります。PlayList アプリケーションでは、曲のリストをソートするタスクは、次の 3 つの部分で構成されています。つまり、リストのソート基準としてのプロパティを見分け、そのプロパティによるソート時に必要なソートオプションを指定し、実際のソート操作を実行することです。

ソートのプロパティ

Song オブジェクトは、曲のタイトル、アーティスト、発表の年、ファイル名、およびユーザーの選曲ジャンル集など、いくつものプロパティを追跡し続けます。言うまでもなく、ソートに実用的なのは、例に挙げた最初の 3 つのプロパティのみです。開発者の便宜のため、例に含めた SortProperty クラスは、ソートに利用可能なプロパティを表す値を持つ列挙として機能します。

```
public static const TITLE:SortProperty = new SortProperty("title");
public static const ARTIST:SortProperty = new SortProperty("artist");
public static const YEAR:SortProperty = new SortProperty("year");
```

SortProperty クラスには、`TITLE`、`ARTIST`、および `YEAR` の 3 つの定数があり、各定数には、ソートに使用できる対応する Song クラスプロパティの実際の名前を含むストリングが格納されます。コードの残り全体を通し、ソートプロパティが指定されると必ず列挙メンバーを使用して行われます。たとえば、PlayList コンストラクタでは、次のように、`sortList()` メソッドを呼び出してリストを最初にソートします。

```
// 最初のソートを設定する
this.sortList(SortProperty.TITLE);
```

ソートのプロパティは `SortProperty.TITLE` として指定されているため、曲はタイトルに従ってソートされます。

プロパティによるソートおよびソートオプションの指定

実際には曲のリストのソートは、次に示すように、PlayList クラスが sortList() メソッドで実行します。

```
/**
 * 指定されたプロパティに従って曲のリストをソートする
 */
public function sortList(sortProperty:SortProperty):void
{
    ...
    var sortOptions:uint;
    switch (sortProperty)
    {
        case SortProperty.TITLE:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.ARTIST:
            sortOptions = Array.CASEINSENSITIVE;
            break;
        case SortProperty.YEAR:
            sortOptions = Array.NUMERIC;
            break;
    }

    // データの実際のソートを実行する
    this._songs.sortOn(sortProperty.propertyName, sortOptions);

    // 現在のソートプロパティを保存する
    this._currentSort = sortProperty;

    // リストがソートされたことを記録する
    this._needToSort = false;
}
```

タイトルまたはアーティストによるソート時にはアルファベット順のソートを、年によるソート時には数値のソートを実行するのが自然です。switch ステートメントを使用して、sortProperty パラメータが指定する値に従った適切なソートオプション (変数 sortOptions に格納) を定義します。ここでも、プロパティの区別には、ハードコードされた値ではなく名前付き列挙メンバーを使用しています。

ソートプロパティおよびソートオプションの決定に続き、_songs 配列を sortOn() メソッドの呼び出しで実際にソートし、それらの 2 つの値をパラメータとして渡します。現在のソートプロパティが記録されますが、それは曲のリストが現時点でソートされたということです。

文字で区切られたストリングへの配列要素の結合

配列を使用して曲のリストを PlayList クラスで維持する以外に、この例では Song クラスでも配列を使用して、曲についてのジャンルリストを管理するのに役立てています。次に示す Song クラスの定義からの抜粋で検討してみます。

```
private var _genres:String;

public function Song(title:String, artist:String, year:uint, filename:String,
   genres:Array)
{
   ...
   // ジャンルは配列として渡されるが、
   // 格納時はセミコロンで区切られたストリングである
   this._genres = genres.join(";");
}
```

新しい Song インスタンスの作成時に、曲についてのジャンルの指定に使用する genres パラメータを Array インスタンスとして定義しています。そのため、複数のジャンルを一緒のグループにまとめ、コンストラクタに渡せる 1 つの変数とするのに便利です。ただし、内部的には Song クラスは、セミコロンで区切られた String インスタンスとして _genres プライベート変数にジャンルを維持します。Array パラメータは、リテラルストリング値 ";" を区切り文字として指定して join() メソッドを呼び出すと、セミコロンで区切られたストリングに変換されます。

同じように、genres アクセサを使用して、ジャンルを配列として設定また取得できます。

```
   public function get genres():Array
   {
      // ジャンルはセミコロンで区切られたストリングとして格納される
      // そのため返すには配列に戻す変換が必要である
      return this._genres.split(";");
   }
   public function set genres(value:Array):void
   {
      // ジャンルは配列として渡されるが、
      // 格納時はセミコロンで区切られたストリングである
      this._genres = value.join(";");
   }
```

genres set アクセサの動作は、コンストラクタと完全に同じです。配列を受け取って join() メソッドを呼び出し、セミコロン区切りのストリングに変換します。get アクセサは反対の操作を実行します。つまり、_genres 変数の split() メソッドが呼び出されると、指定された区切り文字 (リテラルストリング値 ";" など) を使用して、ストリングを値の配列に分割します。

第 9 章

エラー処理

9

" エラー処理 " とは、アプリケーションのコンパイル時またはコンパイル済みアプリケーションの実行時のいずれかに生成されたエラーに応答する、つまりエラーを解決するロジックをアプリケーションに組み入れることを意味します。アプリケーションによるエラー処理では、応答なしや、いかなるプロセスでも説明なしのエラー異常終了になるのとは対照的に、エラー発生時に応答として何かが起きます。エラー処理を正しく使用すれば、予期しない動作からアプリケーションとユーザーを保護するために役立ちます。

ただし、エラー処理は広範な概念で、コンパイル中または実行時に発生する多くの種類のエラーに対する応答を含みます。この章では、ランタイムエラーを処理する方法、生成される可能性のあるさまざまな種類のエラー、および ActionScript 3.0 における新しいエラー処理システムのメリットを中心に説明します。また、エラー処理に関する独自のアプローチをアプリケーション用に実装する方法についても取り上げます。

目次

エラーの種類 256
ActionScript 3.0 におけるエラー処理 258
デバッグ版の Flash Player の操作 260
アプリケーションの同期エラーの処理 261
カスタムエラークラスの作成 266
エラーイベントおよびステータスへの応答 267
Error クラスの比較 271
例 : CustomErrors アプリケーション 277

エラーの種類

アプリケーションの開発中および実行中に発生するエラーは、さまざまな種類があり、使用する用語が異なります。主要なエラーの種類と用語について簡単に説明します。

- コンパイル時エラー : ActionScript コンパイラによるコンパイル処理の際に発生します。コンパイル時エラーは、コードに文法的な問題があるためアプリケーションを構築できないことを示します。
- ランタイムエラー : コンパイル済みのアプリケーションを実行している際に発生します。ランタイムエラーは、SWF ファイルを Adobe Flash Player 9 で再生中に起きるエラーを表します。多くの場合、発生と同時にランタイムエラーを処理してユーザーに報告し、アプリケーションの実行を続けるために必要な処置をとることができます。致命的なエラー (リモート Web サイトへの接続失敗、必須データのロード失敗など) の場合は、アプリケーションを安全に終了するための手段としてエラー処理を使用できます。
- 同期エラー : 関数呼び出しの際に発生するランタイムエラーです。たとえば、特定のメソッドに無効なパラメータを渡して使用しようとすると、Flash Player の例外がスローされる場合です。ほとんどのエラーは同期的に発生 (該当するステートメントを実行した時点で発生) するため、制御フローはすぐに最も適切な catch ステートメントに移りますが、

  次の抜粋コードの例では、プログラムがファイルをアップロードしようとする前に browse() メソッドを呼び出していないためランタイムエラーがスローされます。

  ```
  var fileRef:FileReference = new FileReference();
  try
  {
     fileRef.upload("http://www.yourdomain.com/fileupload.cfm");
  }
  catch (error:IllegalOperationError)
  {
     trace(error);
     // エラー #2037: 関数呼び出しのシーケンスが正しくないか、
     // 以前の呼び出しが失敗しました
  }
  ```

  この場合、ファイルをアップロードしようとする前に browse() メソッドが呼び出されていないことが Flash Player によって検出されたため、ランタイムエラーが発生します。

  同期エラー処理の詳細については、261 ページの「アプリケーションの同期エラーの処理」を参照してください。

■ 非同期エラー：実行時のさまざまな時点で発生するランタイムエラーで、イベントを生成して、イベントリスナーによってキャッチされるエラーです。非同期操作とは、関数が操作を開始してその完了を待たない操作です。アプリケーションまたはユーザーが何らかの操作を試すのを待つために、エラーイベントリスナーを作成し、その操作が失敗した場合にはイベントリスナーでエラーをキャッチしてそのエラーイベントに応答することができます。イベントリスナーはイベントハンドラ関数を呼び出して、エラーイベントに有効な方法で応答します。たとえば、イベントハンドラは、ユーザーにエラーの解決を促すダイアログボックスを起動できます。

前に示した、ファイルアップロードの同期エラーの例で考えてみます。browse() メソッドを正常に呼び出してからファイルのアップロードを開始したとすると、Flash Player からいくつかのイベントが送出されます。たとえば、アップロードが開始したときには open イベントが送出され、ファイルのアップロードが正常に完了したときには complete イベントが送出されます。イベント処理は非同期のため (つまり、事前に設定済みのわかっている時刻に発生するのではないため)、次のように addEventListener() メソッドを使用して、これら特定のイベントをリッスンする必要があります。

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.OPEN, openHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();

function selectHandler(event:Event):void
{
    trace("...select...");
    var request:URLRequest = new URLRequest("http://www.yourdomain.com/
    fileupload.cfm");
    request.method = URLRequestMethod.POST;
    event.target.upload(request.url);
}
function openHandler(event:Event):void
{
    trace("...open...");
}
function completeHandler(event:Event):void
{
    trace("...complete...");
}
```

非同期エラー処理の詳細については、267 ページの「エラーイベントおよびステータスへの応答」を参照してください。

- 不明な (キャッチされない) 例外 : `catch` ステートメントのような､応答のための対応ロジックなしで発生するエラーです。アプリケーションでエラーが発生したときに、エラー処理のための該当する `catch` ステートメントまたはイベントハンドラがそのレベルまたは上位レベルで見つからない場合、そのエラーは不明な例外と見なされます。

  実行時に Flash Player は、エラーによって現在の SWF ファイルが停止しない場合、不明なエラーを意図的に無視して再生を継続しようとします。ユーザー自身が必ずしもエラーを解決できるわけではないからです。不明なエラーを無視するプロセスは " 説明のない異常終了 " と呼ばれ、これはアプリケーションのデバッグを複雑にする可能性があります。不明なエラーに応答するために、デバッグ版の Flash Player では､実行中のスクリプトを終了し､その不明なエラーを `trace` ステートメントの出力に表示するかエラーメッセージをログファイルに書き込みます。また、例外オブジェクトが Error クラスまたはそのサブクラスのインスタンスである場合は、`getStackTrace()` メソッドが呼び出され､スタックトレース情報が `trace` ステートメントまたはログファイルに出力されます。デバッグ版の Flash Player の使用の詳細については、260 ページの「デバッグ版の Flash Player の操作」を参照してください。

ActionScript 3.0 におけるエラー処理

エラーを処理するためのロジックを構築しないで実行できるアプリケーションが数多くあるため、開発者はエラー処理のアプリケーションへの組み入れを後回しにしたい気持ちになります。ただし、エラー処理がないと、何かが予想どおりに働かない場合に、アプリケーションは簡単に機能を停止するかユーザーを欲求不満にさせる可能性があります。ActionScript 2.0 の Error クラスでは、特定のメッセージで例外をスローするためのロジックを独自の関数に組み入れることができました。エラー処理はユーザーフレンドリーなアプリケーションを作成する上で非常に重要であるため、ActionScript 3.0 ではエラーキャッチのために拡張アーキテクチャが追加されています。

> **メモ**
> 『ActionScript 3.0 リファレンスガイド』に、多くのメソッドによってスローされる例外の記載がありますが、メソッドごとに可能性のある例外を網羅してあるわけではありません。メソッドの記述にメソッドがスローする一部の例外が記載されている場合であっても、その記述に明示的に書かれていない構文エラーまたはその他の問題についての例外がメソッドによってスローされることがあります。

ActionScript 3.0 のエラー処理の要素

ActionScript 3.0 には、次に示すように、エラー処理のツールが多数用意されています。

- Error クラス。ECMAScript (ECMA-262) Edition 4 言語仕様案に準拠して、ActionScript 3.0 では、エラーオブジェクトを生成する可能性のある状況の有効範囲を広げるために、幅広い範囲の Error クラスが用意されています。システムエラー (MemoryError 状態など)、コーディングエラー (ArgumentError 状態など)、ネットワークおよび通信のエラー (URIError 状態など)、あるいはその他の状況のいずれであっても、アプリケーションが特定のエラー状態を処理しそれに応答するのに役立つ Error クラスがそれぞれに用意されています。各クラスの詳細については、271 ページの「Error クラスの比較」を参照してください。
- 説明のない異常終了の減少 : Flash Player の以前のバージョンでは、エラーは明示的に `throw` ステートメントを使用した場合にしか生成および報告されませんでした。Flash Player 9 では、ActionScript のネイティブなメソッドやプロパティでランタイムエラーがスローされるようになり、発生するこのような例外をより効果的に処理し、個々の例外に個別に対応することが可能です。
- デバッグ中に表示されるエラーメッセージの明確化 : デバッグ版の Flash Player を使用すると、コードまたは状況に問題があるときに明確な内容を示すエラーメッセージが表示されるため、特定のコードブロックで異常が発生した原因を容易に特定できます。このため、エラーの解決がより効率的になります。詳細については、260 ページの「デバッグ版の Flash Player の操作」を参照してください。
- 正確なエラー情報によって、実行時にユーザーに表示されるエラーメッセージの改良 : Flash Player の以前のバージョンでは、`FileReference.upload()` メソッドの呼び出しに失敗すると Boolean 型の戻り値で `false` が返されましたが、その場合に考えられるエラーの内容には 5 つの種類がありました。ActionScript 3.0 では、`upload()` メソッドの呼び出しにおいてエラーが発生した場合に 4 種類の具体的なエラーをスローでき、エンドユーザーに対して的確なエラーメッセージを表示しやすくなりました。
- エラー処理の再定義 : 別個のエラーが多くの一般的な状況でスローされます。たとえば、ActionScript 2.0 では、FileReference オブジェクトに値を設定する前に、`name` プロパティの値は `null` です。そのため、`name` プロパティを使用または表示する前に、値が設定済みで `null` でないことを確認する必要があります。ActionScript 3.0 では、FileReference オブジェクトの内容が設定される前に `name` プロパティにアクセスしようとすると、値が未設定であることを示す IllegalOperationError 例外がスローされます。`try..catch..finally` ブロックを使用してエラーを処理することができます。詳細については、261 ページの「try..catch..finally ステートメントの使用」を参照してください。
- 従来とほとんど変わらないパフォーマンス : `try..catch..finally` ブロックによるエラー処理に必要とされるリソース量は、ActionScript の以前のバージョンと同等またはわずかに増加する程度に抑えられています。
- ErrorEvent クラス : このクラスを使用すると、特定の非同期エラーイベントにリスナーを構築することができます。詳細については、267 ページの「エラーイベントおよびステータスへの応答」を参照してください。

エラー処理のアプローチ

問題のある状態が見つからない限り、エラー処理ロジックがコードに組み込まれていないアプリケーションはそのまま正常に実行される可能性があります。ただし、積極的なエラー処理がないと、アプリケーションで実際に問題が見つかって異常終了してもユーザーはその原因を知ることができません。

いろいろな方法によってアプリケーションでエラー処理にアプローチすることができます。エラー処理の主要な 3 つの方法の概略を次に示します。

- `try..catch..finally` ステートメントの使用 : これは発生する同期エラーをキャッチします。各種レベルのコード実行で例外をキャッチするために、ステートメントを階層にネストすることができます。詳細については、261 ページの「try..catch..finally ステートメントの使用」を参照してください。
- 独自のカスタムエラーオブジェクトの作成 : Error クラスを使用して、ビルトインのエラーの種類が扱わない特定の操作をアプリケーションで追跡するために、独自のカスタムエラーオブジェクトを作成することができます。そうすると、カスタムエラーオブジェクトで `try..catch..finally` ステートメントを使用できます。詳細については、266 ページの「カスタムエラークラスの作成」を参照してください。
- エラーイベントに応答するイベントリスナーおよびハンドラの記述 : この方法を採用すると、類似のエラーを処理する同じような `try..catch..finally` ブロックをいろいろな場所に記述することなく、グローバルなエラーハンドラでまとめて処理できます。また、このアプローチを使用すると非同期エラーをキャッチする可能性が高くなります。詳細については、267 ページの「エラーイベントおよびステータスへの応答」を参照してください。

デバッグ版の Flash Player の操作

開発者のデバッグ用に特別版の Flash Player が用意されています。このデバッグ版の Flash Player を入手するには Adobe Flash CS3 または Adobe Flex Builder 2 をインストールします。

デバッグ版とリリース版の Flash Player では、エラーの表示方法に重要な差異があります。デバッグ版の場合はエラーのタイプ (一般の Error、IOError、EOFError)、エラー番号、および人間が理解できるエラーメッセージが表示されますが、リリース版の場合はエラーのタイプとエラー番号しか表示されません。たとえば、次のようなコードがあるとします。

```
try
{
   tf.text = myByteArray.readBoolean();
}
catch (error:EOFError)
{
   tf.text = error.toString();
}
```

`readBoolean()` メソッドが EOFError をスローした場合、デバッグ版の Flash Player では、`tf` テキストフィールドに "EOFError: Error #2030: ファイルの終端 (EOF) が検出されました" というメッセージが表示されます。

同じコードをリリース版の Flash Player 上で実行すると、"EOFError: Error #2030" と表示されます。

使用リソース量やサイズを最小限にするため、リリース版の Flash Player にはエラーメッセージのストリングが含まれていません。マニュアル (『ActionScript 3.0 リファレンスガイド』の付録) を参照してエラー番号からエラーメッセージを調べてください。別の方法として、デバッグ版の Flash Player 上で問題を再現することで、省略されていないメッセージを確認することができます。

アプリケーションの同期エラーの処理

最も一般的なエラー処理は同期エラー処理のロジックで、実行時に同期エラーをキャッチするためのステートメントをコードに挿入します。この種類のエラー処理では、関数失敗時のランタイムエラーをアプリケーションで認識してそこから回復できます。同期エラーをキャッチするロジックに `try..catch..finally` ステートメントがあり、これは実際に操作を試し、Flash Player からのエラー応答をキャッチし、最後に、失敗した操作を処理するために何らかの操作を別に実行します。

try..catch..finally ステートメントの使用

同期的なエラーを扱う場合は、`try..catch..finally` ステートメントを使用してエラーをキャッチします。ランタイムエラーが発生すると、Flash Player は例外をスローします。これは、Flash Player が通常の実行処理を中断し、Error 型の特殊なオブジェクトを作成することを意味します。この Error オブジェクトは、該当する最初の `catch` ブロックに送出されます。

`try` ステートメントは、エラーが発生する可能性のあるステートメントを囲むために使用します。`try` には必ず `catch` ステートメントが伴います。`try` ステートメントのブロック内にある何らかのステートメントでエラーが検出されると、その `try` ステートメントに対応する `catch` ステートメントが実行されます。

`finally` ステートメントで囲んだブロックは、`try` ブロックでエラーが発生したかどうかに関係なく実行されます。エラーが発生しなかった場合は、`try` ブロック内のステートメントが終了した後で、`finally` ブロック内のステートメントが実行されます。エラーが発生した場合は、適切な `catch` ステートメントがまず実行され、その後で `finally` ブロック内のステートメントが実行されます。

try..catch..finally ステートメントを使用する際のシンタックスを次のコードに示します。

```
try
{
    // エラーをスローする可能性があるコード
}
catch (err:Error)
{
    // エラーを処理するコード
}
finally
{
    // エラーがスローされたかどうかに関係なく実行されるコード。このコードでは、
   // エラー発生後のクリーンアップや、アプリケーションを続行するために必要な処理を行う。
}
```

個々の catch ステートメントには、そこで処理する特定の例外の型を指定します。catch ステートメントに指定できるのは Error クラスのサブクラスのみです。各 catch ステートメントは上から順にチェックされ、スローされたエラーに該当する最初の catch ステートメントだけが実行されます。したがって、先に上位レベルの Error クラスをチェックし、その Error クラスのサブクラスを後からチェックすると、上位レベルの Error クラスに対する処理しか実行されません。この動作を説明するコードを次に示します。

```
try
{
   throw new ArgumentError("I am an ArgumentError");
}
catch (error:Error)
{
   trace("<Error> " + error.message);
}
catch (error:ArgumentError)
{
   trace("<ArgumentError> " + error.message);
}
```

上のコードを実行すると、次のように表示されます。

```
<Error> I am an ArgumentError
```

ArgumentError を正しくキャッチするには、次のコードのように、最も具体的なエラーのチェックが最初で、下に行くほど一般性の強いエラーのチェックになるような順序にする必要があります。

```
try
{
   throw new ArgumentError("I am an ArgumentError");
}
catch (error:ArgumentError)
{
   trace("<ArgumentError> " + error.message);
}
catch (error:Error)
{
   trace("<Error> " + error.message);
}
```

Flash Player API に含まれるいくつかのメソッドとプロパティでは、実行中にエラーが発生するとランタイムエラーをスローします。たとえば、Sound クラスの close() メソッドでは、オーディオストリームを閉じることができない場合に IOError をスローします。次にコード例を示します。

```
var mySound:Sound = new Sound();
try
{
  mySound.close();
}
catch (error:IOError)
{
  // Error #2029: この URLStream オブジェクトには開いているストリームがありません。
}
```

『ActionScript 3.0 リファレンスガイド』に慣れるに従い、各メソッドの説明に詳細が記載されているとおり、どのメソッドで例外がスローされるかがわかってくるはずです。

throw ステートメント

アプリケーション内で実行時にエラーが発生した場合は、Flash Player が例外をスローします。また、アプリケーション内で throw ステートメントを使用すれば、開発者が例外を明示的にスローすることもできます。明示的にエラーをスローした場合は、Error クラスまたはそのサブクラスのインスタンスをスローすることをお勧めします。次のコードは、Error クラス MyErr のインスタンスをスローする throw ステートメントを示し、最後に、エラーのスロー後に応答するために関数 myFunction() を呼び出しています。

```
var MyError:Error = new Error("Encountered an error with the numUsers value",
  99);
var numUsers:uint = 0;
try
{
  if (numUsers == 0)
  {
    trace("numUsers equals 0");
  }
}
```

```
catch (error:uint)
{
    throw MyError; // 符号なし整数のエラーをキャッチする
}
catch (error:int)
{
    throw MyError; // 整数のエラーをキャッチする
}
catch (error:Number)
{
    throw MyError; // 数値のエラーをキャッチする
}
catch (error:*)
{
    throw MyError; // その他すべてのエラーをキャッチする
}
finally
{
    myFunction(); // 必要に応じてクリーンアップを実行する
}
```

このように、catch ステートメントは最も限定的なデータ型から順に記述するように注意してください。Number データ型に対する catch ステートメントを最初に記述した場合は、uint データ型についても int データ型についても catch ステートメントは実行されなくなります。

メモ	Java プログラミング言語では、例外をスローする可能性があるすべての関数はその旨を宣言することが義務付けられています。宣言するには、スローする可能性がある例外クラスを列挙した throws 節を関数の宣言に含めます。ActionScript では、関数でスローする可能性がある例外を宣言する必要はありません。

単純なエラーメッセージの表示

新しい例外およびエラーイベントモデルがもたらす最大のメリットの 1 つに、アクションが失敗したタイミングと理由をユーザーに示すことができるという点があります。コードを記述する際、メッセージの表示と応答オプションの提示があるようにします。

次のコードは、エラーをテキストフィールドに表示する単純な try..catch ステートメントを示しています。

```
package
{
    import flash.display.Sprite;
    import flash.text.TextField;

    public class SimpleError extends Sprite
    {
        public var employee:XML =
            <EmpCode>
                <costCenter>1234</costCenter>
```

```
            <costCenter>1-234</costCenter>
        </EmpCode>;

    public function SimpleError()
    {
        try
        {
            if (employee.costCenter.length() != 1)
            {
                throw new Error("Error, employee must have exactly one cost center
  assigned.");
            }
        }
        catch (error:Error)
        {
            var errorMessage:TextField = new TextField();
            errorMessage.autoSize = TextFieldAutoSize.LEFT;
            errorMessage.textColor = 0xFF0000;
            errorMessage.text = error.message;
            addChild(errorMessage);
        }
    }
  }
}
```

広範囲のエラークラスおよびビルトインのコンパイルエラーを使用することで、ActionScript 3.0 では以前のバージョンの ActionScript に比べ、何かが失敗したときにその理由に関してより多くの情報が提供されます。このため、優れたエラー処理をする、安定性の向上したアプリケーションを構築することができます。

エラーの再スロー

アプリケーションの開発時には、エラーを適切に処理できず再スローする必要が生じる場合があります。たとえば、次のコードはネストされた `try..catch` ブロックを含んでおり、下位の `catch` ブロックでエラーを処理できない場合には ApplicationError をスローしています。

```
try
{
  try
  {
    trace("<< try >>");
    throw new ArgumentError("some error which will be rethrown");
  }
  catch (error:ApplicationError)
  {
    trace("<< catch >> " + error);
    trace("<< throw >>");
    throw error;
```

```
    }
    catch (error:Error)
    {
        trace("<< Error >> " + error);
    }
}
catch (error:ApplicationError)
{
    trace("<< catch >> " + error);
}
```

このコードを実行すると次のような内容が表示されます。

```
<< try >>
<< catch >> ApplicationError: some error which will be rethrown
<< throw >>
<< catch >> ApplicationError: some error which will be rethrown
```

ネストされた `try` ブロックでカスタム ApplicationError エラーをスローすると、それに続く `catch` ブロックがエラーをキャッチします。このネストされた `catch` ブロックではエラー処理を試みますが、処理できない場合は ApplicationError オブジェクトを上位の `try..catch` ブロックに対してスローします。

カスタムエラークラスの作成

ActionScript では、標準 Error クラスのいずれかを拡張し、目的に特化した独自のエラークラスを作成できます。独自のエラークラスを作成する理由は、次のようにいくつかあります。

- アプリケーションに特有のエラー群を識別するため :

  たとえば、Flash Player でトラップされたエラーに加え、アプリケーションのコードでスローされた特有のエラーに対し、異なる方法で対応することが必要な場合があります。`try..catch` ブロックで新しいエラーデータ型を追跡するために、Error クラスのサブクラスを作成することができます。

- アプリケーション内で生成したエラーについて独自の方法でエラー情報を表示するため :

  たとえば、何らかの決まった方法でエラーメッセージを整形表示するために新しい `toString()` メソッドを作成できます。また、必要に応じて `lookupErrorString()` メソッドを定義することもできます。このメソッドはパラメータで指定したエラーコードに対して、ユーザーの言語環境設定に応じた適切な言語のメッセージを返します。

独自のエラークラスは、ActionScript コア Error クラスを継承している必要があります。ここに示すのは、Error クラスを継承した、特化された AppError クラスの例です。

```
public class AppError extends Error
{
    public function AppError(message:String, errorID:int)
    {
        super(message, errorID);
    }
}
```

次に示すのは、プロジェクトで AppError を使用した例です。

```
try
{
   throw new AppError("Encountered Custom AppError", 29);
}
catch (error:AppError)
{
   trace(error.errorID + ": " + error.message)
}
```

メモ	サブクラスで Error.toString() メソッドをオーバーライドする場合、受け付けるパラメータは ... (残りパラメータ) の 1 つだけとします。これは Error.toString() メソッドの仕様として ECMAScript (ECMA-262) Edition 3 言語仕様に定められていることであり、後方互換性を維持するため ActionScript 3.0 もこの形式に準拠しています。したがって、Error.toString() メソッドをオーバーライドする際は正確にこのパラメータを踏襲する必要があります。ただし、実行時に toString() メソッドにパラメータを渡しても無視されるため、パラメータを指定する意味はありません。

エラーイベントおよびステータスへの応答

ActionScript 3.0 のエラー処理で向上した最も重要な機能の 1 つは、非同期のランタイムエラーに応答するためのエラーイベント処理をサポートすることです。非同期エラーの定義については、256 ページの「エラーの種類」を参照してください。

エラーイベントに応答するためにイベントリスナーとイベントハンドラを作成することができます。多くのクラスが、他のイベントの送出時と同じ方法でエラーイベントを送出します。たとえば、XMLSocket クラスのインスタンスは通常、Event.CLOSE、Event.CONNECT および DataEvent.DATA の 3 種類のイベントを送出します。ただし、問題が発生した場合には、XMLSocket クラスは IOErrorEvent.IOError または SecurityErrorEvent.SECURITY_ERROR を送出できます。イベントリスナーとイベントハンドラの詳細については、345 ページ、第 13 章の「イベントの処理」を参照してください。

エラーイベントは、次の 2 つのカテゴリのいずれかに該当します。

- ErrorEvent クラスを拡張したエラーイベント

  flash.events.ErrorEvent クラスは、ネットワークおよび通信の操作に関係した Flash Player ランタイムエラーを管理するためのプロパティおよびメソッドを含みます。AsyncErrorEvent、IOErrorEvent、および SecurityErrorEvent クラスは ErrorEvent クラスを拡張したものです。デバッグ版の Flash Player を使用している場合は、リスナー関数が検出されなくても、実行時にダイアログボックスでエラーイベントが通知されます。

■ ステータスに基づくエラーイベント

ステータスに基づくエラーイベントは、ネットワークと通信のクラスの netStatus および status プロパティに関係します。データの読み取りまたは書き込み時に Flash Player で問題が検出されると、使用中のクラスオブジェクトに応じて、netStatus.info.level または status.level プロパティが値 "error" に設定されます。このエラーに応答するために、イベントハンドラ関数で level プロパティに値 "error" があるかどうかをチェックします。

エラーイベントの操作

ErrorEvent クラスとそのサブクラスは、データの読み取りまたは書き込みをしようとするときに Flash Player から送出されるエラーを処理するためにエラーの種類を格納しています。

次の例では、ローカルファイルを読み取ろうとして検出されたエラーを表示するために、try..catch ステートメントおよびエラーイベントハンドラの両方を使用しています。もっと高度な処理コードを追加することで、ユーザーにオプションを提供するか、そうでなければ、コメントの「ここにエラー処理コードを記述」の場所でエラーを自動的に処理できます。

```
package
{
  import flash.display.Sprite;
  import flash.errors.IOError;
  import flash.events.IOErrorEvent;
  import flash.events.TextEvent;
  import flash.media.Sound;
  import flash.media.SoundChannel;
  import flash.net.URLRequest;
  import flash.text.TextField;

  public class LinkEventExample extends Sprite
  {
    private var myMP3:Sound;
    public function LinkEventExample()
    {
      myMP3 = new Sound();
      var list:TextField = new TextField();
      list.autoSize = TextFieldAutoSize.LEFT;
      list.multiline = true;
      list.htmlText = "<a href=\"event:track1.mp3\">Track 1</a><br>";
      list.htmlText += "<a href=\"event:track2.mp3\">Track 2</a><br>";
      addEventListener(TextEvent.LINK, linkHandler);
      addChild(list);
    }

    private function playMP3(mp3:String):void
    {
      try
      {
```

```
            myMP3.load(new URLRequest(mp3));
            myMP3.play();
         }
         catch(err:Error)
         {
            trace(err.message);
            // ここにエラー処理コードを記述
         }
         myMP3.addEventListener(IOErrorEvent.IO_ERROR, errorHandler);
      }

      private function linkHandler(linkEvent:TextEvent):void
      {
         playMP3(linkEvent.text);
         // ここにエラー処理コードを記述
      }

      private function errorHandler(errorEvent:IOErrorEvent):void
      {
         trace(errorEvent.text);
         // ここにエラー処理コードを記述
      }
   }
}
```

ステータス変化イベントの操作

Flash Player では、`level` プロパティをサポートするクラスの `netStatus.info.level` または `status.level` プロパティの値は動的に変化します。`netStatus.info.level` プロパティを持つクラスは、NetConnection、NetStream、および SharedObject です。`status.level` プロパティを持つクラスは、HTTPStatusEvent、Camera、Microphone、および LocalConnection です。`level` 値の変化に応答して通信エラーを追跡するために、ハンドラ関数を記述します。

次の例では、`netStatusHandler()` 関数を使用して、`level` プロパティの値をテストしています。`level` プロパティがエラーの検出を示している場合は、"Video stream failed" というメッセージをトレースします。

```
package
{
   import flash.display.Sprite;
   import flash.events.NetStatusEvent;
   import flash.events.SecurityErrorEvent;
   import flash.media.Video;
   import flash.net.NetConnection;
   import flash.net.NetStream;
```

```
    public class VideoExample extends Sprite
    {
        private var videoUrl:String = "Video.flv";
        private var connection:NetConnection;
        private var stream:NetStream;

        public function VideoExample()
        {
            connection = new NetConnection();
            connection.addEventListener(NetStatusEvent.NET_STATUS, 
    netStatusHandler);
            connection.addEventListener(SecurityErrorEvent.SECURITY_ERROR, 
    securityErrorHandler);
            connection.connect(null);
        }

        private function netStatusHandler(event:NetStatusEvent):void
        {
            if (event.info.level = "error")
            {
                trace("Video stream failed")
            }
            else
            {
                connectStream();
            }
        }

        private function securityErrorHandler(event:SecurityErrorEvent):void
        {
            trace("securityErrorHandler: " + event);
        }

        private function connectStream():void
        {
            var stream:NetStream = new NetStream(connection);
            var video:Video = new Video();
            video.attachNetStream(stream);
            stream.play(videoUrl);
            addChild(video);
        }
    }
}
```

Error クラスの比較

ActionScript には多数の定義済み Error クラスがあります。それらの多くは Flash Player で使用されていますが、アプリケーションのコードでも同じ Error クラスを使用して差し支えありません。ActionScript 3.0 には、ActionScript コア Error クラスと flash.error パッケージ Error クラスという 2 つの主要なエラークラスがあります。コア Error クラスは、ECMAScript (ECMA-262) Edition 4 言語仕様案で規定されています。flash.error パッケージの内容は、ActionScript 3.0 のアプリケーション開発およびデバッグを支援するために導入された追加のクラスです。

ECMAScript コア Error クラス

ECMAScript コアエラークラスとしては、Error、EvalError、RangeError、ReferenceError、SyntaxError、TypeError、URIError の各クラスがあります。いずれのクラスもトップレベルの名前空間に属します。

クラス名	説明	メモ
Error	Error クラスは例外をスローするために使用されるクラスであり、ECMAScript に定義されている他の例外クラス、つまり valError、RangeError、ReferenceError、SyntaxError、TypeError および URIError の基本クラスです。	Error クラスは Flash Player からスローされるすべてのランタイムエラーの基本クラスであるだけでなく、独自に作成するエラークラスの基本クラスとすることが推奨されています。
EvalError	EvalError 例外は、Function クラスのコンストラクタに何かパラメータが渡された場合または `eval()` 関数がユーザーコードで呼び出された場合にスローされます。	ActionScript 3.0 では `eval()` 関数のサポートが除外されており、この機能を使用するとエラーがスローされます。 Flash Player の以前のバージョンでは、変数、プロパティ、オブジェクト、ムービークリップに名前でアクセスする際に `eval()` 関数が使用されていました。
RangeError	RangeError 例外は、数値の値が許容される範囲に収まらない場合にスローされます。	たとえば、Timer クラスに対する遅延の指定が負の値や無限である場合は RangeError がスローされます。また、表示オブジェクトを無限の深度に追加しようとした場合も RangeError がスローされます。

クラス名	説明	メモ
ReferenceError	ReferenceError 例外は、sealed 指定された (動的でない) オブジェクトに対して未定義プロパティを参照しようとした場合にスローされます。ActionScript 3.0 より前のバージョンでは、`undefined` であるプロパティにアクセスしようとしても ActionScript コンパイラのエラーは発生しませんでした。しかし、ECMAScript の新しい仕様では、この条件の場合にエラーをスローするよう定められているため、ActionScript 3.0 では ReferenceError 例外をスローするようになっています。	未定義の変数に関する例外は潜在的なバグの発見につながり、ソフトウェア品質の向上に役立ちます。しかし、変数を初期化する必要がない従来の仕様に慣れた開発者は、ActionScript の動作が変更されたことによりコーディング上の習慣を変更する必要があります。
SyntaxError	SyntaxError 例外は、ActionScript コード内で解析エラーが発生した場合にスローされます。 詳細については、Edition 4 が利用可能になるまでは ECMAScript (ECMA-262) Edition 3 言語仕様のセクション 15.11.6.4 (www.ecma-international.org/publications/standards/Ecma-262.htm) と、ECMAScript for XML (E4X) Edition 2 仕様 (ECMA-357) のセクション 10.3.1 (www.ecma-international.org/publications/standards/Ecma-357.htm) を参照してください。	SyntaxError は次の状況でスローされます。 • RegExp クラスで無効な正規表現を解析した場合 • XMLDocument クラスで無効な XML を解析した場合

クラス名	説明	メモ
TypeError	TypeError 例外は、オペランドに要求される型と実際の型が異なる場合にスローされます。 詳細については、ECMAScript 仕様のセクション 15.11.6.5 (www.ecma-international.org/publications/standards/Ecma-262.htm) と、E4X 仕様のセクション 10.3 (www.ecma-international.org/publications/standards/Ecma-357.htm) を参照してください。	TypeError は次の状況でスローされます。 • 関数またはメソッドの実際のパラメータを、形式上必要とされるパラメータの型に変換できない場合 • 変数に割り当てられた値を、その変数の型に変換できない場合 • `is` または `instanceof` 演算子の右辺が有効な型でない場合 • `super` キーワードの使用方法が不正な場合 • プロパティへの参照が複数のバインディングに解決され、結果があいまいである場合 • 互換性のないオブジェクトに対してメソッドが呼び出された場合 (TypeError 例外は RegExp クラスのメソッドが汎用オブジェクトに " 移植 " されてから呼び出された場合などスローされます)
URIError	URIError 例外は、グローバルな URI 処理関数のいずれかが定義に合わない方法で使用された場合にスローされます。 詳細については、ECMAScript 仕様のセクション 15.11.6.6 (www.ecma-international.org/publications/standards/Ecma-262.htm) を参照してください。	URIError は次の状況でスローされます。 有効な URL を必要とする Flash Player API 関数 (`Socket.connect()` など) に対し、無効な URI を指定した場合

ActionScript コア Error クラス

ActionScript には、ECMAScript コア Error クラスの他にも、ActionScript に特有のエラー条件やエラー処理機能に関するクラスがいくつか追加されています。

これらのクラスは ECMAScript Edition 4 言語仕様案に対する ActionScript 独自の拡張ですが、有用な機能追加と判断されて仕様案に反映される可能性を考慮して、flash.error のようなパッケージ内ではなくトップレベルの名前空間に配置されています。

クラス名	説明	注意事項
ArgumentError	ArgumentError クラスは、関数呼び出しで指定されたパラメータの値がその関数の定義に適合していないために発生するエラーを表します。	たとえば、次のような状況でパラメータエラーが発生します。 • メソッドに対して指定されたパラメータの個数が多すぎる、または少なすぎる場合 • 列挙のメンバーを指定することが要求されるパラメータに、それ以外の値が指定された場合
SecurityError	SecurityError 例外は、セキュリティ違反が発生してアクセスが拒否された場合に発生します。	たとえば、次のような状況でセキュリティエラーが発生します。 • セキュリティ Sandbox の境界をまたいで、許可されていないプロパティアクセスやメソッド呼び出しを実行した場合 • セキュリティ Sandbox で許可されていない URL にアクセスしようとした場合 • ポリシーファイルが存在しない環境で、許可されていないポート番号 (たとえば 1024 未満) に対してソケット接続を確立しようとした場合 • ユーザーのカメラやマイクにアクセスしようとしたが、そのアクセス要求がユーザーによって拒否された場合
VerifyError	VerifyError 例外は、不正な形式または破損した SWF ファイルが検出された場合にスローされます。	SWF ファイルが別の SWF ファイルをロードした場合、親の SWF はロードされた SWF が生成する VerifyError をキャッチできます。

flash.error パッケージ Error クラス

flash.error パッケージの Error クラスは、Flash Player API の一部と見なされます。前述した他の Error クラスと違い、flash.error パッケージは Flash Player に特有のエラーイベントを伝達するために使用されます。

クラス名	説明	注意事項
"EOFError:	EOFError 例外は、取得できるデータの末尾よりも後の部分を読み取ろうとした場合にスローされます。	たとえば、IDataInput インターフェイスの読み取りメソッドのいずれかを呼び出したとき、その読み取り要求を満たす量のデータが存在しない場合は EOFError がスローされます。
IllegalOperationError	IllegalOperationError 例外は、メソッドが実装されていないか、使用方法に実装が対応していない場合にスローされます。	たとえば、次のような状況で無効な操作エラーの例外が発生します。 • 基本クラス (DisplayObjectContainer など) に、ステージでサポートされている範囲を超えた機能がある場合。たとえば、Stage のマスクを (`stage.mask` を使用して) 取得または設定しようとすると、Flash Player から "Stage クラスは、このプロパティまたはメソッドを実装しません " というメッセージと共に IllegalOperationError がスローされます。 • サブクラスで、継承する必要がなくサポートすると不都合が生じるようなメソッドを継承した場合 • アクセシビリティのサポートを含めずにコンパイルされた Flash Player の環境で、特定のアクセシビリティ関連メソッドを呼び出した場合 • ランタイム版 Flash Player でオーサリング専用の機能を呼び出した場合 • タイムライン上に配置されたオブジェクトに対して名前を設定しようとした場合
IOError	IOError 例外は、ある種の I/O 例外が発生した場合にスローされます。	たとえば、未接続または切断済みのソケットに対して読み書き操作を実行しようとすると、このエラーが発生します。

クラス名	説明	注意事項
MemoryError	MemoryError 例外は、メモリ割り当て要求が失敗した場合にスローされます。	デフォルトでは、ActionScript 仮想マシン 2 は ActionScript プログラムが割り当てを受けられるメモリの量を制限しません。デスクトップ PC 環境では、メモリ割り当て失敗は頻繁に発生するエラーではありません。操作に必要とされるメモリをシステムが割り当てられないという状況は、デスクトップ PC ではほとんど発生しないからです。32 ビット Windows 版のプログラム (アドレス空間が 2 GB) で 3 GB の割り当てを要求した場合などは、要求を満たすことが不可能であるため、この例外が発生します。
ScriptTimeoutError	ScriptTimeoutError 例外は、スクリプトのタイムアウト間隔である 15 秒が経過した場合にスローされます。ScriptTimeoutError 例外をキャッチすると、スクリプトのタイムアウトをより安全な方法で処理できます。例外ハンドラを用意しない場合は、不明な例外のハンドラによってダイアログボックスにエラーメッセージが表示されます。	悪意のある開発者によってこの例外がキャッチされ、無限ループから脱出できない事態が発生するのを防ぐために、1 つのスクリプトについて実行中にスローされる ScriptTimeoutError 例外は最初の 1 回しかキャッチできません。それ以降の ScriptTimeoutError 例外はアプリケーションのコードではキャッチされず、不明な例外のハンドラが直ちに実行されます。
StackOverflowError	StackOverflowError 例外は、当該スクリプトで使用できるスタックを使い切った場合にスローされます。	StackOverflowError 例外は、無限の再帰が発生したことを示す場合があります。

例 : CustomErrors アプリケーション

ここでは CustomErrors アプリケーションを使って、アプリケーション構築時のカスタムエラーを操作するテクニックについて示します。扱うテクニックは次のとおりです。

- XML パケットの検証
- カスタムエラーの記述
- カスタムエラーのスロー
- エラースロー時のユーザーへの通知

CustomErrors アプリケーションのファイルは、"Samples/CustomError" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
CustomErrors.mxml	MXML ユーザーインターフェイスを構成するメインアプリケーション
com/example/programmingas3/errors/ApplicationError.as	FatalError および WarningError の両方の基本エラークラスとして機能するクラス
com/example/programmingas3/errors/FatalError.as	アプリケーションによるスローの可能性がある FatalError エラーを定義するクラス。このクラスは独自の ApplicationError クラスを拡張します。
com/example/programmingas3/errors/Validator.as	ユーザー指定の従業員 XML パケットを検証する単一のメソッドを定義するクラス
com/example/programmingas3/errors/WarningError.as	アプリケーションによるスローの可能性がある WarningError エラーを定義するクラス。このクラスは独自の ApplicationError クラスを拡張します。

CustomErrors アプリケーションの概要

CustomErrors.mxml ファイルは、カスタムエラーアプリケーション用のユーザーインターフェイスと一部のロジックを含みます。アプリケーションの `creationComplete` イベントが送出されると、`initApp()` メソッドが呼び出されます。このメソッドは、Validator クラスによって検証されるサンプルの XML パケットを定義します。`initApp()` メソッドのコードを次に示します。

```
private function initApp():void
{
  employeeXML =
    <employee id="12345">
      <firstName>John</firstName>
      <lastName>Doe</lastName>
      <costCenter>12345</costCenter>
      <costCenter>67890</costCenter>
    </employee>;
}
```

XML パケットは後で、TextArea コンポーネントインスタンスの Stage に表示されます。このため、再検証を試みる前に XML パケットを変更できます。

ユーザーが Validate ボタンをクリックすると、validateData() メソッドが呼び出されます。このメソッドは、Validator クラスの validateEmployeeXML() メソッドを使用して、従業員 XML パケットを検証します。validateData() メソッドのコードを次に示します。

```
public function validateData():void
{
    try
    {
        var tempXML:XML = XML(xmlText.text);
        Validator.validateEmployeeXML(tempXML);
        status.text = "The XML was successfully validated.";
    }
    catch (error:FatalError)
    {
        showFatalError(error);
    }
    catch (error:WarningError)
    {
        showWarningError(error);
    }
    catch (error:Error)
    {
        showGenericError(error);
    }
}
```

最初に、TextArea コンポーネントインスタンス xmlText の内容を使用して、一時的な XML オブジェクトが作成されます。次に、独自の Validator クラス (com.example.programmingas3/errors/Validator.as) の validateEmployeeXML() メソッドが呼び出され、一時的な XML オブジェクトをパラメータとして渡します。XML パケットが有効な場合、Label コンポーネントインスタンス status が成功メッセージを表示して、アプリケーションが終了します。validateEmployeeXML() メソッドがカスタムエラーをスローした (つまり、FatalError、WarningError、または汎用 Error が発生した) 場合は、適切な catch ステートメントを実行して、showFatalError()、showWarningError()、または showGenericError() メソッドのいずれかを呼び出します。これらの各メソッドは、適切なメッセージを Alert コンポーネントに表示して、発生した具体的なエラーをユーザーに通知します。各メソッドは、具体的なメッセージで Label コンポーネントインスタンス status の更新も行います。

従業員 XML パケットを検証しようとしたときに致命的なエラーが発生した場合は、次のコードが示すように、そのエラーメッセージが Alert コンポーネントに表示され、TextArea コンポーネントインスタンス xmlText および Button コンポーネントインスタンス validateBtn が無効になります。

```
public function showFatalError(error:FatalError):void
{
    var message:String = error.message + "\n\n" + "Click OK to end.";
    var title:String = error.getTitle();
    Alert.show(message, title);
    status.text = "This application has ended.";
    this.xmlText.enabled = false;
    this.validateBtn.enabled = false;
}
```

致命的なエラーではなく警告エラーが発生した場合は、そのエラーメッセージが Alert コンポーネントに表示されますが、TextField および Button コンポーネントインスタンスは無効になりません。showWarningError() メソッドが、カスタムエラーメッセージを Alert コンポーネントインスタンスに表示します。このメッセージでは、XML の検証を続行するかスクリプトを中止するかの質問もユーザーに表示します。showWarningError() メソッドの抜粋コードを次に示します。

```
public function showWarningError(error:WarningError):void
{
    var message:String = error.message + "\n\n" + "Do you want to exit this 
    application?";
    var title:String = error.getTitle();
    Alert.show(message, title, Alert.YES | Alert.NO, null, closeHandler);
    status.text = message;
}
```

Yes または No ボタンを使用してユーザーが Alert コンポーネントインスタンスを閉じると、closeHandler() メソッドが呼び出されます。closeHandler() メソッドの抜粋コードを次に示します。

```
private function closeHandler(event:CloseEvent):void
{
    switch (event.detail)
    {
        case Alert.YES:
            showFatalError(new FatalError(9999));
            break;
        case Alert.NO:
            break;
    }
}
```

警告エラーで Yes をクリックしてユーザーがスクリプトの中止を選択した場合は、FatalError がスローされ、その結果アプリケーションは終了します。

カスタムバリデータの構築

独自の Validator クラスは、1 つのメソッド validateEmployeeXML() を含みます。validateEmployeeXML() メソッドは、1 つのパラメータ employee (検証対象の XML パケット) を受け取ります。validateEmployeeXML() メソッドを次に示します。

```
public static function validateEmployeeXML(employee:XML):void
{
  // XML の項目の完全性チェック
  if (employee.costCenter.length() < 1)
  {
    throw new FatalError(9000);
  }
  if (employee.costCenter.length() > 1)
  {
    throw new WarningError(9001);
  }
  if (employee.ssn.length() != 1)
  {
    throw new FatalError(9002);
  }
}
```

検証のため、従業員は 1 つ (1 つのみ) の原価部門に属していなければなりません。従業員がいずれの原価部門にも属していない場合は、FatalError がスローされてメインアプリケーションファイルの validateData() メソッドに移ります。従業員が複数の原価部門に属している場合は、WarningError がスローされます。XML バリデータの最終チェックは、ユーザーに定義されている社会保障番号が厳密に 1 つであることの確認です (XML パケットの ssn ノード)。存在する ssn ノードが厳密に 1 つでない場合は、FatalError エラーがスローされます。

validateEmployeeXML() メソッドに追加のチェックを加えることができます。たとえば、ssn ノードに格納されている番号が有効であることの確認をするためであったり、従業員に少なくとも 1 つの電話番号と電子メールアドレスを割り当て、その両方の値が有効であることを確認するためであったりします。各従業員に一意の ID を割り当てその上司の ID を指定するために、XML を変更することもできます。

ApplicationError クラスの定義

ApplicationError クラスは、FatalError および WarningError の両方の基本クラスとして機能します。ApplicationError クラスは Error クラスを拡張して、独自にメソッドおよびプロパティを定義します。エラー ID、重大度、独自のエラーコードおよびメッセージを格納する XML オブジェクトなどを定義します。このクラスでは、エラーの種類ごとの重大度を定義するのに使用する 2 つの静的定数も定義します。

ApplicationError クラスのコンストラクタメソッドを次に示します。

```
public function ApplicationError()
{
	messages =
		<errors>
			<error code="9000">
				<![CDATA[Employee must be assigned to a cost center.]]>
			</error>
			<error code="9001">
				<![CDATA[Employee must be assigned to only one cost center.]]>
			</error>
			<error code="9002">
				<![CDATA[Employee must have one and only one SSN.]]>
			</error>
			<error code="9999">
				<![CDATA[The application has been stopped.]]>
			</error>
		</errors>;
}
```

XML オブジェクトの各エラーノードは、一意の数値コードとエラーメッセージを格納します。次の getMessageText() メソッドが示すように、エラーメッセージは、E4X を使用してエラーコードで簡単に調べることができます。

```
public function getMessageText(id:int):String
{
	var message:XMLList = messages.error.(@code == id);
	return message[0].text();
}
```

getMessageText() メソッドは、1 つの整数パラメータ id を受け取り、ストリングを返します。id パラメータには、調べるエラーのエラーコードを指定します。たとえば、id を 9001 で渡すと、「従業員は 1 つのみの原価部門に割り当てなければならない」ことを示すエラーを受け取ります。複数のエラーでエラーコードが同じ場合、ActionScript は見つかった最初の結果のみにエラーメッセージを返します (返された XMLList オブジェクトの message[0])。

このクラスの次のメソッドである getTitle() はパラメータを 1 つも受け取らず、この特定のエラーのエラー ID を格納するストリング値を返します。この値は Alert コンポーネントのタイトルで使用すると、XML パケットの違反時に発生した正確なエラーを簡単に識別するのに役立ちます。getTitle() メソッドの抜粋コードを次に示します。

```
public function getTitle():String
{
	return "Error #" + id;
}
```

ApplicationError クラスの最後のメソッドは toString() です。このメソッドは、エラーメッセージの表現をカスタマイズするために、Error クラスに定義されている関数をオーバーライドします。発生した特定のエラー番号とメッセージを識別するストリングを返します。

```
public override function toString():String
{
    return "[APPLICATION ERROR #" + id + "] " + message;
}
```

FatalError クラスの定義

FatalError クラスはカスタム ApplicationError クラスを拡張して、FatalError コンストラクタ、getTitle() および toString() の 3 つのメソッドを定義します。最初のメソッドである FatalError コンストラクタは、1 つの整数パラメータ errorID を受け取り、ApplicationError クラスに定義されている静的定数値を使用してエラーの重大度を設定し、ApplicationError クラスの getMessageText() メソッドを呼び出して特定のエラーのエラーメッセージを取得します。FatalError コンストラクタを次に示します。

```
public function FatalError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.FATAL;
    message = getMessageText(errorID);
}
```

FatalError クラスの次のメソッドである getTitle() は、ApplicationError クラスで以前に定義された getTitle() メソッドをオーバーライドし、致命的なエラーが発生したことをユーザーに通知するためにタイトルにテキスト "-- FATAL" を追加します。getTitle() メソッドを次に示します。

```
public override function getTitle():String
{
    return "Error #" + id + " -- FATAL";
}
```

このクラスの最後のメソッドである toString() は、ApplicationError に定義されている toString() メソッドをオーバーライドします。toString() メソッドを次に示します。

```
public override function toString():String
{
    return "[FATAL ERROR #" + id + "] " + message;
}
```

WarningError クラスの定義

WarningError クラスは ApplicationError クラスを拡張し、FatalError クラスとほぼ同一です。ただし、次のコードが示すようなマイナー変更があり、エラー重大度は ApplicationError.FATAL ではなく ApplicationError.WARNING に設定されます。

```
public function WarningError(errorID:int)
{
    id = errorID;
    severity = ApplicationError.WARNING;
    message = super.getMessageText(errorID);
}
```

第 10 章

正規表現の使用

10

正規表現とは、ストリング内で条件に一致するテキストの検索および操作に使用するパターンの表記法です。正規表現はストリングのように見えますが、その内容には、パターンや繰り返しを表す特殊なコードが含まれています。たとえば、次の正規表現は、先頭が A であり、その後に 1 桁以上の数字が続くようなストリングに一致します。

```
/A\d+/
```

正規表現では、複雑なパターンや解読困難なパターンが使用されることもあります。たとえば、次の正規表現は有効な電子メールアドレスに一致します。

```
/([0-9a-zA-Z]+[-._+&])*[0-9a-zA-Z]+@([-0-9a-zA-Z]+[.])+[a-zA-Z]{2,6}/
```

この章では、正規表現を作成するための基本的なシンタックスについて説明します。正規表現には複雑な要素や微妙に意味が異なる機能が数多く存在するため、詳細については、Web や書籍で他の参考資料を参照してください。ただし、プログラミング環境が異なると正規表現の実装も異なっている場合があるため注意が必要です。ActionScript 3.0 では、ECMAScript Edition 3 言語仕様 (ECMA-262) の定義に基づいて正規表現を実装しています。

正規表現では、String クラスの `match()` メソッド、`replace()` メソッドおよび `search()` メソッドを使用できます。これらのメソッドの詳細については、216 ページの「ストリング内のパターンの検索およびサブストリングの置換」を参照してください。

目次

正規表現の概要 286
正規表現のシンタックス 287
ストリングに対して正規表現を使用するメソッド 303
例 : Wiki パーサー 304

正規表現の概要

正規表現は、一連の文字に一致するパターンを表します。通常、正規表現は、テキスト値が特定のパターンと一致しているか (ユーザーが入力した電話番号の桁数が正しいかなど) を確認するときや、特定のパターンに一致するテキスト値の一部を置換するときに使用します。たとえば、次の正規表現では、文字 A、B、C が順に連続して出現するパターンを定義しています。

```
/ABC/
```

正規表現リテラルを囲む区切り文字はスラッシュ (/) です。

多くの場合、正規表現は単純な文字の連続ではなく複雑なパターンを見つけるために使用します。たとえば、次の正規表現では、文字 A、B、C が順に連続して出現し、その後に任意の数字が続くパターンを定義しています。

```
/ABC\d/
```

\d というコードが、" 任意の数字 " を意味します。円記号 (\) はエスケープ文字と呼ばれ、正規表現の中では、直後に続く文字 (この場合は d) との組み合わせによって特別な意味を表現します。この章では、各種のエスケープ文字シーケンスと、正規表現シンタックスに関するその他の機能についても説明します。

次の正規表現では、ABC という一連の文字が出現し、その後に任意の個数の数字が続くパターンを定義しています (アスタリスクが付いていることに注意してください)。

```
/ABC\d*/
```

このアスタリスク文字 (*) は、" メタ文字 " の一種です。メタ文字とは、正規表現の中で特別な意味を持つ文字のことです。アスタリスクは " 繰り返し制御文字 " と呼ばれる種類のメタ文字で、文字または文字グループの繰り返しを指定する場合に使用します。詳細については、293 ページの「繰り返し制御文字」を参照してください。

正規表現には、パターンに加え、その正規表現の一致方法を指定するフラグを必要に応じて設定できます。たとえば、次の正規表現で使用している i フラグは、ストリング内の一致箇所を探す際に大文字と小文字を区別しないことを指定するものです。

```
/ABC\d*/i
```

詳細については、299 ページの「フラグとプロパティ」を参照してください。

ストリング内でパターンに一致する箇所を検索および置換するには、String クラスのメソッドに対して正規表現のパラメータを指定します。次に例を示します。

```
var pattern:RegExp = /\d+/; // 連続する 1 桁以上の数字に一致する
var str:String = "Test: 337, 4, or 57.33.";
trace(str.search(pattern)); // 6

trace(str.match(pattern)); // 337

var pattern:RegExp = /\d+/g; // g フラグによるグローバル一致

trace(str.match(pattern)); // 337,4, 57, 33
```

正規表現のパラメータを指定できる String クラスのメソッドとしては、match() メソッド、replace() メソッド、search() メソッドおよび split() があります。これらのメソッドの詳細については、216 ページの「ストリング内のパターンの検索およびサブストリングの置換」を参照してください。

RegExp クラスには、test() メソッドと exec() メソッドがあります。詳細については、303 ページの「ストリングに対して正規表現を使用するメソッド」を参照してください。

正規表現のシンタックス

このセクションでは、ActionScript の正規表現シンタックスを構成するすべての要素について説明します。

正規表現のインスタンスの作成

正規表現のインスタンスを作成するには 2 つの方法があります。1 つは正規表現を区切り文字のスラッシュ (/) で囲んで記述する方法であり、もう 1 つは new コンストラクタを使用する方法です。たとえば、次の正規表現はいずれも同等です。

```
var pattern1:RegExp = /bob/i;
var pattern2:RegExp = new RegExp("bob", "i");
```

ストリングリテラルを引用符で囲むのと同じように、正規表現リテラルを記述する場合はスラッシュで囲みます。正規表現のうち 2 つのスラッシュで囲んだ部分が " パターン " です。それに加え、後ろのスラッシュに続いて " フラグ " を指定することもできます。フラグは正規表現を構成する要素と見なされますが、パターンとは区別されています。

new コンストラクタを使用する場合は、2 つのストリングによって正規表現を定義します。次のように、第 1 のストリングでパターンを指定し、第 2 のストリングでフラグを指定します。

```
var pattern2:RegExp = new RegExp("bob", "i");
```

スラッシュで囲んだ正規表現定義の " 中 " にスラッシュを含める場合は、中のスラッシュの直前にエスケープ文字の円記号 (\) を置く必要があります。たとえば、次の正規表現は 1/2 というパターンに一致します。

```
var pattern:RegExp = /1\/2/;
```

new コンストラクタによる正規表現定義の " 中 " に引用符を含める場合は、中の引用符の直前にエスケープ文字の円記号 (\) を置く必要があります (String リテラルの場合と同じ)。たとえば、次の正規表現は eat at "joe's" というパターンに一致します。

```
var pattern1:RegExp = new RegExp("eat at \"joe's\"", "");
var pattern2:RegExp = new RegExp('eat at "joe\'s"', "");
```

スラッシュで囲んだ正規表現定義の中に引用符を含める場合は、エスケープ文字の円記号を付けないでください。同じように、new コンストラクタによる正規表現定義の中にスラッシュを含める場合は、エスケープ文字の円記号を付けないでください。次の正規表現は同等で、いずれも 1/2 "joe's" というパターンに一致します。

```
var pattern1:RegExp = /1\/2 "joe's"/;
var pattern2:RegExp = new RegExp("1/2 \"joe's\"", "");
var pattern3:RegExp = new RegExp('1/2 "joe\'s"', '');
```

また、new コンストラクタによる正規表現定義の中で、円記号 (\) から始まるメタシーケンス (任意の数字を意味する \d など) を使用する場合は、次のように円記号を二重に入力する必要があります。

```
var pattern:RegExp = new RegExp("\\d+", ""); // 1 桁以上の数字に一致する
```

この場合は、RegExp() コンストラクタメソッドの最初のパラメータがストリングで、ストリングリテラルの中で単一の円記号を認識させるためには円記号を二重に入力する必要があるため、円記号を二重に入力します。

以降のセクションでは、正規表現パターンの定義に使用するシンタックスについて説明します。

フラグの詳細については、299 ページの「フラグとプロパティ」を参照してください。

文字、メタ文字、およびメタシーケンス

最も単純な正規表現は、次のように一連の文字に一致するものです。

```
var pattern:RegExp = /hello/;
```

ただし、次の各文字はメタ文字と呼ばれ、正規表現の中では特別な意味を持ちます。

```
^ $ \ . * + ? ( ) [ ] { } |
```

たとえば、次の正規表現は、文字 A の後に 0 個以上の B が続き (メタ文字のアスタリスクが繰り返しを表す)、さらにその後に C が続くストリングに一致します。

```
/AB*C/
```

メタ文字に該当する文字を正規表現のパターン内に含めて、特別な意味を表さないようにするには、エスケープ文字の円記号 (\) を付ける必要があります。たとえば、次の正規表現は、文字 A の後に B、アスタリスク、および C が続くストリングに一致します。

```
var pattern:RegExp = /AB\*C/;
```

" メタシーケンス " も、メタ文字と同じように正規表現の中で特別な意味を持ちます。1 つのメタシーケンスは複数の文字によって構成されます。以降のセクションでは、メタ文字とメタシーケンスの詳細な使用方法を示します。

メタ文字について

次の表は、正規表現で使用できるメタ文字の一覧です。

メタ文字	説明
^ (キャレット)	ストリングの先頭に一致します。また、m (multiline) フラグを設定した場合は、行の先頭にも一致します (300 ページの「m (multiline) フラグ」を参照してください)。ただし、文字クラスの先頭にキャレットを指定した場合は、ストリングの先頭ではなく補集合を意味します。詳細については、291 ページの「文字クラス」を参照してください。
$ (ドル記号)	ストリングの末尾に一致します。また、m (multiline) フラグを設定した場合は、改行文字 (\n) の直前の位置にも一致します。詳細については、300 ページの「m (multiline) フラグ」を参照してください。
\ (円記号)	特殊文字が持つ特別な意味を無効化します。 また、/1\/2/ のように、正規表現リテラルの中でスラッシュ文字を使用する場合も円記号を使用します (文字 1、スラッシュ、文字 2 の連続に一致)。
. (ドット)	任意の 1 文字に一致します。 ただし、改行文字 (\n) には s (dotall) フラグを設定した場合のみ一致します。 詳細については、301 ページの「s (dotall) フラグ」を参照してください。
* (アスタリスク)	直前のアイテムについて 0 回以上の繰り返しに一致します。 詳細については、293 ページの「繰り返し制御文字」を参照してください。
+ (プラス)	直前のアイテムについて 1 回以上の繰り返しに一致します。 詳細については、293 ページの「繰り返し制御文字」を参照してください。
? (疑問符)	直前のアイテムについて 0 回または 1 回だけ一致します。 詳細については、293 ページの「繰り返し制御文字」を参照してください。
(および)	正規表現の中でグループを定義します。グループは次の場合に使用します。 • 選択制御文字 | の適用範囲を限定する場合。例 : /(a|b|c)d/ • 繰り返し制御文字の適用範囲を指定する場合。例 : /(walla.){1,2}/ • 後方参照を使用する場合。たとえば、次の正規表現に含まれている \1 は、このパターン内にある最初のグループ化括弧に一致したストリングに一致します。 /(\w*) is repeated: \1/ 詳細については、295 ページの「グループ」を参照してください。

メタ文字	説明
[および]	文字クラス (該当位置に一致可能な文字を表す集合) を定義します。 /[aeiou]/ は、角括弧内に指定したいずれか 1 つの文字に一致します。 文字クラス内でハイフン (-) を使用すると、文字の範囲を指定できます。 /[A-Z0-9]/ は、大文字の A ～ Z または 0 ～ 9 に一致します。 文字クラス内で] および - をエスケープするには、円記号を使用します。 /[+\-]\d+/ は、+ または - とそれに続く 1 桁以上の数字に一致します。 文字クラス内では、通常メタ文字として扱われる他の文字は、メタ文字でなく一般の文字として扱われます。円記号でエスケープする必要はありません。 /[$£]/ は、$ または £ に一致します。 詳細については、291 ページの「文字クラス」を参照してください。
| (パイプ)	選択肢を示し、左側または右側のいずれかにある部分に一致します。 /abc|xyz/ は、abc または xyz に一致します。

メタシーケンスについて

メタシーケンスは、正規表現パターンの中で特別な意味を持つ一連の文字です。メタシーケンスについて次の表で説明します。

メタシーケンス	説明
{*n*} {*n*,} および {*n*,*n*}	直前のアイテムについて、繰り返し回数または回数の範囲を指定します。 /A{27}/ は、文字 A の 27 回の繰り返しに一致します。 /A{3,}/ は、文字 A の 3 回以上の繰り返しに一致します。 /A{3,5}/ は、文字 A の 3 回以上 5 回以下の繰り返しに一致します。 詳細については、293 ページの「繰り返し制御文字」を参照してください。
\b	単語構成文字とそれ以外の文字の境界となる位置に一致します。ストリングの先頭または末尾が単語構成文字である場合は、その先頭または末尾にも一致します。
\B	2 つの単語構成文字に挟まれた位置に一致します。また、単語構成文字でない 2 つの文字に挟まれた位置にも一致します。
\d	10 進数の数字 1 文字に一致します。
\D	数字以外の任意の 1 文字に一致します。
\f	改ページ文字に一致します。
\n	改行文字に一致します。
\r	復帰文字に一致します。

メタシーケンス	説明
\s	任意の空白文字 (スペース、タブ、改行、復帰) に一致します。
\S	空白文字以外の任意の 1 文字に一致します。
\t	タブ文字に一致します。
\u*nnnn*	16 進数 *nnnn* で指定される文字コードを持つ Unicode 文字に一致します。たとえば、\u263a はスマイリー文字です。
\v	垂直フィード文字に一致します。
\w	単語構成文字 (A ～ Z、a ～ z、0 ～ 9、または _) に一致します。ただし、英字以外の文字 (é、ñ、ç など) には一致しません。
\W	単語構成文字以外の任意の 1 文字に一致します。
\x*nn*	16 進数 *nn* で指定される文字コードを持つ ASCII 文字に一致します。

文字クラス

文字クラスは、正規表現の中で該当する位置に一致可能な文字を表すリストを指定するために使用します。文字クラスを定義するには、リストを角括弧 ([および]) で囲みます。たとえば、次の正規表現で定義している文字クラスは、bag、beg、big、bog、bug のいずれにも一致します。

```
/b[aeiou]g/
```

文字クラス内のエスケープシーケンス

正規表現において通常は特別な意味を持つメタ文字やメタシーケンスのほとんどは、文字クラス内では特別な意味を持ちません。たとえば、アスタリスクは正規表現の中で繰り返しを意味しますが、文字クラス内に出現するアスタリスクはその例外です。次の文字クラスは、リストで指定されている他の文字と同じようにアスタリスク文字そのものにも一致します。

```
/[abc*123]/
```

ただし、次の表に示す 3 つの文字は、文字クラス内で特別な意味を持つメタ文字として機能します。

メタ文字	文字クラス内での意味
]	文字クラスの末尾を示します。
-	文字の範囲を示します (292 ページの「文字クラス内の文字範囲」を参照してください)。
\	メタシーケンスを指定します。また、メタ文字の特別な意味を無効化します。

これらの文字のいずれかをリテラル文字 (特別な意味のない一般の文字) として認識させるには、その文字の直前にエスケープ文字の円記号を付ける必要があります。たとえば、次の正規表現で定義している文字クラスは、$、\、]、- の 4 つの記号いずれにも一致します。

```
/[$\\\]\-]/
```

文字クラス内でも特別な意味を持つメタ文字があるのと同じように、次のメタシーケンスは文字クラス内でもメタシーケンスとして機能します。

メタシーケンス	文字クラス内での意味
\n	改行文字に一致します。
\r	復帰文字に一致します。
\t	タブ文字に一致します。
\u*nnnn*	16 進数 *nnnn* で指定される文字コードを持つ Unicode 文字に一致します。
\x*nn*	16 進数 *nn* で指定される文字コードを持つ ASCII 文字に一致します。

その他の正規表現メタシーケンスおよびメタ文字は、文字クラス内では一般の文字として扱われます。

文字クラス内の文字範囲

ハイフンを使用すると、文字の範囲を指定できます (例 : A-Z、a-z、0-9)。指定する文字は、使用している文字セットにおいて有効な範囲を構成するものである必要があります。たとえば、次の文字クラスは、a 〜 z の範囲または任意の数字のうち 1 つに一致します。

```
/[a-z0-9]/
```

また、ASCII 文字コードを示す \xnn の形式を使用し、ASCII 値の範囲を指定することもできます。たとえば、次の文字クラスは、拡張 ASCII 文字のセットに属する任意の文字に一致します (é や ê など)。

```
/[\x80-\x9A]/
```

否定文字クラス

文字クラスの先頭にキャレット文字 (^) を指定すると、クラスの指定が反転し、リストに含まれない任意の文字に一致するという意味になります。たとえば、次の文字クラスは、a ～ z の範囲および数字を除く任意の1文字に一致します。

```
/[^a-z0-9]/
```

否定を表すキャレット文字 (^) は、文字クラスの " 先頭 " に記述する必要があります。それ以外の場所に記述した場合は、単に文字クラスにキャレット文字を含める意味になります。たとえば、次の文字クラスは、いろいろな記号 (キャレットを含む) のいずれかに一致します。

```
/[!.,#+*%$&^]/
```

繰り返し制御文字

繰り返し制御文字は、パターンの中で文字またはシーケンスの繰り返しを指定するために使用します。次に例を示します。

繰り返し制御のメタ文字	説明
* (アスタリスク)	直前のアイテムについて 0 回以上の繰り返しに一致します。
+ (プラス)	直前のアイテムについて 1 回以上の繰り返しに一致します。
? (疑問符)	直前のアイテムについて 0 回または 1 回だけ一致します。
{*n*} {*n*,} および {*n*,*n*}	直前のアイテムについて、繰り返し回数または回数の範囲を指定します。 /A{27}/ は、文字 A の 27 回の繰り返しに一致します。 /A{3,}/ は、文字 A の 3 回以上の繰り返しに一致します。 /A{3,5}/ は、文字 A の 3 回以上 5 回以下の繰り返しに一致します。

次のように、繰り返し制御文字は単一の文字に適用することも、グループに適用することもできます。

- /a+/ は、文字 a の 1 回以上の繰り返しに一致します。
- /\d+/ は、1 桁以上の数字に一致します。
- /[abc]+/ は、文字 a、b、c の任意の組み合わせで構成される 1 文字以上の繰り返しに一致します。
- /(very,)*/ は、very という単語の後にカンマと空白が続くストリングの 0 回以上の繰り返しに一致します。

繰り返し制御文字をグループ化括弧内で使用すると、グループに繰り返し制御を適用できます。たとえば、次の繰り返し制御文字は、word というストリングにも、word-word-word というストリングにも一致します。

```
/\w+(-\w+)*/
```

デフォルトでは、正規表現は " 最長一致 " の検索ロジックを使用します。すなわち、正規表現に含まれる各サブパターン (.* など) について、ストリング内で可能な限り多くの文字に一致する方法を探し、それから正規表現内の次の部分に処理を移します。たとえば、次のような正規表現とストリングがあるとします。

```
var pattern:RegExp = /<p>.*<\/p>/;
str:String = "<p>Paragraph 1</p> <p>Paragraph 2</p>";
```

この正規表現はストリング全体に一致します。

```
<p>Paragraph 1</p> <p>Paragraph 2</p>
```

しかし、場合によっては、1 組の <p>...</p> グループのみに一致させることが本来の目的であることもあります。その場合は次のようにします。

```
<p>Paragraph 1</p>
```

任意の繰り返し制御文字の直後に疑問符 (?) を付けると、その繰り返し制御文字は、" 最短一致 " の検索ロジックを使用するように動作が変更されます。たとえば、次の正規表現では、*? という最短一致の繰り返し制御を使用しています。これは、<p> と最小限の文字の後に </p> が続くストリングに一致します。

```
/<p>.*?<\/p>/
```

繰り返し制御文字については、次の点に注意してください。

- {0} または {0,0} と指定しても、対象のアイテムが一致から除外されるわけではありません。
- /abc+*/ のように複数の繰り返し制御文字を重複して指定することはできません。
- ドット (.) に * などの繰り返し制御を付けても、s (dotall) フラグを設定しない限り複数行にわたって一致することはありません。たとえば、次のようなコードがあるとします。

  ```
  var str:String = "<p>Test\n";
  str += "Multiline</p>";
  var re:RegExp = /<p>.*<\/p>/;
  trace(str.match(re)); // null;

  re = /<p>.*<\/p>/s;
  trace(str.match(re));
     // 出力 : <p>Test
     //          Multiline</p>
  ```

詳細については、301 ページの「s (dotall) フラグ」を参照してください。

選択制御

正規表現で | (バー) 文字を使用すると、複数ある選択肢のいずれかに一致することを指定できます。たとえば、次の正規表現は cat、dog、pig、rat のいずれかの単語に一致します。

```
var pattern:RegExp = /cat|dog|pig|rat/;
```

選択制御文字 | の適用範囲を限定するには、グループ化括弧を使用します。次の正規表現は、cat の後に nap または nip が続くストリングに一致します。

```
var pattern:RegExp = /cat(nap|nip)/;
```

詳細については、295 ページの「グループ」を参照してください。

次の 2 つの正規表現は、一方では選択制御文字 | を、もう一方では文字クラス ([と] で指定) を使用していますが、いずれも同等です。

```
/1|3|5|7|9/
/[13579]/
```

詳細については、291 ページの「文字クラス」を参照してください。

グループ

次のように括弧を使用すると、正規表現の中でグループを定義できます。

```
/class-(\d*)/
```

グループはパターンの部分的なセクションであり、次の目的で使用できます。

- 繰り返し制御を複数の文字に適用する場合
- 選択制御 (| 文字) を適用する対象のサブパターンを区切る場合
- 一致したサブストリングをキャプチャする場合。たとえば、正規表現の中で \1 を使用すると、前出のグループが一致したストリングと同じ内容に一致します。また、String クラスの replace() メソッドでは同様の目的に $1 を使用できます。

以降のセクションでは、グループの使用方法について詳しく説明します。

繰り返し制御でのグループの使用

グループを使用しない場合、次のように、繰り返し制御文字はその直前にある 1 つの文字または文字クラスに適用されます。

```
var pattern:RegExp = /ab*/ ;
// 文字 a と、その後に続く
// 文字 b の 0 回以上の繰り返しに一致する

pattern = /a\d+/;
// 文字 a と、その後に続く
// 1 桁以上の数字に一致する

pattern = /a[123]{1,3}/;
// 文字 a と、その後に続く
// 1、2、3 いずれかの 1 回以上 3 回以下の繰り返しに一致する
```

グループを使用すると、次のように、繰り返し制御文字を複数の文字や文字クラスに適用できます。

```
var pattern:RegExp = /(ab)*/;
// 文字 a とその後に続く
// b を組み合わせた 0 回以上の繰り返しに一致する。例 : ababab

pattern = /(a\d)+/;
// 文字 a とその後に続く
// 数字 1 桁を組み合わせた 1 回以上の繰り返しに一致する。例 : a1a5a8a3

pattern = /(spam ){1,3}/;
// spam とそれに続く空白を組み合わせた 1 ～ 3 回の繰り返しに一致する
```

繰り返し制御文字の詳細については、293 ページの「繰り返し制御文字」を参照してください。

選択制御文字 (|) に対するグループの使用

グループは、次のように選択制御文字 (|) の適用対象となる範囲を指定するために使用できます。

```
var pattern:RegExp = /cat|dog/;
// cat または dog に一致する

pattern = /ca(t|d)og/;
// catog または cadog に一致する
```

グループによる一致したサブストリングのキャプチャ

正規表現では、パターン内に通常のグループ化括弧を指定すると、後でそのグループに一致した内容を参照できます。これは " 後方参照 " と呼ばれる機能で、後方参照に使用するグループを " キャプチャグループ " といいます。たとえば、次の正規表現に含まれている \1 というシーケンスは、その前の括弧で指定したキャプチャグループに一致したサブストリングと同じ内容に一致します。

```
var pattern:RegExp = /(\d+)-by-\1/;
// 一致するストリング : 48-by-48
```

後方参照は、\1、\2、...、\99 のように記述することにより、1 つの正規表現の中で最大 99 個まで使用できます。

同じように、String クラスの replace() メソッドでは $1 ～ $99 と記述することにより、一致したサブストリングを置換ストリング内に挿入できます。

```
var pattern:RegExp = /Hi, (\w+)\./;
var str:String = "Hi, Bob.";
trace(str.replace(pattern, "$1, hello."));
// 出力 : Bob, hello.
```

また、RegExp クラスの exec() メソッドまたは String クラスの match() メソッドでキャプチャグループを使用すると、グループに一致したサブストリングをメソッドの戻り値として取得できます。

```
var pattern:RegExp = /(\w+)@(\w+).(\w+)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
    // bob@test.com,bob,example,com
```

非キャプチャグループと先読みグループの使用

非キャプチャグループとは、キャプチャおよび番号による後方参照の機能を持たない、グループ化だけを目的としたグループです。非キャプチャグループを定義するには、次のように (?: と) を使用します。

```
var pattern = /(?:com|org|net);
```

次の例は、(com|org) というパターンをキャプチャグループおよび非キャプチャグループで使用した場合の違いを示しています (全体の一致を調べた後でキャプチャグループの一致内容を列挙するために exec() メソッドを使用します)。

```
var pattern:RegExp = /(\w+)@(\w+).(com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example,com

// 非キャプチャ :
var pattern:RegExp = /(\w+)@(\w+).(?:com|org)/;
var str:String = "bob@example.com";
trace(pattern.exec(str));
  // bob@test.com,bob,example
```

特殊な非キャプチャグループとして、" 先読みグループ " と呼ばれるものがあります。さらにこれは、" 肯定先読みグループ " と " 否定先読みグループ " の 2 つに分類されます。

肯定先読みグループは (?= と) で定義され、該当位置に一致する必要があるサブパターンを示しますが、肯定先読みグループに一致した部分は、同じ正規表現の中にある後続のパターンについても一致の対象となります。たとえば、次のコードに含まれている (?=e) は肯定先読みグループなので、これに一致する文字 e は、正規表現の続き (この例ではキャプチャグループの \w*)) についても一致を調べる対象となります。

```
var pattern:RegExp = /sh(?=e)(\w*)/i;
var str:String = "Shelly sells seashells by the seashore";
trace(pattern.exec(str));
// Shelly,elly
```

否定先読みグループは (?! と) で定義され、該当位置に一致してはならないサブパターンを示します。次に例を示します。

```
var pattern:RegExp = /sh(?!e)(\w*)/i;
var str:String = "She sells seashells by the seashore";
trace(pattern.exec(str));
// shore,ore
```

名前付きグループの使用

名前付きグループとは、正規表現で使用するグループの一種で、名前による識別子を指定したものを指します。名前付きグループを定義するには、(?P< 名前 > と) を使用します。たとえば、次の正規表現では、digits という識別子を持つ名前付きグループを指定しています。

```
var pattern = /[a-z]+(?P<digits>\d+)[a-z]+/;
```

exec( ) メソッドで名前付きグループを使用すると、次のように、名前付きグループに一致したストリングが result 配列にプロパティとして追加されます。

```
var myPattern:RegExp = /([a-z]+)(?P<digits>\d+)[a-z]+/;
var str:String = "a123bcd";
var result:Array = myPattern.exec(str);
trace(result.digits); // 123
```

次の例では、name および dom という識別子を持つ 2 つの名前付きグループを使用しています。

```
var emailPattern:RegExp =
  /(?P<name>(\w|[_.\-])+)@(?P<dom>((\w|-)+))+\.\w{2,4}+/;
var address:String = "bob@example.com";
var result:Array = emailPattern.exec(address);
trace(result.name); // bob
trace(result.dom); // example
```

メモ

名前付きグループは、ECMAScript の言語仕様には規定されていない、ActionScript 3.0 独自の拡張機能です。

フラグとプロパティ

次の表に、正規表現に対して設定できる 5 種類のフラグを示します。各フラグには正規表現オブジェクトのプロパティとしてアクセスできます。

フラグ	プロパティ	説明
g	global	複数箇所に一致します。
i	ignoreCase	一致の判定において大文字と小文字を区別しません。これは A 〜 Z および a 〜 z の文字に対してのみ有効であり、英字以外の É や é には適用されません。
m	multiline	このフラグを設定した場合、$ および ^ はそれぞれ行末と行頭にも一致します。
s	dotall	このフラグを設定した場合、.(ドット) は改行文字 (\n) にも一致します。
x	extended	正規表現の拡張機能を有効にします。パターンの一部と見なされない空白を正規表現の中に挿入できるようになります。これにより、正規表現コードの読みやすさを向上できます。

これらのプロパティは読み取り専用です。次のように、正規表現変数を設定する際にはフラグ (g、i、m、s、x) を設定できます。

```
var re:RegExp = /abc/gimsx;
```

しかし、名前付きプロパティの値を直接設定することはできません。たとえば、次のコードではエラーが発生します。

```
var re:RegExp = /abc/;
re.global = true; // エラーが発生する
```

デフォルトでは、正規表現の宣言内で指定しない限りフラグは設定されず、対応する各プロパティの値は false となります。

正規表現には上記の他にも、次の 2 つのプロパティがあります。

- lastIndex プロパティは、正規表現に対する次回の exec() または test() メソッド呼び出しにおいて使用する、ストリング内のインデックス位置を示します。
- source プロパティは、正規表現の中でパターンを定義する部分のストリングを示します。

g (global) フラグ

g (global) フラグを設定しない場合、正規表現が複数の場所に一致することはありません。たとえば、次の正規表現には g フラグを設定していないので、String.match() メソッドは一致したサブストリングを 1 つしか返しません。

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/;
trace(str.match(pattern)) // 出力 : she
```

次のように g フラグを設定すると、Sting.match() メソッドは複数の一致箇所を返します。

```
var str:String = "she sells seashells by the seashore.";
var pattern:RegExp = /sh\w*/g;
// 同じパターンだが、今度は g フラグを設定する
trace(str.match(pattern)); // 出力 : she,shells,shore
```

i (ignoreCase) フラグ

正規表現の一致を判定する際、デフォルトでは大文字と小文字が区別されますが、i (ignoreCase) フラグを設定した場合は大文字と小文字が区別されません。たとえば、次の正規表現に含まれている小文字の s は、ストリングの先頭にある大文字の S には一致しません。

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/)); // 出力 : 13 -- Not the first character
```

この正規表現に i フラグを設定すると、大文字の S にも一致するようになります。

```
var str:String = "She sells seashells by the seashore.";
trace(str.search(/sh/i)); // 出力 : 0
```

i フラグによる大文字と小文字の同一視は、A 〜 Z および a 〜 z の文字に対してのみ有効であり、英字以外の É や é には適用されません。

m (multiline) フラグ

m (multiline) フラグを設定しない場合、^ はストリングの先頭に、$ はストリングの末尾に一致します。m フラグを設定した場合、これらの文字はそれぞれ行頭と行末にも一致します。次の例では、ストリング内に改行文字が含まれています。

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/g)); // ストリングの先頭にある単語に一致する
```

正規表現に g (global) フラグを設定していても、^ に一致する場所は 1 つ (ストリングの先頭) しか存在しないので、match() メソッドが返すサブストリングは 1 つだけです。したがって、結果の出力は次のようになります。

```
Test
```

次の例も同じコードですが、今度は m フラグを設定しています。

```
var str:String = "Test\n";
str += "Multiline";
trace(str.match(/^\w*/gm)); // 行頭にある単語に一致する
```

この場合、出力にはそれぞれの行頭にある単語が含まれます。

```
Test,Multiline
```

行末を示す文字は、\n の 1 種類だけです。次の文字は行末と見なされません。

- 復帰文字 (\r)
- Unicode の行区切り文字 (\u2028)
- Unicode の段落区切り文字 (\u2029)

s (dotall) フラグ

s (dotall、つまり " ドットを全文字対象にする ") フラグを設定しない場合、正規表現パターン内のドット (.) は改行文字 (\n) には一致しません。したがって、次の例では一致する場所がありません。

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/;
trace(str.match(re));
```

s フラグを設定すると、ドットが改行文字にも一致するようになります。

```
var str:String = "<p>Test\n";
str += "Multiline</p>";
var re:RegExp = /<p>.*?<\/p>/s;
trace(str.match(re));
```

この場合は、<p> タグに囲まれたサブストリング全体 (改行文字を含む) に一致します。

```
<p>Test
Multiline</p>
```

x (extended) フラグ

複雑な正規表現は解読することが困難です。特に、メタ記号やメタシーケンスを多用するとパターンが非常に難解になります。次に例を示します。

```
/<p(>|(\s*[^>]*>)).*?<\/p>/gi
```

正規表現に x (extended) フラグを設定した場合、パターン内に入力した空白文字がすべて無視されます。たとえば、次の正規表現は前の例と同等です。

```
/     <p    (>  |  (\s*  [^>]*  >))     .*?    <\/p>  /gix
```

x フラグを設定している場合に空白文字の一致を調べるには、その空白文字の直前に円記号を付けます。たとえば、次の 2 つの正規表現は同等です。

```
/foo bar/
/foo \ bar/x
```

lastIndex プロパティ

lastIndex プロパティは、ストリング内で次回の検索を開始するインデックス位置を示します。このプロパティは、exec() および test() メソッドの呼び出しに対して g フラグを true に設定した場合に有効です。たとえば、次のようなコードがあるとします。

```
var pattern:RegExp = /p\w*/gi;
var str:String = "Pedro Piper picked a peck of pickled peppers.";
trace(pattern.lastIndex);
var result:Object = pattern.exec(str);
while (result != null)
{
  trace(pattern.lastIndex);
  result = pattern.exec(str);
}
```

lastIndex プロパティは、デフォルトでは 0 (ストリングの先頭から検索を開始) に設定されていますが、検索を実行するたびに、一致した場所に続くインデックス位置に設定されます。したがって、前の例で示したコードの出力は次のようになります。

```
0
5
11
18
25
36
44
```

global フラグを false に設定している場合、exec() メソッドおよび test() メソッドでは lastIndex プロパティを使用または設定しません。

String クラスの match()、replace()、search() 各メソッドでは、検索処理を常にストリングの先頭から実行します。呼び出しに使用する正規表現の lastIndex プロパティにどのような値を設定しても影響はありません (ただし、match() メソッドを呼び出すと lastIndex には 0 が設定されます)。

lastIndex プロパティに値を設定すれば、ストリング内で正規表現による検索を開始する位置を変更できます。

source プロパティ

source プロパティは、正規表現の中でパターンを定義する部分のストリングを示します。次に例を示します。

```
var pattern:RegExp = /foo/gi;
trace(pattern.source); // foo
```

ストリングに対して正規表現を使用するメソッド

RegExp クラスには、exec() と test() という 2 つのメソッドがあります。

RegExp クラスの exec() メソッドおよび test() メソッドに加え、String クラスにも、ストリングに対して正規表現の一致を判定する match() メソッド、replace() メソッド、search() メソッド、splice() メソッドがあります。

test() メソッド

RegExp クラスの test() メソッドは、次の例で示すように、指定したストリング内に正規表現に一致する部分があるかどうかを単に判定します。

```
var pattern:RegExp = /Class-\w/;
var str = "Class-A";
trace(pattern.test(str)); // 出力 : true
```

exec() メソッド

RegExp クラスの exec() メソッドは、指定したストリング内に正規表現に一致する部分があるかどうかを調べ、次の内容を格納した配列を返します。

- パターンに一致したサブストリング
- 正規表現に含まれるいずれかのグループ化括弧に一致したサブストリング

また、この配列には、サブストリングの一致を開始したインデックス位置を示す index プロパティがあります。

たとえば、次のようなコードがあるとします。

```
var pattern:RegExp = /\d{3}\-\d{3}-\d{4}/; // 米国の電話番号
var str:String = "phone: 415-555-1212";
var result:Array = pattern.exec(str);
trace(result.index, " - ", result);
// 7  -  415-555-1212
```

正規表現に g (global) フラグを設定して、exec() メソッドを繰り返し呼び出すことにより、次のように複数のサブストリングについて一致を調べることができます。

```
var pattern:RegExp = /\w*sh\w*/gi;
var str:String = "She sells seashells by the seashore";
var result:Array = pattern.exec(str);

while (result != null)
{
```

```
    trace(result.index, "\t", pattern.lastIndex, "\t", result);
    result = pattern.exec(str);
}
// 出力 :
// 0  3  She
// 10  19  seashells
// 27  35  seashore
```

RegExp パラメータを使用する String のメソッド

正規表現のパラメータを指定できる String クラスのメソッドとしては、`match()` メソッド、`replace()` メソッド、`search()` メソッドおよび `split()` があります。これらのメソッドの詳細については、216 ページの「ストリング内のパターンの検索およびサブストリングの置換」を参照してください。

例 : Wiki パーサー

この単純な Wiki テキスト変換の例で、正規表現の使用方法をいくつか示します。

- ソースの Wiki パターンに一致するテキスト行を適切な HTML 出力ストリングに変換する。
- 正規表現を使用して、URL パターンを HTML の `<a>` ハイパーリンクタグに変換する。
- 正規表現を使用して、米ドルストリング (`"$9.95"` など) をユーロストリング (`"8.24 €"` など) に変換する。

WikiEditor アプリケーションのファイルは、"Samples/WikiEditor" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
WikiEditor.mxml	MXML で記述された Flex 用メインアプリケーションファイル
com/example/programmingas3/regExpExamples/WikiParser.as	正規表現を使用して Wiki 入力テキストパターンを同等の HTML 出力に変換するメソッドが含まれているクラス
com/example/programmingas3/regExpExamples/URLParser.as	正規表現を使用して URL ストリングを HTML の `<a>` ハイパーリンクタグに変換するメソッドが含まれているクラス
com/example/programmingas3/regExpExamples/CurrencyConverter.as	正規表現を使用して米ドルストリングをユーロストリングに変換するメソッドが含まれているクラス

WikiParser クラスの定義

WikiParser クラスには、Wiki 入力テキストを同等の HTML 出力に変換するメソッドが含まれます。この Wiki 変換アプリケーションは堅牢ではありませんが、パターンマッチングおよびストリング変換を行う正規表現の使用方法を示します。

コンストラクタ関数は setWikiData() メソッドと共に、Wiki 入力テキストのサンプルストリングを次のように単純に初期化します。

```
public function WikiParser()
{
    wikiData = setWikiData();
}
```

ユーザーがサンプルアプリケーションの [Test] ボタンをクリックすると、アプリケーションは WikiParser オブジェクトの parseWikiString() メソッドを呼び出します。このメソッドは、他の複数のメソッドを呼び出し、結果の HTML ストリングを組み立てます。

```
public function parseWikiString(wikiString:String):String
{
    var result:String = parseBold(wikiString);
    result = parseItalic(result);
    result = linesToParagraphs(result);
    result = parseBullets(result);
    return result;
}
```

呼び出されるメソッド (parseBold()、parseItalic()、linesToParagraphs()、parseBullets()) は、入力 Wiki テキストを HTML 形式のテキストに変換するために、ストリングの replace() メソッドを使用して、正規表現で定義された一致パターンを置き換えます。

ボールドおよびイタリックのパターンの変換

parseBold() メソッドは、Wiki のボールドテキストパターン ('''foo''' など) を探し、次のように、同等の HTML (foo など) に変換します。

```
private function parseBold(input:String):String
{
    var pattern:RegExp = /'''(.*?)'''/g;
    return input.replace(pattern, "<b>$1</b>");
}
```

正規表現の (.?*) の部分は、2 つの定義パターン ''' の間にある任意の数の文字 (*) に一致します。? 繰り返し制御文字を使用すると、最短一致でマッチングが行われます。そのため、'''aaa''' bbb '''ccc''' のようなストリングの場合、最初に一致するストリングは '''aaa''' になります。''' パターンで開始し、終了するストリング全体ではありません。

正規表現内の括弧は、キャプチャグループを定義します。replace() メソッドは、置換ストリングの $1 コードを使用して、このグループを参照します。正規表現の g (global) フラグによって、replace() メソッドは (最初の 1 つだけではなく) ストリング内のすべての一致を置き換えます。

parseItalic() メソッドは parseBold() メソッドと同じように動作しますが、イタリックテキストの区切り記号として 2 つのアポストロフィ ('') を探します (3 つではありません)。

```
private function parseItalic(input:String):String
{
   var pattern:RegExp = /''(.*?)''/g;
   return input.replace(pattern, "<i>$1</i>");
}
```

箇条書きパターンの変換

次の例で示すように、parseBullet() メソッドは Wiki の箇条書き行パターン (* foo など) を探し、同等の HTML (
foo など) に変換します。

```
private function parseBullets(input:String):String
{
   var pattern:RegExp = /^\*(.*)/gm;
   return input.replace(pattern, "<li>$1</li>");
}
```

正規表現の先頭の ^ 記号は、行の先頭に一致します。正規表現で m (multiline) フラグを使用した場合、^ 記号は、単純にストリングの先頭ではなく、行頭とマッチングされます。

* パターンは、アスタリスク文字に一致します (* 繰り返し制御文字ではなく、リテラルのアスタリスクであることを示すために、円記号を使用します)。

正規表現内の括弧は、キャプチャグループを定義します。replace() メソッドは、置換ストリングの $1 コードを使用して、このグループを参照します。正規表現の g (global) フラグによって、replace() メソッドは (最初の 1 つだけではなく) ストリング内のすべての一致を置き換えます。

Wiki 段落パターンの変換

linesToParagraphs() メソッドは、入力 Wiki ストリングの各行を HTML の <p> 段落タグに変換します。メソッド内の次の行によって、入力 Wiki ストリングから空の行を削除します。

```
var pattern:RegExp = /^$/gm;
var result:String = input.replace(pattern, "");
```

正規表現の ^ 記号は行の先頭に一致し、$ 記号は行の末尾に一致します。正規表現で m (multiline) フラグを使用した場合、^ 記号は単純にストリングの先頭ではなく、行頭とマッチングされます。

replace() メソッドは、すべての一致するサブストリング (空の行) を空のストリング ("") で置き換えます。正規表現の g (global) フラグによって、replace() メソッドは (最初の 1 つだけではなく) ストリング内のすべての一致を置き換えます。

URL から HTML の <a> タグへの変換

ユーザーがサンプルアプリケーションの [Test] ボタンをクリックしたとき、[urlToATag] チェックボックスがオンであると、アプリケーションは URLParser.urlToATag() 静的メソッドを呼び出して、入力 Wiki ストリングの URL ストリングを HTML の <a> タグに変換します。

```
var protocol:String = "((?:http|ftp)://)";
var urlPart:String = "([a-z0-9_-]+\.[a-z0-9_-]+)";
var optionalUrlPart:String = "(\.[a-z0-9_-]*)";
var urlPattern:RegExp = new RegExp(protocol + urlPart + optionalUrlPart,
                        "ig");
var result:String = input.replace(urlPattern,
                "<a href='$1$2$3'><u>$1$2$3</u></a>");
```

RegExp() コンストラクタ関数を使用して、複数の要素部分から得られた正規表現を組み立てます (urlPattern)。これらの要素部分は、正規表現パターンの一部を定義するストリングです。

正規表現パターンの最初の部分は、protocol ストリングで定義され、URL プロトコル (http:// または ftp://) を定義します。括弧は ? 記号で示され、非キャプチャグループを定義します。これは、括弧が | 選択制御パターン用のグループの定義にのみ使用されるという意味です。このグループは replace() メソッドの置換ストリングの後方参照コード ($1、$2、$3) に一致しません。

正規表現のその他の要素部分は、それぞれキャプチャグループを使用し (パターン内の括弧で示されます)、replace() メソッドの置換ストリングの後方参照コード ($1、$2、$3) で使用されます。

urlPart ストリングで定義されたパターン部分は、a 〜 z、0 〜 9、_ または - の 1 つ以上に一致します。+ 繰り返し制御文字は、少なくとも 1 つの文字に一致することを示します。\. は、必須のドット (.) 文字を示します。残りは、a 〜 z、0 〜 9、_ または - の 1 つ以上で構成される別のストリングに一致します。

optionalUrlPart ストリングで定義されたパターン部分は、0 個以上の任意の数の英数字 (_ と - を含む) が続くドット (.) 文字に一致します。* 繰り返し制御文字は、0 個以上の文字が一致することを示します。

replace() メソッドの呼び出しでは、正規表現を使用し、後方参照を使用して置換 HTML ストリングが組み立てられます。

urlToATag() メソッドは、次に emailToATag() メソッドを呼び出します。このメソッドは、似た手法を使用して、電子メールパターンを HTML の <a> ハイパーリンクストリングに置き換えます。このサンプルファイルで HTTP、FTP、および電子メールの URL とのマッチングに使用する正規表現は、例を示すことが目的なので単純化されています。より正確にこれらの URL とマッチングするには、より複雑な正規表現を使用します。

米ドルストリングからユーロストリングへの変換

ユーザーがサンプルアプリケーションの [Test] ボタンをクリックしたとき、[dollarToEuro] チェックボックスがオンであると、アプリケーションは CurrencyConverter.usdToEuro() 静的メソッドを呼び出して、次のように米ドルストリング ("$9.95" など) をユーロストリング ("8.24 €" など) に変換します。

```
var usdPrice:RegExp = /\$([\d,]+.\d+)+/g;
return input.replace(usdPrice, usdStrToEuroStr);
```

最初の行で、米ドルストリングに一致する単純なパターンを定義します。$ 文字の前にバックスラッシュ (\) エスケープ文字が付いていることに注意してください。

replace() メソッドは、正規表現を使用してパターンマッチングを行い、usdStrToEuroStr() 関数を呼び出して置換ストリング (ユーロでの値) を決定します。

関数名を replace() メソッドの第 2 パラメータとして使用すると、次の要素がパラメータとして呼び出し元関数に渡されます。

- ストリング内の一致する部分。
- グループ化括弧によってキャプチャされた一致。この方法で渡される引数の数は、グループ化括弧によってキャプチャされた一致数によって異なります。グループ化括弧によってキャプチャされた一致数は、関数コード内の arguments.length - 3 を確認して特定することができます。
- ストリング内で一致部分が始まる場所のインデックス位置。
- ストリング全体。

usdStrToEuroStr() メソッドは、次のように米ドルストリングパターンをユーロストリングに変換します。

```
private function usdToEuro(...args):String
{
    var usd:String = args[1];
    usd = usd.replace(",", "");
    var exchangeRate:Number = 0.828017;
    var euro:Number = Number(usd) * exchangeRate;
    trace(usd, Number(usd), euro);
    const euroSymbol:String = String.fromCharCode(8364); // €
    return euro.toFixed(2) + " " + euroSymbol;
}
```

`args[1]` は、`usdPrice` 正規表現に一致した、キャプチャされたグループ化括弧を表します。これは、米ドルストリングの数字の部分、つまり $ 記号のないドル金額です。メソッドは、為替レート変換を適用し、結果のストリングを返します (前に $ 記号を付ける代わりに、後ろに € 記号を付けます)。

第 11 章

XML の操作

11

ActionScript 3.0 には、ECMAScript for XML (E4X) 仕様 (ECMA-357 Edition 2) に基づいたクラス群が含まれます。これらのクラスは、XML データの処理に関する強力で扱いやすい機能を備えています。E4X を使用すれば、XML データを処理するコードを、従来のプログラミングテクニックと比べて短期間で開発できます。しかも、コードの読みやすさも従来より向上します。

この章では、E4X による XML データの処理方法について説明します。

目次

XML の基礎 312
XML 処理の E4X アプローチ 313
XML オブジェクト 315
XMLList オブジェクト 317
XML 変数の初期化 319
XML オブジェクトの構築と変形 320
XML 階層構造へのアクセス 322
XML 名前空間の使用 327
XML の型変換 328
外部 XML ドキュメントの読み込み 330
例 : インターネットから RSS データをロードする 330

XML の基礎

この章は、読者が XML の基本的な概念を理解していることを前提としています。しかし、この章に記載した情報を参考にすれば、XML の初心者でも基本的な XML メソッドを使い始めることができます。ここでは、基礎を説明するための例として次のような XML ドキュメントを想定します。

```
<order xmlns = "http://www.example.com/xml">
  <book ISBN="0942407296">
    <title>Baking Extravagant Pastries with Kumquats</title>
    <author>
      <lastName>Contino</lastName>
      <firstName>Chuck</firstName>
    </author>
    <pageCount>238</pageCount>
  </book>
  <book ISBN="0865436401">
    <title>Emu Care and Breeding</title>
    <editor>
      <lastName>Case</lastName>
      <firstName>Justin</firstName>
    </editor>
    <pageCount>115</pageCount>
  </book>
</order>
```

サンプルの XML ドキュメントは、2 つの book " エレメント " (" ノード ") を含んでいます。第1の book エレメントには、それぞれ title、author、pageCount という " 名前 " を持つ 3 つの " 子エレメント " があります。また、author エレメント内には 2 つの子エレメントがあります。第1の book エレメントには ISBN という " 属性 " があり、その値は "0942407296" です。第1の book エレメントに含まれる " 内容 " は、このエレメントに属する子エレメントのコレクションです。firstName エレメントの内容は、"Chuck" というテキストです。この XML ドキュメント全体は、http://www.example.com/xml という架空の URL で定義されているデフォルトの " 名前空間 " を持ちます。このタイプの XML ドキュメントで使用するスキーマは、この名前空間によって定義されます。

ActionScript 3.0 には、XML データの操作に使用する新しい演算子が導入されています。次の例に登場するドット (.) や属性識別用の接頭辞 (@) もその一部です。前述のサンプル XML データを order1 というオブジェクトに割り当てる場合、有効なコードは次のようになります。

```
trace(order1.book[0].author.firstName); // Chuck
trace(order1.book.(@ISBN=="0865436401").pageCount; // 115
delete order1.book[0];
```

新しい演算子と、他の新しい E4X クラス、メソッド、プロパティについては、この章で説明します。

サーバーサイドアプリケーションではデータの構造化に XML が使用されることが多いため、ActionScript で XML クラスを使用することにより、Web サービスへの接続などを行う高度なインターネットアプリケーションを作成できます。Web サービスは複数のアプリケーション間 (たとえば、Adobe Flash Player 9 アプリケーションと Web サーバー上のアプリケーション) を接続する手段の 1 つであり、SOAP (Simple Object Access Protocol) などの一般的な標準規格に基づいています。

XML 処理の E4X アプローチ

ECMAScript for XML 仕様には、XML データを処理するためのクラス群や機能が定義されています。それらは総称して E4X と呼ばれています。ActionScript 3.0 には、XML、XMLList、QName、Namespace の E4X クラスが用意されています。

E4X クラスのメソッド、プロパティ、演算子についての策定作業では、次の特徴を実現することが目標とされています。

- 簡易であること : E4X では、XML データを処理するコードをできるだけ簡単に作成および解読できるようになっています。
- 一貫性があること : E4X で採用されているメソッドや考え方は、E4X それ自体の中で首尾一貫しており、また、ActionScript の他の部分とも一貫性があります。
- なじみやすいこと : XML データの操作には、よく知られているドット (.) などの演算子を使用します。

メモ

ActionScript 2.0 に存在していた従来の XML クラスは、E4X 仕様の一部である ActionScript 3.0 XML クラスと競合するので、ActionScript 3.0 では XMLDocument というクラス名に変更されました。XMLDocument、XMLNode、XMLParser、XMLTag の各クラスは、ActionScript 3.0 では、主にレガシー機能のサポートを目的として flash.xml パッケージに含められています。E4X の新しいクラスはコアクラスなので、使用の際にパッケージを読み込む必要はありません。この章では、ActionScript 2.0 のレガシー XML クラスについては詳しく説明しません。これらの詳細については、『ActionScript 3.0 リファレンスガイド』の flash.xml パッケージを参照してください。

E4X を使用したデータ処理の例を次に示します。

```
var myXML:XML =
  <order>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

XML データは、実際のアプリケーションでは Web サービスや RSS フィードなどの外部ソースからロードすることが多いと考えられますが、ここでは、例を単純にするために XML データをリテラルで割り当てます。

次のコードに示すとおり、E4X には、XML のプロパティや属性へのアクセスに使用するドット (.) や属性識別用の接頭辞 (@) などのわかりやすい演算子が用意されています。

```
trace(myXML.item[0].menuName); // Output: burger
trace(myXML.item.(@id==2).menuName); // Output: fries
trace(myXML.item.(menuName=="burger").price); // Output: 3.95
```

XML のノードに新しい子ノードを割り当てるには、次のように appendChild() メソッドを使用します。

```
var newItem:XML =
  <item id="3">
    <menuName>medium cola</menuName>
    <price>1.25</price>
  </item>

myXML.appendChild(newItem);
```

@ および . 演算子は、読み取りだけでなく、次のようにデータの割り当てにも使用できます。

```
myXML.item[0].menuName="regular burger";
myXML.item[1].menuName="small fries";
myXML.item[2].menuName="medium cola";

myXML.item.(menuName=="regular burger").@quantity = "2";
myXML.item.(menuName=="small fries").@quantity = "2";
myXML.item.(menuName=="medium cola").@quantity = "2";
```

次のように for ループを使用すると、XML の一連のノードに対する繰り返し処理ができます。

```
var total:Number = 0;
for each (var property:XML in myXML.item)
{
  var q:int = Number(property.@quantity);
  var p:Number = Number(property.price);
  var itemTotal:Number = q * p;
  total += itemTotal;
  trace(q + " " + property.menuName + " $" + itemTotal.toFixed(2))
}
trace("Total: $", total.toFixed(2));
```

XML オブジェクト

1 つの XML オブジェクトは、1 つの XML エレメント、属性、コメント、処理命令、またはテキストエレメントを表します。

XML オブジェクトは、" 単純内容 " を持つものと " 複合内容 " を持つものに分類されます。複合内容の XML オブジェクトとは、子ノードを持つ XML オブジェクトです。XML オブジェクトが、属性、コメント、処理命令、またはテキストノードのいずれかである場合、このオブジェクトには単純内容が含まれます。

たとえば、次の XML オブジェクトは、コメントと処理命令を含んだ複合内容の XML オブジェクトです。

```
XML.ignoreComments = false;
XML.ignoreProcessingInstructions = false;
var x1:XML =
  <order>
    <!-- これはコメントです -->
    <?PROC_INSTR sample ?>
    <item id='1'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
    <item id='2'>
      <menuName>fries</menuName>
      <price>1.45</price>
    </item>
  </order>
```

次の例に示すとおり、`comments()` および `processingInstructions()` メソッドを使用して、コメントおよび処理命令の新しい XML オブジェクトを作成できるようになりました。

```
var x2:XML = x1.comments()[0];
var x3:XML = x1.processingInstructions()[0];
```

XML プロパティ

XML クラスには次の 5 つの静的プロパティがあります。

- `ignoreComments` および `ignoreProcessingInstructions` プロパティでは、XML オブジェクトの解析時にコメントまたは処理命令を無視するかどうかを指定します。
- `ignoreWhitespace` プロパティでは、空白文字だけで区切られているエレメントタグや埋め込み式について、空白文字を無視するかどうかを指定します。
- `prettyIndent` および `prettyPrinting` プロパティは、XML クラスの `toString()` および `toXMLString()` メソッドが返すテキストを整形するために使用します。

これらのプロパティの詳細については、『ActionScript 3.0 リファレンスガイド』を参照してください。

XML メソッド

次の各メソッドは、XML オブジェクトの階層構造に関する操作に使用します。

- `appendChild()`
- `child()`
- `childIndex()`
- `children()`
- `descendants()`
- `elements()`
- `insertChildAfter()`
- `insertChildBefore()`
- `parent()`
- `prependChild()`

次の各メソッドは、XML オブジェクトの属性に関する操作に使用します。

- `attribute()`
- `attributes()`

次の各メソッドは、XML オブジェクトのプロパティに関する操作に使用します。

- `hasOwnProperty()`
- `propertyIsEnumerable()`
- `replace()`
- `setChildren()`

次の各メソッドは、修飾名および名前空間に関する操作に使用します。

- `addNamespace()`
- `inScopeNamespaces()`
- `localName()`
- `name()`
- `namespace()`
- `namespaceDeclarations()`
- `removeNamespace()`
- `setLocalName()`
- `setName()`
- `setNamespace()`

次の各メソッドは、XML の内容のタイプに関する操作および判定に使用します。

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `nodeKind()`
- `processingInstructions()`
- `text()`

次の各メソッドは、XML オブジェクトからストリングへの変換および整形に使用します。

- `defaultSettings()`
- `setSettings()`
- `settings()`
- `normalize()`
- `toString()`
- `toXMLString()`

次の各メソッドはその他の用途に使用します。

- `contains()`
- `copy()`
- `valueOf()`
- `length()`

これらのメソッドの詳細については、『ActionScript 3.0 リファレンスガイド』を参照してください。

XMLList オブジェクト

XMLList インスタンスは、XML オブジェクトの任意のコレクションを表し、完全な XML ドキュメント、XML フラグメント、または XML クエリの結果を含むことができます。

次の各メソッドは、XMLList オブジェクトの階層構造に関する操作に使用します。

- `child()`
- `children()`
- `descendants()`
- `elements()`
- `parent()`

次の各メソッドは、XMLList オブジェクトの属性に関する操作に使用します。

- `attribute()`
- `attributes()`

次の各メソッドは、XMLList オブジェクトのプロパティに関する操作に使用します。

- `hasOwnProperty()`
- `propertyIsEnumerable()`

次の各メソッドは、XML の内容のタイプに関する操作および判定に使用します。

- `comments()`
- `hasComplexContent()`
- `hasSimpleContent()`
- `processingInstructions()`
- `text()`

次の各メソッドは、XMLList オブジェクトからストリングへの変換および整形に使用します。

- `normalize()`
- `toString()`
- `toXMLString()`

次の各メソッドはその他の用途に使用します。

- `contains()`
- `copy()`
- `length()`
- `valueOf()`

これらのメソッドの詳細については、『ActionScript 3.0 リファレンスガイド』を参照してください。

XML エレメントを 1 つだけ含んでいる XMLList オブジェクトに対しては、XML クラスのプロパティとメソッドをすべて使用できます。これは、XML エレメントを 1 つだけ含んでいる XMLList オブジェクトは XML オブジェクト 1 つと同等に扱われるためです。たとえば、次のコードにある `doc.div` は、エレメントを 1 つだけ含んでいる XMLList オブジェクトなので、XML クラスの `appendChild()` メソッドを使用できます。

```
var doc:XML =
     <body>
       <div>
         <p>Hello</p>
       </div>
     </body>;
doc.div.appendChild(<p>World</p>);
```

XML プロパティおよびメソッドの一覧については、315 ページの「XML オブジェクト」を参照してください。

XML 変数の初期化

XML オブジェクトに対しては、次のようにして XML リテラルを割り当てることができます。

```
var myXML:XML =
    <order>
        <item id='1'>
            <menuName>burger</menuName>
            <price>3.95</price>
        </item>
        <item id='2'>
            <menuName>fries</menuName>
            <price>1.45</price>
        </item>
    </order>
```

また、次のコードに示すとおり、XML データを含んだストリングから new コンストラクタを使用して XML オブジェクトのインスタンスを作成することもできます。

```
var str:String = "<order><item id='1'><menuName>burger</menuName>"
                + "<price>3.95</price></item></order>";
var myXML:XML = new XML(str);
```

ストリング内の XML データが整形式でない場合 (閉じるタグが欠落している場合など) は、ランタイムエラーが発生します。

また、次のように XML オブジェクトに対してデータを参照で (他の変数から) 渡すこともできます。

```
var tagname:String = "item";
var attributename:String = "id";
var attributevalue:String = “5”;
var content:String = "Chicken";
var x:XML = <{tagname} {attributename}={attributevalue}>{content}</{tagname}>;
trace(x.toXMLString())
    // Output: <item id="5">Chicken</item>
```

指定した URL から XML データをロードするには、次のように URLLoader クラスを使用します。

```
import flash.events.Event;
import flash.net.URLLoader;
import flash.net.URLRequest;

var externalXML:XML;
var loader:URLLoader = new URLLoader();
var request:URLRequest = new URLRequest("xmlFile.xml");
loader.load(request);
loader.addEventListener(Event.COMPLETE, onComplete);

function onComplete(event:Event):void
{
```

```
    var loader:URLLoader = event.target as URLLoader;
    if (loader != null)
    {
        externalXML = new XML(loader.data);
        trace(externalXML.toXMLString());
    }
    else
    {
        trace("loader is not a URLLoader!");
    }
}
```

ソケット接続から XML データを読み取るには、XMLSocket クラスを使用します。詳細については、『ActionScript 3.0 リファレンスガイド』の XMLSocket を参照してください。

XML オブジェクトの構築と変形

XML オブジェクトが持つプロパティリストの先頭または末尾にプロパティを追加するには、次のように prependChild() メソッドまたは appendChild() メソッドを使用します。

```
var x1:XML = <p>Line 1</p>
var x2:XML = <p>Line 2</p>
var x:XML = <body></body>
x = x.appendChild(x1);
x = x.appendChild(x2);
x = x.prependChild(<p>Line 0</p>);
    // x == <body><p>Line 0</p><p>Line 1</p><p>Line 2</p></body>
```

特定のプロパティの直前または直後にプロパティを追加するには、次のように insertChildBefore() メソッドまたは insertChildAfter() メソッドを使用します。

```
var x:XML =
    <body>
        <p>Paragraph 1</p>
        <p>Paragraph 2</p>
    </body>
var newNode:XML = <p>Paragraph 1.5</p>
x = x.insertChildAfter(x.p[0], newNode)
x = x.insertChildBefore(x.p[2], <p>Paragraph 1.75</p>)
```

また、XML オブジェクトの作成時に次のようにして中括弧演算子 ({ および }) を使用し、データを参照で (他の変数から) 渡すこともできます。

```
var ids:Array = [121, 122, 123];
var names:Array = [["Murphy","Pat"], ["Thibaut","Jean"], ["Smith","Vijay"]]
var x:XML = new XML("<employeeList></employeeList>");

for (var i:int = 0; i < 3; i++)
{
  var newnode:XML = new XML();
  newnode =
    <employee id={ids[i]}>
      <last>{names[i][0]}</last>
      <first>{names[i][1]}</first>
    </employee>;

  x = x.appendChild(newnode)
}
```

XML オブジェクトにプロパティまたは属性を割り当てるには、次のように = 演算子を使用します。

```
var x:XML =
  <employee>
    <lastname>Smith</lastname>
  </employee>
x.firstname = "Jean";
x.@id = "239";
```

これを実行すると、x という XML オブジェクトは次のように設定されます。

```
<employee id="239">
  <lastname>Smith</lastname>
  <firstname>Jean</firstname>
</employee>
```

+ 演算子および += 演算子を使用すると、次のように複数の XMLList オブジェクトを連結できます。

```
var x1:XML = <a>test1</a>
var x2:XML = <b>test2</b>
var xList:XMLList = x1 + x2;
xList += <c>test3</c>
```

これを実行すると、xList という XMLList オブジェクトは次のように設定されます。

```
<a>test1</a>
<b>test2</b>
<c>test3</c>
```

XML 階層構造へのアクセス

XML が備えている強力な機能の 1 つは、複雑にネストされたデータを、テキストの文字による連続的なストリングによって表現できることです。データを XML オブジェクトにロードすると ActionScript によって解析され、階層構造のデータとしてメモリにロードされます (XML データが整形式でない場合はランタイムエラーが発生します)。

XML オブジェクトと XMLList オブジェクトの演算子やメソッドを使用すれば、この XML データの構造に簡単にアクセスできます。

XML オブジェクトの子プロパティにアクセスするには、ドット (.) 演算子と子孫アクセサ (..) 演算子を使用します。たとえば、次のような XML オブジェクトがあるとします。

```
var myXML:XML =
  <order>
    <book ISBN="0942407296">
      <title>Baking Extravagant Pastries with Kumquats</title>
      <author>
        <lastName>Contino</lastName>
        <firstName>Chuck</firstName>
      </author>
      <pageCount>238</pageCount>
    </book>
    <book ISBN="0865436401">
      <title>Emu Care and Breeding</title>
      <editor>
        <lastName>Case</lastName>
        <firstName>Justin</firstName>
      </editor>
      <pageCount>115</pageCount>
    </book>
  </order>
```

オブジェクト myXML.book は XMLList オブジェクトであり、myXML オブジェクトの book という名前を持つ子プロパティを含んでいます。それらは、myXML オブジェクトが持つ 2 つの book プロパティに対応する 2 つの XML オブジェクトです。

オブジェクト myXML..lastName は XMLList オブジェクトであり、lastName という名前を持つすべての子孫プロパティを含んでいます。それらは、myXML オブジェクトが持つ 2 つの lastName プロパティに対応する 2 つの XML オブジェクトです。

オブジェクト myXML.book.editor.lastName は XMLList オブジェクトであり、myXML オブジェクトの book という子の editor という子の、lastName という名前を持つすべての子を含んでいます。この場合、これは XML オブジェクトを 1 つだけ含んだ XMLList オブジェクトです (値が "Case" である lastName プロパティ)。

親ノードおよび子ノードへのアクセス

parent() メソッドは、XML オブジェクトの親を返します。

子リストでは、序数のインデックス値を指定することにより特定の子オブジェクトにアクセスできます。たとえば、XML オブジェクト myXML に book という名前の子プロパティが 2 つある場合、book という名前の各子プロパティには、次のようにインデックス番号が対応しています。

```
myXML.book[0]
myXML.book[1]
```

特定の孫にアクセスするには、子の名前と孫の名前の両方に対してインデックス番号を指定します。

```
myXML.book[0].title[0]
```

ただし、x.book[0] に title という子が 1 つしかない場合は、次のようにインデックス参照を省略できます。

```
myXML.book[0].title
```

同じように、x に book という子が 1 つしかなく、それに title オブジェクトが 1 つしか含まれない場合は、次のように両方のインデックス参照を省略できます。

```
myXML.book.title
```

変数や式で子の名前を指定してアクセスするには、次のように child() メソッドを使用します。

```
var myXML:XML =
     <order>
        <book>
           <title>Dictionary</title>
        </book>
     </order>;

var childName:String = "book";

trace(myXML.child(childName).title) // output: Dictionary
```

属性へのアクセス

XML オブジェクトまたは XMLList オブジェクトの属性にアクセスするには、次のように @ シンボル (属性識別用の接頭辞) を使用します。

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@id); // 6401
```

XML オブジェクトまたは XMLList オブジェクトが持つすべての属性にアクセスするには、次のように、@ シンボルにワイルドカードの * シンボルを付けます。

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.@*.toXMLString());
// 6401
// 233
```

また、次のように attribute() または attributes() メソッドを使用して、XML オブジェクトまたは XMLList オブジェクトが持つ特定の属性またはすべての属性にアクセスすることもできます。

```
var employee:XML =
  <employee id="6401" code="233">
    <lastName>Wu</lastName>
    <firstName>Erin</firstName>
  </employee>;
trace(employee.attribute("id")); // 6401
trace(employee.attribute("*").toXMLString());
// 6401
// 233
trace(employee.attributes().toXMLString());
// 6401
// 233
```

メモ : 属性へのアクセスには次のシンタックスを使用することもできます。その場合は下の例のようになります。

```
employee.attribute("id")
employee["@id"]
employee.@["id"]
```

これらはいずれも employee.@id と同等ですが、employee.@id のシンタックスを使用することをお勧めします。

属性またはエレメントの値によるフィルタ処理

括弧演算子の (および) を使用すると、エレメント名または属性値を指定してエレメントをフィルタ処理できます。たとえば、次のような XML オブジェクトがあるとします。

```
var x:XML =
  <employeeList>
    <employee id="347">
      <lastName>Zmed</lastName>
      <firstName>Sue</firstName>
      <position>Data analyst</position>
    </employee>
    <employee id="348">
      <lastName>McGee</lastName>
      <firstName>Chuck</firstName>
      <position>Jr. data analyst</position>
    </employee>
  </employeeList>
```

この場合、次の式はいずれも有効なコードです。

- x.employee.(lastName == "McGee"):2 番目の employee ノード
- x.employee.(lastName == "McGee").firstName:2 番目の employee ノードが持つ firstName プロパティ
- x.employee.(lastName == "McGee").@id:2 番目の employee ノードが持つ id 属性の値
- x.employee.(@id == 347):最初の employee ノード
- x.employee.(@id == 347).lastName:最初の employee ノードが持つ lastName プロパティ
- x.employee.(@id > 300):両方の employee プロパティを含んだ XMLList
- x.employee.(position.toString().search("analyst") > -1):両方の position プロパティを含んだ XMLList

存在しない属性またはエレメントを指定してフィルタ処理を実行しようとすると、Flash Player により例外がスローされます。たとえば、次の例ではコードの最終行でエラーが発生します。これは、2 番目の p エレメントに id 属性がないためです。

```
var doc:XML =
        <body>
          <p id='123'>Hello, <b>Bob</b>.</p>
          <p>Hello.</p>
        </body>;
trace(doc.p.(@id == '123'));
```

同様に、次の例ではコードの最終行でエラーが発生します。これは、2 番目の p エレメントの b プロパティがないためです。

```
var doc:XML =
        <body>
           <p id='123'>Hello, <b>Bob</b>.</p>
           <p>Hello.</p>
        </body>;
trace(doc.p.(b == 'Bob'));
```

このようなエラーを回避するには、次のコードのように attribute() メソッドおよび elements() メソッドを使用して、一致する属性またはエレメントを持つプロパティを特定します。

```
var doc:XML =
        <body>
           <p id='123'>Hello, <b>Bob</b>.</p>
           <p>Hello.</p>
        </body>;
trace(doc.p.(attribute('id') == '123'));
trace(doc.p.(elements('b') == 'Bob'));
```

次のように、hasOwnProperty() メソッドを使用することもできます。

```
var doc:XML =
        <body>
           <p id='123'>Hello, <b>Bob</b>.</p>
           <p>Hello.</p>
        </body>;
trace(doc.p.(hasOwnProperty('@id') && @id == '123'));
trace(doc.p.(hasOwnProperty('b') && b == 'Bob'));
```

for..in および for each..in ステートメントの使用

XMLList オブジェクトに対して繰り返し処理を実行する場合、ActionScript 3.0 では for..in ステートメントと for each..in ステートメントを使用できます。たとえば、次のような XML オブジェクト myXML と XMLList オブジェクト myXML.item があるとします。XMLList オブジェクトの myXML.item は、XML オブジェクトの item というノード 2 つで構成されています。

```
var myXML:XML =
   <order>
      <item id='1' quantity='2'>
         <menuName>burger</menuName>
         <price>3.95</price>
      </item>
      <item id='2' quantity='2'>
         <menuName>fries</menuName>
         <price>1.45</price>
      </item>
   </order>;
```

XMLList が持つプロパティ名のセットに対し、`for..in` ステートメントを使用して繰り返し処理を実行するには、次のようにします。

```
var total:Number = 0;
for (var pname:String in myXML.item)
{
  total += myXML.item.@quantity[pname] * myXML.item.price[pname];
}
```

XMLList が持つ各種のプロパティ名に対し、`for each..in` ステートメントを使用して繰り返し処理を実行するには、次のようにします。

```
var total2:Number = 0;
for each (var prop:XML in myXML.item)
{
  total2 += prop.@quantity * prop.price;
}
```

XML 名前空間の使用

XML オブジェクト (またはドキュメント) に含まれるデータの種類は、そのオブジェクトの名前空間によって特定されます。たとえば、メッセージ通信プロトコルとして SOAP を使用する Web サービスに対して XML データを送信または提供する場合は、その XML の開始タグにおいて次のように名前空間を宣言します。

```
var message:XML =
  <soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
  soap:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
    <soap:Body xmlns:w="http://www.test.com/weather/">
      <w:getWeatherResponse>
        <w:tempurature >78</w:tempurature>
      </w:getWeatherResponse>
    </soap:Body>
  </soap:Envelope>;
```

この名前空間は `soap` という接頭辞を持ち、`http://schemas.xmlsoap.org/soap/envelope/` という URI において定義されています。

ActionScript 3.0 には、XML の名前空間を扱うための Namespace クラスが用意されています。前の例に示した XML オブジェクトに対する Namespace クラスの使用例を次に示します。

```
var soapNS:Namespace = message.namespace("soap");
trace(soapNS); // Output: http://schemas.xmlsoap.org/soap/envelope/

var wNS:Namespace = new Namespace("w", "http://www.test.com/weather/");
message.addNamespace(wNS);
var encodingStyle:XMLList = message.@soapNS::encodingStyle;
var body:XMLList = message.soapNS::Body;

message.soapNS::Body.wNS::GetWeatherResponse.wNS::tempurature = "78";
```

XML クラスには、名前空間の扱いに関するメソッドとして、addNamespace()、inScopeNamespaces()、localName()、name()、namespace()、namespaceDeclarations()、removeNamespace()、setLocalName()、setName()、および setNamespace() があります。

default xml namespace ディレクティブは、XML オブジェクトのデフォルトの名前空間を割り当てるために使用します。たとえば、次の例では、x1 と x2 がいずれも同じデフォルトの名前空間を持ちます。

```
var ns1:Namespace = new Namespace("http://www.example.com/namespaces/");

default xml namespace = ns1;
var x1:XML = <test1 />;
var x2:XML = <test2 />;
```

XML の型変換

XML オブジェクトと XMLList オブジェクトは、String 値に変換できます。同様に、ストリングを XML オブジェクトまたは XMLList オブジェクトに変換することもできます。また、XML の属性値、名前、テキスト値はすべてストリングです。以降のセクションでは、これらすべてに関する XML 型変換について説明します。

XML および XMLList オブジェクトからストリングへの変換

XML クラスと XMLList クラスには、toString() メソッドと toXMLString() メソッドがあります。toXMLString() メソッドは、XML オブジェクトに格納されているすべてのタグ、属性、名前空間宣言、および内容を含んだストリングを返します。複合内容 (子エレメント) を持つ XML オブジェクトの場合、toString() メソッドは toXMLString() メソッドとまったく同じように機能します。単純内容の (テキストエレメントだけを含む) XML オブジェクトの場合、toString() メソッドは、次のように当該エレメントのテキストの内容だけを返します。

```
var myXML:XML =
  <order>
    <item id='1' quantity='2'>
      <menuName>burger</menuName>
      <price>3.95</price>
    </item>
  <order>;

trace(myXML.item[0].menuName.toXMLString());
  // <menuName>burger</menuName>
trace(myXML.item[0].menuName.toString());
  // burger
```

toString() と toXMLString() のいずれかを指定せずに trace() メソッドを使用すると、次のように、データはデフォルトで toString() メソッドを使用して変換されます。

```
var myXML:XML =
   <order>
      <item id='1' quantity='2'>
         <menuName>burger</menuName>
         <price>3.95</price>
      </item>
   <order>;

trace(myXML.item[0].menuName);
   // burger
```

コードのデバッグ用に trace() メソッドを使用する際は、toXMLString() メソッドを使用し、より完全な形でデータを出力することがしばしば役立ちます。

ストリングから XML オブジェクトへの変換

次のように new XML() コンストラクタを使用すると、ストリングから XML オブジェクトを作成できます。

```
var x:XML = new XML("<a>test</a>");
```

XML として有効でないストリングや、整形式の XML になっていないストリングを XML に変換しようとすると、次のようにランタイムエラーが発生します。

```
var x:XML = new XML("<a>test"); // throws an error
```

ストリングから XML の属性値、名前、テキスト値への変換

XML の属性値、名前、テキスト値はすべて String データ型で表現されており、場合によっては他のデータ型に変換する必要があります。たとえば、次のコードでは Number() 関数を使用してテキスト値を数値に変換します。

```
var myXML:XML =
            <order>
               <item>
                  <price>3.95</price>
               </item>
               <item>
                  <price>1.00</price>
               </item>
            </order>;

var total:XML = <total>0</total>;
myXML.appendChild(total);

for each (var item:XML in myXML.item)
```

```
{
   myXML.total.children()[0] = Number(myXML.total.children()[0])
                            + Number(item.price.children()[0]);
}
trace(myXML.total); // 4.35;
```

このコードで Number() 関数を使用しないとすると、+ 演算子がストリング連結の演算子と解釈され、その結果、最終行にある trace() メソッドの出力は次のようになります。

```
01.003.95
```

外部 XML ドキュメントの読み込み

URLLoader クラスを使用すると、指定した URL から XML データをロードできます。次のコードをアプリケーションで使用する際には、XML_URL の値を実際の有効な URL に置き換えてください。

```
var myXML:XML = new XML();
var XML_URL:String = "http://www.example.com/Sample3.xml";
var myXMLURL:URLRequest = new URLRequest(XML_URL);
var myLoader:URLLoader = new URLLoader(myXMLURL);
myLoader.addEventListener("complete", xmlLoaded);

function xmlLoaded(evtObj:Event):void
{
   myXML = XML(myLoader.data);
   trace("Data loaded.");
}
```

また、XMLSocket クラスを使用すると、サーバーとの間に非同期の XML ソケット接続を確立できます。詳細については、『ActionScript 3.0 リファレンスガイド』を参照してください。

例 : インターネットから RSS データをロードする

RSSViewer サンプルアプリケーションを使用して、次のような、ActionScript で XML を操作する多くの機能を示します。

- XML メソッドを使用して、RSS フィード形式の XML データにアクセスする。
- XML メソッドを使用して、テキストフィールドで使用する HTML の形式に XML データを構成する。

RSS 形式は、XML でニュースを受け取るために広く使用されています。単純な RSS データファイルの例を示します。

```
<?xml version="1.0" encoding="UTF-8"?>
<rss version="2.0" xmlns:dc="http://purl.org/dc/elements/1.1/">
<channel>
  <title>Alaska - Weather</title>
  <link>http://www.nws.noaa.gov/alerts/ak.html</link>
  <description>Alaska - Watches, Warnings and Advisories</description>

  <item>
    <title>
      Short Term Forecast - Taiya Inlet, Klondike Highway (Alaska)
    </title>
    <link>
      http://www.nws.noaa.gov/alerts/ak.html#A18.AJKNK.1900
    </link>
    <description>
      Short Term Forecast Issued At: 2005-04-11T19:00:00
      Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
      Homepage: http://pajk.arh.noaa.gov
    </description>
  </item>
  <item>
    <title>
      Short Term Forecast - Haines Borough (Alaska)
    </title>
    <link>
      http://www.nws.noaa.gov/alerts/ak.html#AKZ019.AJKNOWAJK.190000
    </link>
    <description>
      Short Term Forecast Issued At: 2005-04-11T19:00:00
      Expired At: 2005-04-12T01:00:00 Issuing Weather Forecast Office
      Homepage: http://pajk.arh.noaa.gov
    </description>
  </item>
</channel>
</rss>
```

SimpleRSS アプリケーションは、インターネットから RSS データを読み取り、ヘッドライン (タイトル)、リンク、および説明のデータを解析し、そのデータを返します。SimpleRSSUI クラスは UI を提供し、SimpleRSS クラスを呼び出します。SimpleRSS クラスはすべての XML 処理を行います。

RSSViewer アプリケーションのファイルは、"Samples/RSSViewer" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
RSSViewer.mxml	MXML で記述された Flex 用メインアプリケーションファイル
com/example/programmingas3/rssViewer/RSSParser.as	E4X を使用して RSS (XML) データにアクセスし、対応する HTML 表現を生成するメソッドが含まれているクラス
RSSData/ak.rss	サンプル RSS ファイル。アプリケーションは、Adobe がホストする Flex RSS フィードで、Web から RSS データを読み取るように設定されています。ただし、使用するスキーマが Flex RSS フィードのスキーマとやや異なるこのドキュメントの RSS データを読み取るように、簡単にアプリケーションを変更できます。

XML データの読み取りと解析

RSSParser クラスには、`rssXML` 変数に保存されている入力 RSS データを HTML 形式の出力を含むストリングである `rssOutput` に変換する `xmlLoaded()` メソッドが含まれます。

ソース RSS データにデフォルトの名前空間が含まれている場合にデフォルトの XML 名前空間を設定するコードが、メソッドの先頭近くにあります。

```
if (rssXML.namespace("") != undefined)
{
    default xml namespace = rssXML.namespace("");
}
```

次の行は、ソース XML データの内容でループし、`item` という子孫のプロパティを確認します。

```
for each (var item:XML in rssXML..item)
{
    var itemTitle:String = item.title.toString();
    var itemDescription:String = item.description.toString();
    var itemLink:String = item.link.toString();
    outXML += buildItemHTML(itemTitle,
                    itemDescription,
                    itemLink);
}
```

最初の 3 行は、XML データの `item` プロパティの title、description、および link プロパティを表すストリング変数を単純に設定しています。次の行は、3 つの新しいストリング変数をパラメータとして使用して `buildItemHTML()` メソッドを呼び出し、XMLList オブジェクトの形式で HTML データを取得しています。

XMLList データの構成

HTML データ (XMLList オブジェクト) の形式は、次のとおりです。

```
<b>itemTitle</b>
<p>
  itemDescription
  <br />
  <a href="link">
    <font color="#008000">More...</font>
  </a>
</p>
```

メソッドの最初の行で、デフォルトの xml 名前空間を消去します。

```
default xml namespace = new Namespace();
```

`default xml namespace` ディレクティブのスコープは、関数ブロックレベルです。すなわち、この宣言のスコープは `buildItemHTML()` メソッドです。

次の行で、関数に渡されたストリング引数に基づいて、XMLList を構成します。

```
var body:XMLList = new XMLList();
body += new XML("<b>" + itemTitle + "</b>");
var p:XML = new XML("<p>" + itemDescription + "</p>");

var link:XML = <a></a>;
link.@href = itemLink; // <link href="itemLinkString"></link>
link.font.@color = "#008000";
     // <font color="#008000"></font></a>
     // 0x008000 = 緑
link.font = "More...";

p.appendChild(<br/>);
p.appendChild(link);
body += p;
```

この XMLList オブジェクトは、ActionScript HTML テキストフィールドに適したストリングデータを表します。

`xmlLoaded()` メソッドは、`buildItemHTML()` メソッドの戻り値を使用して、ストリングに変換します。

```
XML.prettyPrinting = false;
rssOutput = outXML.toXMLString();
```

RSS フィードのタイトル抽出とカスタムイベントの送信

xmlLoaded() メソッドは、ソース RSS XML データの情報に基づいて、rssTitle ストリングを設定します。

```
rssTitle = rssXML.channel.title.toString();
```

最後に、xmlLoaded() メソッドが、データが解析され使用可能になったことをアプリケーションに通知するイベントを生成します。

```
dataWritten = new Event("dataWritten", true);
```

第 3 部

Flash Player API

3

ここでは、Adobe Flash Player 9 特有の、パッケージとクラスで実装された重要な機能を使用する際の方針について詳しく説明します。

次の章が含まれます。

第 12 章：Flash Player API の概要 337
第 13 章：イベントの処理 345
第 14 章：ネットワーキングとコミュニケーション 371
第 15 章：ジオメトリの操作 417
第 16 章：クライアントのシステム環境. 433
第 17 章：Flash Player セキュリティ 449
第 18 章：プリント 487
第 19 章：External API の使用. 501

第 12 章

Flash Player API の概要

12

この章では、ActionScript 3.0 言語に含まれている Adobe Flash Player 9 パッケージについて説明します。

Flash Player API では、flash パッケージ内のすべてのパッケージ、クラス、関数、プロパティ、定数、イベント、およびエラーが参照されます。これらは、トップレベルのクラス (Date、Math、XML など) や ECMAScript に基づいた言語要素とは異なり、Flash Player に固有です。Flash Player API には、オブジェクト指向プログラミング言語に通常ある関数 (ジオメトリクラスの flash.geom パッケージなど) と、高度なインターネットアプリケーションの要件に固有の関数 (表現力に関する flash.filters パッケージや、サーバーとの間のデータ転送を処理する flash.net パッケージなど) の両方が含まれます。

ActionScript 3.0 の Flash Player API には、オブジェクトとデータを低いレベルで制御できる多数の新しいクラスが追加されています。ActionScript 3.0 のアーキテクチャはまったく新しく、より直観的です。たとえば、API は次の理論パッケージに構成されています。flash.display パッケージには、Flash Player のビジュアル表示リストに関するすべてのクラスが含まれます。flash.media パッケージには、オーディオとビデオを処理するためのクラスが含まれます。flash.text には、Flash Player アプリケーションでテキストを処理するためのクラスが含まれます。

この章では、各パッケージの概要を簡単に示し、特に重要で一般的に使用されるクラスと関数について主に説明します。クラスまたはパッケージレベルの関数の詳細については、『ActionScript 3.0 リファレンスガイド』の対応する項を参照してください。

目次

flash.accessibility パッケージ 338
flash.display パッケージ 338
flash.errors パッケージ 339
flash.events パッケージ 339
flash.external パッケージ 340
flash.filters パッケージ 340
flash.geom パッケージ 340
flash.media パッケージ 341
flash.net パッケージ 341
flash.printing パッケージ 341
flash.profiler パッケージ 342
flash.system パッケージ 342
flash.text パッケージ 342
flash.ui パッケージ 342
flash.utils パッケージ 343
flash.xml パッケージ 343

flash.accessibility パッケージ

flash.accessibility パッケージには、Flash コンテンツおよびアプリケーションのアクセシビリティをサポートするためのクラスが含まれます。Accessibility クラスは、スクリーンリーダーとの通信を管理します。AccessibilityProperties クラスを使用すれば、スクリーンリーダーや、その他のアクセシビリティ補助への Flash オブジェクトの提示方法を制御できます。詳細については、『ActionScript 3.0 リファレンスガイド』の flash.accessibility パッケージを参照してください。

flash.display パッケージ

flash.display パッケージには、Flash Player がビジュアル表示を構築し、表示リストのオブジェクトを制御するために使用するコアクラスが含まれており、Flash Player アプリケーションのすべてのビジュアルエレメントが含まれます。

ActionScript 2.0 では、ほとんどのビジュアルエレメントが MovieClip クラスで制御されるか、Flash Player によって " 内部で " 処理されていました。オブジェクトの低レベル制御は不可能でした。ActionScript 3.0 では、ビジュアルエレメントを制御する API が、機能と使用方法ごとにより論理的に定義されています。flash.display パッケージには、次のクラスが含まれます。

- DisplayObject クラスから継承する基本的な要素。DisplayObjectContainer、Sprite など。MovieClip も基本的な要素で、タイムラインを必要とするオブジェクトに使用します。さまざまなボタンの状態を表すプロパティなど、ボタンに固有の機能は、SimpleButton クラスに含まれます。
- 表現力に富むグラフィックに作成に使用するクラス。ベクターグラフィック用の Graphics、Shape や、ビットマップイメージ用の Bitmap、BitmapData など。
- タイムラインベースのアプリケーションおよびアニメーション用のクラス。MovieClip、Scene など。
- SWF ファイルまたはイメージファイル (JPG、PNG、または GIF) のロードを処理する Loader および LoaderInfo クラス。
- これらのクラスをサポートする追加のクラス。

詳細については、159 ページの「表示のプログラミング」および『ActionScript 3.0 リファレンスガイド』の flash.display パッケージを参照してください。

flash.errors パッケージ

flash.errors パッケージには、IOError (入出力エラー)、IllegalOperationError など、Flash Player に固有の機能に関連するエラークラスが含まれます。ActionScript 3.0 では、例外がランタイムエラーを報告する主要なメカニズムです。詳細については、255 ページの「エラー処理」および『ActionScript 3.0 リファレンスガイド』の flash.errors パッケージを参照してください。

flash.events パッケージ

flash.events パッケージでは、新しい XML DOM (ドキュメントオブジェクトモデル) イベントモデルがサポートされ、EventDispatcher 基本クラスが含まれます。イベントには、エラーイベントが含まれます。エラーイベントは、`Loader.load()` メソッドの呼び出しなど、非同期処理でエラーが発生したときに使用されます。詳細については、345 ページの「イベントの処理」および『ActionScript 3.0 リファレンスガイド』の flash.events パッケージを参照してください。

flash.external パッケージ

flash.external パッケージには、Flash Player 8 で `fscommand()` 関数の代わりに導入された ExternalInterface クラスだけが含まれます。ExternalInterface によって、ActionScript と Flash Player コンテナ (JavaScript を使用している HTML ページや、Flash Player が組み込まれているデスクトップアプリケーションなど) との通信が可能になります。詳細については、501 ページの「External API の使用」および『ActionScript 3.0 リファレンスガイド』の flash.external パッケージを参照してください。

flash.filters パッケージ

flash.filters パッケージには、Flash Player 8 で導入されたビットマップフィルタ効果のクラスが含まれます。フィルタを使用すると、ぼかし、ベベル、グロー、ドロップシャドウなどの豊富な視覚効果を表示オブジェクトに適用できます。flash.filters パッケージのクラスは、BevelFilter、BlurFilter、DisplacementMapFilter、GlowFilter、GradientBevelFilter、および GradientGlowFilter です。このパッケージには、個別のピクセル値にマトリックスを適用することで、より複雑で広範囲な効果を与えることができるクラスが含まれています。それが、ColorMatrixFilter と ConvolutionFilter です。詳細については、『ActionScript 3.0 リファレンスガイド』の flash.filters パッケージを参照してください。

flash.geom パッケージ

flash.geom パッケージには、Bitmap クラスと表示オブジェクトのビットマップキャッシュプロパティをサポートする Point、Rectangle、Matrix などのジオメトリクラスが含まれます。また、カラー値を操作する Transform および ColorTransform クラスも含まれます。詳細については、417 ページの「ジオメトリの操作」および『ActionScript 3.0 リファレンスガイド』の flash.geom パッケージを参照してください。

flash.media パッケージ

flash.media パッケージには、記録済みまたは Flash Player を実行しているクライアントコンピュータから流れるオーディオストリームおよびビデオストリームを処理するためのクラスが含まれます。Sound クラスおよびこのクラスがサポートするクラスを使用すると、外部の MP3 ファイル、および SWF ファイルに埋め込まれたストリーミングサウンドを処理できます。Microphone クラスを使用すると、コンピュータに接続されているマイクロフォンからオーディオをキャプチャできます。Camera および Video クラスを使用すると、コンピュータに接続されているビデオカメラからビデオをキャプチャできます。また、Video クラスを使用すると、記録済みの外部 FLV ファイルを処理できます。詳細については、『ActionScript 3.0 リファレンスガイド』の flash.media パッケージを参照してください。

flash.net パッケージ

flash.net パッケージには、データの送受信を処理するさまざまなクラスが含まれます。Flash Player 9 は、生のバイナリデータ、XML、テキスト、URL エンコードされた変数、アップロードまたはダウンロードできるファイル全体など、多くの種類のデータを処理できます。データは、ネットワークサーバーと Flash Player を実行しているクライアントコンピュータとの間、または 2 つの SWF ファイルの間で転送できます。

パッケージの中心は、URLLoader クラスと URLRequest クラスです。URLLoader クラスを使用して、オブジェクト内の変数を指定の URL に送ったり、指定された URL にある変数をオブジェクトにロードしたりできます。URLRequest クラスは、HTTP 要求のすべてのデータをキャプチャします。

このパッケージには、`navigateToURL()` や `sendToURL()` など、サーバーとの対話をそれほど必要としない単純な操作に使用できるパッケージレベルの関数も含まれています。

詳細については、371 ページの「ネットワーキングとコミュニケーション」および『ActionScript 3.0 リファレンスガイド』の flash.net パッケージを参照してください。

flash.printing パッケージ

このパッケージには、Flash コンテンツを印刷するためのクラスが含まれます。詳細については、487 ページの「プリント」および『ActionScript 3.0 リファレンスガイド』の flash.printing パッケージを参照してください。

flash.profiler パッケージ

このパッケージには、コードのデバッグに役立つ showRedrawRegions() 関数が含まれます。詳細については、『ActionScript 3.0 リファレンスガイド』の flash.profiler パッケージを参照してください。

flash.system パッケージ

flash.system パッケージには、システムレベルの機能を提供するクラスと、Flash Player を実行しているクライアントコンピュータの機能について制限された指定の情報を取得できるクラスが含まれます。Capabilities および IME クラスを使用して、Flash Player を実行しているコンピュータの特定のハードウェアおよびソフトウェア性能を判断できます。このパッケージには、セキュリティに関連するクラス、つまり Security、SecurityDomain、LoaderContext が含まれています。

flash.system パッケージには、ApplicationDomain クラスも含まれます。このクラスを使用して、カスタムクラスを別のアプリケーションドメインに分割し、定義を再利用できます。

詳細については、433 ページの「クライアントのシステム環境」および『ActionScript 3.0 リファレンスガイド』の flash.system パッケージを参照してください。

flash.text パッケージ

このパッケージには、テキストフィールド、テキストのフォーマット、フォント、アンチエイリアス、テキストのメトリック、スタイルシート、およびレイアウトを扱うクラスが含まれます。高度なアンチエイリアスは、TextFormat および TextRenderer クラスを使用することで、Flash Player 9 で使用できます。TextField クラスの新しいさまざまなメソッドを使用して、テキストフィールドの低レベルの詳細を取得できます。詳細については、『ActionScript 3.0 リファレンスガイド』の flash.text パッケージを参照してください。

flash.ui パッケージ

flash.ui パッケージには、ユーザーインターフェイス (具体的には、マウスカーソル、コンピュータキーボード、コンテキストメニュー) をカスタマイズするためのクラスが含まれます。詳細については、『ActionScript 3.0 リファレンスガイド』の flash.ui パッケージを参照してください。

flash.utils パッケージ

このパッケージには、バイトレベルでデータにアクセスして処理できる ByteArray、指定したタイムシーケンスでコードを実行できる Timer クラスなど、さまざまなユーティリティクラスが含まれます。また、ActionScript コードを実行する遅延時間や間隔を制御できるパッケージレベルの関数も多数含まれます。詳細については、『ActionScript 3.0 リファレンスガイド』の flash.utils パッケージを参照してください。

flash.xml パッケージ

flash.xml パッケージは、古い XML サポートを提供します。トップレベルの E4X 準拠 XML クラスが導入されたため、ActionScript 1.0 および 2.0 XML オブジェクトと互換性がある古い XML 関連クラスは、flash.xml パッケージにまとめられました。詳細については、311 ページの「XML の操作」および『ActionScript 3.0 リファレンスガイド』の XML パッケージと XMLList パッケージを参照してください。

第 13 章

イベントの処理

13

イベント処理システムは、ユーザー入力やシステムイベントにプログラムが応答するための便利な仕組みです。ActionScript 3.0 の新しいイベントモデルは単に便利であるだけでなく、標準規格の仕様に準拠しており、しかも、Adobe Flash Player 9 の新しい表示リストの機能とよく統合されています。業界標準のイベント処理アーキテクチャであるドキュメントオブジェクトモデル (DOM) Level 3 Events 仕様に基づいた、新しいイベントモデルにより、ActionScript による強力でわかりやすいイベント処理が可能になりました。

この章は 5 つのセクションで構成されます。そのうち最初の 2 つのセクションでは、ActionScript のイベント処理に関する基本事項について説明します。残り 3 つのセクションでは、新しいイベントモデルのベースとなっている主要な概念、すなわち、イベントフロー、イベントオブジェクト、およびイベントリスナーについて説明します。ActionScript 3.0 のイベント処理システムは表示リストと緊密に協調して機能するため、この章の内容は、表示リストについて基本事項を理解している読者が対象となっています。詳細については、159 ページの「表示のプログラミング」を参照してください。

目次

イベント処理の概要 346
ActionScript 3.0 のイベント処理と以前のバージョンにおけるイベント処理との違い . . 347
イベントフロー 350
イベントオブジェクト 352
イベントリスナー 357
例 : Alarm Clock. 365

イベント処理の概要

イベントとは、SWF ファイルに発生する各種の事象のうち、プログラマにとって意味のあるすべてのことを指す言葉と考えて差し支えありません。たとえば、ほとんどの SWF ファイルは何らかの形でユーザーとのやりとりをサポートします (その内容は、マウスクリックに反応するだけの単純なものから、フォームに入力されるデータを受け付けて処理するような複雑なものまでさまざまです)。ユーザーと SWF ファイルとの間で行われるそのような対話操作は、イベントの一種と考えられます。また、ユーザーによる直接的な操作が行われていないときも、サーバーからのデータのロードが終了した場合や、接続されたカメラがアクティブになった場合などにイベントが発生することがあります。

ActionScript 3.0 では、1 回のイベントを 1 個のイベントオブジェクト (Event クラスまたはそのサブクラスのインスタンス) として表現します。イベントオブジェクトには、具体的なイベントに関する情報が格納されているのに加え、イベントオブジェクトを操作する際に役立つメソッドが備わっています。たとえば Flash Player では、マウスクリックを検出すると、その特定のマウスクリックイベントを表すイベントオブジェクトを作成します。その場合のイベントオブジェクトは MouseEvent クラスのインスタンスです。

次に、Flash Player は作成したイベントオブジェクトを " 送出 " します。これは、イベントのターゲットとなるオブジェクトにイベントオブジェクトを引き渡すという意味です。イベントオブジェクトの送出先となるオブジェクトを、" イベントターゲット " と呼びます。たとえば、コンピュータに取り付けられているカメラがアクティブになった場合、Flash Player はイベントターゲット (この場合はカメラを表すオブジェクト) に対して直接にイベントオブジェクトを送出します。ただし、イベントターゲットが表示リスト内にある場合、イベントオブジェクトは表示リストの階層内を下ってイベントターゲットまで伝達されていきます。場合によっては、この表示リスト階層内の経路をイベントオブジェクトが逆に " 浮上 " (バブリング) していくこともあります。このように表示リストの階層内をイベントが伝達されていくことを イベントフロー といいます。

アプリケーションのコードでイベントを受け取る (リッスンする) には、イベントリスナーを使用します。" イベントリスナー " とは、特定のイベントに応答するために記述する関数やメソッドです。プログラムがイベントに対して確実に応答できるようにするには、イベントターゲットにイベントリスナーを登録するか、イベントオブジェクトのイベントフローにおいて伝達経路の一部として使用される表示リストオブジェクトにイベントリスナーを追加する必要があります。

新しいイベントモデルは、イベントフロー、イベントオブジェクト、およびイベントリスナーという 3 つの基本概念によって構成されています。SWF ファイルで発生するイベントに応答するためには、これらの概念を確実に理解しておくことが重要です。このイベントモデルでは上級開発者が独自のイベントを作成および送出することもできますが、この章の内容では、ActionScript 3.0 にあらかじめ定義された、Flash Player によって直接送出されるイベントについての説明を主体とします。

ActionScript 3.0 のイベント処理と以前のバージョンにおけるイベント処理との違い

ActionScript 3.0 のイベント処理と、ActionScript の以前のバージョンにおけるイベント処理との最も目立つ違いは、ActionScript 3.0 にはイベント処理に関するシステムが 1 つしかないのに対し、以前のバージョンには複数の異なるイベント処理システムが存在していたことです。このセクションでは、まず、以前のバージョンにおけるイベント処理の仕組みを概観し、続いて、それが ActionScript 3.0 でどのように変更されたかを説明します。

ActionScript の以前のバージョンにおけるイベント処理

ActionScript 3.0 より前のバージョンでは、イベント処理に関して次のように複数の方法が存在していました。

- Button インスタンスや MovieClip インスタンスに直接配置できる `on()` イベントハンドラ
- MovieClip インスタンスに直接配置できる `onClipEvent()` イベントハンドラ
- `XML.onload` や `Camera.onActivity` などのコールバック関数プロパティ
- `addListener()` メソッドを使用して登録するイベントリスナー
- DOM イベントモデルの一部を実装した UIEventDispatcher クラス

いずれのメカニズムにも、それぞれに特有のメリットとデメリットがあります。`on()` ハンドラと `onClipEvent()` ハンドラの場合は、気軽に使用できる反面、ボタンやムービークリップにコードを直接配置すると見通しが悪いため以後のプロジェクト保守作業が困難になります。コールバック関数の場合も、実装作業は簡単ですが、1 種類のイベントについてコールバック関数を 1 つしか作成できないという制約があります。イベントリスナーの場合は、リスナーオブジェクトおよび関数の作成に加え、イベントを生成するオブジェクトに対してリスナーを登録する必要があるため、実装作業が複雑になりますが、その代わり 1 種類のイベントに対して複数のリスナーオブジェクトを作成および登録できます。

ActionScript 2.0 用コンポーネントが開発される際にもう 1 つのイベントモデルが導入され、UIEventDispatcher クラスとして実装されました。このモデルは DOM イベント仕様のサブセットに準拠していたため、コンポーネントのイベント処理に慣れた開発者は、ActionScript 3.0 の新しいイベントモデルに比較的スムーズに移行できます。

従来の各種イベントモデルの間には、使用するシンタックスがさまざまな形で重複していたり、また別の面では違っていたりするという問題がありました。たとえば、ActionScript 2.0 における一部のプロパティ (`TextField.onChanged` など) は、コールバック関数として使用することも、イベントリスナーとして使用することもできました。ところが、リスナーをサポートする 6 つのクラスと UIEventDispatcher クラスとではリスナーオブジェクトを登録する際のシンタックスが違っていました。つまり、Key、Mouse、MovieClipLoader、Selection、Stage、TextField の各クラスでは `addListener()` メソッドを使用する一方、コンポーネントのイベント処理については `addEventListener()` メソッドを呼び出す必要がありました。

他にも、イベント処理モデルが複数あることに起因して、イベントハンドラ関数のスコープが使用メカニズムによって異なるという問題がありました。このため、`this` キーワードの指す対象が一貫せず、使用するイベント処理システムによって異なっていました。

ActionScript 3.0 におけるイベント処理

ActionScript 3.0 ではイベント処理モデルが一本化され、以前のバージョンに存在していた複数のイベント処理メカニズムはそれによって置き換えられました。新しいイベントモデルは、ドキュメントオブジェクトモデル (DOM) Level 3 Events 仕様に基づいています。SWF ファイルの形式は DOM の標準仕様に準拠していませんが、表示リストと DOM の構造には、DOM イベントモデルを実装するために十分な程度の共通点があります。表示リスト内におけるオブジェクトは、DOM 階層構造におけるノードに対応するものと考えることができます。このため、この章の説明では *表示リストオブジェクト* および *ノード* という 2 つの用語をいずれも同じ意味で使用することがあります。

Flash Player における DOM イベントモデルの実装には、" デフォルト動作 " という新しい概念が導入されています。*デフォルト動作* とは、ある種のイベントに対する通常の結果として Flash Player で実行されるアクションのことです。

デフォルト動作

イベントに応答するコードは、開発者が記述しなくてはならないのが普通です。しかし、イベントに対して何らかの決まった動作が非常によく行われる場合については、一般的な処理が Flash Player によって自動的に実行されるようになっています (開発者が特にこの動作をキャンセルするためのコードを記述した場合を除く)。Flash Player によって自動的に実行されるそうした動作を、デフォルト動作と呼びます。

たとえば、ユーザーによって TextField オブジェクトにテキストが入力されたとき、たいていの場合は、入力されたテキストをその TextField オブジェクトに表示する必要があります。これは非常に一般的な動作なので、Flash Player に組み込まれています。この動作が不要な場合は、新しいイベント処理システムを使用してデフォルト動作をキャンセルできます。ユーザーによって TextField オブジェクトにテキストが入力されると、その入力を表す TextEvent クラスのインスタンスが Flash Player によって作成されます。入力テキストが Flash Player によって TextField オブジェクト内に表示されるのを防ぐには、その特定の TextEvent インスタンスにアクセスして `preventDefault()` メソッドを呼び出す必要があります。

デフォルト動作の中には、キャンセルできないものもあります。たとえば、ユーザーによって TextField オブジェクト内の単語がダブルクリックされると、Flash Player から MouseEvent オブジェクトが生成されます。この場合のデフォルト動作である、カーソル位置の単語をハイライトする処理はキャンセルできません。

多くの種類のイベントオブジェクトには、デフォルト動作が関連付けられていません。たとえば、ネットワーク接続が確立すると Flash Player から connect イベントが送出されますが、このイベントに関連付けられたデフォルト動作はありません。API ドキュメントの Event クラスとそのサブクラスに関する項目には、各種イベントの一覧と、関連付けられたデフォルト動作がある場合はその内容、およびデフォルト動作をキャンセルできるかどうかが説明されています。

デフォルト動作は Flash Player から送出されるイベントオブジェクトにしか関連付けられないため、ActionScript のプログラムで送出したイベントオブジェクトに対しては実行されません。この点を理解しておくことは重要です。たとえば、EventDispatcher クラスのメソッドを使用して `textInput` タイプのイベントオブジェクトを送出することはできますが、その場合はイベントオブジェクトにデフォルト動作が関連付けられません。したがって、プログラムから `textInput` イベントを送出しても、その結果 Flash Player によって TextField オブジェクトに文字が表示されることはありません。

ActionScript 3.0 のイベントリスナーに関する新機能

ActionScript 2.0 の `addListener()` メソッドを使用した経験があれば、ActionScript 2.0 のイベントリスナーモデルと ActionScript 3.0 のイベントモデルとの違いがよくわかります。2 つのイベントモデル間の主な差異点を次の一覧に示します。

- イベントリスナーを登録する際、ActionScript 2.0 では `addListener()` を使用する場合と `addEventListener()` を使用する場合がありましたが、ActionScript 3.0 では常に `addEventListener()` を使用します。
- ActionScript 2.0 にはイベントフローの仕組みがないため、`addListener()` メソッドを呼び出せるのは当該イベントをブロードキャストするオブジェクトに対してのみでした。ActionScript 3.0 では、イベントフローの一部となっているすべてのオブジェクトに対して `addEventListener()` メソッドを呼び出せます。
- ActionScript 2.0 では、関数、メソッド、オブジェクトのいずれをイベントリスナーとして使用することもできました。ActionScript 3.0 では、関数またはメソッドのみイベントリスナーとして使用できます。

イベントフロー

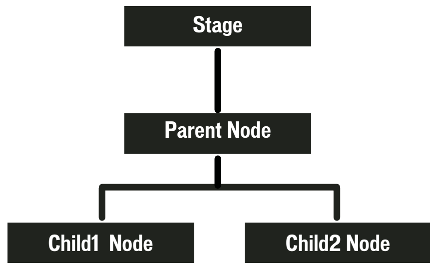
Flash Player では、イベントが発生するたびにイベントオブジェクトが送出されます。イベントターゲットが表示リスト内にない場合､Flash Player からイベントターゲットに対して直接にイベントオブジェクトが送出されます。たとえば、progress イベントオブジェクトは URLStream オブジェクトに対して直接に送出されます。イベントターゲットが表示リスト内にある場合、イベントオブジェクトは表示リストに対して送出され、表示リストの階層内を経由してイベントターゲットに到達します。

イベントフローは、表示リスト内をイベントオブジェクトが伝わる経路です。表示リストの構造はツリー状の階層になっており、そのトップの階層はステージです。ステージは特殊な表示オブジェクトコンテナであり、表示リストのルートとして機能します。ステージは flash.display.Stage クラスによって表され、表示オブジェクトを介してのみアクセスできます。すべての表示オブジェクトには、当該アプリケーションのステージを参照する `stage` というプロパティがあります。

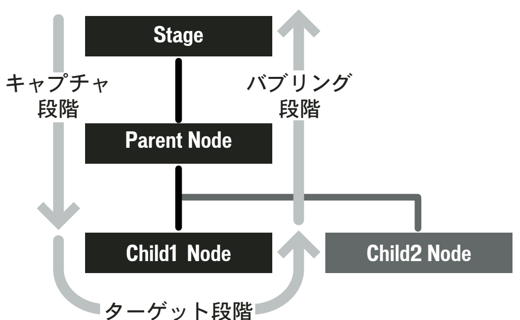
Flash Player によってイベントオブジェクトが送出されると、そのイベントオブジェクトは、ステージから " ターゲットノード " までの経路を往復して移動します。DOM Events 仕様では、ターゲットノードとはイベントターゲットを表すノードのことであると定義されています｡これを言い換えれば､ターゲットノードとは当該イベントが発生した表示リストオブジェクトのことです。たとえば、ユーザーが `child1` という表示リストオブジェクトをクリックすると、Flash Player により、`child1` をターゲットノードとしてイベントオブジェクトが送出されます。

概念上、イベントフローは 3 つの部分に分けられます。第 1 の部分はキャプチャ段階と呼ばれ、ステージからターゲットノードの親までのノードすべてがこれに含まれます。第 2 の部分はターゲット段階と呼ばれ、ターゲットノードだけがこれに含まれます。第 3 の部分はバブリング段階と呼ばれ、ターゲットノードの親からステージまでの帰り道を構成するすべてのノードがこれに含まれます。

次の図のように、ステージを最上位とする縦の階層として表示リストの構造を考えると、各段階に付けられた名前の意味がよりよくわかります。



ユーザーによって Child1 Node がクリックされると、Flash Player により、イベントオブジェクトがイベントフローに対して送出されます。次の図で示すように、オブジェクトは Stage から始まる伝達経路を下っていき、Parent Node を経て Child1 Node に到達してから、もう一度 "バブリング" によって Parent Node から Stage へと逆方向の経路を浮上していきます。



この例で、イベントオブジェクトが下に向かって伝達されていくキャプチャ段階には、Stage および Parent Node が含まれています。ターゲット段階には Child1 Node が含まれています。また、イベントオブジェクトがルートノードへと浮上していくバブリング段階の経路には、Parent Node および Stage が含まれています。

このイベントフローは、従来よりいっそう強力な ActionScript 用イベント処理システムの実現に寄与しています。ActionScript の以前のバージョンでは、イベントフローの仕組みが存在しなかったため、イベントの生成元となるオブジェクトにイベントリスナーを登録することしかできませんでした。ActionScript 3.0 では、ターゲットノード自体の他、イベントフローに含まれる任意のノードにイベントリスナーを登録できます。

イベントリスナーをイベントフローに追加する機能は、ユーザーインターフェイスコンポーネントが複数のオブジェクトで構成されている場合に役立ちます。たとえば、ボタンオブジェクトには、ボタンのラベルとして機能するテキストオブジェクトが含まれる場合があります。リスナーをイベントフローに追加する機能がないと、ボタンの任意の場所で発生したクリックイベントに関する通知を受信できるように、ボタンオブジェクトとテキストオブジェクトの両方にリスナーを追加する必要が生じます。しかし、イベントフローがあることで、単一のイベントリスナーをボタンオブジェクトに配置し、テキストオブジェクトと、テキストオブジェクトで隠されていないボタンオブジェクトの領域の両方で発生したクリックイベントを処理できます。

ただし、イベントオブジェクトのタイプによっては、イベントフローの 3 段階のうち一部しか処理されないことがあります。たとえば、enterFrame や init などのタイプはターゲットノードに対して直接に送出され、キャプチャ段階とバブリング段階を経ることがありません。また、その他のイベントでは表示リストにないオブジェクトがターゲットになることがあります。Socket クラスのインスタンスに対して送出されるイベントなどがこれに該当します。それらのイベントオブジェクトもまた、キャプチャ段階やバブリング段階を経ることなく、ターゲットノードに対して直接に送出されます。

具体的な各種イベントタイプの動作については、API ドキュメントを参照するか、実際のイベントオブジェクトのプロパティを確認してください。イベントオブジェクトのプロパティについては、次のセクションで説明します。

イベントオブジェクト

新しいイベント処理システムにおいて、イベントオブジェクトには主として 2 つの用途があります。1 つは、具体的な個々のイベントを表現し、当該イベントに関する情報を各種プロパティに格納することです。もう 1 つは、各種メソッドを使用してイベントオブジェクトを操作し、イベント処理システムの動作を変化させることです。

それらのプロパティとメソッドにアクセスしやすいよう、Flash Player API には、イベントオブジェクトすべての基本クラスとなる Event クラスが定義されています。Event クラスは、すべてのイベントオブジェクトに共通する基本的なプロパティとメソッドを備えています。

このセクションでは、まず Event クラスのプロパティと、次に Event クラスのメソッドについて順に説明し、最後に、Event クラスにサブクラスが存在する理由について説明します。

Event クラスのプロパティについて

Event クラスには、イベントオブジェクトに関する重要な情報を提供する多数の読み取り専用プロパティと定数があります。次のプロパティは特に重要です。

- イベントオブジェクトのタイプは定数によって表され、`Event.type` プロパティに格納されます。
- イベントのデフォルト動作をキャンセルできるかどうかは Boolean 値によって表され、`Event.cancelable` プロパティに格納されます。
- イベントフローに関する情報はその他のプロパティに格納されます。

イベントオブジェクトのタイプ

すべてのイベントオブジェクトには、それぞれイベントタイプが設定されます。イベントタイプはストリング値として `Event.type` プロパティに格納されます。コードでイベントオブジェクトのタイプを知ることができると、タイプに応じてオブジェクトの処理方法を区別できて便利です。たとえば、次のコードでは、`myDisplayObject` に渡されるマウスクリックのイベントオブジェクトすべてに対して `clickHandler()` リスナー関数が応答するように指定しています。

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler)
```

Event クラス自体に関連付けられ、Event クラス定数によって表されるイベントタイプの数は 20 種類を超えます。それらの一部について、Event クラス定義から抜粋した次のコードに示します。

```
package flash.events
{
    public class Event
    {
        // クラス定数
        public static const ACTIVATE:String = "activate";
        public static const ADDED:String    = "added";
        // その他の定数は省略
    }
}
```

これらの定数を使用すると、特定のイベントタイプを簡単に参照できます。各定数が表すストリングを直接に使用することは避け、定数名で参照するようにしてください。コードに入力した定数名にスペルミスがあった場合はコンパイラによって検出できます。しかし、ストリングの値にスペルミスがあった場合はコンパイル時のエラーとならず実行時に予期しない動作が生じ、困難なデバッグ作業が必要となる可能性があります。たとえば、イベントリスナーを登録する際は次のようなコードを使用してください。

```
myDisplayObject.addEventListener(MouseEvent.CLICK, clickHandler);
```

次のコードはお勧めできません。

```
myDisplayObject.addEventListener("click", clickHandler);
```

デフォルト動作情報

コードで cancelable プロパティを調べると、特定のイベントオブジェクトに対するデフォルト動作をキャンセルできるかどうか知ることができます。cancelable プロパティの値は Boolean 型で、デフォルト動作に対するキャンセルの可否を示します。デフォルト動作のキャンセルが可能なイベントは少数ですが、該当する場合は、関連付けられた動作を preventDefault() メソッドで無効化できます。詳細については、356 ページの「イベントのデフォルト動作のキャンセル」を参照してください。

イベントフロー情報

Event クラスの他のプロパティには、次に示すとおり、イベントオブジェクト自体やイベントフローとの関係に関する重要な情報が格納されます。

- bubbles プロパティは、そのイベントオブジェクトの伝達経路となるイベントフローの構成要素に関する情報を含みます。
- eventPhase プロパティは、イベントフローが現在どの段階にあるかを示します。
- target プロパティは、イベントターゲットへの参照です。
- currentTarget プロパティは、当該イベントオブジェクトを現在処理している表示リストオブジェクトへの参照です。

bubbles プロパティ

イベントオブジェクトがバブリング段階にあるとき、そのオブジェクトによって表されるイベントは " 浮上 " 中、つまり、ターゲットノードからステージへと階層の上位に向かって戻る経路の途中にあります。Event.bubbles プロパティには、そのイベントオブジェクトがバブリング段階を経由するかどうかを示す Boolean 値が格納されます。バブリングの対象となるイベントは、必ずキャプチャ段階とターゲット段階の対象にもなるので、イベントフローの 3 段階すべてを経ることになります。このプロパティの値が true の場合、当該イベントオブジェクトは 3 つの段階をすべて経由します。値が false の場合、当該イベントオブジェクトはバブリング段階を経由しません。

eventPhase プロパティ

eventPhase プロパティを調べると、そのイベントオブジェクトがイベントのどの段階にあるかを知ることができます。eventPhase プロパティには、イベントフローの 3 つの段階いずれかを示す符号なし整数値が格納されます。次に示すコードの抜粋にあるとおり、Flash Player API には、それらの符号なし整数値に対応する 3 つの定数を含んだ EventPhase クラスが別途定義されています。

```
package flash.events
{
  public final クラス EventPhase
  {
    public static const CAPTURING_PHASE:uint = 1;
    public static const AT_TARGET:uint       = 2;
    public static const BUBBLING_PHASE:uint  = 3;
  }
}
```

これらの定数は、eventPhase プロパティにおいて有効な 3 つの値に対応します。定数を使用すると、コードの読みやすさを向上できます。たとえば、イベントオブジェクトがターゲット段階にある場合のみ myFunc() という関数を呼び出すようにするには、次のようなコードで条件を判断します。

```
if (e.eventPhase == EventPhase.AT_TARGET)
{
  myFunc();
}
```

target プロパティ

target プロパティには、当該イベントのターゲットであるオブジェクトへの参照が格納されます。ターゲットの扱いは単純な場合とそうでない場合があります。たとえば、マイクがアクティブになったことを示すイベントでは、イベントオブジェクトのターゲットは Microphone オブジェクトであることが明確です。しかし、ターゲットが表示リスト内にある場合は、表示リストの階層構造を考慮する必要があります。たとえば、複数の表示リストオブジェクトの重なり合う地点がユーザーによってマウスでクリックされた場合、Flash Player では、ステージから最も遠い階層にあるオブジェクトがイベントターゲットとして選ばれます。

複雑な SWF ファイルでは、`target` プロパティがうまく機能しないことがあります。各ボタンに対して子オブジェクトを使用した定型的な修飾処理が施されているような場合は、target プロパティの参照先がボタンではなく修飾用の子オブジェクトになってしまう状況が発生しやすいためです。このような状況では、ボタンにイベントリスナーを登録し、`currentTarget` プロパティを使用するのが一般的です。そうすれば、`target` プロパティが子オブジェクトを参照してしまう場合にも currentTarget プロパティでボタンを参照できます。

currentTarget プロパティ

`currentTarget` プロパティには、当該イベントオブジェクトを現在処理している表示リストオブジェクトへの参照が格納されます。調べる対象のイベントオブジェクトが現在どのノードで処理されているかが事前にわかっていないとは奇妙な状況のようにも聞こえますが、実際は特別なことではありません。なぜなら、当該イベントオブジェクトのイベントフローに含まれる任意の表示オブジェクトにリスナー関数が登録されている可能性があり、また、リスナー関数自体が任意の場所に配置されている可能性があるからです。さらに、1 つのリスナー関数が複数の表示オブジェクトに登録されている場合もあります。プロジェクトの規模が大きくなり、複雑さが増すにつれて、`currentTarget` プロパティの有用性は高まります。

Event クラスのメソッドについて

Event クラスのメソッドは、次の 3 種類に大別されます。

- ユーティリティメソッド : イベントオブジェクトのコピーを作成する、またはストリングに変換する
- イベントフローメソッド : イベントオブジェクトをイベントフローから取り除く
- デフォルト動作メソッド : デフォルト動作をキャンセルする、またはキャンセル済みかどうかを確認する

Event クラスのユーティリティメソッド

Event クラスには 2 つのユーティリティメソッドがあります。`clone()` メソッドは、イベントオブジェクトのコピーを作成します。`toString()` メソッドは、イベントオブジェクトの各種プロパティとそれらの値を表すストリング表現を生成します。いずれのメソッドもイベントモデルシステムの内部で使用されているものですが、アプリケーション開発者にも一般用途向けに公開されています。

上級開発者が Event クラスのサブクラスを作成する場合は、イベントサブクラスを正常に機能させるために、両方のユーティリティメソッドをオーバーライドして独自のバージョンを実装する必要があります。

イベントフローの停止

イベントオブジェクトがイベントフローで処理されるのを停止するには、`Event.stopPropogation()` メソッドまたは `Event.stopImmediatePropogation()` メソッドを呼び出します。これら 2 つのメソッドはほとんど同じですが、現在のノードに登録されている残りのイベントリスナーについて実行を許可するかどうかという点のみが異なります。

- `Event.stopPropogation()` メソッドを呼び出すと、当該イベントオブジェクトがイベントフローで次のノードに伝達されるのを防ぐことができます。ただし、現在のノードに登録されているイベントリスナーはすべて実行されます。
- `Event.stopImmediatePropogation()` メソッドを呼び出すと、当該イベントオブジェクトがイベントフローで次のノードに伝達されるのを防ぎ、また、現在のノードに登録されている他のイベントリスナーの実行も防ぐことができます。

いずれのメソッドを呼び出した場合も、イベントに関連付けられているデフォルト動作が実行されるかどうかには影響しません。デフォルト動作をキャンセルするには、Event クラスのデフォルト動作メソッドを使用する必要があります。

イベントのデフォルト動作のキャンセル

デフォルト動作のキャンセルに関しては、`preventDefault()` メソッドと `isDefaultPrevented()` メソッドの 2 つがあります。`preventDefault()` メソッドを使用すると、イベントに関連付けられたデフォルト動作をキャンセルできます。既に `preventDefault()` を呼び出したかどうかを確認するには、`isDefaultPrevented()` メソッドを使用します。呼び出し済みの場合は戻り値として `true` が返され、まだ呼び出していない場合は `false` が返されます。

`preventDefault()` メソッドは、デフォルト動作のキャンセルが可能であるイベントに対してのみ使用できます。キャンセルの可否については、API ドキュメントで該当するイベントタイプの項目を参照するか、ActionScript で当該イベントオブジェクトの `cancelable` プロパティを確認してください。

デフォルト動作をキャンセルしても、イベントフローによるイベントオブジェクト処理の進行には影響しません。イベントオブジェクトをイベントフローから取り除くには、イベントフローメソッドを使用する必要があります。

Event クラスのサブクラス

多くのイベントは、Event クラスに定義されている共通のプロパティセットを使用すれば十分に表現できます。しかし、Event クラスのプロパティでは表現できない独特の性質を持ったイベントもあります。そのようなイベントのために、Flash Player API には Event クラスのサブクラスがいくつか定義されています。

各サブクラスには、そのイベントのカテゴリに特有のプロパティおよびイベントタイプが追加されています。たとえば、マウスの入力に関するイベントには、Event クラスに定義されているプロパティでは表現できない独特の性質があります。MouseEvent クラスは、Event クラスを拡張し、マウスイベントの発生位置や、発生時に特定のキーが押されていたかどうかなどの情報を格納するプロパティを追加したものです。

また、Event のサブクラスには、当該サブクラスに関連付けられたイベントタイプを表す定数が格納されます。たとえば MouseEvent クラスの場合、マウス関連のイベントタイプである `click`、`doubleClick`、`mouseDown`、`mouseUp` を表す定数がそれぞれ定義されています。

355 ページの「Event クラスのユーティリティメソッド」で説明したとおり、Event クラスを作成するときは、`clone()` および `toString()` メソッドをオーバーライドして、サブクラスに固有の機能を提供する必要があります

イベントリスナー

イベントリスナーとは、特定のイベントに応答して Flash Player により実行される関数のことであり、イベントハンドラとも呼ばれます。イベントリスナーを登録するには 2 つの手順を実行します。まず、目的のイベントに応答して Flash Player に実行させるための関数またはクラスメソッドを作成します。これは、リスナー関数またはイベントハンドラ関数と呼ばれることもあります。次に `addEventListener()` メソッドを使用して、そのリスナー関数を、イベントのターゲットまたは該当するイベントフローに含まれる任意の表示リストオブジェクトに登録します。

リスナー関数の作成

ActionScript 3.0 のイベントモデルのうち、リスナー関数の作成に関する部分は DOM イベントモデルに準拠していません。DOM イベントモデルでは、イベントリスナーとリスナー関数が明確に区別されています。イベントリスナーは EventListener インターフェイスを実装したクラスのインスタンスを意味し、リスナー関数はそのクラスが備える `handleEvent()` というメソッドを意味します。DOM イベントモデルにおいては、リスナー関数自体を登録するのではなく、リスナー関数を含んだクラスインスタンスを登録する形式をとります。

ActionScript 3.0 のイベントモデルには、イベントリスナーとリスナー関数との区別がありません。ActionScript 3.0 には EventListener インターフェイスがなく、リスナー関数はクラス内、クラス外のいずれに定義することもできます。また、リスナー関数の名前が handleEvent() である必要はなく、識別子として有効な任意の名前を付けることができます。ActionScript 3.0 においては、リスナー関数自体の名前を登録する形式をとります。

クラス外に定義したリスナー関数

次のコードは、赤い正方形のシェイプを表示する単純な SWF ファイルです。クラス内にある clickHandler() というリスナー関数で、赤い正方形に対するマウスクリックイベントを受け付けます。

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
}

function clickHandler(event:MouseEvent):void
{
    trace("clickHandler detected an event of type: " + event.type);
    trace("the this keyword refers to: " + this);
}
```

この SWF ファイルで、ユーザーによって正方形がクリックされると、Flash Player は次のトレース出力を生成します。

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

イベントオブジェクトが、`clickHandler()` にパラメータとして渡されています。これにより、リスナー関数の中でイベントオブジェクトを調べることができます。この例では、イベントオブジェクトの `type` プロパティを使用し、イベントがクリックイベントであることを確認しています。

また、この例では `this` キーワードの値も確認しています。`this` はグローバルオブジェクトを参照しています。なぜなら、リスナー関数をカスタムクラスやオブジェクトに属さない外部で定義しているからです。

クラスメソッドとして定義したリスナー関数

次の例では、前出の例と同じく ClickExample クラスを定義していますが、`clickHandler()` 関数を ChildSprite クラスのメソッドとして定義している点が異なります。

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, clickHandler);
    }
    private function clickHandler(event:MouseEvent):void
    {
        trace("clickHandler detected an event of type: " + event.type);
```

```
        trace("the this keyword refers to: " + this);
    }
}
```

この SWF ファイルで、ユーザーによって赤い正方形がクリックされると、Flash Player は次のトレース出力を生成します。

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

this キーワードは child という ChildSprite インスタンスを参照しています。この点に関しては動作が変更されており、ActionScript 2.0 とは異なります。ActionScript 2.0 のコンポーネントでは、UIEventDispatcher.addEventListener() に対してクラスメソッドを指定した場合、リスナーメソッドのスコープは、メソッドが定義されているクラスではなくイベントをブロードキャストしたコンポーネントに基づいて決まるようになっていました。つまり、この例と同じテクニックを ActionScript 2.0 で使用したとすると、this キーワードは、ChildSprite インスタンスではなくイベントをブロードキャストしたコンポーネントを参照することになります。

この従来の仕様では、リスナーメソッドからそのメソッドがあるクラス内の他のメソッドやプロパティにアクセスできないため、場合によっては非常に大きな問題が生じていました。これを回避するために、ActionScript 2.0 には、mx.util.Delegate クラスを使用してリスナーメソッドのスコープを変更するという方法が用意されていました。ActionScript 3.0 では addEventListener() の呼び出し時にバインドメソッドが作成されるため、この回避策は不要です。結果として、this キーワードの参照先は ChildSprite インスタンス child となり、プログラムで ChildSprite クラス内の他のメソッドやプロパティにアクセスできるようになりました。

望ましくないイベントリスナー登録の方法

第 3 のテクニックとして、汎用オブジェクトを作成し、そのオブジェクトのプロパティによって動的にリスナー関数を割り当てる方法がありますが、これを使用することはお勧めできません。この方法を説明するのは、ActionScript 2.0 で一般的に使用されていたという理由からです。ActionScript 3.0 では使用しないでください。このテクニックでは this キーワードの参照先がリスナーオブジェクトではなくグローバルオブジェクトになるため、望ましくありません。

次の例では、前出の例と同じく ClickExample クラスを定義していますが、リスナー関数を myListenerObj という汎用オブジェクトの一部として定義している点が異なります。

```
package
{
    import flash.display.Sprite;

    public class ClickExample extends Sprite
    {
        public function ClickExample()
        {
            var child:ChildSprite = new ChildSprite();
            addChild(child);
```

```
        }
    }
}

import flash.display.Sprite;
import flash.events.MouseEvent;

class ChildSprite extends Sprite
{
    public function ChildSprite()
    {
        graphics.beginFill(0xFF0000);
        graphics.drawRect(0,0,100,100);
        graphics.endFill();
        addEventListener(MouseEvent.CLICK, myListenerObj.clickHandler);
    }
}

var myListenerObj:Object = new Object();
myListenerObj.clickHandler = function (event:MouseEvent):void
{
        trace("clickHandler detected an event of type: " + event.type);
        trace("the this keyword refers to: " + this);
}
```

トレースの結果は次のようになります。

```
clickHandler detected an event of type: click
the this keyword refers to: [object global]
```

`this` の参照先は `myListenerObj` であるかのように見え、トレース出力も `[object Object]` になっていますが、実際の参照先はグローバルオブジェクトです。動的なプロパティ名を `addEventListener()` のパラメータとして指定した場合、Flash Player はバインドメソッドを作成できません。なぜなら、その場合に `listener` パラメータとして渡されるのはリスナー関数の単なるメモリアドレスであり、Flash Player がそのメモリアドレスと `myListenerObj` インスタンスを結び付ける手段がないからです。

イベントリスナーの管理

リスナー関数を管理するには、IEventDispatcher インターフェイスのメソッドを使用します。IEventDispatcher インターフェイスは、DOM イベントモデルにおける EventTarget インターフェイスの ActionScript 3.0 バージョンです。IEventDispatcher という名前からは、Event オブジェクトを送信 (送出) することが主な目的であるように思われますが、実際には、イベントリスナーを登録、確認、および削除する目的でメソッドを使用する機会が最も多いといえます。IEventDispatcher インターフェイスには、次のコードに示す 5 つのメソッドが定義されています。

```
package flash.events
{
```

```
    public interface IEventDispatcher
    {
        function addEventListener(eventName:String,
                    listener:Object,
                    useCapture:Boolean=false,
                    priority:Integer=0,
                    useWeakReference:Boolean=false):Boolean;

        function removeEventListener(eventName:String,
                listener:Object,
                useCapture:Boolean=false):Boolean;

        function dispatchEvent(eventObject:Event):Boolean;

        function hasEventListener(eventName:String):Boolean;
        function willTrigger(eventName:String):Boolean;
    }
}
```

Flash Player API では、EventDispatcher クラスに IEventDispatcher インターフェイスが実装されています。このクラスは、イベントフローの一部やイベントターゲットとして機能するすべてのクラスに対する基本クラスです。たとえば、DisplayObject クラスは EventDispatcher クラスを継承しています。このため、表示リスト内の任意のオブジェクトは IEventDispatcher インターフェイスのメソッドにアクセスできます。

イベントリスナーの追加

`addEventListener()` は、IEventDispatcher インターフェイスで最もよく使用されるメソッドです。リスナー関数を登録する際にはこのメソッドを使用します。`type` および `listener` の 2 つのパラメータは必須です。`type` パラメータでは、イベントのタイプを指定します。`listener` パラメータでは、目的のイベントが発生したときに実行するリスナー関数を指定します。`listener` パラメータには、関数への参照またはクラスメソッドへの参照のいずれかを指定できます。

> **メモ** `listener` パラメータを指定するときは括弧を使用しないでください。たとえば、次の `addEventListener()` メソッド呼び出しで、`clickHandler()` 関数は括弧なしで指定されます。
> `addEventListener(MouseEvent.CLICK, clickHandler)`

`addEventListener()` メソッドの `useCapture` パラメータを使用すると、イベントフローのどの段階においてリスナーをアクティブにするかを制御できます。`useCapture` に `true` を指定すると、そのリスナーはイベントフローのキャプチャ段階でアクティブになります。`useCapture` に `false` を指定すると、そのリスナーはイベントフローのターゲット段階とバブリング段階でアクティブになります。イベントフローのすべての段階においてイベントを受け取るには、`useCapture` に `true` を指定した場合と、`useCapture` に `false` を指定した場合の両方について、合計 2 回 `addEventListener()` を呼び出す必要があります。

addEventListener() メソッドの priority パラメータは、DOM Level 3 イベントモデルの正式なパラメータではありません。これは、イベントリスナーをより柔軟に編成できるようにするため、ActionScript 3.0 で独自に提供されているパラメータです。addEventListener() を呼び出す際に priority パラメータに整数値を指定することで、イベントリスナーの優先度を設定できます。デフォルト値は 0 ですが、負の整数値または正の整数値を指定することができます。指定した数値が高いほど、イベントリスナーが早い順序で実行されます。同じ優先度で登録された複数のイベントリスナーがある場合、それらは登録順に実行されます。つまり、先に登録したイベントリスナーほど早く実行されます。

useWeakReference パラメータを使用すると、そのリスナー関数に対する参照を通常の参照にするか、弱参照にするかを指定できます。このパラメータに true を指定すると、リスナー関数が使用されなくなった後もメモリ上に残り続けるのを防ぐことができます。Flash Player では、" ガベージコレクション " というテクニックによって、使用されなくなったオブジェクトをメモリから削除します。あるオブジェクトに対して参照が 1 つも存在しなくなったとき、そのオブジェクトはもう使用されないと見なされます。弱参照はガベージコレクションのシステムで無視されるため、弱参照による参照しか受けていないリスナー関数はガベージコレクションの対象となります。

イベントリスナーの削除

不要になったイベントリスナーを削除するには、removeEventListener() メソッドを使用します。不要になったリスナーは削除することをお勧めします。必須のパラメータは eventName および listener の 2 つで、addEventListener() メソッドの必須パラメータと同じです。前述したとおり、イベントフローのすべての段階においてイベントを受け取る場合は、useCapture に true を指定した場合と false を指定した場合の両方について、合計 2 回 addEventListener() を呼び出します。その後で両方のイベントリスナーを削除するには、useCapture に true を指定した場合と false を指定した場合の両方について合計 2 回 removeEventListener() を呼び出す必要があります。

イベントの送出

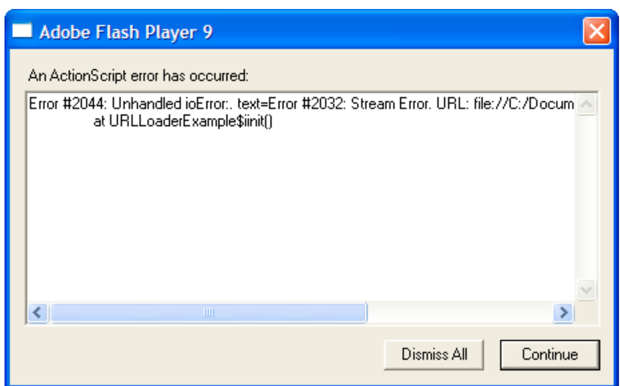
dispatchEvent() メソッドは、独自のイベントオブジェクトをイベントフローに送出する場合に使用する上級開発者向けのメソッドです。このメソッドに指定できるパラメータは 1 つだけで、Event クラスまたはそのサブクラスのインスタンスを指定します。送出されたイベントオブジェクトの target プロパティには、dispatchEvent() が呼び出されたオブジェクトが設定されます。

既存のイベントリスナーの確認

IEventDispatcher インターフェイスが備える残り 2 つのメソッドは、イベントリスナーの存在に関する有用な情報を取得するために使用します。指定した表示リストオブジェクトに、指定したイベントタイプのイベントリスナーが登録されている場合、hasEventListener() メソッドは true を返します。willTrigger() メソッドも、指定した表示リストオブジェクトについてリスナーが登録されている場合に true を返します。ただし、willTrigger() では指定した表示オブジェクト自体を確認するだけでなく、イベントフローのすべての段階におけるその表示リストオブジェクトの祖先もすべて確認します。

エラーイベントのリスナーを登録しない場合

エラー処理に関して、ActionScript 3.0 における最も重要なメカニズムはイベントではなく例外ですが、非同期の操作 (ファイルのロードなど) には例外処理を使用できません。非同期の操作を実行中にエラーが発生すると、Flash Player によりエラーイベントオブジェクトが送出されます。エラーイベントに対するリスナーを作成していない場合、デバッグ版の Flash Player では、発生したエラーに関する情報がダイアログボックスに表示されます。たとえば、ファイルのロード処理において無効な URL を指定すると、デバッグ版 Flash Player では、エラーを示すこのダイアログボックスが表示されます。



エラーイベントのほとんどは ErrorEvent クラスに基づいているため、Flash Player で表示するためのエラーメッセージを格納する text というプロパティがあります。ただし、StatusEvent クラスと NetStatusEvent クラスの 2 つはこれに該当しません。これらのクラスには level プロパティ (StatusEvent.level および NetStatusEvent.info.level) があります。level プロパティの値が "error" の場合、これらのイベントタイプはエラーイベントと見なされます。

エラーイベントが発生しても、それによって SWF ファイルの実行が停止することはありません。単に、デバッグ版のブラウザプラグインまたはスタンドアローンの Player ではダイアログボックスによって、オーサリング環境の Player では出力パネルのメッセージによって、また、Adobe Flex Builder 2 ではログファイルのエントリによって、エラーの情報が示されるだけです。リリース版の Flash Player では何も表示されません。

例 : Alarm Clock

Alarm Clock の例は、アラームが鳴り出す時刻をユーザーが指定できる時計と、その時刻に表示されるメッセージで構成されます。Alarm Clock の例は、第 6 章の「日付と時刻の操作」の SimpleClock アプリケーションの上に構築されています。Alarm Clock で、次のような、ActionScript 3.0 によるイベント処理のいくつかの側面を示します。

- イベントのリスンと応答
- リスナーへのイベントの通知
- カスタムイベントタイプの作成

Alarm Clock アプリケーションのファイルは、"Samples/AlarmClock" フォルダにあります。アプリケーションには次のファイルが含まれます。

ファイル	説明
AlarmClockApp.mxml	MXML で記述された Flex 用メインアプリケーションファイル
com/example/programmingas3/clock/AlarmClock.as	SimpleClock クラスを拡張し、目覚まし時計の機能を追加するクラス
com/example/programmingas3/clock/AlarmEvent.as	AlarmClock クラスの `alarm` イベントのイベントオブジェクトとして機能するカスタムイベントクラス (flash.events.Event のサブクラス)
com/example/programmingas3/clock/AnalogClockFace.as	時計の丸い文字盤と時間に合わせた時針、分針、秒針を描画するコード (SimpleClock の例で説明)
com/example/programmingas3/clock/SimpleClock.as	単純な時刻合わせ機能を持つ時計インターフェイスコンポーネント (SimpleClock の例で説明)

Alarm Clock の概要

この例の時計の主な機能である時刻の追跡と文字盤の表示などは、204 ページの「例 : 単純なアナログ時計」で説明した SimpleClock アプリケーションのコードを再利用しています。AlarmClock クラスは、アラーム時刻の設定やアラームが " 鳴り出す " ときの通知の指定など、目覚まし時計に必要な機能を追加することで、前の例の SimpleClock クラスを拡張します。

イベントが作成された目的は、何かが発生したときの通知を行うためです。AlarmClock クラスが Alarm イベントを公開し、他のオブジェクトは Alarm イベントをリスンして、必要なアクションを実行できます。また、AlarmClock クラスは Timer クラスのインスタンスを使用して、アラームをトリガする時刻を判断します。AlarmClock クラスと同様に、Timer クラスは指定された時間が経過すると、他のオブジェクト (この例では AlarmClock インスタンス) に通知するイベントを提供します。ほとんどの ActionScript アプリケーションと同様に、イベントは Alarm Clock サンプルアプリケーションの機能の重要な部分を形成します。

アラームのトリガ

既に説明したように、AlarmClock クラスが実際に提供する機能は、アラームの設定とトリガに関する機能だけです。組み込みの Timer クラス (flash.utils.Timer) を使用して、指定された時間が経過した後で実行するようにコードを定義できます。AlarmClock クラスは Timer インスタンスを使用して、アラームを鳴らす時刻を判断します。

```
import flash.events.TimerEvent;
import flash.utils.Timer;

/**
 * アラームに使用する Timer
 */
public var alarmTimer:Timer;
...
/**
 * 指定されたサイズの新しい AlarmClock のインスタンス化
 */
public override function initClock(faceSize:Number = 200):void
{
   super.initClock(faceSize);
   alarmTimer = new Timer(0, 1);
   alarmTimer.addEventListener(TimerEvent.TIMER, onAlarm);
}
```

AlarmClock クラスで定義された Timer インスタンスの名前は alarmTimer です。AlarmClock インスタンスに必要なセットアップ操作を実行する initClock() メソッドは、alarmTimer 変数を使用して 2 つの処理を行います。まず、Timer インスタンスが 0 ミリ秒待機し、タイマーイベントを 1 回だけトリガするように指示するパラメータを使用して、変数をインスタンス化します。alarmTimer をインスタンス化した後、コードは変数の addEventListener() メソッドを呼び出し、変数の timer イベントをリスンしようとしていることを示します。指定された時間が経過した後、timer イベントを送出することで、Timer インスタンスが動作します。AlarmClock クラスは、独自のアラームを鳴らすために、いつ timer イベントが送出されたかを知る必要があります。addEventListener() を呼び出すことで、AlarmClock コードは自分自身をリスナーとして alarmTimer に登録します。2 つのパラメータは、AlarmClock クラスが timer イベントをリスンしようとしていることを示し (定数 TimerEvent.TIMER で指定)、イベントが発生したときに、イベントに対する応答として AlarmClock クラスの onAlarm() メソッドを呼び出す必要があることを示しています。

実際にアラームを設定するために、AlarmClock クラスの setAlarm() メソッドが次のように呼び出されます。

```
/**
 * アラームを鳴らす時刻を設定する
 * パラメータ hour は、アラーム時刻の時間部分
 * パラメータ minutes は、アラーム時刻の分部分
 * パラメータ message は、アラームが鳴り出したときに表示するメッセージ
 * 戻り値は、アラームが鳴り出す時刻
 */
public function setAlarm(hour:Number = 0, minutes:Number = 0, message:String = 
"Alarm!"):Date
{
   this.alarmMessage = message;
   var now:Date = new Date();
   // 今日の日付でこの時刻を作成
   alarmTime = new Date(now.fullYear, now.month, now.date, hour, minutes);

   // 今日、指定された時刻が既に過ぎているかどうかを判断
   if (alarmTime <= now)
   {
      alarmTime.setTime(alarmTime.time + MILLISECONDS_PER_DAY);
   }

   // 現在設定されている場合、アラームタイマーを停止
   alarmTimer.reset();
   // アラームを鳴らすまでに経過する時間を計算
   // ( アラーム時刻と現在時刻の差 ) をミリ秒単位で計算し、
   // アラームタイマーの遅延として設定
   alarmTimer.delay = Math.max(1000, alarmTime.time - now.time);
   alarmTimer.start();

   return alarmTime;
}
```

このメソッドは、アラームメッセージの保存、アラームを鳴らす時刻の実際の瞬間を表す Date オブジェクト (alarmTime) の作成など、複数の処理を実行します。この章と最も関係がある部分として、メソッドの最後の数行で、alarmTimer 変数のタイマーが設定され、アクティブ化されます。まず、reset() メソッドが呼び出され、タイマーが停止され、既に実行されていた場合はリセットされます。次に、現在時刻 (now 変数で表現) が alarmTime 変数の値から減算され、アラームを鳴らすまでに経過する時間がミリ秒単位で決定されます。Timer クラスは timer イベントを絶対時間ではトリガしないため、この相対的な時間差が alarmTimer の delay プロパティに割り当てられます。最後に start() メソッドが呼び出され、タイマーが実際に開始されます。

指定された時間が経過すると、alarmTimer は timer イベントを送出します。AlarmClock クラスは、このイベントのリスナーとして onAlarm() メソッドを登録しているため、timer イベントが発生すると、onAlarm() が呼び出されます。

```
/**
 * timer イベントが送出されたときに呼び出される
 */
public function onAlarm(event:TimerEvent):void
{
    trace("Alarm!");
    var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
    this.dispatchEvent(alarm);
}
```

イベントリスナーとして登録されているメソッドは、適切なシグネチャ (メソッドのパラメータと戻り値の型のセット) で定義されている必要があります。メソッドを Timer クラスの timer イベントのリスナーとして使用するには、データ型が Event クラスのサブクラス TimerEvent (flash.events.TimerEvent) であるパラメータを 1 つ定義する必要があります。Timer インスタンスはイベントリスナーを呼び出すときに、TimerEvent インスタンスをイベントオブジェクトとして渡します。

他へのアラームの通知

Timer クラスと同様に、AlarmClock クラスはイベントを提供し、これによってアラームが鳴り出すときに他のコードが通知を受信できます。ActionScript に組み込まれているイベント処理フレームワークをクラスで使用するには、そのクラスで flash.events.IEventDispatcher インターフェイスを実装する必要があります。最も一般的な方法として、IEventDispatcher の標準実装を提供する flash.events.EventDispatcher クラスを拡張する方法があります (または、EventDispatcher のサブクラスのいずれかを拡張します)。既に説明したように、AlarmClock クラスは SimpleClock クラスを拡張し、その結果、mx.core.UIComponent クラスが拡張され、(継承のチェーンを通じて) EventDispatcher クラスが拡張されます。すなわち、AlarmClock クラスには、既に独自のイベントを提供する組み込み機能があります。

他のコードは、AlarmClock が EventDispatcher から継承した addEventListener() メソッドを呼び出すことで、AlarmClock クラスの alarm イベントの通知を受けるように登録できます。alarm イベントが生成されたことを AlarmClock インスタンスが他のコードに通知する準備ができている場合は、同様に EventDispatcher から継承された dispatchEvent() メソッドを呼び出して通知できます。

```
var alarm:AlarmEvent = new AlarmEvent(this.alarmMessage);
this.dispatchEvent(alarm);
```

これらのコード行は、AlarmClock クラスの onAlarm() メソッドにあります (コードの全体は既に示しています)。AlarmClock インスタンスの dispatchEvent() メソッドが呼び出され、次に、AlarmClock インスタンスの alarm イベントがトリガされたことが登録済みのすべてのリスナーに通知されます。dispatchEvent() に渡されるパラメータは、リスナーメソッドに渡されるイベントオブジェクトです。この例では、特に作成された Event サブクラスである AlarmEvent クラスのインスタンスです。

カスタムアラームイベントの提供

すべてのイベントリスナーが、トリガされる特定のイベントに関する情報が設定されているイベントオブジェクトパラメータを受け取ります。多くの場合、イベントオブジェクトは Event クラスのインスタンスです。ただし、場合によっては、追加の情報をイベントリスナーに提供すると便利なことがあります。この章で既に説明したように、一般的にはこれを実現するために、新しいクラス (Event クラスのサブクラス) を定義し、このクラスのインスタンスをイベントオブジェクトとして使用します。この例では、AlarmClock クラスの alarm イベントが送出されたときに、AlarmEvent インスタンスがイベントオブジェクトとして使用されています。ここで示す AlarmEvent クラスは、alarm イベントに関する追加の情報 (具体的にはアラームメッセージ) を提供します。

```
import flash.events.Event;

/**
 * このカスタム Event クラスによって、基本の Event に message プロパティを追加
 */
public class AlarmEvent extends Event
{
    /**
     * 新しい AlarmEvent タイプの名前
     */
    public static const ALARM:String = "alarm";

    /**
     * このイベントオブジェクトと共にイベントハンドラに渡すことができる
     * テキストメッセージ
     */
    public var message:String;

    /**
     *  コンストラクタ
```

```
     * パラメータ message は、アラームが鳴り出したときに表示するテキスト
     */
    public function AlarmEvent(message:String = "ALARM!")
    {
        super(ALARM);
        this.message = message;
    }
    ...
}
```

カスタムイベントオブジェクトクラスを作成する最適な方法は、前の例で示したように、Event クラスを拡張するクラスを定義することです。継承された機能を補うために、AlarmEvent クラスでは、イベントに関連付けられたアラームメッセージのテキストを含むプロパティ message を定義しています。message 値はパラメータとして AlarmEvent コンストラクタに渡されます。AlarmEvent クラスは、定数 ALARM も定義します。これは、AlarmClock クラスの addEventListener() メソッドを呼び出すときに、特定のイベント (alarm) を参照するために使用できます。

カスタム機能を追加するほかに、Event サブクラスでは、ActionScript イベント処理フレームワークの一部として継承された clone() メソッドをオーバーライドする必要があります。Event サブクラスでは、オプションで、toString() メソッドが呼び出されたときに返される値にカスタムイベントのプロパティを含めるように、継承された toString() をオーバーライドすることもできます。

```
/**
 * 現在のインスタンスのコピーを作成して返す
 * 戻り値は、現在のインスタンスのコピー
 */
public override function clone():Event
{
    return new AlarmEvent(message);
}

/**
 * 現在のインスタンスのすべてのプロパティを含む String を
 * 返す
 * 戻り値は、現在のインスタンスのストリング表現
 */
public override function toString():String
{
    return formatToString("AlarmEvent", "type", "bubbles", "cancelable",
"eventPhase", "message");
}
```

オーバーライドされた clone() メソッドは、現在のインスタンスと一致するようにすべてのカスタムプロパティが設定されているカスタム Event サブクラスの新しいインスタンスを返す必要があります。オーバーライドされた toString() メソッドは、ユーティリティメソッド formatToString() (Event から継承) を使用して、カスタムタイプの名前およびカスタムタイプのすべてのプロパティの名前と値が設定されたストリングを提供します。

第 14 章

ネットワーキングとコミュニケーション

14

flash.net パッケージには、インターネットを介したデータ送受信に関するクラスが含まれています。たとえば、リモート URL からのコンテンツのロード、他の Flash Player インスタンスとの通信、リモート Web サイトへの接続などに使用するクラスがあります。flash.net パッケージに属するクラスの多くは、ActionScript の以前のバージョンではトップレベルに属していました。ActionScript 3.0 では、それらの多数のクラスが flash.net パッケージに移動しました。

flash.net パッケージに含まれるクラスは、FileReference、LocalConnection、NetConnection、NetStream、SharedObject、Socket、URLLoader、URLRequest、URLStream、URLVariables、および XMLSocket です。また、flash.net には `navigateToURL()`、`sendToURL()`、`registerClassAlias()`、および `getClassByAlias()` の 4 つのメソッドが含まれています。

この章では、SWF ファイルで外部ファイルおよび他の Adobe Flash Player 9 インスタンスと通信できるようにする方法について説明します。また、外部ソースからデータをロードする方法、Java サーバーと Flash Player の間でメッセージを送信する方法、FileReference および FileReferenceList クラスを使用してファイルのアップロードとダウンロードを行う方法についても説明します。

目次

外部データの操作 372
他の Flash Player インスタンスへの接続 379
ソケット接続 385
ローカルデータの保存 390
ファイルのアップロードおよびダウンロードに関する操作 394
例 : Telnet クライアントの構築 404
例 : ファイルのアップロードとダウンロード 408

外部データの操作

大規模な ActionScript アプリケーションを作成する場合、サーバーサイドスクリプトとの通信や、外部の XML またはテキストファイルのロードが必要になることがよくあります。これに関する動作は ActionScript 2.0 と ActionScript 3.0 では大きく異なります。ActionScript の以前のバージョンでは、リモートのテキストファイルをロードする際に LoadVars クラスと `LoadVars.onData()` イベントハンドラを使用しました。ActionScript 3.0 では、外部ファイルをロードするには URLLoader クラスと URLRequest クラスを使用します。リモートコンテンツが名前 / 値のペアの形式になっている場合は、URLVariables クラスを使用してサーバーの結果を解析できます。

リモート XML ドキュメントをロードするには、URLLoader および URLRequest クラスを使用します。その後、XML クラスのコンストラクタ、XMLDocument クラスのコンストラクタ、または `XMLDocument.parseXML()` メソッドを使用して XML ドキュメントを解析できます。これにより、URLVariables クラスと XML クラスのいずれを使用する場合でも外部ファイルのロードに関するコードを同じにすることができ、ActionScript コードが簡潔になります。

URLLoader および URLVariables クラスの使用

ActionScript 3.0 では、LoadVars クラスが URLLoader クラスと URLVariables クラスに置き換えられました。URLLoader クラスは、指定した URL からテキスト、バイナリデータ、または URL エンコード形式の変数をダウンロードする際に使用します。データ駆動の動的な ActionScript アプリケーションで使用するテキストファイル、XML、その他の情報をダウンロードする場合には URLLoader クラスが便利です。URLLoader クラスでは ActionScript 3.0 の高度なイベント処理モデルを利用しているため、`complete`、`httpStatus`、`ioError`、`open`、`progress`、`securityError` などのイベントを受け付けて処理できます。新しいイベント処理モデルはエラーやイベントをより効率よく扱うことができ、ActionScript 2.0 における `LoadVars.onData`、`LoadVars.onHTTPStatus`、`LoadVars.onLoad` 各イベントハンドラのサポートに比べて大きく改良されています。イベントの処理の詳細については、第 13 章の「イベントの処理」を参照してください。

ActionScript の以前のバージョンにおける XML クラスおよび LoadVars クラスの場合と同様に、URLLoader でも、指定した URL のデータはダウンロードが完了するまで利用可能となりません。ダウンロードの進捗状況 (ロード済みバイト数と合計バイト数) は、`flash.events.ProgressEvent.PROGRESS` イベントをリッスンすることにより監視できますが、ファイルのロードが非常に早く完了した場合は `ProgressEvent.PROGRESS` イベントが送出されないことがあります。ファイルのダウンロードが正常に完了すると、`flash.events.Event.COMPLETE` イベントが送出されます。ロードされたデータは、UTF-8 または UTF-16 のエンコード形式からストリングにデコードされます。

メモ

`URLRequest.contentType` が設定されていない場合、値は `application/x-www-form-urlencoded` 形式で送信されます。

URLLoader.load() メソッドのパラメータは、URLRequest クラスのオブジェクトを指定する request パラメータの 1 つだけです (URLLoader クラスのコンストラクタにも、必要に応じて同じパラメータを指定できます)。URLRequest オブジェクトには、1 件の HTTP リクエストに関して、ターゲット URL、リクエストメソッド (GET または POST)、付加的なヘッダー情報、MIME タイプ (XML コンテンツのアップロード時など)、その他すべての情報が格納されます。

たとえば、XML パケットをサーバーサイドスクリプトにアップロードする場合の ActionScript 3.0 コードは次のようになります。

```
var secondsUTC:Number = new Date().time;
var dataXML:XML =
  <login>
    <time>{secondsUTC}</time>
    <username>Ernie</username>
    <password>guru</password>
  </login>;
var request:URLRequest = new URLRequest("http://www.yourdomain.com/login.cfm");
request.contentType = "text/xml";
request.data = dataXML.toXMLString();
request.method = URLRequestMethod.POST;
var loader:URLLoader = new URLLoader();
try
{
  fr.download(request);
}
catch (error:ArgumentError)
{
  trace("An ArgumentError has occurred.");
}
catch (error:SecurityError)
{
  trace("A SecurityError has occurred.");
}
```

このコードでは、サーバーに送信する XML パケットを含んだ dataXml という XML インスタンスを作成します。次に URLRequest の contentType プロパティに "text/xml" を設定し、URLRequest の data プロパティに、XML.toXMLString() メソッドでストリングに変換した XML パケットの内容を設定します。最後に、新しい URLLoader インスタンスを作成し、URLLoader.load() メソッドを使用してリモートスクリプトに要求を送信します。

URL リクエストで渡すパラメータを指定するには、次の 3 つの方法があります。

- URLVariables コンストラクタで指定する方法
- URLVariables.decode() メソッドで指定する方法
- URLVariables オブジェクト自体の特定のプロパティとして設定する方法

URLVariables のコンストラクタまたは `URLVariables.decode()` メソッドで変数を定義する場合、アンパサンド文字 (&) は必ず URL エンコードしておく必要があります。アンパサンドには特別な意味があり、区切り文字として機能するからです。URL エンコードすると、アンパサンド `&` は `%26` になります。

外部ドキュメントからのデータのロード

動的なアプリケーションを ActionScript 3.0 で作成する場合、外部のファイルまたはサーバーサイドスクリプトからデータをロードするようにすると、ActionScript ファイルの編集と再コンパイルを必要としない動的なアプリケーションを実現できます。たとえば、" 今日の一言 " アプリケーションを作成する場合、サーバーサイドスクリプトで、データベースからランダムに選んだ一言を取得して 1 日 1 回テキストファイルに保存するようにします。そうすれば、ActionScript アプリケーションでは静的なテキストファイルをロードするだけでよく、毎回データベースへのクエリを発行する必要はなくなります。

次のコードでは、URLRequest オブジェクトと URLLoader オブジェクトを作成し、それによって "params.txt" という外部のテキストファイルから内容をロードします。

```
var request:URLRequest = new URLRequest("params.txt");
var loader:URLLoader = new URLLoader();
loader.load(request);
```

また、この例は次のように簡略化することもできます。

```
var loader:URLLoader = new URLLoader(new URLRequest("params.txt"));
```

リクエストメソッドを指定しない場合、Flash Player では、デフォルトで HTTP `GET` メソッドを使用してコンテンツをロードします。`POST` メソッドでデータを送信する必要がある場合は、次のように、静的定数 `URLRequestMethod.POST` を使用して `request.method` プロパティに `POST` を設定します。

```
var request:URLRequest = new URLRequest("sendfeedback.cfm");
request.method = URLRequestMethod.POST;
```

実行時にロードされる外部ドキュメント "params.txt" の内容は次のようなデータです。

```
monthNames=January,February,March,April,May,June,July,August,September,October,N
   ovember,December&dayNames=Sunday,Monday,Tuesday,Wednesday,Thursday,Friday,Sat
   urday
```

このファイルには、`monthNames` および `dayNames` の 2 つのパラメータが含まれています。各パラメータの内容は、ストリングとして解析されるカンマ区切りリストです。このリストは `String.split()` メソッドを使用して分割し、配列に変換できます。

ヒント	外部データファイル内の変数名には、予約語やプログラミング言語の構成要素を含めないでください。そのような変数名を使用すると、コードの解読やデバッグ作業が困難になります。

データのロードが完了すると Event.COMPLETE イベントが送出され、次のように、URLLoader の data プロパティで外部ドキュメントの内容にアクセスできるようになります。

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    trace(loader2.data);
}
```

リモートドキュメントに名前と値のペアが格納されている場合は、URLVariables クラスを使用し、ロードしたファイルの内容を次のように渡すことでデータを解析できます。

```
private function completeHandler(event:Event):void
{
    var loader2:URLLoader = URLLoader(event.target);
    var variables:URLVariables = new URLVariables(loader2.data);
    trace(variables.dayNames);
}
```

外部ファイルからロードした個々の名前 / 値ペアから、URLVariables オブジェクトの個別のプロパティが作成されます。このコード例にある variables オブジェクトの各プロパティは、ストリングとして扱われます。名前 / 値ペアの値がアイテムのリストである場合は、次のように String.split() メソッドを呼び出すことでストリングから配列に変換できます。

```
var dayNameArray:Array = variables.dayNames.split(",");
```

ヒント	外部テキストファイルから数値データをロードするには、トップレベル関数の int()、uint()、Number() などを使用して値を数値に変換する必要があります。

リモートファイルの内容をロードして新しい URLVariables オブジェクトを作成する方法とは別に、URLLoader.dataFormat プロパティに URLLoaderDataFormat クラスの静的プロパティのいずれかを設定する方法もあります。URLLoader.dataFormat プロパティに設定できる値は次の 3 つのうちいずれかです。

- URLLoaderDataFormat.BINARY : URLLoader.data プロパティには、ByteArray オブジェクトとしてバイナリデータが格納されます。
- URLLoaderDataFormat.TEXT : URLLoader.data プロパティには、String オブジェクトとしてテキストが格納されます。
- URLLoaderDataFormat.VARIABLES : URLLoader.data プロパティには、URLVariables オブジェクトとして URL エンコード形式の変数が格納されます。

次のコードでは URLLoader.dataFormat プロパティを URLLoaderDataFormat.VARIABLES に設定しているため、ロードしたデータは自動的に解析されて URLVariables オブジェクトが作成されます。

```
package
{
    import flash.display.Sprite;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLLoaderDataFormat;
    import flash.net.URLRequest;

    public class URLLoaderDataFormatExample extends Sprite
    {
        public function URLLoaderDataFormatExample()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
    params.txt");
            var variables:URLLoader = new URLLoader();
            variables.dataFormat = URLLoaderDataFormat.VARIABLES;
            variables.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                variables.load(request);
            }
            catch (error:Error)
            {
                trace("Unable to load URL: " + error);
            }
        }
        private function completeHandler(event:Event):void
        {
            var loader:URLLoader = URLLoader(event.target);
            trace(loader.data.dayNames);
        }
    }
}
```

メモ

URLLoader.dataFormat のデフォルト値は URLLoaderDataFormat.TEXT です。

次の例に示すように、外部ファイルから XML をロードする方法も、URLVariables をロードする場合と同じです。URLRequest インスタンスと URLLoader インスタンスを作成し、それらを使用してリモート XML ドキュメントをダウンロードします。ファイルのダウンロードが完了すると、`Event.COMPLETE` イベントが送出され、外部ファイルの内容は XML インスタンスに変換されます。このインスタンスは XML のメソッドとプロパティを使用して解析できます。

```
package
{
    import flash.display.Sprite;
    import flash.errors.*;
    import flash.events.*;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class ExternalDocs extends Sprite
    {
        public function ExternalDocs()
        {
            var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
    data.xml");
            var loader:URLLoader = new URLLoader();
            loader.addEventListener(Event.COMPLETE, completeHandler);
            try
            {
                loader.load(request);
            }
            catch (error:ArgumentError)
            {
                trace("An ArgumentError has occurred.");
            }
            catch (error:SecurityError)
            {
                trace("A SecurityError has occurred.");
            }
        }
        private function completeHandler(event:Event):void
        {
            var dataXML:XML = XML(event.target.data);
            trace(dataXML.toXMLString());
        }
    }
}
```

外部スクリプトとの通信

URLVariables クラスを使用すると、外部データファイルからデータをロードできるだけでなく、サーバーサイドスクリプトに変数を送信し、サーバーの応答を処理することもできます。たとえば、ゲームを作成する場合、ユーザーの得点をサーバーに送信して高得点者リストに掲載できるかどうか確認することや、ユーザーのログイン情報をサーバーに送信して検証することなどが考えられます。サーバーサイドスクリプトでは、ユーザー名とパスワードを処理してデータベースに照会し、入力されたユーザー情報が有効かどうかの確認を応答として返します。

次のコードでは、variables という URLVariables オブジェクトを作成し、このオブジェクトを使用して name という新しい変数を作成します。次に、変数の送信先となるサーバーサイドスクリプトの URL を指定する URLRequest オブジェクトを作成します。その後、URLRequest オブジェクトの method プロパティに、HTTP POST リクエストで変数を送信することを設定します。URL リクエストに URLVariables オブジェクトを追加するために、先ほど作成した URLVariables オブジェクトを URLRequest オブジェクトの data プロパティに設定します。最後に、URLLoader インスタンスを作成し、URLLoader.load() メソッドを呼び出してリクエストを送信します。

```
var variables:URLVariables = new URLVariables("name=Franklin");
var request:URLRequest = new URLRequest();
request.url = "http://www.[yourdomain].com/greeting.cfm";
request.method = URLRequestMethod.POST;
request.data = variables;
var loader:URLLoader = new URLLoader();
loader.dataFormat = URLLoaderDataFormat.VARIABLES;
loader.addEventListener(Event.COMPLETE, completeHandler);
try
{
  loader.load(request);
}
catch (error:Error)
{
  trace("Unable to load URL");
}

function completeHandler(event:Event):void
{
  trace(event.target.data.welcomeMessage);
}
```

次のコードは、前の例で使用されている ColdFusion® ドキュメント "greeting.cfm" の内容です。

```
<cfif NOT IsDefined("Form.name") OR Len(Trim(Form.Name)) EQ 0>
  <cfset Form.Name = "Stranger" />
</cfif>
<cfoutput>welcomeMessage=#UrlEncodedFormat("Welcome, " & Form.name)#
</cfoutput>
```

他の Flash Player インスタンスへの接続

LocalConnection クラスを使用すると、他の Flash Player インスタンス (HTML コンテナ内の SWF、組み込み Player やスタンドアローン Player 上の SWF など) との間で通信ができます。これにより、たとえば Web ブラウザ上の SWF ファイルや C# アプリケーションに組み込まれた SWF など複数の Flash Player インスタンス間でデータの共有を実現し、非常に用途の広いアプリケーションを作成できます。

LocalConnection クラス

LocalConnection クラスを使用すれば、`fscommand()` メソッドや JavaScript を使用することなく、他の SWF ファイルに対して指示を送る機能を備えた SWF ファイルを作成できます。LocalConnection オブジェクトでの通信相手は、同じクライアントコンピュータ上で動作する SWF ファイルに限られますが、通信相手が同じアプリケーションで動作している必要はありません。たとえば、ブラウザ上で動作している SWF ファイルとプロジェクタの SWF ファイルで情報を共有しつつ、プロジェクタがローカルの情報を管理し、ブラウザ上の SWF がリモート接続を実行するといった分担もできます。プロジェクタとは、スタンドアローンアプリケーションとして実行可能な形式で保存した SWF ファイルです。つまり、プロジェクタは Flash Player がインストールされていることを必要としません。なぜなら、プロジェクタは実行可能ファイルの中に埋め込まれるからです。

ActionScript 3.0 で作成された LocalConnection オブジェクトは、ActionScript 1.0 または 2.0 で作成された LocalConnection オブジェクトと通信できます。また、その逆も可能です。ActionScript 1.0 または 2.0 で作成された LocalConnection オブジェクトは、ActionScript 3.0 で作成された LocalConnection オブジェクトと通信できます。Flash Player は、このバージョンが異なる LocalConnection オブジェクト間の通信を自動的に処理します。

LocalConnection オブジェクトの最も簡単な使用方法は、同じドメイン内の LocalConnection オブジェクト間だけで通信することです。なぜなら、その場合はセキュリティの問題がないからです。しかし、異なるドメイン間で通信を行う必要がある場合は、セキュリティ対策を実施する必要があります。いくつかの方法があります。詳細については、『ActionScript 3.0 リファレンスガイド』で、`send()` の `connectionName` パラメータに関する説明と、`allowDomain()` と `domain` の項を参照してください。

ヒント	LocalConnection オブジェクトを使用して 1 つの SWF ファイル内でデータを送受信することは可能ですが、お勧めしません。代わりに、共有オブジェクトを使用してください。

LocalConnection オブジェクトにコールバックメソッドを追加するには、次の 3 つの方法があります。

- LocalConnection クラスのサブクラスを作成してメソッドを追加する方法
- `LocalConnection.client` プロパティに、メソッドを実装したオブジェクトを設定する方法
- LocalConnection を拡張した動的なクラスを作成し、動的にメソッドを追加する方法

第 1 の方法では、LocalConnection クラスを拡張することでコールバックメソッドを追加します。LocalConnection インスタンスに動的にメソッドを追加するのではなく、カスタムクラス内にメソッドを定義します。この方法を使用する例を次のコードに示します。

```
package
{
   import flash.net.LocalConnection;
   public class CustomLocalConnection extends LocalConnection
   {
      public function CustomLocalConnection(connectionName:String)
      {
         try
         {
            connect(connectionName);
         }
         catch (error:ArgumentError)
         {
            // サーバーは既にあり、接続済みとする
         }
      }
      public function onMethod(timeString:String):void
      {
         trace("onMethod called at: " + timeString);
      }
   }
}
```

DynamicLocalConnection クラスの新しいインスタンスを作成するには、次のコードを使用します。

```
var serverLC:CustomLocalConnection;
serverLC = new CustomLocalConnection("serverName");
```

第 2 の方法では、`LocalConnection.client` プロパティを使用してコールバックメソッドを追加します。次のように、カスタムクラスを作成し、そのクラスの新しいインスタンスを `client` プロパティに設定します。

```
var lc:LocalConnection = new LocalConnection();
lc.client = new CustomClient();
```

LocalConnection.client プロパティは、呼び出しが必要なオブジェクトコールバックメソッドを示します。このコードでは、client プロパティに CustomClient というカスタムクラスの新しいインスタンスを設定しています。client プロパティのデフォルト値は、現在の LocalConnection インスタンスです。client プロパティは、同じメソッド群を備えているが動作が異なる 2 つのデータハンドラがある場合に使用できます。たとえば、あるウィンドウのボタンによって別のウィンドウの表示を切り替えるようなアプリケーションが該当すると考えられます。

CustomClient クラスを作成するには、次のようなコードを使用します。

```
package
{
   public class CustomClient extends Object
   {
      public function onMethod(timeString:String):void
      {
         trace("onMethod called at: " + timeString);
      }
   }
}
```

第 3 の方法では、次のように、動的なクラスを作成して動的にコールバックメソッドを追加します。これは、ActionScript の以前のバージョンで LocalConnection クラスを使用する方法と非常によく似ています。

```
import flash.net.LocalConnection;
dynamic class DynamicLocalConnection extends LocalConnection {}
```

このクラスにコールバックメソッドを動的に追加するには、次のコードを使用します。

```
var connection:DynamicLocalConnection = new DynamicLocalConnection();
connection.onMethod = this.onMethod;
// ここにコードを追加する
public function onMethod(timeString:String):void
{
   trace("onMethod called at: " + timeString);
}
```

上記の方法でコールバックメソッドを追加することは、コードの移植性がそれほどよくないので、お勧めできません。また、この方法でローカル接続を作成すると、ダイナミックプロパティへのアクセスは sealed プロパティにアクセスするより大幅に時間を要するため、パフォーマンスの問題が起きるおそれがあります。

2 つの Flash Player インスタンス間でのメッセージ送信

LocalConnection クラスを使用すると、Flash Player の異なるインスタンス間での通信ができます。たとえば、1 つの Web ページ上に複数の Flash Player インスタンスがある場合や、ポップアップウィンドウ内の Flash Player インスタンスから提供されるデータを別の Flash Player インスタンスで取得する場合などが考えられます。

次のコードでは、サーバーとして機能するローカル接続オブジェクトを定義し、他の Flash Player インスタンスからの接続を受け付けます。

```
package
{
  import flash.net.LocalConnection;
  import flash.display.Sprite;
  public class ServerLC extends Sprite
  {
    public function ServerLC()
    {
      var lc:LocalConnection = new LocalConnection();
      lc.client = new CustomClient1();
      try
      {
        lc.connect("conn1");
      }
      catch (error:Error)
      {
        trace("error:: already connected");
      }
    }
  }
}
```

このコードでは、まず lc という LocalConnection オブジェクトを作成し、その client プロパティに CustomClient1 というカスタムクラスを設定しています。このローカル接続インスタンスに対して、他の Flash Player インスタンスからメソッドが呼び出されると、Flash Player は CustomClient1 クラスの該当するメソッドを探します。

何らかの Flash Player インスタンスがこの SWF ファイルに接続し、当該ローカル接続に対して何らかのメソッドを呼び出そうとすると、そのたびに、client プロパティに設定したクラス (CustomClient1 クラス) に対してリクエストが送信されます。

```
package
{
  import flash.events.*;
  import flash.system.fscommand;
  import flash.utils.Timer;
  public class CustomClient1 extends Object
  {
    public function doMessage(value:String = ""):void
    {
      trace(value);
    }
    public function doQuit():void
    {
      trace("quitting in 5 seconds");
      this.close();
      var quitTimer:Timer = new Timer(5000, 1);
      quitTimer.addEventListener(TimerEvent.TIMER, closeHandler);
```

```
        }
        public function closeHandler(event:TimerEvent):void
        {
            fscommand("quit");
        }
    }
}
```

LocalConnection サーバーを作成するには、`LocalConnection.connect()` メソッドを呼び出して一意の接続名を指定します。指定した名前の接続が既に存在する場合は `ArgumentError` エラーが発生します。このエラーは、オブジェクトが接続済みであるために接続の試行が失敗したことを示します。

次のコードでは、`conn1` という名前で新しいソケット接続を作成します。

```
try
{
    connection.connect("conn1");
}
catch (error:ArgumentError)
{
    trace("Error!Server already exists\n");
}
```

メモ	ActionScript の以前のバージョンでは、指定した接続名が既に使用されている場合に `LocalConnection.connect()` メソッドが Boolean 値を返しました。ActionScript 3.0 では、指定した接続名が既に使用されている場合はエラーが発生します。

第 1 の SWF ファイルに対して第 2 の SWF ファイルが接続するには、送信元の LocalConnection オブジェクト内で新しい LocalConnection オブジェクトを作成してから、接続名と実行するメソッドの名前を指定して `LocalConnection.send()` メソッドを呼び出す必要があります。たとえば、前の例で作成した LocalConnection オブジェクトに接続するには次のコードを使用します。

```
sendingConnection.send("conn1", "doQuit");
```

このコードでは、既存の LocalConnection オブジェクトに接続名 `conn1` で接続し、リモート SWF ファイルの `doQuit()` メソッドを呼び出しています。リモート SWF ファイルにパラメータを送信する場合は、次のように、`send()` メソッドに対するメソッド名の指定の後に追加パラメータを指定します。

```
sendingConnection.send("conn1", "doMessage", "Hello world");
```

異なるドメインの SWF ドキュメントへの接続

特定ドメインからの接続だけを許可するには、LocalConnection クラスの allowDomain() または allowInsecureDomain() メソッドを呼び出し、当該 LocalConnection オブジェクトへのアクセスを許可するドメインのリストを指定します。

ActionScript の以前のバージョンでは、開発者が LocalConnection.allowDomain() および LocalConnection.allowInsecureDomain() をコールバックメソッドとして実装する必要があり、また、いずれも戻り値として Boolean 値を返すようにする必要がありました。ActionScript 3.0 では、LocalConnection.allowDomain() および LocalConnection.allowInsecureDomain() はいずれも内蔵メソッドとなりました。開発者はアクセスを許可するドメイン名のリストを指定して、これらのメソッドを Security.allowDomain() や Security.allowInsecureDomain() と同様に呼び出すだけで済みます。

LocalConnection.allowDomain() および LocalConnection.allowInsecureDomain() メソッドに対して指定できる特別な値として、* と localhost の 2 つがあります。アスタリスク値 (*) は、すべてのドメインにアクセスを許可することを示します。localhost というストリングは、ローカル環境にインストールされている SWF ファイルから当該 SWF ファイルへの呼び出しを許可することを示します。

Flash Player 8 から、ローカル SWF ファイルに関するセキュリティ制限が導入されました。これにより、インターネットへのアクセスを許可された SWF ファイルは、ローカルファイルシステムにはアクセスできません。localhost を指定した場合、すべてのローカル SWF ファイルは当該 SWF ファイルにアクセスできます。セキュリティ Sandbox 内から、呼び出し元コードがアクセスを認められていない SWF ファイルに対して LocalConnection.send() メソッドで接続しようとすると、securityError イベント (SecurityErrorEvent.SECURITY_ERROR) が送出されます。このエラーを回避するには、受信側の LocalConnection.allowDomain() メソッドで、呼び出し元のドメインを指定してください。

同じドメイン内の SWF ファイル間でのみ通信を行う場合は、connectionName パラメータに対して、先頭がアンダースコア (_) 以外で始まり、しかもドメイン名を含まない名前を指定します (例 : myDomain:connectionName)。また、それと同じストリングを LocalConnection.connect(connectionName) コマンドでも使用します。

異なるドメインにある SWF ファイル間での通信を実装する場合は、connectionName パラメータに対して、先頭がアンダースコア (_) で始まる名前を指定します。このアンダースコアを付けると、受信側 LocalConnection オブジェクトを含む SWF ファイルのドメイン間におけるポータビリティが高まります。考えられる 2 つの状況を次に示します。

■ connectionName のストリングがアンダースコア (_) で始まっていない場合、Flash Player によってスーパードメイン名にプレフィクスとコロンが追加されます (例 : myDomain:connectionName)。この処理には他のドメインにある同じ名前の接続との競合を回避する意味がありますが、その代わり、このスーパードメインをすべての送信側 LocalConnection オブジェクトで指定する必要があります (例 : myDomain:connectionName)。受信側 LocalConnection オブジェクトを含んだ SWF ファイルを別のドメインに移動すると、Flash Player によって追加されるプレフィクスは、スーパードメインの変更を反映して変化します (例 : anotherDomain:connectionName)。したがって、新しいスーパードメインを参照するようにすべての送信側 LocalConnection オブジェクトを手動で編集する必要が生じます。

■ connectionName のストリングがアンダースコアで始まる場合 (例 : _connectionName)、Flash Player によってストリングにプレフィクスが追加されることはありません。そのため、受信側と送信側の両方の LocalConnection オブジェクトで、connectionName に同じストリングを使用できます。すべてのドメインからの接続を受け付けることを受信側オブジェクトが LocalConnection.allowDomain() によって指定していれば、受信側 LocalConnection オブジェクトを含んだ SWF ファイルを別のドメインに移動した場合でも、送信側 LocalConnection オブジェクトに変更を加える必要はありません。

ソケット接続

ActionScript 3.0 で使用できるソケット接続には、XML ソケット接続とバイナリソケット接続の 2 種類があります。XML ソケットでリモートサーバーに接続する場合は、明示的に接続を閉じるまで維持されるサーバー接続を確立できます。これにより、サーバー接続を確立し直す処理を繰り返すことなくサーバー / クライアント間で XML データの交換を継続できます。XML ソケットサーバーを使用するもう 1 つのメリットは、ユーザーが明示的にデータを要求する必要がないことです。要求がなくてもサーバーからのデータ送信ができ、また、XML ソケットサーバーに接続済みのクライアントすべてに対してデータを送信できます。

バイナリソケット接続は、XML ソケットに似ていますが、クライアント / サーバー間で交換するデータが XML パケットである必要はないという点が異なります。データはバイナリ情報として伝送されます。このため、電子メールサーバー (POP3、SMTP、IMAP) やニュースサーバー (NNTP) などさまざまなサービスとの接続に使用できます。

Socket クラス

ActionScript 3.0 で新設された Socket クラスを使用すると、ActionScript でソケット接続を確立して生のバイナリデータを読み書きできます。これは XMLSocket クラスに似ていますが、送受信するデータの形式に制約がありません。Socket クラスは、バイナリプロトコルを使用するサーバーとの通信に役立ちます。バイナリソケット接続を使用すれば、POP3、SMTP、IMAP、NNTP などさまざまなインターネットプロトコルによる通信のコードを記述できます。したがって、Flash Player から電子メールサーバーやニュースサーバーに接続することもできます。

Flash Player からサーバーに直接接続するには、サーバーが使用するバイナリプロトコルに従う必要があります。バイト順序にビッグエンディアンを使用するサーバーと、リトルエンディアンを使用するサーバーがあります。標準の " ネットワークバイト順序 " がビッグエンディアンとされているため、インターネット上の大部分のサーバーはビッグエンディアンのバイト順序を使用しています。また、Intel x86 アーキテクチャでリトルエンディアンが採用されているため、リトルエンディアンのバイト順序が使用されることもよくあります。データを送受信する対象のサーバーに適したエンディアンバイト順序を使用してください。IDataInput および IDataOutput インターフェイスで実行されるすべての操作と、これらのインターフェイスを実装するクラス (ByteArray、Socket、URLStream) では、デフォルトでビッグエンディアン形式 (先頭が最上位バイト) を使用します。これは、Java および公式なネットワークバイト順序に適合するためです。使用するエンディアンを変更する場合は、`endian` プロパティに `Endian.BIG_ENDIAN` または `Endian.LITTLE_ENDIAN` を設定します。

ヒント

Socket クラスでは、IDataInput および IDataOutput インターフェイス (flash.utils パッケージ) で実装するすべてのメソッドを継承しています。Socket の読み書きにはそれらのメソッドを使用してください。

XMLSocket クラス

ActionScript には、サーバーとの連続的な接続を確立できる XMLSocket ビルトインクラスがあります。接続が維持されるため待ち時間の問題が解消されるので、チャットやマルチプレーヤーゲームなどのリアルタイムアプリケーションによく使用されます。従来の HTTP ベースによるチャットソリューションでは、サーバーへのポーリング処理を頻繁に実行し、HTTP リクエストを使用して新しいメッセージをダウンロードします。それに対し、XMLSocket によるチャットソリューションではサーバーとの接続を開いたまま維持するため、クライアントから要求されなくても、新着メッセージをサーバーから直ちに送信できます。

ソケット接続を確立するには、ソケット接続の要求を受け付けて SWF ファイルに応答を送るサーバーサイドアプリケーションを作成する必要があります。このようなサーバーサイドアプリケーションは、Java、Python、Perl などのプログラミング言語で作成できます。XMLSocket クラスを使用するには、サーバーコンピュータで動作するデーモンが XMLSocket クラスで使用するプロトコルを処理できる必要があります。プロトコルの説明を次の一覧に示します。

- XML メッセージは、全二重 TCP/IP ストリームソケット接続を介して送られます。
- 個々の XML メッセージは完全な XML ドキュメントであり、ゼロ (0) バイトで終了します。
- 1 つの XMLSocket 接続を使用して送受信できる XML メッセージの数に制限はありません。

> **メモ** XMLSocket クラスは、ファイアウォールをトンネリングによって自動的に通過することはできません。RTMP プロトコルとは異なり、XMLSocket には HTTP トンネリング機能が備わっていないためです。HTTP トンネリングが必要な場合は、Flash Remoting または (RTMP をサポートする) Flash Media Server の使用をお勧めします。

XMLSocket オブジェクトがサーバーに接続する方法と接続先については、次の制限があります。

- `XMLSocket.connect()` メソッドが接続できる TCP ポートの番号は、1024 以上です。この制限により、`XMLSocket` オブジェクトと通信するサーバーデーモンにも、1024 以上のポート番号を割り当てる必要があります。1024 未満のポート番号は、FTP (21)、Telnet (23)、SMTP (25)、HTTP (80)、POP3 (110) などのシステムサービスによって使用されることが多いため、セキュリティ上の理由から、XMLSocket オブジェクトではこれらのポートにアクセスできません。こうしたリソースが不適切な方法でアクセスされたり悪用されたりする可能性を小さくするために、ポート番号が制限されています。
- `XMLSocket.connect()` メソッドは、SWF ファイルが存在するのと同じドメイン内のコンピュータにしか接続できません。この制限は、ローカルディスクから再生される SWF ファイルには適用されません (この制限は、`URLLoader.load()` のセキュリティ規則と同じです)。特定のドメインからのアクセスを許可するセキュリティポリシーファイルを作成すると、SWF ファイルが存在するドメイン以外で実行されるサーバーデーモンに接続できます。

> **メモ** XMLSocket オブジェクトと通信できるようにサーバーを設定することは、場合によっては困難が伴います。リアルタイムのインタラクティブ機能を必要としないアプリケーションでは、XMLSocket クラスではなく URLLoader クラスを使用してください。

XMLSocket クラスの `XMLSocket.connect()` メソッドと `XMLSocket.send()` メソッドを使用すると、ソケット接続を介してサーバーとの間で XML を転送できます。`XMLSocket.connect()` メソッドは、Web サーバーのポートに対してソケット接続を確立します。`XMLSocket.send()` メソッドは、ソケット接続で指定されたサーバーに XML オブジェクトを送信します。

XMLSocket.connect() メソッドを呼び出すと、Flash Player によってサーバーへの TCP/IP 接続が開かれ、次のいずれかのイベントが発生するまでその接続が開いたまま維持されます。

- XMLSocket クラスの XMLSocket.close() メソッドが呼び出される。
- XMLSocket オブジェクトへの参照が 1 つも存在しなくなる。
- Flash Player が終了する。
- 接続が切れる (モデムの切断など)。

Java XML ソケットサーバーの作成および接続

次のコードは、Java で作成した単純な XMLSocket サーバーの例です。接続を受け付け、受信したメッセージをコマンドプロンプトウィンドウに表示します。デフォルトでは新しいサーバーをローカルマシンのポート番号 8080 に作成しますが、サーバーを起動する際のコマンドラインでポート番号を変更することもできます。

新しいテキストドキュメントを作成し、次のコードを入力します。

```
import java.io.*;
import java.net.*;

class SimpleServer
{
  private static SimpleServer server;
  ServerSocket socket;
  Socket incoming;
  BufferedReader readerIn;
  PrintStream printOut;

  public static void main(String[] args)
  {
    int port = 8080;

    try
    {
      port = Integer.parseInt(args[0]);
    }
    catch (ArrayIndexOutOfBoundsException e)
    {
      // 例外をキャッチし、処理を続行する
    }

    server = new SimpleServer(port);
  }

  private SimpleServer(int port)
  {
    System.out.println(">> Starting SimpleServer");
    try
```

```
    {
      socket = new ServerSocket(port);
      incoming = socket.accept();
      readerIn = new BufferedReader(new
  InputStreamReader(incoming.getInputStream()));
      printOut = new PrintStream(incoming.getOutputStream());
      printOut.println("Enter EXIT to exit.\r");
      out("Enter EXIT to exit.\r");
      boolean done = false;
      while (!done)
      {
        String str = readerIn.readLine();
        if (str == null)
        {
          done = true;
        }
        else
        {
          out("Echo: " + str + "\r");
          if(str.trim().equals("EXIT"))
          {
            done = true;
          }
        }
        incoming.close();
      }
    }
    catch (Exception e)
    {
      System.out.println(e);
    }
  }

  private void out(String str)
  {
    printOut.println(str);
    System.out.println(str);
  }
}
```

このドキュメントを "SimpleServer.java" という名前でハードディスクに保存し、Java コンパイラでコンパイルして、"SimpleServer.class" という Java クラスを作成します。

コマンドプロンプトを開き、`java SimpleServer` と入力して XMLSocket サーバーを起動します。"SimpleServer.class" ファイルはローカルコンピュータ上またはネットワーク上のどこに配置しても使用できます。Web サーバーのルートディレクトリに置く必要はありません。

ヒント	ファイルを置いた場所が Java クラスパスに含まれていないためにサーバーを起動できない場合は、`java -classpath .SimpleServer` と入力すれば起動できることがあります。

ActionScript アプリケーションから XMLSocket に接続するには、次のように XMLSocket クラスの新しいインスタンスを作成し、ホスト名とポート番号を指定して XMLSocket.connect() メソッドを呼び出します。

```
var xmlsock:XMLSocket = new XMLSocket();
xmlsock.connect("127.0.0.1", 8080);
```

呼び出し元のセキュリティサンドボックスの外にあるサーバーや 1024 未満のポート番号に対して XMLSocket.connect() で接続しようとした場合は、securityError (flash.events.SecurityErrorEvent) イベントが発生します。

サーバーからデータを受信すると、そのたびにデータイベント (flash.events.DataEvent.DATA) が送出されます。

```
xmlsock.addEventListener(DataEvent.DATA, onData);
private function onData(event:DataEvent):void
{
   trace("[" + event.type + "] " + event.data);
}
```

XMLSocket サーバーにデータを送信するには、パラメータに XML オブジェクトまたはストリングを指定して XMLSocket.send() メソッドを呼び出します。Flash Player によってパラメータが String オブジェクトに変換され、その内容に終端を示すゼロ (0) バイトを付加したデータが XMLSocket サーバーに送信されます。

```
xmlsock.send(xmlFormattedData);
```

XMLSocket.send() メソッドは、データが正常に転送されたかどうかを示す値を返しません。データの送信中にエラーが発生した場合は、IOError エラーがスローされます。

ヒント	XML ソケットサーバーに送信する個々のメッセージには、末尾に終端の改行文字 (\n) を付ける必要があります。

ローカルデータの保存

共有オブジェクト ("Flash クッキー" とも呼ばれる) は、アクセス先サイトの必要に応じてコンピュータ上に作成されるデータファイルです。共有オブジェクトは主として、ユーザーの Web ブラウズ時の使い勝手を改善するために使用されます、たとえば、よく利用する Web サイトの外観や操作性をパーソナライズする場合などです。共有オブジェクト自体がコンピュータ上にあるデータにアクセスすることはできません。さらに、ユーザーの電子メールアドレスなどの個人情報を共有オブジェクトでアクセスしたり保存したりすることも、ユーザーが自発的にそうした情報を提示しない限り、不可能です。

共有オブジェクトの新しいインスタンスを作成するには、静的メソッドの SharedObject.getLocal() または SharedObject.getRemote() を使用します。getLocal() メソッドでは、現在のクライアントだけがアクセスできるローカルの永続共有オブジェクトに対してロードを試みます。一方、getRemote() メソッドでは、何らかのサーバー (Flash Media Server など) によって複数クライアントでの共有ができるリモートの共有オブジェクトに対してロードを試みます。目的のローカル共有オブジェクトやリモート共有オブジェクトが存在しない場合、getLocal() メソッドと getRemote() メソッドは新しい SharedObject インスタンスを返します。

次のコードでは、test というローカル共有オブジェクトをロードしようとしています。この共有オブジェクトが存在しなければ、指定した名前の新しい共有オブジェクトが作成されます。

```
var so:SharedObject = SharedObject.getLocal("test");
trace("SharedObject is" + so.size + "bytes");
```

test という共有オブジェクトが見つからない場合は、サイズが 0 バイトの新しい共有オブジェクトが作成されます。該当する共有オブジェクトが既に存在する場合は、現在のサイズ (バイト単位) が返されます。

共有オブジェクトにデータを格納するには、次のようにデータオブジェクトに値を割り当てます。

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.now = new Date().time;
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

test という名前の共有オブジェクトとパラメータ now が既に存在する場合は、既存の値が上書きされます。共有オブジェクトが既に存在するかどうかは、次の例に示すように、SharedObject.size プロパティを使用して判別できます。

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
   // 共有オブジェクトが存在しない
   trace("created...");
   so.data.now = new Date().time;
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
```

このコードでは、指定した名前の共有オブジェクトインスタンスが既に存在するかどうかを、size パラメータによって判別しています。次のコードでは、実行するたびに共有オブジェクトを改めて作成します。ユーザーのハードディスクに共有オブジェクトを保存しておくには、次の例が示すように明示的に SharedObject.flush() メソッドを呼び出す必要があります。

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
   // 共有オブジェクトが存在しない
   trace("created...");
   so.data.now = new Date().time;
```

```
}
trace(so.data.now);
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

ユーザーのハードディスクに共有オブジェクトを保存するために flush() メソッドを使用する際は、次の例に示すように、Flash Player 設定マネージャ (http://www.macromedia.com/support/documentation/jp/flashplayer/help/settings_manager07.html) でユーザーがローカル記憶域の使用を明示的に無効にしていないか慎重に確認してください。

```
var so:SharedObject = SharedObject.getLocal("test");
trace("Current SharedObject size is " + so.size + " bytes.");
so.flush();
```

共有オブジェクトから値を取り出すには、共有オブジェクトの data プロパティに含まれるプロパティ名を指定します。たとえば、次のコードでは、SharedObject インスタンスが作成された時点からの経過時間 (分単位) を Flash Player に表示します。

```
var so:SharedObject = SharedObject.getLocal("test");
if (so.size == 0)
{
   // 共有オブジェクトが存在しない
   trace("created...");
   so.data.now = new Date().time;
}
var ageMS:Number = new Date().time - so.data.now;
trace("SharedObject was created " + Number(ageMS / 1000 / 60).toPrecision(2) + "
   minutes ago");
trace("SharedObject is " + so.size + " bytes");
so.flush();
```

このコードは初めて実行されると、test という名前でサイズが 0 バイトの新しい SharedObject インスタンスを作成します。初期サイズは 0 バイトで if ステートメントの条件が true になるため、now という新しいプロパティが共有オブジェクトに追加されます。共有オブジェクト作成時からの経過時間は、現在時刻から now プロパティの値を減算して求めています。以後、このコードが再度実行される際には共有オブジェクトのサイズが 0 より大きくなっているため、共有オブジェクト作成時からの経過時間をトレース出力します。

共有オブジェクトの内容の表示

共有オブジェクトに保存する値は、data プロパティ内に格納されます。ループで共有オブジェクトインスタンス内の個々の値にアクセスするには、次のように for..in ループを使用します。

```
var so:SharedObject = SharedObject.getLocal("test");
so.data.hello = "world";
so.data.foo = "bar";
so.data.timezone = new Date().timezoneOffset;
for (var i:String in so.data)
{
```

```
  trace(i + ":\t" + so.data[i]);
}
```

セキュアな SharedObject の作成

getLocal() または getRemote() でローカルまたはリモートの SharedObject を作成する際にオプションの secure というパラメータを指定すると、その共有オブジェクトについて、HTTPS 経由で配信される SWF ファイルのみにアクセスを制限するかどうかを定義できます。HTTPS 経由で配信される SWF ファイルにおいて、このパラメータに true を指定すると、セキュアな共有オブジェクトを新しく作成、または既存のセキュアな共有オブジェクトに対する参照を取得できます。このセキュアな共有オブジェクトは、HTTPS 経由で配信される SWF ファイルで、secure パラメータに true を指定して SharedObject.getLocal() を呼び出した場合に限り読み書きできます。HTTPS 経由で配信される SWF ファイルにおいて、このパラメータに false を指定すると、共有オブジェクトを新しく作成、または既存の共有オブジェクトに対する参照を取得でき、

この場合の共有オブジェクトは HTTPS 以外の接続を経由して配信される SWF ファイルからも読み書きできます。HTTPS 以外の接続を経由して配信される SWF ファイルにおいて、このパラメータに true を指定しようとすると、新しい共有オブジェクトの作成 (または既存のセキュアな共有オブジェクトへのアクセス) に失敗してエラーがスローされ、共有オブジェクトには null が設定されます。次のコードを、HTTPS 以外の接続を経由して配信される SWF ファイルで実行しようとすると、SharedObject.getLocal() メソッドでエラーがスローされます。

```
try
{
  var so:SharedObject = SharedObject.getLocal("contactManager", null, true);
}
catch (error:Error)
{
  trace("Unable to create SharedObject.");
}
```

作成した共有オブジェクトには、このパラメータの値がいずれであっても関係なく、当該ドメインで使用可能な合計ディスク容量の枠が適用されます。

ファイルのアップロードおよびダウンロードに関する操作

FileReference クラスを使用すると、クライアントコンピュータとサーバーとの間でファイルのアップロードおよびダウンロード機能を実現できます。ユーザーはダイアログボックス (Windows OS の [開く] ダイアログボックスなど) によって、アップロードするファイルやダウンロード先の場所を指定するよう要求されます。

ActionScript で作成した個々の FileReference オブジェクトは、ユーザーのハードディスク上にある同じファイルを参照します。このオブジェクトには、ファイルのサイズ、種類、名前、作成日、および変更日の情報を格納するプロパティがあります。

メモ

creator プロパティは Macintosh でのみサポートされており、その他すべてのプラットフォームでは null を返します。

FileReference クラスのインスタンスを作成する方法は 2 種類あります。1 つは、次のコードのように new 演算子を使用する方法です。

```
import flash.net.FileReference;
var myFileReference:FileReference = new FileReference();
```

もう 1 つは、FileReferenceList.browse() メソッドを呼び出す方法です。このメソッドは、ユーザーのシステム上でダイアログボックスを開き、アップロードするファイルの指定 (場合によっては複数) をユーザーに要求して、正常にファイルが指定されると FileReference オブジェクトの配列を作成します。個々の FileReference オブジェクトは、ユーザーがダイアログボックスで指定した 1 つのファイルを表します。FileReference オブジェクトに含まれる各種の FileReference プロパティ (name、size、modificationDate など) には、次のいずれかが行われるまでは何もデータが格納されていません。

- FileReference.browse() メソッドまたは FileReferenceList.browse() メソッドが呼び出され、ユーザーによってファイルピッカー上でファイルが選択されます。
- FileReference.download() メソッドが呼び出され、ユーザーによってファイルピッカー上でファイルが選択されます。

ダウンロードを実行する場合、FileReference.name プロパティだけはダウンロードの完了より前に設定されます。ダウンロードが完了すると、すべてのプロパティの値が設定されます。

FileReference.browse()、FileReferenceList.browse()、FileReference.download() いずれかのメソッドに対する呼び出しの実行中、ほとんどの Player では SWF の再生が続けられます。

FileReference クラス

FileReference クラスを使用すると、ユーザーのコンピュータとサーバーとの間でファイルのアップロードおよびダウンロード機能を実現できます。ユーザーに対してアップロードするファイルやダウンロード先の場所の指定を求めるために、OS のダイアログボックスを表示します。個々の FileReference オブジェクトはユーザーのディスク上にある 1 つのファイルを表し、ファイルのサイズ、タイプ、名前、作成日、変更日、クリエータに関する情報をプロパティとして保持します。

FileReference インスタンスを作成するには、次の 2 つの方法があります。

- 次のように new 演算子で FileReference コンストラクタを使用する方法。
  `var myFileReference:FileReference = new FileReference();`
- `FileReferenceList.browse()` を呼び出す方法 (このメソッドによって FileReference オブジェクトの配列が作成される)。

アップロード処理およびダウンロード処理では、当該 SWF ファイル自体が属するドメイン内のファイルと、クロスドメインポリシーファイルで指定されたドメイン内のファイルに対してのみアクセスできます。アップロードまたはダウンロードを開始する SWF ファイルがファイルサーバーと同じドメインに属していない場合は、ファイルサーバー上にポリシーファイルを配置する必要があります。

メモ	`browse()` または `download()` のアクションを複数並行して実行することはできません。これは、一度に 1 つしかダイアログボックスを表示できないためです。

ファイルのアップロードを処理するサーバースクリプトは、次の要素を含んだ HTTP `POST` リクエストを受け付ける必要があります。

- `Content-Type`:設定されている値は `multipart/form-data`。
- `Content-Disposition` : `name` 属性に `"Filedata"`、`filename` 属性に元のファイルの名前。`FileReference.upload()` メソッドの `uploadDataFieldName` パラメータを指定すると、`name` 属性に独自の値を指定できます。
- ファイルの内容を表すバイナリデータ。

HTTP `POST` リクエストの例を次に示します。

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache
```

```
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Filedata"; filename="sushi.jpg"
Content-Type: application/octet-stream

テストファイル
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
(actual file data,,,)
```

次に示すサンプルの HTTP POST 要求は、api_sig、api_key、および auth_token という 3 つの POST 変数を送信します。使用するカスタムアップロードデータフィールド名の値は、"photo" です。

```
POST /handler.asp HTTP/1.1
Accept: text/*
Content-Type: multipart/form-data;
boundary=----------Ij5ae0ae0KM7GI3KM7ei4cH2ei4gL6
User-Agent: Shockwave Flash
Host: www.mydomain.com
Content-Length: 421
Connection: Keep-Alive
Cache-Control: no-cache

------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Filename"

sushi.jpg
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_sig"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="api_key"

XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="auth_token"

XXXXXXXXXXXXXXXXXXXXXXX
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="photo"; filename="sushi.jpg"
Content-Type: application/octet-stream
```

```
(actual file data,,,)
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7
Content-Disposition: form-data; name="Upload"

Submit Query
------------Ij5GI3GI3ei4GI3ei4KM7GI3KM7KM7--
```

サーバーに対するファイルのアップロード

サーバーにファイルをアップロードするには、まず browse() メソッドを呼び出して、ユーザーにファイルの選択を求めます。次に、FileReference.upload() メソッドが呼び出されたときに、選択されたファイルがサーバーへ転送されます。FileReferenceList.browse() メソッドで、ユーザーにより複数のファイルが選択された場合は、それらのファイルを表す FileReferenceList.fileList という配列が作成されます。その後、各ファイルを個別に FileReference.upload() メソッドでアップロードします。

メモ	FileReference.browse() メソッドを使用する場合は、ファイルを 1 つだけアップロードできます。複数のファイルをアップロードできるようにするには、FileReferenceList.browse() メソッドを使用する必要があります。

デフォルトでは、ユーザーは OS のファイルピッカーダイアログボックスを使用して、ローカルコンピュータ上にある任意のタイプのファイルを選択できます。ただし、次のように FileFilter クラスを使用してファイルタイプのフィルタを作成し、ファイルフィルタインスタンスの配列を browse() メソッドで指定すれば、選択できるファイルを限定することもできます。

```
var imageTypes:FileFilter = new FileFilter("Images (*.jpg, *.jpeg, *.gif,
  *.png)", "*.jpg; *.jpeg; *.gif; *.png");
var textTypes:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt;
  *.rtf");
var allTypes:Array = new Array(imageTypes, textTypes);
var fileRef:FileReference = new FileReference();
fileRef.browse(allTypes);
```

ユーザーが OS のファイルピッカーでファイルを選択し、[開く] ボタンをクリックすると、Event.SELECT イベントが送出されます。FileReference.browse() メソッドを使用してファイルをアップロードする場合は、次のコードで Web サーバーにファイルを送信する必要があります。

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
try
{
  var success:Boolean = fileRef.browse();
}
catch (error:Error)
{
  trace("Unable to browse for files.");
}
function selectHandler(event:Event):void
```

```
{
  var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
  fileUploadScript.cfm")
  try
  {
    fileRef.upload(request);
  }
  catch (error:Error)
  {
    trace("Unable to upload file.");
  }
}
function completeHandler(event:Event):void
{
  trace("uploaded");
}
```

ヒント FileReference.upload() メソッドでは、URLRequest.method および URLRequest.data プロパティを使用して、POST または GET メソッドで変数のデータをサーバーに送信することもできます。

FileReference.upload() メソッドを使用してファイルをアップロードしようとする際には、次のイベントが送出されることがあります。

- Event.OPEN: アップロード処理が開始するときに送出されます。
- ProgressEvent.PROGRESS: ファイルのアップロード処理中に定期的に送出されます。
- Event.COMPLETE: ファイルのアップロード処理が正常に完了したときに送出されます。
- SecurityErrorEvent.SECURITY_ERROR: セキュリティ侵害が原因でアップロードが失敗した場合に送出されます。
- HTTPStatusEvent.HTTP_STATUS: HTTP エラーが原因でアップロードが失敗した場合に送出されます。
- IOErrorEvent.IO_ERROR: 次のいずれかの原因でアップロードが失敗した場合に送出されます。
  - Flash Player によるファイルの読み込み中、書き込み中、または転送中に入出力エラーが発生した場合。
  - SWF ファイルが、認証 (例 : ユーザー名とパスワード) を必要とするサーバーにファイルをアップロードしようとした場合。Flash Player には、アップロード時にユーザーがパスワードを入力する手段が用意されていません。

- `url` パラメータに無効なプロトコルが指定されている場合。`FileReference.upload()` メソッドでは、HTTP または HTTPS を使用する必要があります。

> **ヒント** 認証を必要とするサーバーに対する Flash Player のサポートは完全ではありません。認証用のユーザー名とパスワードを入力するダイアログボックスを表示できるのは、ブラウザプラグインまたは ActiveX コントロールを使用してブラウザ上で実行される SWF ファイルにおいて、ダウンロードを実行する場合のみです。プラグインまたは ActiveX コントロールを使用する場合のアップロードと、スタンドアローン Player または外部 Player を使用する場合のアップロードおよびダウンロードでは、ファイル転送に失敗します。

Flash Player によるファイルのアップロードを受け付けるスクリプトを ColdFusion で作成する場合は、次のようなコードを使用します。

```
<cffile action="upload" filefield="Filedata" destination="#ExpandPath('./')#"
  nameconflict="OVERWRITE" />
```

この ColdFusion コードでは、Flash Player から送信されたファイルをアップロードし、ColdFusion テンプレートと同じディレクトリに保存します。同じ名前のファイルが既に存在する場合は上書きされます。このコードではファイルのアップロードに必要な最低限の処理しか行っていないため、実際の運用環境ではこのスクリプトを使用しないでください。データ検証のコードを付け加えて、特定のファイルタイプ (イメージなど) についてのみアップロードを許可し、サーバーサイドスクリプトなど危険が生じる可能性のあるファイルを受け付けないようにすることをお勧めします。

次のコードは PHP によるファイルのアップロード処理の例で、データ検証も実行します。このスクリプトでは、アップロード先ディレクトリにアップロードするファイルの数を 10 個まで、ファイルサイズを 200 KB までに制限し、しかも、JPEG、GIF、PNG のいずれかのファイルだけをアップロードしてファイルシステムに保存できるようにしています。

```
<?php
$MAXIMUM_FILESIZE = 1024 * 200; // 200KB
$MAXIMUM_FILE_COUNT = 10; // サーバー上のファイル数を 10 個までに制限する
echo exif_imagetype($_FILES['Filedata']);
if ($_FILES['Filedata']['size'] <= $MAXIMUM_FILESIZE)
{
  move_uploaded_file($_FILES['Filedata']['tmp_name'], "./temporary/
  ".$_FILES['Filedata']['name']);
  $type = exif_imagetype("./temporary/".$_FILES['Filedata']['name']);
  if ($type == 1 || $type == 2 || $type == 3)
  {
    rename("./temporary/".$_FILES['Filedata']['name'], "./images/
  ".$_FILES['Filedata']['name']);
  }
  else
  {
    unlink("./temporary/".$_FILES['Filedata']['name']);
  }
}
$directory = opendir('./images/');
```

```
$files = array();
while ($file = readdir($directory))
{
  array_push($files, array('./images/'.$file, filectime('./images/'.$file)));
}
usort($files, sorter);
if (count($files) > $MAXIMUM_FILE_COUNT)
{
  $files_to_delete = array_splice($files, 0, count($files) -
  $MAXIMUM_FILE_COUNT);
  for ($i = 0; $i < count($files_to_delete); $i++)
  {
    unlink($files_to_delete[$i][0]);
  }
}
print_r($files);
closedir($directory);

function sorter($a, $b)
{
  if ($a[1] == $b[1])
  {
    return 0;
  }
  else
  {
    return ($a[1] < $b[1]) ? -1 : 1;
  }
}
?>
```

このアップロードスクリプトには、POST または GET リクエストメソッドを使用して追加の変数を渡すこともできます。追加の POST 変数をアップロードスクリプトに送信するには、次のようなコードを使用します。

```
var fileRef:FileReference = new FileReference();
fileRef.addEventListener(Event.SELECT, selectHandler);
fileRef.addEventListener(Event.COMPLETE, completeHandler);
fileRef.browse();
function selectHandler(event:Event):void
{
  var params:URLVariables = new URLVariables();
  params.date = new Date();
  params.ssid = "94103-1394-2345";
  var request:URLRequest = new URLRequest("http://www.yourdomain.com/
  FileReferenceUpload/fileupload.cfm");
  request.method = URLRequestMethod.POST;
  request.data = params;
  fileRef.upload(request, "Custom1");
```

```
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

この例では、新しい URLVariables オブジェクトを作成してリモートのサーバーサイドスクリプトに送信しています。ActionScript の以前のバージョンでは、変数をサーバーアップロードスクリプトに送信するには値をクエリ文字列に含めて渡す必要がありました。ActionScript 3.0 では、URLRequest オブジェクトを使用して POST または GET メソッドで変数をリモートスクリプトに送信します。これにより、データのセット数が多い場合でも容易に、簡潔に送信処理を記述できるようになりました。変数の受け渡しに使用する GET または POST リクエストメソッドを指定するには、URLRequest.method プロパティに URLRequestMethod.GET (GET メソッドの場合) または URLRequestMethod.POST (POST メソッドの場合) を設定します。

また、ActionScript 3.0 では、upload() メソッドの第 2 パラメータを指定することで、デフォルトの Filedata アップロードファイルフィールド名を変更できます。前のコード例では、この操作の例としてデフォルト値の Filedata を Custom1 に置き換えています。

Flash Player のデフォルトではテストアップロードが試行されませんが、upload() メソッドの第 3 パラメータに true を指定するとこの動作を変更できます。テストアップロードの目的は、実際のファイルアップロードを正常に実行できるかどうか試すことと、(認証が必要な場合に) 認証を通過できるかどうか確認することです。

テストアップロードは、現在のところ Windows 版の各種 Flash Player でのみ実行されます。

サーバーからのファイルのダウンロード

FileReference.download() メソッドを使用すると、サーバー上のファイルをユーザーがダウンロードできます。このメソッドには、request および defaultFileName の 2 つのパラメータがあります。第 1 パラメータには、ダウンロードするファイルの URL を表す URLRequest オブジェクトを指定します。第 2 パラメータはオプションで、ファイルダウンロードのダイアログボックスに表示するデフォルトのファイル名を指定します。第 2 パラメータの defaultFileName を省略すると、指定した URL に含まれているファイル名が使用されます。

次のコードでは、SWF ドキュメントと同じディレクトリから "index.xml" というファイルをダウンロードします。

```
var request:URLRequest = new URLRequest("index.xml");
var fileRef:FileReference = new FileReference();
fileRef.download(request);
```

デフォルトの名前を "index.xml" ではなく "currentnews.xml" に設定するには、次のように defaultFileName パラメータを指定します。

```
var request:URLRequest = new URLRequest("index.xml");
var fileToDownload:FileReference = new FileReference();
fileToDownload.download(request, "currentnews.xml");
```

サーバーにあるファイルに意味のわかりにくい名前が付いている場合や、ファイル名がサーバーで自動生成される場合、ファイル名の変更は非常に役立ちます。また、ファイルを直接にダウンロードするのではなくサーバーサイドスクリプトによってダウンロードする場合も、defaultFileName パラメータを明示的に指定することをお勧めします。たとえば、指定した URL 変数に基づいて特定のファイルをダウンロードするようなサーバーサイドスクリプトに対しては、defaultFileName パラメータを指定する必要があります。指定しない場合、ダウンロードするファイルのデフォルトの名前はサーバーサイドスクリプトの名前と同じになります。

データ解析用サーバースクリプトを表す URL にパラメータを追加すると、download() メソッドを使用してサーバーにデータを送信できます。次の ActionScript 3.0 コードでは、ColdFusion スクリプトに渡すパラメータに基づいてドキュメントをダウンロードします。

```
package
{
    import flash.display.Sprite;
    import flash.net.FileReference;
    import flash.net.URLRequest;
    import flash.net.URLRequestMethod;
    import flash.net.URLVariables;

    public class DownloadFileExample extends Sprite
    {
        private var fileToDownload:FileReference;
        public function DownloadFileExample()
        {
            var request:URLRequest = new URLRequest();
            request.url = "http://www.[yourdomain].com/downloadfile.cfm";
            request.method = URLRequestMethod.GET;
            request.data = new URLVariables("id=2");
            fileToDownload = new FileReference();
            try
            {
                fileToDownload.download(request, "file2.txt");
            }
            catch (error:Error)
            {
                trace("Unable to download file.");
            }
        }
    }
}
```

次のコードは "download.cfm" という ColdFusion スクリプトの例です。URL 変数の値に応じて、サーバー上にある 2 つのファイルのうちいずれかをダウンロードします。

```
<cfparam name="URL.id" default="1" />
<cfswitch expression="#URL.id#">
    <cfcase value="2">
        <cfcontent type="text/plain" file="#ExpandPath('two.txt')#" deletefile="No"
    />
    </cfcase>
    <cfdefaultcase>
        <cfcontent type="text/plain" file="#ExpandPath('one.txt')#" deletefile="No"
    />
    </cfdefaultcase>
</cfswitch>
```

FileReferenceList クラス

FileReferenceList クラスを使用すると、サーバーサイドスクリプトにアップロードするファイルをユーザーが指定できます。ファイルのアップロードについては、ユーザーが指定した個々のファイルに対して FileReference.upload() メソッドを呼び出すことにより処理する必要があります。

次のコードでは、2 つの FileFilter オブジェクト (imageFilter および textFilter) を作成し、それらを配列として FileReferenceList.browse() メソッドに渡しています。これにより、OS のファイルダイアログボックスにはファイルタイプを指定するフィルタが 2 つ表示されます。

```
var imageFilter:FileFilter = new FileFilter("Image Files (*.jpg, *.jpeg, *.gif,
    *.png)", "*.jpg; *.jpeg; *.gif; *.png");
var textFilter:FileFilter = new FileFilter("Text Files (*.txt, *.rtf)", "*.txt;
    *.rtf");
var fileRefList:FileReferenceList = new FileReferenceList();
try
{
    var success:Boolean = fileRefList.browse(new Array(imageFilter, textFilter));
}
catch (error:Error)
{
    trace("Unable to browse for files.");
}
```

FileReferenceList クラスを使用して複数のファイルを選択およびアップロードできるようにする場合も、ファイルの選択は FileReference.browse() と同様に行いますが、複数のファイルを選択できる点が異なります。複数のファイルをアップロードする際は、次のように、個々のファイルについて FileReference.upload() を呼び出す必要があります。

```
var fileRefList:FileReferenceList = new FileReferenceList();
fileRefList.addEventListener(Event.SELECT, selectHandler);
fileRefList.browse();
function selectHandler(event:Event):void
```

```
{
    var request:URLRequest = new URLRequest("http://www.[yourdomain].com/
    fileUploadScript.cfm");
    var file:FileReference;
    var files:FileReferenceList = FileReferenceList(event.target);
    var selectedFileArray:Array = files.fileList;
    for (var i:uint = 0; i < selectedFileArray.length; i++)
    {
        file = FileReference(selectedFileArray[i]);
        file.addEventListener(Event.COMPLETE, completeHandler);
        try
        {
            file.upload(request);
        }
        catch (error:Error)
        {
            trace("Unable to upload files.");
        }
    }
}
function completeHandler(event:Event):void
{
    trace("uploaded");
}
```

配列内の各 FileReference オブジェクトに対して `Event.COMPLETE` イベントのリスナーを登録しているため、各ファイルのアップロードが完了するたびに、Flash Player によって `completeHandler()` メソッドが呼び出されます。

例 : Telnet クライアントの構築

この Telnet の例では、Socket クラスを使用してリモートサーバーに接続し、データを伝送する手法を示します。この例が示す手法は、次のとおりです。

- Socket クラスを使用したカスタム Telnet クライアントの作成
- ByteArray オブジェクトを使用したリモートサーバーへのテキストの送信
- リモートサーバーから受信したデータの処理

Telnet アプリケーションのファイルは、Samples/Telnet フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
TelnetSocket.mxml	MXML ユーザーインターフェイスからなるメインアプリケーションファイル。
com/example/programmingas3/Telnet/Telnet.as	リモートサーバーへの接続、データの送受信と表示など、アプリケーションに Telnet クライアント機能を提供します。

Telnet ソケットアプリケーションの概要

メイン TelnetSocket.mxml ファイルは、アプリケーション全体で使用するユーザーインターフェイス (UI) の作成に責任を負います。

このファイルは、UI のほかに、ユーザーを指定されたサーバーに接続するために、`login()` と `sendCommand()` の 2 つのメソッドも定義しています。

次のコードは、メインアプリケーションファイル内の ActionScript を示しています。

```
import com.example.programmingas3.socket.Telnet;
private var telnetClient:Telnet;
private function connect():void
{
  telnetClient = new Telnet(serverName.text, int(portNumber.text), output);
  console.title = "Connecting to " + serverName.text + ":" + portNumber.text;
  console.enabled = true;
}
private function sendCommand():void
{
  var ba:ByteArray = new ByteArray();
  ba.writeMultiByte(command.text + "\n", "UTF-8");
  telnetClient.writeBytesToSocket(ba);
  command.text = "";
}
```

コードの 1 行目は、Telnet クラスをカスタム com.example.programmingas.socket パッケージからインポートします。コードの 2 行目は、Telnet クラスのインスタンス、telnetClient を宣言します。このインスタンスは、後で connect() メソッドによって初期化されます。次に、connect() メソッドが宣言され、前に宣言された telnetClient 変数を初期化します。このメソッドは、ユーザーが指定した Telnet サーバーの名前とポートを引き渡すほか、ソケットサーバーからのテキスト応答の表示に使用される表示リスト上の TextArea コンポーネントへの参照を引き渡します。connect() メソッドの最後の 2 行は、Panel の title プロパティを設定し、Panel コンポーネントを有効にします。このコンポーネントを使用すると、リモートサーバーへデータを送信できます。メインアプリケーションファイル内の最後のメソッド、sendCommand() は、ユーザーのコマンドを ByteArray オブジェクトとしてリモートサーバーへ送信するために使用されます。

Telnet クラスの概要

Telnet クラスは、リモート Telnet サーバーへの接続と、データの送受信に責任を負います。

Telnet クラスは、次のプライベート変数を宣言します。

```
private var serverURL:String;
private var portNumber:int;
private var socket:Socket;
private var ta:TextArea;
private var state:int = 0;
```

最初の変数、serverURL は、ユーザーが指定した接続先サーバーアドレスを格納します。

2 番目の変数、portNumber は、Telnet サーバーが現在稼働しているポートの番号です。デフォルトでは、Telnet サービスはポート 23 上で稼働します。

3 番目の変数、socket は、serverURL 変数と portNumber 変数によって定義されたサーバーへの接続を試みる Socket インスタンスです。

4 番目の変数、ta は、ステージ上の TextArea コンポーネントインスタンスへの参照です。このコンポーネントは、リモート Telnet サーバーからの応答やエラーメッセージを表示するために使用されます。

最後の変数、state は、その Telnet クライアントがどのオプションをサポートするかを決定するために使用される数値です。

前に述べたように、Telnet クラスのコンストラクタ関数は、メインアプリケーションファイル内の connect() メソッドによって呼び出されます。

Telnet コンストラクタは server、port、output の 3 つのパラメータを受け取ります。server パラメータと port パラメータは、Telnet サーバーが稼働しているサーバーの名前とポート番号を指定します。最後のパラメータ、output は、サーバーの出力がユーザーに対して表示されるステージ上の TextArea コンポーネントインスタンスへの参照です。

```
public function Telnet(server:String, port:int, output:TextArea)
{
    serverURL = server;
    portNumber = port;
    ta = output;
    socket = new Socket();
    socket.addEventListener(Event.CONNECT, connectHandler);
    socket.addEventListener(Event.CLOSE, closeHandler);
    socket.addEventListener(ErrorEvent.ERROR, errorHandler);
    socket.addEventListener(IOErrorEvent.IO_ERROR, ioErrorHandler);
    socket.addEventListener(ProgressEvent.SOCKET_DATA, dataHandler);
    Security.loadPolicyFile("http://" + serverURL + "/crossdomain.xml");
    try
    {
        msg("Trying to connect to " + serverURL + ":" + portNumber + "\n");
        socket.connect(serverURL, portNumber);
    }
    catch (error:Error)
    {
        msg(error.message + "\n");
        socket.close();
    }
}
```

ソケットへのデータの書き込み

Socket 接続にデータを書き込むには、Socket クラス内のいずれかの書き込みメソッド (writeBoolean()、writeByte()、writeBytes()、writeDouble() など) を呼び出した後、flush() メソッドで出力バッファ内のデータをフラッシュします。Telnet サーバー内では、データは writeBytes() メソッドを使用してソケット接続へ書き込まれます。このメソッドは、パラメータとしてバイト配列を受け取り、それを出力バッファへ送信します。writeBytesToSocket() メソッドは、次のとおりです。

```
public function writeBytesToSocket(ba:ByteArray):void
{
    socket.writeBytes(ba);
    socket.flush();
}
```

このメソッドは、メインアプリケーションファイルの sendCommand() メソッドによって呼び出されます。

ソケットサーバーからのメッセージの表示

ソケットサーバーからメッセージを受け取るか、イベントが発生すると、カスタム msg() メソッドが呼び出されます。このメソッドは、ステージ上の TextArea にストリングを付加し、カスタム setScroll() メソッドを呼び出します。これにより、TextArea コンポーネントは最下部までスクロールします。msg() メソッドは、次のとおりです。

```
private function msg(value:String):void
{
    ta.text += value;
    setScroll();
}
```

TextArea コンポーネントの内容を自動的にスクロールしなかった場合、サーバーからの最新の応答を参照するには、テキスト領域にあるスクロールバーを手動でドラッグする必要があります。

TextArea コンポーネントのスクロール

setScroll() メソッドには、TextArea コンポーネントの内容を垂直方向にスクロールする 1 行だけの ActionScript が入っており、これにより、ユーザーは返されたテキストの最後の行を見ることができます。次のコードは、setScroll() メソッドを示しています。

```
public function setScroll():void
{
    ta.verticalScrollPosition = ta.maxVerticalScrollPosition;
}
```

このメソッドは、現在表示されている文字の一番上の行の行番号を示す verticalScrollPosition プロパティを設定し、そのプロパティを maxVerticalScrollPosition プロパティの値に設定します。

例 : ファイルのアップロードとダウンロード

FileIO は、Flash Player でファイルのダウンロードとアップロードを行う手法を示した例です。それらの手法は、次のとおりです。

- ユーザーのコンピュータへのファイルのダウンロード
- ユーザーのコンピュータからサーバーへのファイルのアップロード
- 進行中のダウンロードのキャンセル
- 進行中のアップロードのキャンセル

FileIO アプリケーションのファイルは、Samples/FileIO フォルダにあります。このアプリケーションは、次のファイルで構成されています。

ファイル	説明
FileIO.mxml	MXML ユーザーインターフェイスからなるメインアプリケーションファイル。
com/example/programmingas3/fileio/FileDownload.as	サーバーからファイルをダウンロードするためのメソッドを含んでいるクラス。
com/example/programmingas3/fileio/FileUpload.as	ファイルをサーバーへアップロードするためのメソッドを含んでいるクラス。

FileIO アプリケーションの概要

File IO アプリケーションには、ユーザーが Flash Player を使用してファイルをアップロードまたはダウンロードできるユーザーインターフェイスが含まれています。このアプリケーションは最初に、FileUpload と FileDownload という 1 対のカスタムコンポーネントを定義します。これらのコンポーネントは、com.example.programmingas3.fileio パッケージに入っています。それぞれのカスタムコンポーネントが `contentComplete` イベントを送出した後、コンポーネントの `init()` メソッドが呼び出され、ProgressBar および Button コンポーネントインスタンスへの参照を引き渡します。これらを使用して、ユーザーはファイルのアップロードまたはダウンロードの進行状況を知ることができ、進行中のファイル転送をキャンセルすることもできます。

FileIO.mxml ファイル内の関連するコードは、次のとおりです。

```
<example:FileUpload id="fileUpload"
  creationComplete="fileUpload.init(uploadProgress, cancelUpload);" />
<example:FileDownload id="fileDownload"
  creationComplete="fileDownload.init(downloadProgress, cancelDownload);" />
```

次のコードは、プログレスバーと 2 つのボタンが入っている [ファイルのアップロード] パネルを示しています。最初のボタン、`startUpload` は `FileUpload.startUpload()` メソッドを呼び出し、このメソッドは `FileReference.browse()` メソッドを呼び出します。次のコードは、[ファイルのアップロード] パネルのコードです。

```
<mx:Panel title="Upload File" paddingTop="10" paddingBottom="10"
  paddingLeft="10" paddingRight="10">
  <mx:ProgressBar id="uploadProgress" label="" mode="manual" />
  <mx:ControlBar horizontalAlign="right">
    <mx:Button id="startUpload" label="Upload..."
  click="fileUpload.startUpload();" />
    <mx:Button id="cancelUpload" label="Cancel"
  click="fileUpload.cancelUpload();" enabled="false" />
  </mx:ControlBar>
</mx:Panel>
```

このコードは、ProgressBar コンポーネントインスタンスと、2 つの Button コンポーネントボタンインスタンスをステージ上に配置します。ユーザーが [アップロード] ボタンをクリックすると (startUpload)、オペレーティングシステムのダイアログボックスが起動し、リモートサーバーへアップロードするファイルをユーザーが選択できるようになります。もう一方のボタン、cancelUpload は、デフォルトでは無効にされていますが、ユーザーがファイルのアップロードを開始すると有効になり、ユーザーが任意の時点でファイル転送を中止できるようにします。

[ファイルのダウンロード] パネルのコードは、次のとおりです。

```
<mx:Panel title="Download File" paddingTop="10" paddingBottom="10"
   paddingLeft="10" paddingRight="10">
   <mx:ProgressBar id="downloadProgress" label="" mode="manual" />
   <mx:ControlBar horizontalAlign="right">
      <mx:Button id="startDownload" label="Download..."
   click="fileDownload.startDownload();" />
      <mx:Button id="cancelDownload" label="Cancel"
   click="fileDownload.cancelDownload();" enabled="false" />
   </mx:ControlBar>
</mx:Panel>
```

このコードはファイルアップロードコードに非常によく似ています。ユーザーが [ダウンロード] ボタンをクリックすると (startDownload)、FileDownload.startDownload() メソッドが呼び出され、FileDownload.DOWNLOAD_URL 変数で指定されたファイルのダウンロードが開始されます。ファイルのダウンロードが進行するにつれて、プログレスバーが更新され、ファイルのどれくらいのパーセンテージがダウンロードされたかが示されます。ユーザーは、cancelDownload ボタンをクリックすることにより、任意の時点でダウンロードをキャンセルできます。このボタンは、進行中のファイルダウンロードを即時に停止します。

リモートサーバーからのファイルのダウンロード

リモートサーバーからのファイルのダウンロードは、flash.net.FileReference クラスとカスタム com.example.programmingas3.fileio.FileDownload クラスによって処理されます。ユーザーが [ダウンロード] ボタンをクリックすると、Flash Player は FileDownload クラスの DOWNLOAD_URL 変数で指定されたファイルのダウンロードを開始します。

FileDownload クラスは、次のコードが示すように、冒頭で com.example.programmingas3.fileio パッケージ内の 4 つの変数を定義します。

```
/**
 * ユーザーのコンピュータへダウンロードするファイルの URL をハードコーディングする
 */
private const DOWNLOAD_URL:String = "http://www.yourdomain.com/
file_to_download.zip";

/**
 * ファイルのダウンロードを処理する FileReference インスタンスを作成する
 */
private var fr:FileReference;
```

```
/**
 * ダウンロードの ProgressBar コンポーネントへの参照を定義する
 */
private var pb:ProgressBar;

/**
 * 現在進行中のダウンロードを即時に停止する [ キャンセル ] ボタンへの
 * 参照を定義する
 */
private var btn:Button;
```

最初の変数、DOWNLOAD_URL には、ユーザーがメインアプリケーションファイル内の [ダウンロード] ボタンをクリックしたときに、ユーザーのコンピュータへダウンロードされるファイルへのパスが入っています。

2 番目の変数、fr は FileReference オブジェクトで、これは FileDownload.init() メソッド内で初期化され、ユーザーのコンピュータへのリモートファイルのダウンロードを処理します。

最後の 2 つの変数、pb と btn には、ステージ上の ProgressBar および Button コンポーネントインスタンスへの参照が入っており、これらのインスタンスは、FileDownload.init() メソッドによって初期化されます。

FileDownload コンポーネントの初期化

FileDownload コンポーネントは、FileDownload クラス内の init() メソッドを呼び出すことによって初期化されます。このメソッドは pb と btn の 2 つのパラメータを受け取ります。これらはそれぞれ、ProgressBar コンポーネントと Button コンポーネントのインスタンスです。

init() メソッドのコードは次のとおりです。

```
/**
 * コンポーネントへの参照を設定し、OPEN、
 * PROGRESS、および COMPLETE イベントのリスナーを追加する
 */
public function init(pb:ProgressBar, btn:Button):void
{
    // プログレスバーとキャンセルボタンへの参照を設定する。これらは、
    // 呼び出し元のスクリプトから渡される。
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

ファイルのダウンロードの開始

ユーザーがステージ上の [ダウンロード] ボタンコンポーネントをクリックすると、startDownload() メソッドがファイルダウンロードプロセスを開始します。次のコードは、startDownload() メソッドを示しています。

```
/**
 * DOWNLOAD_URL 定数で指定されたファイルのダウンロードを開始する
 */
public function startDownload():void
{
    var request:URLRequest = new URLRequest();
    request.url = DOWNLOAD_URL;
    fr.download(request);
}
```

最初に、startDownload() メソッドは新しい URLRequest オブジェクトを作成し、ターゲットの URL を DOWNLOAD_URL 変数で指定された値に設定します。次に、FileReference.download() メソッドが呼び出され、新規に作成された URLRequest オブジェクトがパラメータとして引き渡されます。これにより、オペレーティングシステムはユーザーのコンピュータ上にダイアログボックスを表示し、ユーザーに、要求したドキュメントを保存する場所を選択するよう求めます。ユーザーが場所を選択すると、open イベント (Event.OPEN) が送出され、openHandler() メソッドが起動されます。

openHandler() メソッドは、ProgressBar コンポーネントの label プロパティのテキストフォーマットを設定し、[キャンセル] ボタンを有効にします。このボタンにより、ユーザーは進行中のダウンロードを即時に停止できます。openHandler() メソッドは、次のとおりです。

```
/**
 * OPEN イベントが送出されたら、プログレスバーのラベルを変更し、
 * [ キャンセル ] ボタンを有効にする。これにより、ユーザーは
 * ダウンロード操作を中止できる
 */
private function openHandler(event:Event):void
{
    pb.label = "DOWNLOADING %3%%";
    btn.enabled = true;
}
```

ファイルのダウンロード進行状況の監視

ファイルがリモートサーバーからユーザーのコンピュータへダウンロードされるとき、一定の間隔で progress イベント (ProgressEvent.PROGRESS) が送出されます。progress イベントが送出されたときは、常に progressHandler() メソッドが起動され、ステージ上の ProgressBar コンポーネントインスタンスが更新されます。progressHandler() メソッドのコードは、次のとおりです。

```
/**
 * ファイルのダウンロード中、プログレスバーのステータスを更新する
 */
private function progressHandler(event:ProgressEvent):void
{
    pb.setProgress(event.bytesLoaded, event.bytesTotal);
}
```

progress イベントには bytesLoaded と bytesTotal の 2 つのプロパティが含まれており、これらは、ステージ上の ProgressBar コンポーネントを更新するために使用されます。これにより、ユーザーはファイルのどれくらいの分量が既にダウンロードを完了し、どれくらいの分量がまだ残っているかを知ることができます。ユーザーは、プログレスバーの下にある [キャンセル] ボタンをクリックすることにより、任意の時点でファイル転送を中止できます。

ファイルのダウンロードが正常に完了した場合は、complete イベント (Event.COMPLETE) が completeHandler() メソッドを起動します。このメソッドはユーザーにファイルのダウンロードが完了したことを通知し、[キャンセル] ボタンを無効にします。completeHandler() メソッドのコードは次のとおりです。

```
/**
 * ダウンロードが完了した後、プログレスバーのラベルを
 * 最後に 1 回だけ変更し、[ キャンセル ] ボタンを無効にする。なぜなら、ダウンロードは
 * 既に完了しているからである
 */
private function completeHandler(event:Event):void
{
    pb.label = "DOWNLOAD COMPLETE";
    btn.enabled = false;
}
```

ファイルのダウンロードのキャンセル

ユーザーは、ステージ上の [キャンセル] ボタンをクリックすることにより、任意の時点でファイル転送を中止し、それ以上のバイトのダウンロードを停止することができます。次のコードは、ダウンロードをキャンセルするためのコードを示しています。

```
/**
 * 現在のファイルのダウンロードをキャンセルする
 */
public function cancelDownload():void
{
    fr.cancel();
```

```
    pb.label = "DOWNLOAD CANCELLED";
    btn.enabled = false;
}
```

最初に、このコードはファイル転送を即時に停止し、それ以上データがダウンロードされないようにします。次に、プログレスバーの label プロパティが更新され、ダウンロードが正常にキャンセルされたことをユーザーに通知します。最後に、[キャンセル] ボタンが無効にされます。これにより、ユーザーは再びファイルのダウンロードを試みるまで、このボタンをクリックできなくなります。

リモートサーバーへのファイルのアップロード

ファイルのアップロードプロセスは、ファイルのダウンロードプロセスに非常によく似ています。FileUpload クラスは、次のコードに示すように、同じ 4 つの変数を宣言します。

```
private const UPLOAD_URL:String = "http://www.yourdomain.com/
  your_upload_script.cfm";
private var fr:FileReference;
private var pb:ProgressBar;
private var btn:Button;
```

FileDownload.DOWNLOAD_URL 変数と異なり、UPLOAD_URL 変数にはユーザーのコンピュータからファイルをアップロードするサーバーサイドスクリプトへの URL が入っています。残りの 3 つの変数は、それらに対応する FileDownload クラス内の変数と同じ働きをします。

FileUpload コンポーネントの初期化

FileUpload コンポーネントには init() メソッドが入っており、このメソッドはメインアプリケーションから呼び出されます。このメソッドは pb と btn の 2 つのパラメータを受け取ります。これらのパラメータは、ステージ上の ProgressBar および Button コンポーネントインスタンスへの参照です。次に、init() メソッドは前に FileUpload クラス内で定義された FileReference オブジェクトを初期化します。最後に、このメソッドは 4 つのイベントリスナーを FileReference オブジェクトに割り当てます。init() メソッドのコードは次のとおりです。

```
public function init(pb:ProgressBar, btn:Button):void
{
    this.pb = pb;
    this.btn = btn;

    fr = new FileReference();
    fr.addEventListener(Event.SELECT, selectHandler);
    fr.addEventListener(Event.OPEN, openHandler);
    fr.addEventListener(ProgressEvent.PROGRESS, progressHandler);
    fr.addEventListener(Event.COMPLETE, completeHandler);
}
```

ファイルのアップロードの開始

ファイルのアップロードは、ユーザーがステージ上の [アップロード] ボタンをクリックしたときに開始されます。このボタンは、`FileUpload.startUpload()` メソッドを起動します。このメソッドは FileReference クラスの `browse()` メソッドを呼び出し、後者のメソッドにより、オペレーティングシステムは、ユーザーにリモートサーバーへアップロードするファイルを選択するよう求めるシステムダイアログボックスを表示します。次のコードは、`startUpload()` メソッドのコードです。

```
public function startUpload():void
{
    fr.browse();
}
```

ユーザーがアップロードするファイルを選択すると、`select` イベント (`Event.SELECT`) が送出され、`selectHandler()` メソッドが呼び出されます。`selectHandler()` メソッドは新しい URLRequest オブジェクトを作成し、`URLRequest.url` プロパティを前にコード内で定義された `UPLOAD_URL` 定数の値に設定します。最後に、FileReference オブジェクトは選択されたファイルを指定されたサーバーサイドスクリプトへアップロードします。`selectHandler()` メソッドのコードは次のとおりです。

```
private function selectHandler(event:Event):void
{
    var request:URLRequest = new URLRequest();
    request.url = UPLOAD_URL;
    fr.upload(request);
}
```

FileUpload クラス内の残りのコードは、FileDownload クラス内で定義されたコードと同じです。ユーザーは、任意の時点でアップロードを終了する場合、[キャンセル] ボタンをクリックします。これにより、プログレスバー上のラベルが設定され、ファイル転送が即時に停止します。プログレスバーは、`progress` イベント (`ProgressEvent.PROGRESS`) が送出されると必ず更新されます。同様に、アップロードが完了すると、プログレスバーが更新され、ファイルが正常にアップロードされたことをユーザーに通知します。その後、[キャンセル] ボタンは、ユーザーが新しいファイル転送を開始するまで無効にされます。

第 15 章

ジオメトリの操作

15

flash.geom パッケージには、ポイント、矩形、変換マトリックスなどのジオメトリオブジェクトを定義するクラスがあります。これらのクラスは、他のクラスで使用されるオブジェクトのプロパティを定義する際に使用します。

目次

Point オブジェクトの使用 417
Rectangle オブジェクトの使用 420
Matrix オブジェクトの使用 423
例 : 表示オブジェクトに対するマトリックス変換の適用 428

Point オブジェクトの使用

Point オブジェクトは、1 ペアの座標値で表される直行座標上のポイントを定義します。2 次元の座標系において、水平軸を表す x と垂直軸を表す y により 1 つのポイントを表現します。

Point オブジェクトを定義する際は、次のように x プロパティと y プロパティを設定します。

```
import flash.geom.*;
var pt1:Point = new Point(10, 20); // x == 10; y == 20
var pt2:Point = new Point();
pt2.x = 10;
pt2.y = 20;
```

2 ポイント間の距離の取得

Point クラスの distance() メソッドを使用すると、同じ座標空間内にある 2 つのポイント間について距離を調べることができます。たとえば、次のコードでは、同じ表示オブジェクトコンテナ内にある 2 つの表示オブジェクト circle1 および circle2 が持つ基準点間の距離を取得します。

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
var pt2:Point = new Point(circle2.x, circle2.y);
var distance:Number = Point.distance(pt1, pt2);
```

座標空間の変換

2 つの表示オブジェクトがそれぞれ異なる表示オブジェクトコンテナ内にある場合は、それぞれの属する座標区間が異なっている可能性があります。DisplayObject クラスの localToGlobal() メソッドを使用すると、それらの座標を共通のグローバルな座標空間 (ステージ座標) に変換できます。たとえば、次のコードでは、別々の表示オブジェクトコンテナ内にある 2 つの表示オブジェクト circle1 および circle2 が持つ基準点間の距離を測定します。

```
import flash.geom.*;
var pt1:Point = new Point(circle1.x, circle1.y);
pt1 = circle1.localToGlobal(pt1);
var pt2:Point = new Point(circle1.x, circle1.y);
pt2 = circle2.localToGlobal(pt2);
var distance:Number = Point.distance(pt1, pt2);
```

同様に、target という表示オブジェクトの基準点とステージ上にある特定のポイントとの距離を調べる場合は、DisplayObject クラスの localToGlobal() メソッドを次のように使用します。

```
import flash.geom.*;
var stageCenter:Point = new Point();
stageCenter.x = this.stage.stageWidth / 2;
stageCenter.y = this.stage.stageHeight / 2;
var targetCenter:Point = new Point(target.x, target.y);
targetCenter = target.localToGlobal(targetCenter);
var distance:Number = Point.distance(stageCenter, targetCenter);
```

角度と距離の指定による表示オブジェクトの移動

移動距離と角度を指定してオブジェクトを移動するには、Point クラスの polar() メソッドを使用します。たとえば、次のコードでは、myDisplayObject オブジェクトを 60 度の方向に 100 ピクセル分だけ移動します。

```
import flash.geom.*;
var distance:Number = 100;
var angle:Number = 2 * Math.PI * (90 / 360);
var translatePoint:Point = Point.polar(distance, angle);
myDisplayObject.x += translatePoint.x;
myDisplayObject.y += translatePoint.y;
```

Point クラスの他の使用方法

Point オブジェクトは次のメソッドおよびプロパティで使用できます。

クラス	メソッドまたはプロパティ	説明
DisplayObjectContainer	areInaccessibleObjectsUnderPoint() getObjectsUnderPoint()	表示オブジェクトコンテナ内のポイントに存在するオブジェクトのリストを返すために使用します。
BitmapData	hitTest()	BitmapData オブジェクト内のピクセルと、衝突のチェックに使用するポイントを定義するために使用します。
BitmapData	applyFilter() copyChannel() merge() paletteMap() pixelDissolve() threshold()	操作の対象とする長方形の位置を定義するために使用します。
Matrix	deltaTransformPoint() transformPoint()	変換を適用するポイントを定義するために使用します。
Rectangle	bottomRight size topLeft	これらのプロパティを定義するために使用します。

Rectangle オブジェクトの使用

Rectangle オブジェクトは、長方形の領域を定義します。Rectangle オブジェクトには、位置 (左上隅の x 座標と y 座標で定義される) と、width プロパティ、height プロパティがあります。新しい Rectangle オブジェクトに対してこれらのプロパティを指定するには、次のように Rectangle() コンストラクタを呼び出します。

```
import flash.geom.Rectangle;
var rx:Number = 0;
var ry:Number = 0;
var rwidth:Number = 100;
var rheight:Number = 50;
var rect1:Rectangle = new Rectangle(rx, ry, rwidth, rheight);
```

Rectangle オブジェクトのサイズ変更と移動

Rectangle オブジェクトをサイズ変更および移動するには多数の方法があります。

x および y プロパティを変更すると、直接に Rectangle オブジェクトの位置を移動できます。ただし、Rectangle オブジェクトの幅と高さをこの方法で変更しても効果はありません。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.x = 20;
rect1.y = 30;
trace(rect1); // (x=20, y=30, w=100, h=50)
```

次のコードで示すように、left または top プロパティを変更すると、Rectangle オブジェクトの位置が移動し、left および top それぞれの値に対応して x および y プロパティの値が変化します。ただし、Rectangle オブジェクトの右下隅の位置は変化せず、その代わりに長方形のサイズが変化します。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.left = 20;
rect1.top = 30;
trace(rect1); // (x=30, y=20, w=70, h=30)
```

同様に、次の例で示すように、Rectangle オブジェクトの `bottom` または `right` プロパティを変更すると、左上隅の位置は変化せずに長方形のサイズが変化します。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.right = 60;
trect1.bottom = 20;
trace(rect1); // (x=0, y=0, w=60, h=20)
```

次のように、Rectangle オブジェクトの `offset()` メソッドでも位置を移動できます。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.offset(20, 30);
trace(rect1); // (x=20, y=30, w=100, h=50)
```

`offsetPt()` メソッドも同様に機能しますが、パラメータとして x および y のオフセット値ではなく Point オブジェクトを指定します。

Rectangle オブジェクトの `inflate()` メソッドに `dx` および `dy` の 2 つのパラメータを指定してサイズを変更することもできます。`dx` パラメータは、長方形の中心に対して右辺と左辺を移動する量 (ピクセル単位) を表します。`dy` パラメータは、長方形の中心に対して上辺と下辺を移動する量 (ピクセル単位) を表します。

```
import flash.geom.Rectangle;
var x1:Number = 0;
var y1:Number = 0;
var width1:Number = 100;
var height1:Number = 50;
var rect1:Rectangle = new Rectangle(x1, y1, width1, height1);
trace(rect1) // (x=0, y=0, w=100, h=50)
rect1.inflate(6,4);
trace(rect1); // (x=-6, y=-4, w=112, h=58)
```

The `inflatePt()` メソッドも同様に機能しますが、パラメータとして `dx` および `dy` の値ではなく Point オブジェクトを指定します。

2 つの Rectangle オブジェクトから成る結合および交差の取得

2 つの長方形を合わせた境界線によって形成される長方形領域を取得するには、union() メソッドを使用します。

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(120, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.union(rect2)); // (x=0, y=0, w=220, h=160)
```

2 つの長方形が重なり合う部分を表す長方形領域を取得するには、intersection() メソッドを使用します。

```
import flash.display.*;
import flash.geom.Rectangle;
var rect1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(rect1); // (x=0, y=0, w=100, h=100)
var rect2:Rectangle = new Rectangle(80, 60, 100, 100);
trace(rect2); // (x=120, y=60, w=100, h=100)
trace(rect1.intersection(rect2)); // (x=80, y=60, w=20, h=40)
```

2 つの長方形に重なり合う部分があるかどうかを調べるには、intersects() メソッドを使用します。また、表示オブジェクトがステージ上の特定の領域内に配置されているかどうかを調べる場合にも intersects() メソッドを使用できます。次のコード例では、circle オブジェクトを含んだ表示オブジェクトコンテナの座標空間がステージの座標空間と同じであることが前提となっています。この例は、intersects() メソッドを使用して、表示オブジェクト circle が target1 および target2 Rectangle オブジェクトで定義されたステージの指定された領域と交差しているかどうかを判断する方法を示しています。

```
import flash.display.*;
import flash.geom.Rectangle;
var circle:Shape = new Shape();
circle.graphics.lineStyle(2, 0xFF0000);
circle.graphics.drawCircle(250, 250, 100);
addChild(circle);
var circleBounds:Rectangle = circle.getBounds(stage);
var target1:Rectangle = new Rectangle(0, 0, 100, 100);
trace(circleBounds.intersects(target1)); // false
var target2:Rectangle = new Rectangle(0, 0, 300, 300);
trace(circleBounds.intersects(target2)); // true
```

同様に、2 つの表示オブジェクトの境界を示す矩形に重なり合う部分があるかどうかを調べる場合にも intersects() メソッドを使用できます。表示オブジェクトの境界線内の領域に加え、線の太さによって余分なスペースが必要となる場合は、その分を加味した領域を取得するために DisplayObject クラスの getRect() メソッドを使用できます。

Rectangle オブジェクトの他の使用方法

Rectangle オブジェクトは次のメソッドおよびプロパティで使用されます。

クラス	メソッドまたはプロパティ	説明
BitmapData	applyFilter()、colorTransform()、copyChannel()、copyPixels()、draw()、fillRect()、generateFilterRect()、getColorBoundsRect()、getPixels()、merge()、paletteMap()、pixelDissolve()、setPixels()、および threshold()	BitmapData オブジェクトの領域を定義するパラメータの型として使用します。
DisplayObject	getBounds()、getRect()、scrollRect、scale9Grid	プロパティのデータ型または返されるデータ型として使用します。
PrintJob	addPage()	printArea パラメータを定義するために使用します。
Sprite	startDrag()	bounds パラメータを定義するために使用します。
TextField	getCharBoundaries()	戻り値型として使用します。
Transform	pixelBounds	データ型として使用します。

Matrix オブジェクトの使用

Matrix クラスは、座標空間の間でポイントをマッピングする方法を決定する変換行列を表します。Matrix オブジェクトのプロパティを設定し、その Matrix オブジェクトを Transform オブジェクトの matrix プロパティに適用し、さらに、その Transform オブジェクトを表示オブジェクトの transform プロパティに適用することで、表示オブジェクトに対してさまざまなグラフィック変換を実行できます。実現できる変換機能としては、平行移動 (x 位置および y 位置の移動)、回転、拡大・縮小、傾斜などがあります。

Matrix オブジェクトの定義

マトリックスは、プロパティ (a、b、c、d、tx、ty) を調整して直接に定義することもできますが、createBox() メソッドを使用して定義するほうが簡単です。このメソッドのパラメータでは、作成するマトリックスに定義する拡大・縮小、回転、平行移動の効果を直接に定義できます。たとえば、次のコードで作成している Matrix オブジェクトには、水平方向 2.0 倍の拡大、垂直方向 3.0 倍の拡大、45 度の回転、右に 10 ピクセルの移動、および下に 20 ピクセルの移動の効果があります。

```
var matrix:Matrix = new Matrix();
var scaleX:Number = 2.0;
var scaleY:Number = 3.0;
var rotation:Number = 2 * Math.PI * (45 / 360);
var tx:Number = 10;
var ty:Number = 20;
matrix.createBox(scaleX, scaleY, rotation, tx, ty);
```

また、scale()、rotate()、translate() の各メソッドを使用すると、Matrix オブジェクトに設定した拡大・縮小、回転、平行移動の効果を変更できます。これらのメソッドで指定した値は、既存の Matrix オブジェクトに設定されている値と組み合わされます。たとえば、次のコードでは scale() と rotate() を 2 回呼び出しているため、Matrix オブジェクトの効果は 4 倍の拡大と 60 度の回転になります。

```
var matrix:Matrix = new Matrix();
var rotation:Number = 2 * Math.PI * (30 / 360); // 30°
var scaleFactor:Number = 2;
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);
matrix.scale(scaleX, scaleY);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

傾斜変形を Matrix オブジェクトに適用するには、b プロパティまたは c プロパティを調整します。b プロパティを調整するとマトリックスが垂直方向に傾斜し、c プロパティを調整するとマトリックスが水平方向に傾斜します。次のコードは、myMatrix Matrix オブジェクトを垂直方向に 2 倍傾斜します。

```
var skewMatrix:Matrix = new Matrix();
skewMatrix.b = Math.tan(2);
myMatrix.concat(skewMatrix);
```

Matrix による変換は、表示オブジェクトの transform プロパティに適用できます。たとえば、次のコードでは myDisplayObject というオブジェクトにマトリックス変換を適用しています。

```
var matrix:Matrix = myDisplayObject.transform.matrix;
var scaleFactor:Number = 2;
var rotation:Number = 2 * Math.PI * (60 / 360); // 60°
matrix.scale(scaleFactor, scaleFactor);
matrix.rotate(rotation);

myDisplayObject.transform.matrix = matrix;
```

最初の行で、`myDisplayObject` 表示オブジェクトが使用する既存の変換マトリックス (`myDisplayObject` 表示オブジェクトの `transformation` プロパティの `matrix` プロパティ) を Matrix オブジェクトに設定します。こうすることで、表示オブジェクトの現状の位置、拡大率、回転角をふまえた上に、Matrix クラスメソッドの実行による効果を累積的に適用できます。

メモ	flash.geometry パッケージには ColorTransform クラスも含まれていますが、このクラスは Transform オブジェクトの `colorTransform` プロパティを設定する際に使用するものです。幾何学変換には使用しないため、この章では説明しません。詳細については、『ActionScript 3.0 リファレンスガイド』の ColorTransform クラスを参照してください。

グラデーションに関する Matrix オブジェクトの定義

シェイプで使用するグラデーションを定義するには、flash.display.Graphics クラスの `beginGradientFill()` および `lineGradientStyle()` メソッドを使用します。
グラデーションを定義する際、これらのメソッドのパラメータでマトリックスを指定します。

この目的でマトリックスを作成するには、`createGradientBox()` メソッドを使用してグラデーション定義用の長方形を定義するのが最も簡単な方法です。`createGradientBox()` メソッドに指定するパラメータでは、グラデーションの拡大・縮小、回転、および位置を定義します。

たとえば、次の性質を持つグラデーションを作成するとします。

- `GradientType.LINEAR`
- グリーンとブルーの２色、`ratios` 配列を `[0, 255]` に設定
- `SpreadMethod.PAD`
- `InterpolationMethod.LINEAR_RGB`

次の例で示すグラデーションは、createGradientBox() メソッドの rotation パラメータが異なりますが、その他の設定はすべて同じです。

```
width = 100;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```

```
width = 100;
height = 100;
rotation = Math.PI/4; // 45°
tx = 0;
ty = 0;
```

```
width = 100;
height = 100;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```

次の例で示すグリーン 〜 ブルーの線状グラデーション効果は、createGradientBox() メソッドの rotation、tx、ty パラメータが異なりますが、その他の設定はすべて同じです。

```
width = 50;
height = 100;
rotation = 0;
tx = 0;
ty = 0;
```

```
width = 50;
height = 100;
rotation = 0
tx = 50;
ty = 0;
```

```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 0;
```

```
width = 100;
height = 50;
rotation = Math.PI/2; // 90°
tx = 0;
ty = 50;
```

createGradientBox() メソッドの width、height、tx、ty パラメータは、次の例で示すように、放射状グラデーションの塗りに関するサイズと位置にも影響します。

```
width = 50;
height = 100;
rotation = 0;
tx = 25;
ty = 0;
```

次のコードは、前の図の最後に示した放射状グラデーションを生成します。

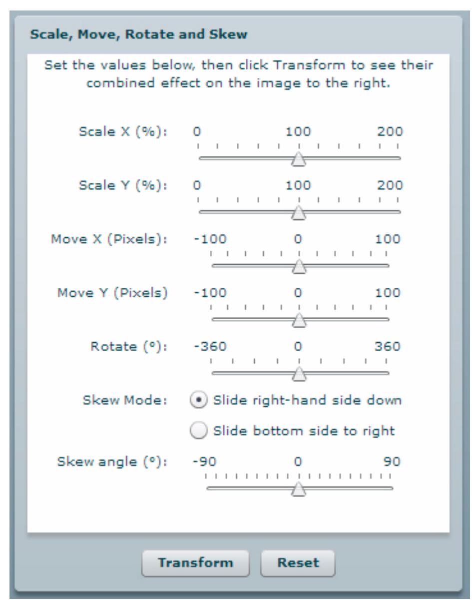
```
import flash.display.Shape;
import flash.display.GradientType;
import flash.geom.Matrix;

var type:String = GradientType.RADIAL;
var colors:Array = [0x00FF00, 0x000088];
var alphas:Array = [1, 1];
var ratios:Array = [0, 255];
var spreadMethod:String = SpreadMethod.PAD;
var interp:String = InterpolationMethod.LINEAR_RGB;
var focalPtRatio:Number = 0;

var matrix:Matrix = new Matrix();
var boxWidth:Number = 50;
var boxHeight:Number = 100;
var boxRotation:Number = Math.PI/2; // 90°
var tx:Number = 25;
var ty:Number = 0;
matrix.createGradientBox(boxWidth, boxHeight, boxRotation, tx, ty);

var square:Shape = new Shape;
square.graphics.beginGradientFill(type,
                 colors,
                 alphas,
                 ratios,
                 matrix,
                 spreadMethod,
                 interp,
                 focalPtRatio);
square.graphics.drawRect(0, 0, 100, 100);
addChild(square);
```

例 : 表示オブジェクトに対するマトリックス変換の適用

DisplayObjectTransformer サンプルアプリケーションで、Matrix クラスを使用して表示オブジェクトを変形する次のような多くの機能を示します。

- 表示オブジェクトの回転
- 表示オブジェクトの拡大 / 縮小
- 表示オブジェクトの平行移動 (再配置)
- 表示オブジェクトの傾斜

アプリケーションには、次のようなマトリックス変換のパラメータを調整するインターフェイスがあります。



ユーザーが [Transform] ボタンを押すと、アプリケーションは適切な変換を適用します。

元の表示オブジェクトと、-45 度回転して 50% 縮小された表示オブジェクト

DisplayObjectTransformer アプリケーションのファイルは、"Samples/DisplayObjectTransformer" フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
DisplayObjectTransformer.mxml	MXML で記述された Flex 用メインアプリケーションファイル
com/example/programmingas3/geometry/MatrixTransformer.as	マトリックス変換を適用するためのメソッドが含まれているクラス
img/	アプリケーションが使用するサンプルイメージファイルが保存されているディレクトリ

MatrixTransformer クラスの定義

MatrixTransformer クラスには、Matrix オブジェクトの図形変換を適用する静的メソッドが含まれます。

transform() メソッド

`transform()` メソッドには、次の各パラメータが含まれます。

- `sourceMatrix`：メソッドが変換する入力マトリックス
- `xScale` および `yScale`：x および y の縮尺率
- `dx` および `dy`：x および y の平行移動の量 (ピクセル単位)
- `rotation`：回転量 (度単位)
- `skew`：傾斜係数 (パーセント単位)
- `skewType`：傾斜する方向 (`"right"` または `"left"`)

戻り値は、結果のマトリックスです。

`transform()` メソッドは、クラスの次の静的メソッドを呼び出します。

- `skew()`
- `scale()`
- `translate()`
- `rotate()`

それぞれ、ソースマトリックスに変換を適用して返します。

skew() メソッド

skew() メソッドは、マトリックスの b および c プロパティを調整して、マトリックスを傾斜します。オプションのパラメータ unit によって、傾斜角度の定義に使用する単位が決まります。必要に応じて、メソッドは angle 値をラジアンに変換します。

```
if (unit == "degrees")
{
    angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
    angle = Math.PI * 2 * angle / 100;
}
```

傾斜変換を適用するために、skewMatrix Matrix オブジェクトが作成され、調整されます。最初は、次のように、同一のマトリックスです。

```
var skewMatrix:Matrix = new Matrix();
```

skewSide パラメータによって、傾斜を適用する側が決まります。"right" が設定されている場合、次のコードでマトリックスの b プロパティが設定されます。

```
skewMatrix.b = Math.tan(angle);
```

それ以外の場合、次のように Matrix の c プロパティが調整され、下側が傾斜されます。

```
skewMatrix.c = Math.tan(angle);
```

次の例で示すように、2 つのマトリックスを連結することによって、結果の傾斜が既存のマトリックスに適用されます。

```
sourceMatrix.concat(skewMatrix);
return sourceMatrix;
```

scale () メソッド

次の例で示すように、scale() メソッドはまず、縮尺率がパーセントで指定されている場合、調整します。次に、マトリックスオブジェクトの scale() メソッドを使用します。

```
if (percent)
{
    xScale = xScale / 100;
    yScale = yScale / 100;
}
sourceMatrix.scale(xScale, yScale);
return sourceMatrix;
```

translate () メソッド

translate() メソッドは、次のようにマトリックスオブジェクトの translate() メソッドを呼び出し、単純に、dx および dy 平行移動係数を適用します。

```
sourceMatrix.translate(dx, dy);
return sourceMatrix;
```

rotate () メソッド

rotate() メソッドは、入力回転角度をラジアンに変換し (度またはグラデーションで指定されている場合)、マトリックスオブジェクトの rotate() メソッドを呼び出します。

```
if (unit == "degrees")
{
   angle = Math.PI * 2 * angle / 360;
}
if (unit == "gradients")
{
   angle = Math.PI * 2 * angle / 100;
}
sourceMatrix.rotate(angle);
return sourceMatrix;
```

アプリケーションからの MatrixTransformer.transform() メソッドの呼び出し

アプリケーションには、ユーザーが変換パラメータを指定するためのユーザーインターフェイスがあります。パラメータが指定されると、次のように、パラメータと表示オブジェクトの transform プロパティの matrix プロパティを Matrix.transform() メソッドに渡します。

```
tempMatrix = MatrixTransformer.transform(tempMatrix,
                                         xScaleSlider.value,
                                         yScaleSlider.value,
                                         dxSlider.value,
                                         dySlider.value,
                                         rotationSlider.value,
                                         skewSlider.value,
                                         skewSide );
```

次に、アプリケーションは戻り値を表示オブジェクトの transform プロパティの matrix プロパティに適用します。これによって、変換がトリガされます。

```
img.content.transform.matrix = tempMatrix;
```

第 16 章

クライアントのシステム環境

16

クライアントのシステム環境は、flash.system パッケージ内のクラスのコレクションです。これらのクラスを使用すると、システムレベルの機能にアクセスでき、たとえば、SWF を実行しているアプリケーションおよびセキュリティドメインの判別、ユーザーの Flash Player の機能の判別、IME (Input Method Editor) を使用した複数言語サイトの構築、Flash Player のコンテナ (HTML ページや コンテナアプリケーションの場合もある) の操作、ユーザーのクリップボードへの情報の保存などを行うことができます。また、flash.system パッケージには IMEConversionMode および SecurityPanel クラスも含まれています。これらでは、それぞれ IME クラスと Security クラスで使用する静的定数が定義されています。

この章では、ユーザーのシステムを操作する方法について説明します。ここでは、サポートされている機能を判別する方法と、ユーザーが IME をインストールしている場合に IME を使用して複数言語の SWF ファイルを構築する方法を示します。また、アプリケーションドメインの一般的な使用方法も示します。

目次

System クラス 433
Capabilities クラス 435
ApplicationDomain クラス 436
IME クラス 439
例 : システム機能の検出.. 444

System クラス

System クラスには、ユーザーのオペレーティングシステムを操作し、Flash Player の現在のメモリ使用状況を取得するために使用できるメソッドとプロパティがあります。System クラスのメソッドとプロパティを使用すると、`imeComposition` イベントの待ち受け、ユーザーの現行コードページを使用した外部テキストファイルのロードまたは Unicode での外部テキストファイルのロードを行うための Flash Player への指示、ユーザーのクリップボードの内容の設定も行うこともできます。

実行時におけるユーザーのシステムに関するデータの取得

System.totalMemory プロパティを調べると、Flash Player が現在使用しているメモリの容量をバイト単位で判別できます。このプロパティを使用してメモリ使用量を監視すれば、空きメモリの状況の変化に応じてアプリケーションの動作を最適に調整できます。たとえば、特定のビジュアル効果を使用するとメモリ使用量が大幅に増加する場合、状況によってその効果を変化または完全に無効にすることが考えられます。

System.ime プロパティは、現在インストールされている IME (Input Method Editor) への参照です。このプロパティを使用すると、addEventListener() メソッドで imeComposition イベント (flash.events.IMEEvent.IME_COMPOSITION) を待ち受けることができます。

System クラスの 3 番目のプロパティは、useCodePage です。useCodePage を true に設定すると、Flash Player を実行しているオペレーティングシステムの通常のコードページを使用して外部テキストファイルがロードされます。このプロパティを false に設定すると、Flash Player は外部ファイルを Unicode として解釈します。

System.useCodePage を true に設定した場合、Flash Player を実行している OS の通常のコードページに属する文字を使用して外部テキストファイルの内容が記述されていないと、テキストは表示されないことに注意してください。たとえば、中国語を含んだ外部テキストファイルをロードする場合、English Windows コードページを使用しているシステム上では文字が表示されません。これは、English Windows コードページには中国語の文字が含まれないためです。

SWF ファイル内で使用される外部テキストファイルをすべてのプラットフォームのユーザーが表示できるようにするには、すべての外部テキストファイルを Unicode で保存し、System.useCodePage をデフォルトの false の設定のままにします。そうすれば、Flash Player 6 以降ではテキストが Unicode として解釈されます。

クリップボードへのテキストの保存

System クラスには setClipboard() というメソッドがあります。このメソッドを使用すると、Flash Player で、指定したストリングを使用してユーザーのクリップボードの内容を設定できます。セキュリティ上の理由により、Security.getClipboard() メソッドはありません。そのようなメソッドは、悪意あるサイトがユーザーのクリップボードに最後にコピーされたデータにアクセスすることを許す可能性があるからです。

次のコードは、セキュリティエラーが発生したときに、ユーザーのクリップボードにエラーメッセージをコピーする方法の例です。このエラーメッセージは、ユーザーがアプリケーションの潜在的なバグを報告する場合に役に立ちます。

```
private function securityErrorHandler(event:SecurityErrorEvent):void
{
    var errorString:String = "[" + event.type + "] " + event.text;
    trace(errorString);
    System.setClipboard(errorString);
}
```

Capabilities クラス

Capabilities クラスを使用すると、開発者は SWF ファイルの実行環境に関する情報を取得できます。Capabilities クラスのさまざまなプロパティにより、ユーザーのシステムにおける画面解像度、アクセシビリティソフトウェアに対するサポートの有無、ユーザーのオペレーティングシステムの言語、および現在インストールされている Flash Player のバージョンを知ることができます。

Capabilities クラスのプロパティを確認すれば、実際のユーザー環境に応じてアプリケーションの動作を最適に調整できます。たとえば、`Capabilities.screenResolutionX` および `Capabilities.screenResolutionY` プロパティを調べることにより、ユーザーのシステムで使用されているディスプレイ解像度を判別し、どのサイズのビデオが最適かを判断できます。また、`Capabilities.hasMP3` プロパティを調べれば、外部 MP3 ファイルをロードする前にユーザーのシステムにおける MP3 再生のサポート状況を知ることができます。

次のコードでは、クライアント環境にインストールされている Flash Player のバージョン情報を正規表現で解析しています。

```
var versionString:String = Capabilities.version;
var pattern:RegExp = /^(\w*) (\d*),(\d*),(\d*),(\d*)$/;
var result:Object = pattern.exec(versionString);
if (result != null)
{
   trace("input: " + result.input);
   trace("platform: " + result[1]);
   trace("majorVersion: " + result[2]);
   trace("minorVersion: " + result[3]);
   trace("buildNumber: " + result[4]);
   trace("internalBuildNumber: " + result[5]);
}
else
{
   trace("Unable to match RegExp.");
}
```

ユーザーのシステムの機能に関する情報をサーバーサイドスクリプトに伝えてデータベースに保存するには、次のような ActionScript コードを使用します。

```
var url:String = "log_visitor.cfm";
var request:URLRequest = new URLRequest(url);
request.method = URLRequestMethod.POST;
request.data = new URLVariables(Capabilities.serverString);
var loader:URLLoader = new URLLoader(request);
```

ApplicationDomain クラス

ApplicationDomain クラスの目的は、ActionScript 3.0 の定義のテーブルを保存することです。SWF ファイル内のすべてのコードは、アプリケーションドメイン内に存在するように定義されます。アプリケーションドメインは、同じセキュリティドメイン内にある複数のクラスを分離するために使用します。それにより、1 つのクラスについて複数の定義を用意することや、子が親の定義を再利用することが可能になります。

ActionScript 3.0 で作成された外部 SWF ファイルを Loader クラス API でロードするときは、アプリケーションドメインを使用できます。ActionScript 1.0 または ActionScript 2.0 で作成された SWF ファイルのイメージをロードするときは、アプリケーションドメインを使用できないことに注意してください。ロードされたクラスに含まれているすべての ActionScript 3.0 定義は、アプリケーションドメイン内に格納されます。SWF ファイルをロードするとき、LoaderContext オブジェクトの `applicationDomain` パラメータを `ApplicationDomain.currentDomain` に設定することにより、ファイルを Loader オブジェクトと同じアプリケーションドメインに含めるよう指定できます。ロードされた SWF ファイルを同じアプリケーションドメインに置くと、そのクラスに直接アクセスできます。これは、関連付けられたクラス名によってアクセスできるメディアが埋め込まれた SWF ファイルをロードする場合や、ロードされた SWF ファイルのメソッドにアクセスする場合に便利です。次に例を示します。

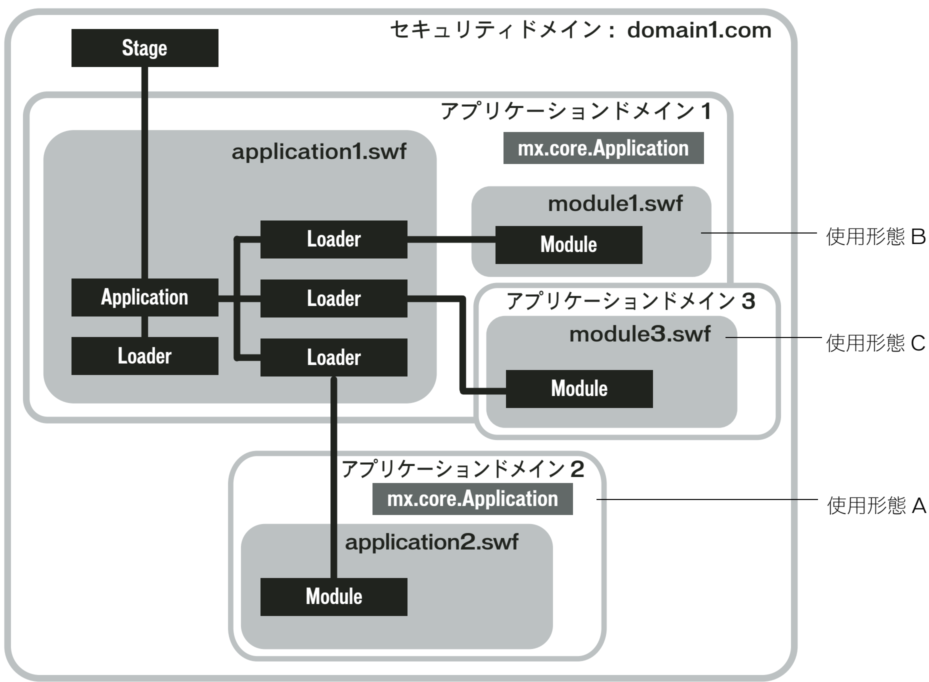
```
package
{
  import flash.display.Loader;
  import flash.display.Sprite;
  import flash.events.*;
  import flash.net.URLRequest;
  import flash.system.ApplicationDomain;
  import flash.system.LoaderContext;

  public class ApplicationDomainExample extends Sprite
  {
    private var ldr:Loader;
    public function ApplicationDomainExample()
    {
      ldr = new Loader();
      var req:URLRequest = new URLRequest("Greeter.swf");
      var ldrContext:LoaderContext = new LoaderContext(false,
  ApplicationDomain.currentDomain);
      ldr.contentLoaderInfo.addEventListener(Event.COMPLETE, completeHandler);
      ldr.load(req, ldrContext);
    }
    private function completeHandler(event:Event):void
    {
      ApplicationDomain.currentDomain.getDefinition("Greeter");
      var myGreeter:Greeter = Greeter(event.target.content);
      var message:String = myGreeter.welcome("Tommy");
      trace(message); // Hello, Tommy
```

```
        }
    }
}
```

それ以外に、アプリケーションドメインを操作するときに注意すべき点は、次のとおりです。

- SWF ファイル内のすべてのコードは、アプリケーションドメイン内に存在するように定義されます。" 現在のドメイン " とは、メインアプリケーションが実行される場所のことです。現在のドメインおよび他のすべてのアプリケーションドメインは " システムドメイン " 内に属します。つまり、システムドメインにはすべての Flash Player クラスが含まれます。
- システムドメインを除くすべてのアプリケーションドメインには、親ドメインが関連付けられています。メインアプリケーションのアプリケーションドメインの親ドメインは、システムドメインです。ロードされるクラスは、そのクラスの親によってまだ定義されていない場合にのみ定義されます。ロードされるクラスの定義を新しい定義でオーバーライドすることはできません。

次の図に示すアプリケーションでは、さまざまな SWF ファイルから domain1.com という単一のドメイン内にコンテンツをロードしています。ロードするコンテンツによっては、異なるアプリケーションドメインを使用することもできます。このアプリケーションに含まれる個々の SWF ファイルでは、それぞれに適した異なる形態でアプリケーションドメインを使用しています。各形態の使い分け方について次に説明します。



メインアプリケーションファイルは application1.swf です。これには、他の SWF ファイルからコンテンツをロードする Loader オブジェクトが入っています。このシナリオでは、現在のドメインは Application domain 1 です。使用形態 A、使用形態 B、および使用形態 C は、アプリケーション内の各 SWF ファイルに適切なアプリケーションドメインを設定するさまざまな手法の例を示しています。

使用形態 A : システムドメインの子ドメインを作成することにより、子 SWF ファイルを分離します。図では、システムドメインの子ドメインとして Application domain 2 を作成しています。"application2.swf" ファイルを Application domain 2 内にロードしているため、このファイルに含まれるクラス定義は "application1.swf" のクラス定義と分離されます。

この形態の用途としては、古いアプリケーションにおいて同じアプリケーションの新バージョンを動的にロードする場合に競合の発生を防ぐことが考えられます。異なるバージョンを異なるアプリケーションドメインに分離すれば、それぞれにおいて同じクラス名を使用していても競合が発生しません。

次のコードでは、システムドメインの子ドメインとしてアプリケーションドメインを作成します。

```
request.url = "application2.swf";
request.applicationDomain = new ApplicationDomain();
```

使用形態 B : 現在のクラス定義に新しいクラス定義を追加します。図では、"module1.swf" に対するアプリケーションドメインとして現在のドメイン (Application domain 1) を設定しています。この場合、アプリケーションに現在含まれている各種のクラス定義に、新しいクラス定義を加えることになります。この形態は、メインアプリケーションのランタイム共有ライブラリに使用できます。ロードされた SWF は、リモート共有ライブラリ (RSL) として扱われます。アプリケーションが開始する前にプリローダーによって RSL をロードするには、このテクニックを使用します。

次のコードでは、現在のドメインをアプリケーションドメインとして設定します。

```
request.url = "module1.swf";
request.applicationDomain = ApplicationDomain.currentDomain;
```

使用形態 C : 現在のドメインの子ドメインを新しく作成することにより、親のクラス定義を使用します。図では、"module3.swf" に対するアプリケーションドメインとして現在のドメインの子ドメインを設定しており、子ドメインでは、親ドメインに含まれるすべてのクラス定義を使用します。この形態の用途としては、複数画面の RIA (高度なインターネットアプリケーション) を構成するモジュールをメインアプリケーションの子としてロードし、メインアプリケーションで定義されている型をモジュール内でも使用できるようにすることが考えられます。クラスを更新される際に必ず後方互換性を維持するようにし、また、ロードされるモジュールよりもメインアプリケーションを常に新しくしておけば、メインアプリケーションに含まれるクラス定義はモジュール内でも使用されます。また、このようにして新しいアプリケーションドメイン内にロードしたモジュールは、アンロードすれば、すべてのクラス定義をガベージコレクションの対象とすることができます (ただし、子 SWF への参照を確実に破棄する必要があります)。

このテクニックでは、ロードする側にあるシングルトンオブジェクトや静的クラスメンバーを、ロードされる側のモジュールでも共有できます。

次のコードでは、現在のドメインの子ドメインを新しく作成します。

```
request.url = "module3.swf";
request.applicationDomain = new
    ApplicationDomain(ApplicationDomain.currentDomain);
```

IME クラス

IME クラスを使用すると、オペレーティングシステムの IME (Input Method Editor) を Flash Player 内で操作できます。

ActionScript から、次の事項を調べることができます。

- ユーザーのコンピュータに IME がインストールされているかどうか (`Capabilities.hasIME`)
- ユーザーのコンピュータで IME が有効になっているかどうか (`IME.enabled`)
- 現在の IME で使用されている変換モード (`IME.conversionMode`)

テキスト入力フィールドを特定の IME コンテキストに関連付けることができます。それにより、入力フィールド間をフォーカスが移動するのに従って IME の変換モード (ひらがな、全角数字、半角数字、直接入力など) の切り替えができます。

IME を使用すると、中国語、日本語、韓国語といったマルチバイトの言語で ASCII 以外の文字を入力できるようになります。

IME の使用方法の詳細については、アプリケーションの開発対象であるオペレーティングシステムのマニュアルを参照してください。追加情報については、次の Web サイトを参照してください。

- http://www.microsoft.com/globaldev/default.mspx
- http://developer.apple.com/documentation/
- http://java.sun.com/

メモ

IME がユーザーのコンピュータ上でアクティブでない場合、IME のメソッドまたはプロパティの呼び出しは、`Capabilities.hasIME` を除き、失敗します。IME を手動でアクティブにすると、それ以降の ActionScript による IME のメソッドまたはプロパティの呼び出しは、意図したとおりに動作します。たとえば、日本語 IME を使用する場合は、IME のメソッドまたはプロパティを呼び出す前に IME をアクティブにする必要があります。

IME がインストールされ有効になっているかどうかの確認

IME のメソッドまたはプロパティを呼び出す前に、必ずユーザーのコンピュータを調べ、IME が現在インストールされ有効になっているかどうかを確認する必要があります。次のコードは、いずれかのメソッドを呼び出す前に、ユーザーが IME をインストール済みで、それがアクティブであるかどうかを確認する方法を示しています。

```
if (Capabilities.hasIME)
{
    if (IME.enabled)
    {
        trace("IME is installed and enabled.");
    }
    else
    {
        trace("IME is installed but not enabled.Please enable your IME and try
    again.");
    }
}
else
{
    trace("IME is not installed.Please install an IME and try again.");
}
```

上記のコードは、最初に、Capabilities.hasIME プロパティを使用して、ユーザーが IME をインストールしているかどうかを確認します。このプロパティが true に設定されている場合は、次に IME.enabled プロパティを使用して、ユーザーの IME が現在有効になっているかどうかを確認します。

どの IME 変換モードが現在有効であるかの判別

複数言語アプリケーションを構築するときは、ユーザーが現在どの変換モードをアクティブにしているかを判別することが必要な場合もあります。次のコードは、ユーザーが IME をインストールしてあるかどうか、また、インストールしてある場合は、どの IME 変換モードが現在アクティブであるかを確認する方法を示しています。

```
if (Capabilities.hasIME)
{
    switch (IME.conversionMode)
    {
        case IMEConversionMode.ALPHANUMERIC_FULL:
            tf.text = "Current conversion mode is alphanumeric (full-width).";
            break;
        case IMEConversionMode.ALPHANUMERIC_HALF:
            tf.text = "Current conversion mode is alphanumeric (half-width).";
            break;
        case IMEConversionMode.CHINESE:
            tf.text = "Current conversion mode is Chinese.";
            break;
        case IMEConversionMode.JAPANESE_HIRAGANA:
```

```
        tf.text = "Current conversion mode is Japananese Hiragana.";
        break;
    case IMEConversionMode.JAPANESE_KATAKANA_FULL:
        tf.text = "Current conversion mode is Japanese Katakana (full-width).";
        break;
    case IMEConversionMode.JAPANESE_KATAKANA_HALF:
        tf.text = "Current conversion mode is Japanese Katakana (half-width).";
        break;
    case IMEConversionMode.KOREAN:
        tf.text = "Current conversion mode is Korean.";
        break;
    default:
        tf.text = "Current conversion mode is " + IME.conversionMode + ".";
        break;
  }
}
else
{
  tf.text = "Please install an IME and try again.";
}
```

上記のコードでは、最初に、ユーザーが IME をインストールしているかどうかを確認します。次に、現在の IME がどの変換モードを使用しているかを確認するために、IME.conversionMode プロパティを IMEConversionMode クラス内のそれぞれの定数と照合します。

IME 変換モードの設定

ユーザーの IME の変換モードを変更するときは、コードが try..catch ブロック内で折り返されていることを確認する必要があります。conversionMode プロパティを使用して変換モードを設定すると、IME が変換モードを設定できない場合にエラーがスローされる可能性があるからです。次のコードは、IME.conversionMode プロパティを設定するときの try..catch ブロックの使用方法を示しています。

```
var statusText:TextField = new TextField;
statusText.autoSize = TextFieldAutoSize.LEFT;
addChild(statusText);
if (Capabilities.hasIME)
{
  try
  {
    IME.enabled = true;
    IME.conversionMode = IMEConversionMode.KOREAN;
    statusText.text = "Conversion mode is " + IME.conversionMode + ".";
  }
  catch (error:Error)
  {
    statusText.text = "Unable to set conversion mode.\n" + error.message;
  }
}
```

上記のコードは、最初に、ユーザーに対してステータスメッセージを表示するためのテキストフィールドを作成します。次に、このコードは、IME がインストールされている場合、IME を有効にして変換モードを韓国語に設定します。ユーザーのコンピュータに韓国語 IME がインストールされていない場合は Flash Player によってエラーがスローされ、try..catch ブロックによってキャッチされます。try..catch ブロックは、前に作成されたテキストフィールドにエラーメッセージを表示します。

特定のテキストフィールドに対する IME の無効化

場合によっては、ユーザーが文字を入力する間、ユーザーの IME を無効にすることもできます。たとえば、数値入力だけを受け入れるテキストフィールドがある場合、IME が起動されてデータ入力操作が遅くなることは望ましくありません。

次の例では、FocusEvent.FOCUS_IN イベントと FocusEvent.FOCUS_OUT イベントを待ち受け、それらのイベントに応じてユーザーの IME を無効にする方法を示しています。

```
var phoneTxt:TextField = new TextField();
var nameTxt:TextField = new TextField();

phoneTxt.type = TextFieldType.INPUT;
phoneTxt.addEventListener(FocusEvent.FOCUS_IN, focusInHandler);
phoneTxt.addEventListener(FocusEvent.FOCUS_OUT, focusOutHandler);
phoneTxt.restrict = "0-9";
phoneTxt.width = 100;
phoneTxt.height = 18;
phoneTxt.background = true;
phoneTxt.border = true;
addChild(phoneTxt);

nameField.type = TextFieldType.INPUT;
nameField.x = 120;
nameField.width = 100;
nameField.height = 18;
nameField.background = true;
nameField.border = true;
addChild(nameField);

function focusInHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = false;
  }
}
function focusOutHandler(event:FocusEvent):void
{
  if (Capabilities.hasIME)
  {
    IME.enabled = true;
  }
}
```

この例では、phoneTxt と nameTxt という 2 つの入力テキストフィールドを作成した後、2 つのイベントリスナーを phoneTxt テキストフィールドに追加します。ユーザーが phoneTxt テキストフィールドにフォーカスを設定すると、FocusEvent.FOCUS_IN イベントが送出され、IME は無効になります。phoneTxt テキストフィールドがフォーカスを失うと、FocusEvent.FOCUS_OUT イベントが送出され、IME が再び有効になります。

IME 入力イベントの待ち受け

IME 入力イベントは、入力ストリングが設定されようとしているときに送出されます。たとえば、ユーザーが IME を有効にしてあり、それをアクティブにして日本語でストリングを入力する場合、ユーザーが入力ストリングを選択すると同時に IMEEvent.IME_COMPOSITION イベントが送出されます。IMEEvent.IME_COMPOSITION イベントを待ち受けるためには、次の例に示すように、System クラス (flash.system.System.ime.addEventListener(...)) の静的プロパティ ime にイベントリスナーを追加する必要があります。

```
var inputTxt:TextField;
var outputTxt:TextField;

inputTxt = new TextField();
inputTxt.type = TextFieldType.INPUT;
inputTxt.width = 200;
inputTxt.height = 18;
inputTxt.border = true;
inputTxt.background = true;
addChild(inputTxt);

outputTxt = new TextField();
outputTxt.autoSize = TextFieldAutoSize.LEFT;
outputTxt.y = 20;
addChild(outputTxt);

if (Capabilities.hasIME)
{
  IME.enabled = true;
  try
  {
    IME.conversionMode = IMEConversionMode.JAPANESE_HIRAGANA;
  }
  catch (error:Error)
  {
    outputTxt.text = "Unable to change IME.";
  }
  System.ime.addEventListener(IMEEvent.IME_COMPOSITION, imeCompositionHandler);
}
else
{
```

```
    outputTxt.text = "Please install IME and try again.";
}

function imeCompositionHandler(event:IMEEvent):void
{
    outputTxt.text = "you typed: " + event.text;
}
```

上記のコードは 2 つのテキストフィールドを作成し、表示リストに追加します。最初のテキストフィールド、inputTxt は、ユーザーが日本語テキストを入力できる入力テキストフィールドです。2 番目のテキストフィールド、outputTxt は、ユーザーに対するエラーメッセージを表示するか、ユーザーが inputTxt テキストフィールドに入力した日本語ストリングをエコーするための動的テキストフィールドです。

例 : システム機能の検出

この CapabilitiesExplorer の例は、flash.system.Capabilities クラスを使用してユーザーの Flash Player がどのような機能をサポートしているかを判別する方法を示しています。この例から、次の手法を学習できます。

- Capabilities クラスを使用してユーザーの Flash Player がサポートする機能を検出する方法
- ExternalInterface クラスを使用してユーザーのブラウザがサポートしているブラウザ設定を検出する方法

CapabilitiesExplorer アプリケーションのファイルは、Samples/CapabilitiesExplorer フォルダにあります。このアプリケーションは、次のファイルで構成されています。

ファイル	説明
CapabilitiesExplorer.mxml	MXML で記述された Flex 用アプリケーションのユーザーインターフェイス
com/example/programmingas3/capabilities/CapabilitiesGrabber.as	配列へのシステム Capabilities の追加、項目のソート、ExternalInterface クラスを使用したブラウザ機能の取得など、アプリケーションの主要な機能を提供するクラス。
capabilities.html	External API との通信に必要な JavaScript を格納する HTML コンテナ。

CapabilitiesExplorer の概要

CapabilitiesExplorer.mxml ファイルは、CapabilitiesExplorer アプリケーションのユーザーインターフェイスを設定することに責任を負います。ユーザーの Flash Player 機能は、ステージ上の DataGrid コンポーネントインスタンス内に表示されます。また、ユーザーが HTML コンテナからアプリケーションを実行している場合、しかも External API が使用可能な場合は、ユーザーのブラウザ機能も表示されます。

メインアプリケーションファイルの creationComplete イベントが送出されると、initApp() メソッドが呼び出されます。initApp() メソッドは com.example.programmingas3.capabilities.CapabilitiesGrabber クラスの中から getCapabilities() メソッドを呼び出します。initApp() メソッドのコードは次のとおりです。

```
private function initApp():void
{
    var dp:Array = CapabilitiesGrabber.getCapabilities();
    capabilitiesGrid.dataProvider = dp;
}
```

CapabilitiesGrabber.getCapabilities() メソッドは、Flash Player 機能およびブラウザ機能のソートされた配列を返し、その後、ステージ上の capabilitiesGrid DataGrid コンポーネントインスタンスの dataProvider プロパティに設定されます。

CapabilitiesGrabber クラスの概要

CapabilitiesGrabber クラスの静的 getCapabilities() メソッドは、flash.system.Capabilities クラスからの各プロパティを配列 (capDP) に追加します。その後、CapabilitiesGrabber クラス内の静的 getBrowserObjects() メソッドを呼び出します。getBrowserObjects() メソッドは、External API を使用して、ブラウザのナビゲータオブジェクトをループ処理します。このオブジェクトには、ブラウザの機能が含まれます。getCapabilities() メソッドは、次のとおりです。

```
public static function getCapabilities():Array
{
    var capDP:Array = new Array();
    capDP.push({name:"Capabilities.avHardwareDisable",
    value:Capabilities.avHardwareDisable});
    capDP.push({name:"Capabilities.hasAccessibility",
    value:Capabilities.hasAccessibility});
    capDP.push({name:"Capabilities.hasAudio", value:Capabilities.hasAudio});
    ...
    capDP.push({name:"Capabilities.version", value:Capabilities.version});
    var navArr:Array = CapabilitiesGrabber.getBrowserObjects();
    if (navArr.length > 0)
    {
        capDP = capDP.concat(navArr);
```

```
    }
    capDP.sortOn("name", Array.CASEINSENSITIVE);
    return capDP;
}
```

getBrowserObjects() メソッドは、ブラウザのナビゲータオブジェクトに含まれる各プロパティの配列を返します。その配列の長さが 1 項目以上である場合、ブラウザ機能の配列 (navArr) が Flash Player 機能の配列 (capDP) に付加され、配列全体がアルファベット順でソートされます。最後に、ソートされた配列がメインアプリケーションファイルへ返され、メインアプリケーションファイルがデータグリッドにデータを入れます。getBrowserObjects() メソッドのコードは次のとおりです。

```
private static function getBrowserObjects():Array
{
    var itemArr:Array = new Array();
    var itemVars:URLVariables;
    if (ExternalInterface.available)
    {
        try
        {
            var tempStr:String = ExternalInterface.call("JS_getBrowserObjects");
            itemVars = new URLVariables(tempStr);
            for (var i:String in itemVars)
            {
                itemArr.push({name:i, value:itemVars[i]});
            }
        }
        catch (error:SecurityError)
        {
            // 無視する
        }
    }
    return itemArr;
}
```

現在のユーザー環境で External API が使用可能な場合、Flash Player は JavaScript の JS_getBrowserObjects() メソッドを呼び出します。このメソッドはブラウザのナビゲータオブジェクトをループ処理し、URL エンコードされた値のストリングを ActionScript に返します。このストリングは、その後、URLVariables オブジェクト (itemVars) に変換され、itemArr 配列に追加されます。その配列が、呼び出し元のスクリプトへ返されます。

JavaScript との通信

CapabilitiesExplorer アプリケーションを構築する最後の操作は、ブラウザのナビゲータオブジェクト内の各項目をループ処理して名前と値のペアを一時配列に追加するために必要な JavaScript を作成することです。container.html ファイルに入っている JavaScript `JS_getBrowserObjects()` メソッドのコードは、次のとおりです。

```
<script language="JavaScript">
    function JS_getBrowserObjects()
    {
        // ブラウザの各項目を保持する配列を作成する
        var tempArr = new Array();

        // ブラウザのナビゲータオブジェクトに含まれている各項目をループ処理する
        for (var name in navigator)
        {
            var value = navigator[name];

            // 現在の値がストリングオブジェクトまたは Boolean オブジェクトである場合は、それを
            // 配列に追加し、それ以外の場合は項目を無視する
            switch (typeof(value))
            {
                case "string":
                case "boolean":

                    // 配列に追加される一時ストリングを作成する
                    // 必ず JavaScript の escape() 関数を使用して値を
                    // URL エンコードする
                    var tempStr = "navigator."+ name + "=" + escape(value);
                    // URL エンコードした名前と値のペアを配列にプッシュする
                    tempArr.push(tempStr);
                    break;
            }
        }
        // ブラウザの画面オブジェクトに入っている各項目をループ処理する
        for (var name in screen)
        {
            var value = screen[name];

            // 現在の値が数値なら、それを配列に追加し、数値でなければ、
            // その項目を無視する
            switch (typeof(value))
            {
                case "number":
                    var tempStr = "screen."+ name + "=" + escape(value);
                    tempArr.push(tempStr);
                    break;
            }
        }
        // 配列を、URL エンコードされた名前と値のペアからなるストリングとして返す
```

```
        return tempArr.join("&");
    }
</script>
```

このコードは最初に、ナビゲータオブジェクト内にあるすべての名前と値のペアを保持する一時配列を作成します。次に、ナビゲータオブジェクトが for..in ループでループ処理され、現在の値のデータ型が評価されます。これにより、不要な値が除外されます。このアプリケーションでは、String 値または Boolean 値だけが注目され、それ以外のデータ型 (関数または配列など) は無視されます。ナビゲータオブジェクト内のそれぞれの String 値または Boolean 値は、tempArr 配列に付加されます。次に、ブラウザの画面オブジェクトが、for..in ループでループ処理され、それぞれの数値が tempArr 配列に追加されます。最後に、一時配列オブジェクトが、Array.join() メソッドでストリングに変換されます。この配列は、アンパサンド (&) を区切り記号として使用します。これにより、ActionScript は URLVariables クラスを使用してデータを簡単に解析できます。

第 17 章

Flash Player セキュリティ

17

セキュリティは、Adobe、ユーザー、Web サイト所有者、およびコンテンツ開発者にとって大きな問題です。このため、Adobe Flash Player 9 には、ユーザー、Web サイト所有者、およびコンテンツ開発者を保護するための一連のセキュリティルールとコントロールが用意されています。この章では、Flash アプリケーションを開発する場合の Flash Player セキュリティモデルの操作方法について解説します。この章で取り上げる SWF ファイルは、特に説明がない限り、ActionScript 3.0 でパブリッシュされ、したがって Flash Player 9 以降で実行されるものとします。

この章の目的はセキュリティの概要を説明することで、あらゆる実装の詳細、使用方法、特定の API を使用する際の問題などについて包括的な説明をするものではありません。Flash Player セキュリティの概念の詳細については、Flash Player 9 セキュリティに関するホワイトペーパー (www.adobe.com/go/fp9_0_security) を参照してください。

目次

Flash Player セキュリティの概要 450
アクセス許可管理の概要 452
セキュリティサンドボックス 461
ネットワーク API の制限 464
フルスクリーンモードのセキュリティ 465
コンテンツのロード 467
クロススクリプト 470
ロードされたメディアへのデータとしてのアクセス 474
データのロード 477
セキュリティドメインにインポートされたSWF ファイルからの埋め込みコンテンツのロード . . 479
古いコンテンツの操作 480
LocalConnection 許可の設定 481
ホスト Web ページのスクリプトへのアクセス制御 481
共有オブジェクト 483
カメラ、マイク、クリップボード、マウス、キーボードアクセス 484

Flash Player セキュリティの概要

Flash Player セキュリティの大部分は、ロードされた SWF ファイル、メディア、およびその他のアセットの元のドメインに基づいています。www.example.com などの特定のインターネットドメインの SWF ファイルは、常にそのドメインのすべてのデータにアクセスできます。これらのアセットは " セキュリティサンドボックス " と呼ばれる同じセキュリティグループに配置されます。(詳細については、461 ページの「セキュリティサンドボックス」を参照してください。)

たとえば、SWF ファイルは、ドメイン内の SWF ファイル、ビッドマップ、オーディオ、テキストファイル、およびその他のアセットをダウンロードできます。また、同じドメインの 2 つの SWF 間のクロススクリプトも、両方のファイルが ActionScript 3.0 を使用して記述されていれば可能です。" クロススクリプト " は、SWF ファイルが ActionScript を使用して別の SWF ファイルのプロパティ、メソッド、およびオブジェクトにアクセスする機能です。しかし、ActionScript 3.0 で記述された SWF ファイルと旧バージョンの ActionScript で記述された SWF ファイルではクロススクリプトはサポートされません。この場合は、LocalConnection クラスを使用して通信できます。詳細については、470 ページの「クロススクリプト」を参照してください。

デフォルトでは、次の基本的なセキュリティルールが常に適用されます。

- 同じセキュリティサンドボックス内のリソースは、常に互いにアクセスできます。
- リモートサンドボックス内の SWF ファイルは、ローカルファイルおよびデータにはアクセスできません。

Flash Player では、以下を個別のドメインとみなし、それぞれに対して個別のセキュリティサンドボックスを設定します。

- `http://example.com`
- `http://www.example.com`
- `http://store.example.com`
- `https://www.example.com`
- `http://192.0.34.166`

http://example.com などの名前付きドメインが http://192.0.34.166 などの特定の IP アドレスに対応している場合も、Flash Player は両方に対して別々のセキュリティサンドボックスを設定します。

開発者が SWF ファイルが他のファイルのサンドボックスのアセットへアクセスを許可するために使用できる基本的な方法には、次の 2 つがあります。

- `Security.allowDomain()` メソッド (460 ページの「作成者 (開発者) の管理」参照)
- クロスドメインポリシーファイル (456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」参照)

デフォルトでは、SWF ファイルが他のドメインの ActionScript 3.0 SWF ファイルをクロススクリプトし、他のドメインのデータをロードする機能は使用できません。この機能は、ロードされた SWF ファイルの `Security.allowDomain()` メソッドへの呼び出しを使用して許可できます。詳細については、470 ページの「クロススクリプト」を参照してください。

Flash Player セキュリティモデルでは、コンテンツのロードとデータへのアクセスまたはロードは区別されています。

- コンテンツのロード – " コンテンツ " はメディアとして定義され、これには、Flash Player が表示できるビジュアルメディア、オーディオファイル、ビデオファイル、または表示されるメディアが入っている SWF ファイルが含まれます。" データ " は、ActionScript コードからのみアクセスできるものとして定義されます。コンテンツは、Loader クラス、Sound クラス、NetStream クラスなどのクラスを使用してロードできます。
- データまたはロードデータとしてのコンテンツへのアクセス - データには、ロードされたメディアコンテンツからデータを抽出する、または外部ファイル (XML ファイルなど) からデータを直接ロードすることでアクセスできます。ロードしたメディアからデータを抽出するには、Bitmap オブジェクト、`BitmapData.draw()` メソッド、`Sound.id3` プロパティ、または `SoundMixer.computeSpectrum()` メソッドを使用します。データをロードするには、たとえば URLStream、URLLoader、Socket、XMLSocket などのクラスを使用します。

Flash Player セキュリティモデルは、コンテンツのロードとデータへのアクセスのために、さまざまな規則を定義しています。一般に、コンテンツのロードには、データへのアクセスより制限が少なくなっています。

一般に、コンテンツ (SWF ファイル、ビットマップ、MP3 ファイル、およびビデオ) はどこからでもロードできますが、コンテンツがロードする側の SWF ファイルのドメイン以外のドメインにある場合は、別々のセキュリティサンドボックスに分割されます。

コンテンツのロードには、次のような制限があります。

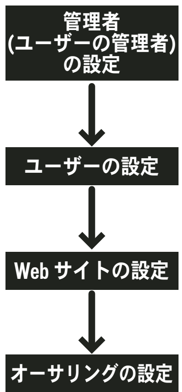
- デフォルトでは、ローカルの SWF ファイル (ユーザーのハードディスクなどの非ネットワークアドレスからロードされた SWF ファイル) は、local-with-filesystem システムサンドボックスに分類されます。これらのファイルでは、ネットワークからコンテンツをロードできません。詳細については、462 ページの「ローカルサンドボックス」を参照してください。
- RTMP (Real-Time Messaging Protocol) サーバーでは、コンテンツへのアクセスを制限することができます。詳細については、470 ページの「RTMP サーバーを使用して配信されるコンテンツ」を参照してください。

ロードされたメディアがイメージ、オーディオ、またはビデオの場合、セキュリティサンドボックス外部にある SWF ファイルのドメインがメディアの元のドメインにあるクロスドメインポリシーファイルに含まれていなければ、ピクセルデータやサウンドデータなどのメディアのデータにその SWF ファイルからアクセスすることができません。詳細については、474 ページの「ロードされたメディアへのデータとしてのアクセス」を参照してください。

その他のロードされたデータの形態としては、テキストファイルまたは XML ファイルがあり、これらは URLLoader オブジェクトを使用してロードされます。この場合も、別のセキュリティサンドボックスからのデータにアクセスするには、元のドメインでのクロスドメインポリシーファイルによって許可が与えられている必要があります。詳細については、477 ページの「URLLoader と URLStream の使用」を参照してください。

アクセス許可管理の概要

Flash Player クライアントランタイムのセキュリティモデルは、SWF ファイル、ローカルデータ、インターネット URL などのオブジェクトであるリソースに基づいて設計されています。ステークホルダーは、これらのリソースの所有者または使用者です。ステークホルダーはそのリソースを管理 (セキュリティ設定) できます。また、各リソースにはステークホルダーが 4 つあります。次の図に示すように、Flash Player ではこれらの管理の権限階層が厳密に適用されます。



セキュリティ管理の階層

これは、たとえば、管理者がリソースへのアクセスを制限した場合、他のステークホルダーはこの制限をオーバーライドすることはできないことを示します。

管理者、ユーザー、および Web サイトの管理については、以降のセクションで詳しく説明します。この章では、作成者 (開発者) 設定についても説明します。

管理ユーザーの管理

コンピュータの管理ユーザー (管理者権限でログインするユーザー) は、そのコンピュータの全ユーザーに有効な Flash Player セキュリティ設定を適用できます。家庭用コンピュータなどのエンタープライズ環境以外では、通常は管理者権限も持つユーザーは 1 人です。エンタープライズ環境でも、個々のユーザーが管理者権限を持つ場合があります。

管理ユーザーの管理には次の 2 種類があります。

- mms.cfg ファイル
- Global Flash Player Trust ディレクトリ

mms.cfg ファイル

Mac OS X システムでは、mms.cfg ファイルの場所は /Library/Application Support/Macromedia/mms.cfg です。Microsoft Windows システムでは、ファイルはシステムディレクトリの Macromedia Flash Player フォルダにあります (たとえば、Windows XP のデフォルトのインストールでは C:¥windows¥system32¥macromed¥flash¥mms.cfg です)。

Flash Player が起動すると、このファイルからセキュリティ設定を読み取り、機能を制限するために使用します。

mms.cfg ファイルには、管理者が次のタスクを実行するために使用する設定が含まれます。

- **データロード ー** ローカル SWF ファイルの読み取りを制限し、ファイルのダウンロードとアップロードを無効にし、永続共有オブジェクトの記憶領域制限を設定します。
- **プライバシーコントロール ー** マイクおよびカメラへのアクセスを無効にし、SWF ファイルのウィンドウなしコンテンツを再生しないようにし、ブラウザウィンドウに表示される URL と一致しないドメイン内の SWF ファイルが永続共有オブジェクトにアクセスできないようにします。
- **Flash Player のアップデート ー** 最新バージョンの Flash Player をチェックする間隔を設定し、Flash Player の更新情報をチェックする URL を指定し、最新バージョンの Flash Player をダウンロードする URL を指定し、Flash Player の自動アップデートをすべて無効にします。
- **古いファイルのサポートー**古いバージョンの SWF ファイルを local-trusted サンドボックス内に配置するかどうかを指定します。
- **ローカルファイルのセキュリティー**ローカルファイルを local-trusted サンドボックス内に配置するかどうかを指定します。
- **フルスクリーンモード ー** フルスクリーンモードを無効にします。

SWF ファイルは、`Capabilities.avHardwareDisable` および `Capabilities.localFileReadDisable` プロパティを呼び出すことにより、無効にされている機能に関する一部の情報にアクセスできます。しかし、mms.cfg ファイル内の設定のほとんどは、ActionScript からは照会できません。

コンピュータのセキュリティとプライバシーに関するセキュリティを、アプリケーションから独立して適用するには、システム管理者のみが mms.cfg ファイルを変更する必要があります。mms.cfg ファイルは、アプリケーションインストーラが使用するためのものではありません。管理権限を持って実行しているインストーラは mms.cfg ファイルの内容を変更できますが、Adobe では、そのような使用状況はユーザーの信頼を裏切るものであると考えており、インストーラの作成者に mms.cfg ファイルを決して変更しないように強く要請しています。

Global Flash Player Trust ディレクトリ

管理ユーザーおよびインストーラアプリケーションは、指定されたローカル SWF ファイルを信頼できるファイルとして登録できます。これらの SWF ファイルは、local-trusted サンドボックスに割り当てられます。割り当てられたファイルは、他の SWF ファイルと自由に通信し、リモートまたはローカルを問わずどこからでもデータをロードできます。ファイルは、Global Flash Player Trust ディレクトリで信頼できるファイルとして指定されます。このディレクトリは mms.cfg ファイルと同じディレクトリ内にあり、現在のユーザーに固有の次の場所にあります。

- Windows: system¥Macromed¥Flash¥FlashPlayerTrust

  (例 : C:¥windows¥system32¥Macromed¥Flash¥FlashPlayerTrust)
- Mac: app support/Macromedia/FlashPlayerTrust

  (例 : /Library/Application Support/Macromedia/FlashPlayerTrust)

Flash Player Trust ディレクトリには、任意の数のテキストファイルを格納でき、それぞれのファイルは、信頼できるパスを 1 行に 1 パスずつ示したリストです。1 つのパスを 1 つの SWF ファイル、HTML ファイル、またはディレクトリとすることもできます。コメント行は、# 記号で始まります。たとえば、次のテキストが入っている Flash Player トラスト構成ファイルは、指定されたディレクトリとそのすべてのサブディレクトリにあるすべてのファイルに、信頼されている状態を付与します。

```
# Trust files in the following directories:
C:¥Documents and Settings¥All Users¥Documents¥SampleApp
```

トラスト構成ファイル内に示されるパスは、常にローカルパスであるか SMB ネットワークパスであることが必要です。トラスト構成ファイル内の HTTP パスは、すべて無視されます。信頼できるのは、ローカルファイルだけです。

競合を避けるため、それぞれのトラスト構成ファイルにはインストールするアプリケーションに対応したファイル名を付け、.cfg のファイル拡張子を使用してください。

ローカルで実行する SWF ファイルをインストーラアプリケーションから配布する開発者は、インストーラアプリケーションに Global Flash Player Trust ディレクトリへ構成ファイルを追加させて、配布するファイルに対するすべての権限を付与することができます。インストーラアプリケーションは、管理者権限のあるユーザーが実行する必要があります。mms.cfg ファイルと異なり、Global Flash Player Trust ディレクトリはトラスト許可を付与するインストーラアプリケーションのために組み込まれています。管理ユーザーとインストーラアプリケーションは、どちらも Global Flash Player Trust ディレクトリを使用して、信頼できるローカルアプリケーションを指定できます。

個々のユーザー用の Flash Player Trust ディレクトリもあります。次のセクションの「ユーザーの管理」を参照してください。

ユーザーの管理

Flash Player には、権限を設定するための 3 つのユーザーレベルのメカニズム (設定 UI、設定マネージャ、および User Flash Player Trust ディレクトリ) があります。

設定 UI および設定マネージャ

設定 UI は、特定のドメインの設定をすばやくインタラクティブに行うメカニズムです。設定マネージャは、詳細なインターフェイスを提供し、多数またはすべてのドメインのアクセス許可を対象とするグローバルな変更を行う機能を備えています。また、SWF ファイルによって新しいアクセス許可が要求され、セキュリティやプライバシーに関する実行時決定が必要なとき、ユーザーが Flash Player 設定を調整できるダイアログボックスが表示されます。

設定マネージャおよび設定 UI には、次のセキュリティ関連のオプションがあります。

- カメラおよびマイクの設定 ー ユーザーは、コンピュータのカメラおよびマイクへの Flash Player のアクセスを制御できます。ユーザーは、すべてのサイトまたは特定のサイトのアクセスを許可または拒否できます。すべてのサイトまたは特定のサイトの設定を指定しなかった場合、SWF ファイルがカメラまたはマイクにアクセスしようとするとダイアログボックスが表示され、ユーザーは SWF ファイルのデバイスへのアクセスを許可するかどうかを選択できます。使用するカメラまたはマイクを指定することもでき、マイクの感度も設定できます。
- 共有オブジェクトの記憶領域設定 ー ユーザーは、ドメインで永続共有オブジェクトを格納するために使用できるディスク容量を選択できます。任意の数のドメインについてこれらの設定を行うことができ、新しいドメインのデフォルト設定を指定できます。デフォルトのディスク容量は 100 KB です。永続共有オブジェクトの詳細については、『ActionScript 3.0 リファレンスガイド』の SharedObject クラスを参照してください。

メモ	mms.cfg ファイル (452 ページの「管理ユーザーの管理」参照) の中で行った設定は、設定マネージャには反映されません。

設定マネージャの詳細については、http://www.adobe.com/go/settingsmanager を参照してください。

User Flash Player Trust ディレクトリ

ユーザーおよびインストーラアプリケーションは、指定されたローカル SWF ファイルを信頼できるファイルとして登録できます。これらの SWF ファイルは、local-trusted サンドボックスに割り当てられます。割り当てられたファイルは、他の SWF ファイルと自由に通信し、リモートまたはローカルを問わずどこからでもデータをロードできます。ユーザーは、User Flash Player Trust ディレクトリでファイルを信頼できるファイルに指定します。このディレクトリは、Flash 共有オブジェクトの記憶領域と同じディレクトリ内にあり、現在のユーザーに固有の次の場所にあります。

- Windows: app data¥Macromedia¥Flash Player¥#Security¥FlashPlayerTrust

  (例 : C:¥Documents and Settings¥JohnD\Application Data¥Macromedia¥Flash Player¥#Security¥FlashPlayerTrust)
- Mac: app data/Macromedia/Flash Player/#Security/FlashPlayerTrust

  (例 : /Users/JohnD/Library/Preferences/Macromedia/Flash Player/#Security/FlashPlayerTrust)

この設定は、現在のユーザーだけに影響を及ぼし、コンピュータにログインする他のユーザーには影響を及ぼしません。管理者権限のないユーザーがシステムの独自部分にアプリケーションをインストールすると、インストーラは User Flash Player Trust ディレクトリで、アプリケーションをそのユーザーの信頼できるアプリケーションとして登録できます。

ローカルで実行する SWF ファイルをインストーラアプリケーションによって配布する開発者は、インストーラアプリケーションに User Flash Player Trust ディレクトリへ構成ファイルを追加させて、配布するファイルに対するすべての権限を付与することができます。この場合でも、User Flash Player Trust ディレクトリファイルは、管理ユーザーのアクション (インストール) によって開始されるため、管理ユーザーの管理と見なされます。

管理ユーザーまたはインストーラが、コンピュータの全ユーザー用にアプリケーションを登録するために使用する Global Flash Player Trust ディレクトリもあります (452 ページの「管理ユーザーの管理」参照)。

Web サイトの管理 (クロスドメインポリシーファイル)

Web サーバーにあるデータを他のドメインの SWF ファイルから利用できるようにするために、サーバー上にクロスドメインポリシーファイルを作成できます。クロスドメインポリシーファイルは XML ファイルであり、サーバーのデータとドキュメントをどのドメインの SWF ファイルが利用できるかを示すものです。特定のドメインのみを指定することも、すべてのドメインを指定することもできます。サーバーのポリシーファイルに指定されたドメインが提供する SWF ファイルであれば、サーバーのデータやアセットにアクセスすることができます。

クロスドメインポリシーファイルは、次のようなアセットへのアクセスに影響を及ぼします。

- ビットマップ、サウンド、およびビデオのデータ
- XML ファイルとテキストファイルのロード
- ソケットと XML ソケット接続へのアクセス
- 他のセキュリティドメインから、ロードする SWF ファイルのセキュリティドメインへの SWF ファイルのインポート

詳細は、この章の残りの部分で説明します。

ポリシーファイルのシンタックス

次の例は、`*.example.com`, `www.friendOfExample.com` および `192.0.34.166` に置かれている SWF ファイルへのアクセスを許可するポリシーファイルを示しています。

```
<?xml version="1.0"?>
<cross-domain-policy>
   <allow-access-from domain="*.example.com" />
   <allow-access-from domain="www.friendOfExample.com" />
   <allow-access-from domain="192.0.34.166" />
</cross-domain-policy>
```

SWF ファイルが別のドメインのデータにアクセスしようとすると、Flash Player は自動的にそのドメインからポリシーファイルをロードしようとします。データをアクセスしようとしている SWF ファイルのドメインがポリシーファイルに含まれている場合、データへのアクセスが自動的に許可されます。

デフォルトでは、ポリシーファイルの名前は `crossdomain.xml` にし、ファイルはサーバーのルートディレクトリに置く必要があります。しかし、SWF ファイルは、`Security.loadPolicyFile()` メソッドを呼び出して、別の名前または別のディレクトリ内をチェックできます。クロスドメインポリシーファイルは、ロード元のディレクトリとその子ディレクトリにのみ、適用されます。したがって、ルートディレクトリのポリシーファイルはサーバー全体に適用されますが、任意のサブディレクトリからロードされたポリシーファイルは、そのディレクトリとそのサブディレクトリにのみ適用されます。

ポリシーファイルは、そのファイルが存在する特定のサーバーへのアクセスにのみ影響を及ぼします。たとえば、https://www.adobe.com:8080/crossdomain.xml にあるポリシーファイルは、HTTPS 経由でポート 8080 から www.adobe.com に対して行われるデータロード呼び出しにのみ適用されます。

クロスドメインポリシーファイルには、`<cross-domain-policy>` タグが 1 つ含まれています。このタグには、`<allow-access-from>` タグが含まれている場合もあれば含まれていない場合もあります。各 `<allow-access-from>` タグには、`domain` という属性が 1 つ含まれています。この属性は、正確な IP アドレス、正確なドメイン、ワイルドカードドメイン (任意のドメイン) のいずれかを指定します。ワイルドカードドメインを指定するには、アスタリスク (*) 1 つを使用するか、アスタリスクの後に接尾辞を続けます。前者はすべてのドメインとすべての IP アドレスを意味し、後者は指定の接尾辞で終わるドメインのみを意味します。接尾辞はドットで始める必要があります。ただし、この先頭のドットは検索には使用されません。ドットを除いた接尾辞に一致するドメインだけが検索されます。たとえば、foo.com は *.foo.com の一部と見なされます。IP ドメインの指定にはワイルドカードを使用できません。

IP アドレスを指定した場合は、IP シンタックス (http://65.57.83.12/flashmovie.swf など) を使用してその IP アドレスからロードされた SWF ファイルにのみアクセスが付与されます。ドメイン名シンタックスを使用してロードされた SWF ファイルには付与されません。Flash Player は、DNS 解決を行いません。

次のように、すべてのドメインのドキュメントに対してアクセスを許可できます。

```
<?xml version="1.0"?>
<!-- http://www.foo.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

各 `<allow-access-from>` タグには、オプションで `secure` 属性を指定することもできます。この属性は、デフォルトで `true` に設定されています。ポリシーファイルが HTTPS サーバー上に置かれているとき、非 HTTPS サーバー上の SWF ファイルによって、HTTPS サーバーからデータをロードするには、この属性を `false` に設定します。

`secure` 属性を `false` に設定すると、HTTPS のセキュリティが影響を受けます。特に、この属性を `false` に設定すると、セキュアなコンテンツがスヌープ攻撃やスプーフィング攻撃にさらされる可能性があります。`secure` 属性を `false` に設定しないことを強くお勧めします。

ロードするデータが HTTPS サーバー上にあり、それをロードする SWF ファイルが HTTP サーバー上にある場合は、セキュアなデータのすべてのコピーを HTTPS の保護下に置いておくことができるよう、ロードする SWF ファイルを HTTPS サーバーへ移動してください。しかし、ロードする SWF ファイルを HTTP サーバー上に置いておく必要があると判断した場合は、次のコードに示すように、secure="false" 属性を <allow-access-from> タグに追加します。

```
<allow-access-from domain="www.example.com" secure="false" />
```

ポリシーファイルに <allow-access-from> タグが 1 つも含まれていない場合は、サーバー上にポリシーファイルがないのと同じことになります。

ソケットポリシーファイル

ActionScript オブジェクトは、2 種類のサーバー接続 (ドキュメントベースのサーバー接続とソケット接続) をインスタンス化します。Loader、Sound、URLLoader、URLStream のような ActionScript オブジェクトは、ドキュメントベースのサーバー接続をインスタンス化し、URL からファイルをロードします。ActionScript の Socket オブジェクトと XMLSocket オブジェクトはソケット接続を作成し、ロードされたドキュメントでなくストリーミングデータを処理します。Flash Player は 2 種類のポリシーファイル (ドキュメントベースのポリシーファイルとソケットポリシーファイル) をサポートします。ドキュメントベースの接続にはドキュメントベースのポリシーファイルが必要であり、ソケット接続にはソケットポリシーファイルが必要です。

Flash Player では、試みられている接続で使用するプロトコルと同じ種類のプロトコルを使用してポリシーファイルが伝送される必要があります。たとえば、ポリシーファイルを HTTP サーバーに置いた場合、他のドメインの SWF ファイルは、その HTTP サーバーからデータをロードできます。しかし、同じサーバーでソケットポリシーファイルを提供しなければ、他のドメインの SWF ファイルはそのサーバーにソケットレベルで接続できなくなります。ソケットポリシーファイルを取得する手段は、接続の手段に一致する必要があります。

ソケットサーバーで提供するポリシーファイルのシンタックスは、他のポリシーファイルとほぼ同じですが、アクセスを許可するポートも指定する必要がある点が異なります。ポリシーファイルが 1024 未満のポート番号から提供される場合、任意のポートへのアクセスが許可されます。ポリシーファイルがポート 1024 以上のポートから提供される場合、1024 以上のポートへのアクセスのみが許可されます。許可するポートは、<allow-access-from> タグの to-ports 属性で指定します。指定できる値は、単一のポート番号、ポート範囲、およびワイルドカードです。

次に、XMLSocket ポリシーファイルの例を示します。

```
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example2.com" to-ports="516-523" />
   <allow-access-from domain="www.example2.com" to-ports="507,516-523" />
   <allow-access-from domain="www.example3.com" to-ports="*" />
</cross-domain-policy>
```

ポリシーファイルが最初に Flash Player 6 に導入されたとき、ソケットポリシーファイルはサポートされていませんでした。ソケットサーバーへの接続は、ソケットサーバーと同じホストのポート 80 上にある HTTP サーバーのクロスドメインポリシーファイルのデフォルトの場所に置かれているポリシーファイルによって許可されていました。既存のサーバー配置を保存できるよう、Flash Player 9 では、この機能が引き続きサポートされています。しかし、現在の Flash Player はデフォルトで、ソケット接続と同じポート上のソケットポリシーファイルを取得します。HTTP ベースのポリシーファイルを使用してソケット接続を許可する場合は、次のようなコードを使用して HTTP ポリシーファイルを明示的に要求する必要があります。

```
Security.loadPolicyFile("http://socketServerHost.com/crossdomain.xml")
```

さらに、ソケット接続を許可するためには、HTTP ポリシーファイルがクロスドメインポリシーファイルのデフォルトの場所からのみ提供され、それ以外の HTTP の場所から提供されない必要があります。HTTP サーバーから取得されたポリシーファイルは、ポート 1024 以上のすべてのポートへのアクセスを暗黙に許可します。HTTP ポリシーファイル内に `to-ports` 属性があっても、それは無視されます。

ソケットポリシーファイルの詳細については、477 ページの「ソケットへの接続」を参照してください。

ポリシーファイルのプリロード

サーバーからのデータのロードまたはソケットへの接続は非同期操作であり、Flash Player は、単にクロスドメインポリシーファイルがダウンロードを終了するのを待ってから、主たる操作を開始します。しかし、イメージからのピクセルデータの抽出や、サウンドからのサンプルデータの抽出は同期操作です。したがって、データを抽出するためには、事前にクロスドメインポリシーファイルをロードしておく必要があります。メディアをロードするときは、クロスドメインポリシーファイルの有無をチェックするよう指定する必要があります。

- `Loader.load()` メソッドを使用するときは、LoaderContext オブジェクトである `context` パラメータの `checkPolicyFile` プロパティを設定します。
- `<img>` タグを使用してテキストフィールドにイメージを埋め込むときは、次のように、`<img>` タグの `checkPolicyFile` 属性を `"true"` に設定します。`<img checkPolicyFile = "true" src = "example.jpg">`。
- `Sound.load()` メソッドを使用するときは、SoundLoaderContext オブジェクトである `context` パラメータの `checkPolicyFile` プロパティを設定します。
- NetStream クラスを使用するときは、NetStream オブジェクトの `checkPolicyFile` プロパティを設定します。

これらのいずれかのパラメータを設定すると、Flash Player は最初に、既にそのドメイン用にダウンロードされたポリシーファイルがあるかどうかをチェックします。次に、`Security.loadPolicyFile()` メソッドへの保留中の呼び出しがスコープ内にあるかどうかを調べ、もしそうであれば、それらの呼び出しを待ちます。その後、サーバー上のデフォルトの場所にあるクロスドメインポリシーファイルを探します。

作成者 (開発者) の管理

セキュリティ権限を付与するために使用される主な ActionScript API は、`Security.allowDomain()` メソッドで、これは、指定されたドメイン内の SWF ファイルに権限を付与します。次の例では、SWF ファイルは www.example.com ドメインに置かれた SWF ファイルへのアクセスを許可します。

```
Security.allowDomain("www.example.com")
```

このメソッドは、次の許可を付与します。

- SWF ファイル間のクロススクリプト (470 ページの「クロススクリプト」参照)
- 表示リストへのアクセス (473 ページの「表示リスト内の移動」参照)
- イベント検出 (473 ページの「イベントのセキュリティ」参照)
- Stage オブジェクトのプロパティおよびメソッドへのフルアクセス (472 ページの「ステージのセキュリティ」参照)

`Security.allowDomain()` メソッドを呼び出す主な目的は、外部ドメインにある SWF ファイルに、`Security.allowDomain()` メソッドを呼び出す SWF ファイルをスクリプトする許可を与えることです。詳細については、470 ページの「クロススクリプト」を参照してください。

`Security.allowDomain()` メソッドにパラメータとして IP アドレスを指定しても、指定された IP アドレスに存在するすべてのアクセス元からのアクセスが許可されるわけではありません。許可されるのは、その IP アドレスに対応するドメイン名ではなく、URL として指定された IP アドレスを含むアクセス元からのアクセスだけです。たとえば、ドメイン名 www.example.com が IP アドレス 192.0.34.166 に対応している場合、`Security.allowDomain("192.0.34.166")` の呼び出しでは www.example.com へのアクセスは許可されません。

ワイルドカード "*" を `Security.allowDomain()` メソッドに渡して、すべてのドメインからのアクセスを許可できます。"*" ワイルドカードは " すべての " ドメインにある SWF ファイルに、呼び出しを行う SWF ファイルをスクリプトする許可を付与するので、使用に注意してください。

ActionScript には、`Security.allowInsecureDomain()` という第 2 の許可 API が組み込まれています。このメソッドは `Security.allowDomain()` メソッドとほとんど同じことをしますが、異なる点は、セキュアな HTTPS 接続によってサービスされた SWF ファイルから呼び出された場合に、HTTP などのセキュアでないプロトコルからサービスされる他の SWF ファイルに、呼び出し側の SWF ファイルへのアクセスを追加的に許可することです。しかし、セキュアなプロトコル (HTTPS) からのファイルと、セキュアでないプロトコル (HTTP など) からのファイルの間でスクリプティングを許可することはセキュリティ上望ましくありません。そのようなことをすると、セキュアなコンテンツがスヌープ攻撃やスプーフィング攻撃にさらされる可能性があります。そのような攻撃が可能になる理由は、次のとおりです。`Security.allowInsecureDomain()` メソッドは、HTTP 接続によってサービスされる SWF ファイルがセキュアな HTTPS データにアクセスすることを許可するので、HTTP サーバーとユーザーの間に割り込んだ攻撃者は、HTTP の SWF ファイルを独自のものに置き換えることにより、HTTPS データにアクセスできます。

セキュリティに関連したもう 1 つの重要なメソッドは、`Security.loadPolicyFile()` です。このメソッドは、Flash Player に標準以外の場所にクロスドメインポリシーファイルがあるかどうかをチェックさせます。詳細については、456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照してください。

セキュリティサンドボックス

クライアントコンピュータは、外部 Web サイトやローカルファイルシステムなどの多数のソースから個々の SWF ファイルを取得できます。Flash Player では、SWF ファイル、および共有オブジェクト、ビットマップ、サウンド、ビデオ、データファイルなどの他のリソースが Flash Player にロードされると、元の場所に基づいてセキュリティサンドボックスに個別に割り当てられます。次のセクションでは、Flash Player で適用される、任意のサンドボックス内の SWF ファイルがアクセスできるものを管理する規則について説明します。

セキュリティサンドボックスの詳細については、Flash Player 9 セキュリティに関するホワイトペーパーを参照してください。

リモートサンドボックス

Flash Player では、インターネットのアセット (SWF ファイルを含む) が Web サイトの元のドメインに対応する個別のサンドボックスに分類されます。デフォルトでは、それらのファイルは、それら自体のサーバーにあるリソースへのアクセスを許可されています。リモート SWF ファイルは、クロスドメインポリシーファイルと `Security.allowDomain()` メソッドなどの明示的な Web サイトおよび作成者の許可によって、他のドメインのその他のデータにアクセスできます。詳細については、456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」および 460 ページの「作成者 (開発者) の管理」を参照してください。

リモート SWF ファイルは、ローカルファイルまたはローカルリソースをロードできません。

詳細については、Flash Player 9 セキュリティに関するホワイトペーパーを参照してください。

ローカルサンドボックス

" ローカルファイル " は、`file:` プロトコルまたは UNC (Universal Naming Convention) パスを使用して参照されるすべてのファイルを表します。ローカル SWF ファイルは、次の 3 つのローカルサンドボックスのいずれかに配置されます。

- local-with-filesystem サンドボックス ー セキュリティ上の目的から、Flash Player はすべてのローカル SWF ファイルおよびアセットをデフォルトで local-with-filesystem サンドボックスに配置します。このサンドボックスから、SWF ファイルは (たとえば、URLLoader クラスを使用して) ローカルファイルを読み取ることができますが、ネットワークとの通信はできません。その結果、ローカルデータがネットワークに漏れ出したり、不正に共有されたりすることがなくなります。
- local-with-networking サンドボックス ーSWF ファイルをコンパイルするときに、その SWF ファイルがローカルファイルとして実行された場合にネットワークにアクセスできるように指定できます (463 ページの「ローカル SWF ファイルのサンドボックスタイプの設定」参照)。これらのファイルは、local-with-networking サンドボックスに配置されます。SWF ファイルが local-with-networking サンドボックスに割り当てられると、ローカルファイルへのアクセス権が失われます。代わりに、SWF ファイルはネットワークのデータにアクセスできるようになります。しかし、local-with-networking SWF ファイルは、そのための許可を得ない限り、クロスドメインポリシーファイルを使用するか、`Security.allowDomain()` メソッドの呼び出しを使用してネットワークからのデータを読み取ることはできません。そのような許可を付与するためには、クロスドメインポリシーファイルは、すべてのドメインへ許可を与える必要があり、それには、`<allow-access-from domain="*"/>` を使用するか、`Security.allowDomain("*")` を使用します。詳細については、456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」、および 460 ページの「作成者 (開発者) の管理」を参照してください。
- local-trusted サンドボックス ー ユーザーまたはインストーラプログラムによって信頼できるとして登録されたローカル SWF ファイルは、local-trusted サンドボックスに配置されます。システム管理者およびユーザーは、セキュリティについての考慮事項に基づいて、ローカル SWF ファイルを local-trusted サンドボックスに再割り当て (移動) またはそのサンドボックスから再割り当て (移動) することもできます (452 ページの「管理ユーザーの管理」および 454 ページの「ユーザーの管理」参照)。local-trusted サンドボックスに割り当てられた SWF ファイルは、他の SWF ファイルと自由に通信したり、どこからでも (リモートでもローカルでも) データをロードしたりすることができます。

local-with-networking サンドボックスと local-with-filesystem サンドボックスの間の通信は、local-with-filesystem サンドボックスとリモートサンドボックスの間の通信と同様に、厳しく禁じられています。そのような通信を可能にする許可を、Flash アプリケーションまたはユーザーや管理者が付与することはできません。

ローカル HTML ファイルとローカル SWF ファイル間のどちらの方向のスクリプティング (たとえば、ExternalInterface クラスを使用したもの) も、関与する HTML ファイルと SWF ファイルの両方が local-trusted サンドボックスに入っている必要があります。その理由は、ブラウザのローカルセキュリティモデルが Flash Player のローカルセキュリティモデルと異なるからです。

local-with-networking サンドボックス内の SWF ファイルは、local-with-filesystem サンドボックス内の SWF ファイルをロードできません。local-with-filesystem サンドボックス内の SWF ファイルは、local-with-networking サンドボックス内の SWF ファイルをロードできません。

ローカル SWF ファイルのサンドボックスタイプの設定

local-with-filesystem サンドボックスまたは local-with-networking サンドボックス用の SWF ファイルは、Flex コンパイラで use-network フラグを設定することで設定できます。詳細については、『Flex アプリケーションの構築と展開』の 197 ページの「アプリケーションコンパイラのオプションについて」を参照してください。

コンピュータのエンドユーザーまたは管理者は、ローカル SWF ファイルが信頼できることを指定し、ローカルおよびネットワークの他のドメインからデータをロードできるようにすることができます。これは、Global Flash Player Trust ディレクトリおよび User Flash Player Trust ディレクトリで指定します。詳細については、452 ページの「管理ユーザーの管理」および454 ページの「ユーザーの管理」を参照してください。

ローカルサンドボックスの詳細については、462 ページの「ローカルサンドボックス」を参照してください。

Security.sandboxType プロパティ

SWF ファイルの作成者は、読み取り専用の静的プロパティ `Security.sandboxType` を使用して、Flash Player で SWF ファイルを割り当てたサンドボックスのタイプを指定できます。Security クラスは、`Security.sandboxType` プロパティに指定できる値を表す次のような定数を含んでいます。

- `Security.REMOTE`ー この SWF ファイルはインターネット URL からのものであり、ドメインベースのサンドボックス規則に従って機能します。
- `Security.LOCAL_WITH_FILE`ー この SWF ファイルはローカルファイルですが、ユーザーから信頼されておらず、ネットワークを指定してパブリッシュされていません。この SWF ファイルはローカルデータソースからの読み取りはできますが、インターネットとの通信はできません。
- `Security.LOCAL_WITH_NETWORK`ー この SWF ファイルはローカルファイルで、ユーザーから信頼されていませんが、ネットワークを指定してパブリッシュされています。この SWF ファイルはインターネットとの通信はできますが、ローカルデータソースからの読み取りはできません。
- `Security.LOCAL_TRUSTED`ー この SWF ファイルはローカルファイルであり、設定マネージャまたは Flash Player トラスト構成ファイルを使用してユーザーから信頼されています。ローカルのデータソースから読み取ることも、インターネットでやり取りすることもできます。

ネットワーク API の制限

SWF ファイルのネットワーク機能へのアクセスは、SWF コンテンツを含む HTML ページの <object> および <embed> タグで allowNetworking パラメータを設定することで制御できます。

allowNetworking に有効な値は、次のとおりです。

- "all" (デフォルト)ー すべてのネットワーク API が SWF でのアクセスを許可されます。
- "internal"ーSWF ファイルは、このセクションの後半で一覧表示されているブラウザナビゲーションまたはブラウザインタラクション API を呼び出せませんが、他のネットワーク API は呼び出せます。
- "none"ーSWF ファイルは、このセクションの後半で一覧表示されているブラウザナビゲーションまたはブラウザインタラクション API を呼び出せません。また、SWF 対 SWF コミュニケーション API (後半で一覧表示されています) は使用できません。

禁止 API が SecurityError 例外をスローする呼び出し。

allowNetworking パラメータを設定するには、次の例に示すように、SWF ファイルの参照を含む HTML ページの <object> と <embed> タグに allowNetworking パラメータを追加して、値を設定します。

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
   swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowNetworking" value="none" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowNetworking="none" bgcolor="#333333"
   width="600" height="400"
   name="test" align="middle" type="application/x-shockwave-flash"
   pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML ページでも、SWF 埋め込みタグを生成するためにスクリプトを使用することもできます。スクリプトは、正しい allowNetworking 設定が挿入されるように変更する必要があります。Flash および Flex Builder で生成された HTML ページで、SWF ファイルに参照を埋め込むために AC_FL_RunContent() 関数を使用する場合は、次のように、allowNetworking パラメータ設定をスクリプトに追加する必要があります。

```
AC_FL_RunContent( ..."allowNetworking", "none", ...)
```

以下の API は、allowNetworking が "internal" に設定されている場合は禁止されます。

- navigateToURL()
- fscommand()
- ExternalInterface.call()

前述の API に加え、以下の API も、allowNetworking が "none" に設定されている場合は禁止されます。

- sendToURL()
- FileReference.download()
- FileReference.upload()
- Loader.load()
- LocalConnection.connect()
- LocalConnection.send()
- NetConnection.connect()
- NetStream.play()
- Security.loadPolicyFile()
- SharedObject.getLocal()
- SharedObject.getRemote()
- Socket.connect()
- Sound.load()
- URLLoader.load()
- URLStream.load()
- XMLSocket.connect()

選択した allowNetworking の設定により、SWF ファイルでネットワーク API を使用できる場合でも、この章に説明してあるセキュリティサンドボックスの制限に基づいた他の制限が適用されることがあります。

allowNetworking が "none" に設定されている場合は、TextField オブジェクト (SecurityError 例外スロー済み) htmlText のプロパティ内の <img> タグにある外部メディアは参照できません。

フルスクリーンモードのセキュリティ

Flash Player の 9.0.27.0 以降のバージョンでは、Flash コンテンツを画面全体で表示できるフルスクリーンモードがサポートされています。フルスクリーンモードを開始するには、ステージの displayState プロパティを StageDisplayState.FULL_SCREEN 定数に設定します。詳細については、173 ページの「フルスクリーンモードの操作」を参照してください。

ブラウザで実行される SWF ファイルには、いくつかのセキュリティ上の注意事項があります。

フルスクリーンモードを有効にするには、次の例に示すように、SWF ファイルの参照を含む HTML ページの <object> および <embed> タグに、allowFullScreen パラメータを追加して、値を "true" に設定します (デフォルト値は "false" です)。

```
<object classid="clsid:d27cdb6e-ae6d-11cf-96b8-444553540000"
 codebase="http://fpdownload.macromedia.com/pub/shockwave/cabs/flash/
   swflash.cab#version=9,0,18,0"
 width="600" height="400" id="test" align="middle">
<param name="allowFullScreen" value="true" />
<param name="movie" value="test.swf" />
<param name="bgcolor" value="#333333" />
<embed src="test.swf" allowFullScreen="true" bgcolor="#333333"
   width="600" height="400"
   name="test" align="middle" type="application/x-shockwave-flash"
   pluginspage="http://www.macromedia.com/go/getflashplayer" />
</object>
```

HTML ページでは SWF 埋め込みタグを生成するためにスクリプトを使用することもできます。スクリプトは、正しい allowFullScreen 設定が挿入されるように変更する必要があります。Flash および Flex Builder で生成された HTML ページで、SWF ファイルに参照を埋め込むために AC_FL_RunContent() 関数を使用する場合は、次のように、allowFullScreen パラメータ設定を追加する必要があります。

```
AC_FL_RunContent( ..."allowFullScreen", "true", ...)
```

フルスクリーンモードを開始する ActionScript は、マウスイベントまたはキーボードイベントに対する応答でのみ呼び出すことができます。他の状況で呼び出された場合、Flash Player により例外がスローされます。

フルスクリーンモードにある間は、テキスト入力フィールドにテキストを入力することはできません。フルスクリーンモードの間は、アプリケーションを通常モードに戻す Esc キーの押下げなどのキーボードショートカットを除き、すべてのキーボード入力およびキーボード関連 ActionScript が無効になります。

コンテンツがフルスクリーンモードになると、フルスクリーンモードを終了して、通常モードに戻る方法を示すメッセージが表示されます。メッセージは数秒間表示されて消えます。

Stage オブジェクトの displayState プロパティを呼び出すと、ステージ所有者と同じセキュリティサンドボックス内にない呼び出し元について例外がスローされます。この場合のステージ所有者とはメイン SWF ファイルです。詳細については、472 ページの「ステージのセキュリティ」を参照してください。

管理者は mms.cfg ファイルで FullScreenDisable = 1 を設定することで、ブラウザで実行される SWF ファイルのフルスクリーンモードを無効にすることができます。詳細については、452 ページの「管理ユーザーの管理」を参照してください。

ブラウザでフルスクリーンモードにするには、SWF ファイルが HTML ページに含まれている必要があります。

スタンドアロンプレイヤーまたはプロジェクタファイルでは、フルスクリーンモードは常に許可されます。

コンテンツのロード

SWF ファイルでは、次のタイプのコンテンツをロードできます。

- SWF ファイル
- イメージ
- サウンド
- ビデオ

SWF ファイルとイメージのロード

SWF ファイルおよびイメージ (JPG、GIF、または PNG ファイル) をロードするには、Loader クラスを使用します。local-with-filesystem サンドボックス以外にある SWF ファイルは、任意のネットワークドメインから SWF ファイルおよびイメージをロードできます。ローカルサンドボックス内にある SWF ファイルだけが、SWF ファイルとイメージをローカルファイルシステムからロードできます。しかし、local-with-networking サンドボックス内のファイルは、local-trusted サンドボックスまたは local-with-networking サンドボックス内のローカル SWF ファイルしかロードできません。local-with-networking サンドボックス内の SWF ファイルは、SWF ファイル以外のローカルコンテンツ (イメージなど) をロードしますが、ロードしたコンテンツの中のデータにはアクセスできません。

信頼できないソース (Loader オブジェクトのルート SWF ファイルのドメイン以外のドメインなど) から SWF ファイルをロードするとき、次のコードに示すように、Loader オブジェクトのマスクを定義して、ロードされたコンテンツ (Loader オブジェクトの子) がそのマスクの外部にあるステージの一部分に描画されないようにすることができます。

```
import flash.display.*;
import flash.net.URLRequest;
var rect:Shape = new Shape();
rect.graphics.beginFill(0xFFFFFF);
rect.graphics.drawRect(0, 0, 100, 100);
addChild(rect);
var ldr:Loader = new Loader();
ldr.mask = rect;
var url:String = "http://www.unknown.example.com/content.swf";
var urlReq:URLRequest = new URLRequest(url);
ldr.load(urlReq);
addChild(ldr);
```

Loader オブジェクトの `load()` メソッドを呼び出す場合、LoaderContext オブジェクトである `context` パラメータを指定できます。LoaderContext クラスには、ロードされたコンテンツの使用方法のコンテキストを定義できる次の 3 つのプロパティが含まれています。

- `checkPolicyFile`—SWF ファイルではなく、イメージファイルをロードする場合にのみ、このプロパティを使用します。Loader オブジェクトを含んでいるファイルのドメイン以外のドメインのイメージファイルに対して、このプロパティを指定します。このプロパティを `true` に設定すると、Loader によってオリジンサーバーでクロスドメインポリシーファイルの有無がチェックされます (456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」参照)。サーバーが Loader ドメインに許可を与えた場合、Loader ドメイン内にある SWF ファイルの ActionScript は、ロードされたイメージ内のデータにアクセスできます。つまり、`Loader.content` プロパティを使用して、ロードされたイメージを表す Bitmap オブジェクトへの参照を取得するか、`BitmapData.draw()` メソッドを使用して、ロードされたイメージのピクセルにアクセスできます。
- `securityDomain`— イメージではなく、SWF ファイルをロードする場合にのみ、このプロパティを使用します。Loader オブジェクトを含むファイルのドメイン以外のドメインの SWF ファイルに対してこのプロパティを指定します。現時点で `securityDomain` プロパティに指定できるのは、`null` (デフォルト) と `SecurityDomain.currentDomain` の 2 つだけです。`SecurityDomain.currentDomain` を指定した場合、ロードされた SWF ファイルを、ロードする側の SWF ファイルのサンドボックスに " インポート " するように要求されます。つまり、ロードされた SWF ファイルがロードする側の SWF ファイルのサーバーからロードされたかのように動作することを意味します。これが許可されるのは、ロードされた SWF ファイルのサーバー上でクロスドメインポリシーファイルがあり、ロードする側の SWF ファイルのドメインによるアクセスを許可している場合だけです。要求されたポリシーファイルが検出された場合、ロードする側とロードされる側は、同じサンドボックス内にあるので、ロードが開始された後、自由にお互いをスクリプトすることができます。サンドボックスのインポートは、ほとんどの場合、通常のロードを実行した後に、ロードされた SWF ファイルに `Security.allowDomain()` メソッドを呼び出させることに置き換えることができます。後者の方法の方が使いやすい場合もあります。なぜなら、ロードされた SWF ファイルはそれ自体の自然なサンドボックス内に入り、それ自体の実際のサーバー上にあるリソースにアクセスできるからです。
- `applicationDomain`—ActionScript 1.0 または 2.0 で記述されたイメージまたは SWF ファイルではなく、ActionScript 3.0 で記述された SWF ファイルをロードする場合にのみ、このプロパティを使用します。ファイルをロードするときに、ファイルを特定のアプリケーションドメインに配置するよう指定できます。デフォルトのままでは、ファイルは新しいアプリケーションドメイン (ロードする側の SWF ファイルのアプリケーションドメインの子) に配置されます。アプリケーションドメインはセキュリティドメインのサブユニットです。したがって、ターゲットアプリケーションドメインを指定できるのは、ロードしようとしている SWF ファイルがユーザー自身のセキュリティドメインのものである場合だけです。なぜなら、それはユーザー自身のサーバーに入っているか、`securityDomain` プロパティを使用してセキュリティドメインへ正しくインポートされた SWF ファイルだからです。アプリケーションドメインを指定し、ロードされた SWF ファイルが別のセキュリティドメインに属している場合、`applicationDomain` で指定したドメインは無視されます。詳細については、436 ページの「ApplicationDomain クラス」を参照してください。

詳細については、181 ページの「LoaderContext クラス」を参照してください。

Loader オブジェクトの重要なプロパティの 1 つに、contentLoaderInfo プロパティがあります。これは LoaderInfo オブジェクトです。他のほとんどのオブジェクトと異なり、LoaderInfo オブジェクトはロードする側の SWF ファイルとロードされたコンテンツの間で共有され、常に双方からアクセスできます。ロードされたコンテンツが SWF ファイルである場合、その SWF ファイルは DisplayObject.loaderInfo プロパティを通じて LoaderInfo オブジェクトにアクセスできます。LoaderInfo オブジェクトには、ロードの進行状況、ロードする側とロードされる側の URL、ロードする側とロードされる側の信頼関係、およびその他の情報が含まれています。詳細については、180 ページの「LoaderInfo クラス」を参照してください。

サウンドとビデオのロード

local-with-filesystem サンドボックス内に入っている以外のすべての SWF ファイルで、Sound.load()、NetConnection.connect()、および NetStream.play() メソッドを使用して、ネットワークからサウンドおよびビデオをロードできます。

ローカル SWF ファイルだけが、ローカルファイルシステムからメディアをロードできます。それらのロードされたファイルのデータにアクセスできるのは、local-with-filesystem サンドボックスまたは local-trusted サンドボックスに入っている SWF ファイルだけです。

それ以外にも、ロードされたメディアのデータへのアクセスには制限があります。詳細については、474 ページの「ロードされたメディアへのデータとしてのアクセス」を参照してください。

テキストフィールド内の <img> タグを使用した SWF ファイルとイメージのロード

SWF ファイルおよびビットマップをテキストフィールドにロードするには、次のコードに示すように、<img> タグを使用します。

```
<img src = 'filename.jpg' id = 'instanceName' >
```

この方法でロードされたコンテンツにアクセスするには、次のコードに示すように、TextField インスタンスの getImageReference() メソッドを使用します。

```
var loadedObject:DisplayObject = myTextField.getImageReference('instanceName');
```

ただし、この方法でロードされた SWF ファイルおよびイメージは元の場所に対応するサンドボックスに配置されます。

テキストフィールド内の <img> タグを使用してイメージファイルをロードする場合、イメージ内のデータへのアクセスは、クロスドメインポリシーファイルによって許可できます。ポリシーファイルの有無をチェックするには、次のコードに示すように、checkPolicyFile 属性を <img> タグに追加します。

```
<img src = 'filename.jpg' checkPolicyFile = 'true' id = 'instanceName' >
```

テキストフィールド内の <img> タグを使用して SWF をロードする場合、Security.allowDomain() メソッドを呼び出すことにより、その SWF ファイルのデータへのアクセスを許可できます。

SWF 内に埋め込んだ Bitmap クラスを使用せず、テキストフィールド内の `<img>` タグを使用して外部ファイルをロードする場合、Loader オブジェクトは TextField オブジェクトの子として自動的に作成され、外部ファイルは、ActionScript 内の Loader オブジェクトを使用してファイルをロードする場合とまったく同じように、Loader の中へロードされます。その場合、`getImageReference()` メソッドは、自動的に作成された Loader を返します。この Loader オブジェクトにアクセスするのに、セキュリティチェックの必要はありません。その理由は、オブジェクトが呼び出し側のコードと同じセキュリティサンドボックス内にあるからです。

しかし、ロードされたメディアにアクセスするために Loader オブジェクトの `content` プロパティを参照する場合は、セキュリティルールが適用されます。コンテンツがイメージである場合は、クロスドメインポリシーファイルを実装する必要があり、コンテンツが SWF ファイルである場合は、その SWF ファイル内のコードで `allowDomain()` メソッドを呼び出す必要があります。

RTMP サーバーを使用して配信されるコンテンツ

Flash Media Server は、RTMP (Real-Time Media Protocol) を使用してデータ、オーディオ、およびビデオを提供します。SWF ファイルは、NetConnection クラスの `connect()` メソッドを使用してこのメディアをロードし、パラメータとして RTMP URL を渡します。Flash Media Server は、要求元ファイルのドメインに基づいて、接続を制限し、コンテンツがダウンロードされないようにすることができます。詳細については、Flash Media Server のマニュアルを参照してください。

RTMP ソースからロードしたメディアの場合、`BitmapData.draw()` および `SoundMixer.computeSpectrum()` メソッドを使用してランタイムグラフィックおよびサウンドデータを抽出することはできません。

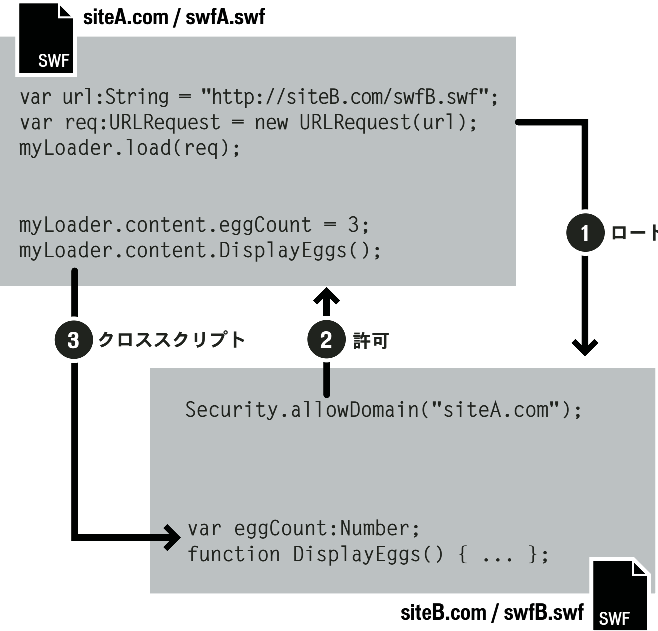
クロススクリプト

ActionScript 3.0 で記述された 2 つの SWF ファイルが同じドメインに所属している場合 (たとえば、一方の SWF ファイルの URL が http://www.example.com/swfA.swf、もう一方の URL が http://www.example.com/swfB.swf)、一方の SWF ファイルでもう一方の SWF ファイル内の変数、オブジェクト、プロパティ、メソッドなどを調査、変更でき、逆の場合も同様のことを実行できます。これを " クロスドメインスクリプト " と呼びます。

AVM1 SWF ファイルと AVM2 SWF ファイル間でのクロススクリプトはサポートされません。AVM1 SWF ファイルは、ActionScript 1.0 または ActionScript 2.0 を使用して作成されたファイルです。AVM1 および AVM2 は、ActionScript 仮想マシンを意味します。しかし、LocalConnection クラスを使用すると、AVM1 と AVM2 間でデータを送信できます。

ActionScript 3.0 で記述された 2 つの SWF ファイルが異なるドメインに所属している場合 (たとえば、http://siteA.com/swfA.swf および http://siteB.com/siteB.swf)、デフォルトでは、swfA.swf で swfB.swf をスクリプトすることも、swfB.swf で swfA.swf をスクリプトすることも許可されません。`Security.allowDomain()` を呼び出すことで、SWF ファイルによって他のドメインの SWF ファイルに許可を与えます。swfB.swf は `Security.allowDomain("siteA.com")` を呼び出すことにより、siteA.com からの SWF ファイルに swfB.swf をスクリプトする許可を与えます。

クロスドメインの場合は、関与する 2 つのドメインを明確にすることが重要です。説明のため、ここでは、クロススクリプトを実行する側を " アクセス元 " (通常、アクセスする SWF) と呼び、他を " アクセス先 " (通常、アクセスされる SWF) と呼びます。次の図に示すように、siteA.swf が siteB.swf をスクリプトする場合、siteA.swf がアクセス元で、siteB.swf がアクセス先です。



`Security.allowDomain()` メソッドを使用して確立されるクロスドメイン許可は非対称です。前の例では、siteA.swf は siteB.swf をスクリプトできますが、siteB.swf は siteA.swf をスクリプトできません。siteA.swf で、siteB.com の SWF ファイルに siteA.swf をスクリプトする許可を与えるための `Security.allowDomain()` メソッドを呼び出していないためです。対称的な許可を設定するには、両方の SWF ファイルで `Security.allowDomain()` メソッドを呼び出すようにする必要があります。

Flash Player では、SWF ファイルを他の SWF ファイルによるクロスドメインスクリプトから保護するだけでなく、HTML ファイルによるクロスドメインスクリプトからも保護します。HTML から SWF へのスクリプトは、ExternalInterface.addCallback() メソッドを通じて確立されたコールバックで発生させることができます。HTML から SWF へのスクリプトがドメインを越える場合、アクセス先 SWF ファイルは、アクセス元が SWF ファイルの場合と同様に、Security.allowDomain() メソッドを呼び出す必要があります。さもなければ、操作は失敗します。詳細については、460 ページの「作成者 (開発者) の管理」を参照してください。

また、Flash Player は SWF から HTML へのスクリプトのセキュリティ管理も行います。詳細については、481 ページの「ホスト Web ページのスクリプトへのアクセス制御」を参照してください。

ステージのセキュリティ

Stage オブジェクトの一部のプロパティとメソッドは、表示リストのスプライトやムービークリップで使用できます。

しかし、Stage オブジェクトには所有者、つまり、ロードされた最初の SWF ファイルがあると考えられています。デフォルトでは、Stage オブジェクトの次のプロパティおよびメソッドは、ステージ所有者と同じセキュリティサンドボックス内の SWF ファイルでのみ使用できます。

プロパティ		メソッド
align	showDefaultContextMenu	addChild()
displayState	stageFocusRect	addChildAt()
frameRate	stageHeight	addEventListener()
height	stageWidth	dispatchEvent()
mouseChildren	tabChildren	hasEventListener()
numChildren	textSnapshot	setChildIndex()
quality	width	willTrigger()
scaleMode		

ステージ所有者のサンドボックス以外のサンドボックス内の SWF ファイルが上記のプロパティおよびメソッドにアクセスするには、ステージ所有者の SWF ファイルは、Security.allowDomain() メソッドを呼び出して、外部サンドボックスのドメインを許可する必要があります。詳細については、460 ページの「作成者 (開発者) の管理」を参照してください。

frameRate プロパティは、特殊な事例です。すべての SWF ファイルは frameRate プロパティを読み取ることができます。しかし、このプロパティを変更できるのは、ステージ所有者のサンドボックスに入っているか、Security.allowDomain() メソッドの呼び出しによって許可を付与された SWF ファイルだけです。

Stage オブジェクトの `removeChildAt()` および `swapChildrenAt()` メソッドについての制限もありますが、それらは他の制限と異なっています。これらのメソッドを呼び出すには、ステージ所有者と同じドメイン内にあることが必要なのではなく、コードが、影響を受ける子オブジェクトの所有者と同じドメイン内にあることが必要です。あるいは、子オブジェクトは `Security.allowDomain()` メソッドを呼び出すことができます。

表示リスト内の移動

SWF ファイルから他のサンドボックスからロードされた表示オブジェクトにアクセスする機能は制限されます。SWF ファイルが別のサンドボックス内にある別の SWF ファイルによって作成された表示オブジェクトにアクセスするには、アクセスされる側の SWF ファイルが `Security.allowDomain()` メソッドを呼び出し、アクセスする側の SWF ファイルのドメインによるアクセスを許可する必要があります。詳細については、460 ページの「作成者 (開発者) の管理」を参照してください。

Loader オブジェクトによってロードされた Bitmap オブジェクトにアクセスするには、イメージファイルのオリジンサーバー上にクロスドメインポリシーファイルが存在する必要があり、そのクロスドメインポリシーファイルが、Bitmap オブジェクトにアクセスを試みている SWF ファイルのドメインに許可を付与する必要があります (456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」参照)。

ロードされたファイル (および Loader オブジェクト) に対応する LoaderInfo オブジェクトは、次の 3 つのプロパティを含んでおり、これらのプロパティはロードされたオブジェクトと Loader オブジェクト間の関係を定義します。`childAllowsParent`、`parentAllowsChild`、および `sameDomain`。

イベントのセキュリティ

表示オブジェクトに関連するイベントには、イベントを送出する表示オブジェクトのサンドボックスに基づくセキュリティアクセスの制限があります。表示リスト内のイベントには、バブリングおよびキャプチャ段階があります (345 ページ、第 13 章の「イベントの処理」参照)。バブリングおよびキャプチャ段階では、イベントは表示リスト内のソース表示オブジェクトから親表示オブジェクトに移行します。親オブジェクトがソースの表示オブジェクトと異なるセキュリティサンドボックス内にある場合、キャプチャおよびバブリング段階はその親オブジェクトの下で停止します。ただし、親オブジェクトの所有者とソースオブジェクトの所有者の間に相互の信頼関係がある場合は除きます。この相互の信頼関係は、次のようにして達成できます。

1. 親オブジェクトを所有する SWF ファイルは、`Security.allowDomain()` メソッドを呼び出して、ソースオブジェクトを所有する SWF ファイルのドメインを信頼する必要があります。
2. ソースオブジェクトを所有する SWF ファイルは、`Security.allowDomain()` メソッドを呼び出して、親オブジェクトを所有する SWF ファイルのドメインを信頼する必要があります。

ロードされたファイル (および Loader オブジェクト) に対応する LoaderInfo オブジェクトは、プロパティ (`childAllowsParent` および `parentAllowsChild`) を含んでおり、これらのプロパティはロードされるオブジェクトと Loader オブジェクト間の関係を定義します。

表示オブジェクト以外のオブジェクトから送出されたイベントの場合、セキュリティチェックやセキュリティ関連の影響はありません。

ロードされたメディアへのデータとしてのアクセス

ロードされたデータにアクセスするには、`BitmapData.draw()` や `SoundMixer.computeSpectrum()` などのメソッドを使用します。デフォルトでは、あるセキュリティサンドボックスの SWF ファイルでは、別のサンドボックス内にあるロードされたメディアによってレンダリングまたは再生されるグラフィックオブジェクトまたはオーディオオブジェクトからピクセルデータやオーディオデータを取得できません。しかし、次のメソッドを使用してこのアクセス許可を付与することができます。

- ロードされた SWF ファイルで、`Security.allowDomain()` メソッドを呼び出して、別のドメインにある SWF ファイルへのデータアクセスを許可します。
- ロードされたイメージ、サウンド、またはビデオの場合は、ロードされたファイルのサーバー上でクロスドメインポリシーファイルを追加します。このポリシーファイルでは、`BitmapData.draw()` または `SoundMixer.computeSpectrum()` メソッドを呼び出してファイルからデータを抽出しようとする SWF ファイルのドメインにアクセスを許可する必要があります。

次のセクションでは、ビットマップ、サウンド、およびビデオのデータにアクセスする方法を詳しく説明します。

ビットマップデータへのアクセス

BitmapData オブジェクトの `draw()` メソッドを使用すると、任意の表示オブジェクトの現在表示されているピクセルを BitmapData オブジェクトに描画できます。描画できるピクセルには、MovieClip オブジェクト、Bitmap オブジェクト、または任意の表示オブジェクトのピクセルがあります。`draw()` メソッドで BitmapData オブジェクトへピクセルを描画するには、次の各条件が満たされる必要があります。

- ロードされたビットマップ以外のソースオブジェクトの場合、そのソースオブジェクトおよび (Sprite または MovieClip オブジェクトの場合には) そのすべての子オブジェクトは、`draw()` メソッドを呼び出すオブジェクトと同じドメインに属しているか、`Security.allowDomain()` メソッドの呼び出しを済ませていて、呼び出し側からアクセスできる SWF ファイル内に存在する必要があります。
- ロードされたビットマップソースオブジェクトの場合、ソースオブジェクトは `draw()` メソッドを呼び出すオブジェクトと同じドメインに属しているか、そのソースサーバーが、呼び出し側ドメインに許可を与えるクロスドメインポリシーファイルを含んでいる必要があります。

これらの条件が満たされない場合は、SecurityError 例外が発生します。

イメージをロードするとき、Loader クラスの load() メソッドを使用すると、LoaderContext オブジェクトである context パラメータを指定できます。LoaderContext オブジェクトの checkPolicyFile プロパティを true に設定した場合、Flash Player は、ロードされるイメージが入っているサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルが存在し、そのファイルがロードを行おうとしている SWF ファイルのドメインを許可している場合、その SWF ファイルは Bitmap オブジェクト内のデータにアクセスできます。それ以外の場合は、アクセスできません。

テキストフィールド内の <img> タグによってロードされるイメージの中で、checkPolicyFile プロパティを指定することもできます。詳細については、469 ページの「テキストフィールド内の <img> タグを使用した SWF ファイルとイメージのロード」を参照してください。

サウンドデータへのアクセス

次のサウンド関連の ActionScript 3.0 API にはセキュリティ制限があります。

- SoundMixer.computeSpectrum() メソッド ー サウンドファイルと同じセキュリティサンドボックス内にある SWF ファイルで常に使用できます。他のサンドボックス内のファイルには、セキュリティチェックがあります。
- SoundMixer.stopAll() メソッド ー サウンドファイルと同じセキュリティサンドボックス内にある SWF ファイルで常に使用できます。他のサンドボックス内のファイルには、セキュリティチェックがあります。
- Sound クラスの id3 プロパティ ー サウンドファイルと同じセキュリティサンドボックス内にある SWF ファイルで常に使用できます。他のサンドボックス内のファイルには、セキュリティチェックがあります。

すべてのサウンドには、2 種類のサンドボックスが関連付けられています。コンテンツサンドボックスと所有者サンドボックスです。

- サウンドのオリジンドメインはコンテンツサンドボックスを決定し、コンテンツサンドボックスは、サウンドの id3 プロパティと SoundMixer.computeSpectrum() メソッドによってサウンドからデータを抽出できるかどうかを決定します。
- サウンドの再生を開始したオブジェクトは、所有者サンドボックスを決定し、所有者サンドボックスは、SoundMixer.stopAll() メソッドを使用してサウンドを停止できるかどうかを決定します。

サウンドをロードするとき、Sound クラスの load() メソッドを使用すると、SoundLoaderContext オブジェクトである context パラメータを指定できます。SoundLoaderContext オブジェクトの checkPolicyFile プロパティを true に設定した場合、Flash Player は、サウンドがロードされるサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルがあり、そのファイルでロードする側の SWF ファイルのドメインが許可されている場合、SWF ファイルは Sound オブジェクトの id プロパティにアクセスできます。それ以外の場合はアクセスできません。また、checkPolicyFile プロパティを設定すると、ロードされたサウンドに対して、SoundMixer.computeSpectrum() メソッドを有効にすることができます。

SoundMixer.areSoundsInaccessible() メソッドを使用すると、SoundMixer.stopAll() メソッドの呼び出しが、呼び出し側からサウンド所有者のサンドボックスにアクセスできないためにすべてのサウンドを停止できないかどうかを知ることができます。

SoundMixer.stopAll() メソッドを呼び出すと、所有者サンドボックスが stopAll() の呼び出し側のそれと同じものであるサウンドが停止します。また、Security.allowDomain() メソッド (これによって stopAll() メソッドを呼び出す SWF ファイルのドメインによるアクセスを許可する) を呼び出した SWF ファイルによって再生が開始されたサウンドも停止します。それ以外のサウンドは停止せず、そのようなサウンドの存在は、SoundMixer.areSoundsInaccessible() メソッドを呼び出すことによって知ることができます。

computeSpectrum() メソッドを呼び出すには、再生中のすべてのサウンドが、このメソッドを呼び出すオブジェクトと同じサンドボックスに属するか、呼び出し側のサンドボックスに許可を与えたソースに属する必要があります。それ以外の場合は、SecurityError 例外が発生します。SWF ファイル内でライブラリの埋め込みサウンドからロードされたサウンドの場合、ロードされた SWF ファイルの Security.allowDomain() メソッドへの呼び出しを使用して許可を付与します。SWF ファイル以外のソースからロードされたサウンド (ロードされた MP3 ファイルまたは Flash ビデオからのサウンド) の場合、ソースサーバー上のクロスドメインポリシーファイルが、ロードされたメディア内のデータへのアクセスを許可します。サウンドが RTMP ストリームからロードされた場合は、computeSpectrum() メソッドを使用できません。

詳細については、460 ページの「作成者 (開発者) の管理」、および 456 ページの「Web サイトの管理 (クロスドメインポリシーファイル)」を参照してください。

ビデオデータへのアクセス

ビデオの現在のフレームのピクセルデータをキャプチャするには、BitmapData.draw() メソッドを使用します。

ビデオには、次の 2 種類があります。

- RTMP ビデオ
- プログレッシブビデオ (RTMP サーバーなしに FLV ファイルからロードされる)

BitmapData.draw() メソッドを使用して RTMP ビデオにアクセスすることはできません。

プログレッシブビデオを source パラメータとして BitmapData.draw() メソッドを呼び出す場合は、BitmapData.draw() の呼び出し側が FLV ファイルと同じサンドボックスに属しているか、FLV ファイルのサーバーが、呼び出しを行う SWF ファイルのドメインに許可を与えるポリシーファイルを持っている必要があります。NetStream オブジェクトの checkPolicyFile プロパティを true に設定することにより、ポリシーファイルのダウンロードを要求できます。

データのロード

SWF ファイルはサーバーから ActionScript 内にデータをロードでき、ActionScript からサーバーにデータを送信できます。データのロードはメディアのロードとは異なる種類の操作です。なぜなら、ロードされた情報は、メディアとして表示されず、ActionScript 内で直接表示されるからです。一般に、SWF ファイルは、それ自体のドメインからデータをロードできます。しかし、他のドメインからデータをロードするためには、通常ではクロスドメインポリシーファイルが必要になります。

URLLoader と URLStream の使用

データは、XML ファイルまたはテキストファイルとしてロードできます。URLLoader および URLStream クラスの `load()` メソッドは、クロスドメインポリシーファイルのアクセス許可によって管理されます。

`load()` メソッドを使用して、このメソッドを呼び出している SWF ファイルのドメイン以外のドメインからコンテンツをロードする場合、Flash Player はロードされるアセットのサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルが存在し、そのファイルでロードする側の SWF ファイルのドメインにアクセスが許可されている場合は、データをロードできます。

ソケットへの接続

デフォルトでは、ソケットおよび XML ソケット接続へのクロスドメインアクセスは無効になっています。また、1024 未満のポート上での、SWF ファイルと同じドメイン内のソケット接続に対するアクセスもデフォルトでは無効になります。これらのポートへのアクセスを許可するには、次のいずれかの場所からクロスドメインポリシーファイルを提供します。

- メインソケット接続と同じポート
- 別のポート
- ソケットサーバーと同じドメイン内のポート 80 上の HTTP サーバー

メインソケット接続と同じポート、または別のポートからクロスドメインポリシーファイルを提供する場合、次の例に示すように、クロスドメインポリシーファイル内で `to-ports` 属性を使用して許可されるポートを列挙します。

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy
  SYSTEM "http://www.adobe.com/xml/dtds/cross-domain-policy.dtd">
<!-- xmlsocket://socks.mysite.com のポリシーファイル -->
<cross-domain-policy>
   <allow-access-from domain="*" to-ports="507" />
   <allow-access-from domain="*.example.com" to-ports="507,516" />
   <allow-access-from domain="*.example.org" to-ports="516-523" />
   <allow-access-from domain="adobe.com" to-ports="507,516-523" />
   <allow-access-from domain="192.0.34.166" to-ports="*" />
</cross-domain-policy>
```

メインソケット接続と同じポートからソケットポリシーファイルを取得するには、単に Socket.connect() または XMLSocket.connect() メソッドを呼び出します。また、指定されたドメインが、呼び出しを行っている SWF ファイルのドメインと同じでなければ、Flash Player が、試みられているメイン接続と同じポートからポリシーファイルを取得しようと自動的に試みます。メイン接続と同じサーバー上で別のポートからソケットポリシーファイルを取得するには、次のように特殊な "xmlsocket" シンタックスを使用して、Security.loadPolicyFile() メソッドを呼び出します。

```
Security.loadPolicyFile("xmlsocket://server.com:2525");
```

Socket.connect() メソッドまたは XMLSocket.connect() メソッドを呼び出す前に、Security.loadPolicyFile() メソッドを呼び出してください。その場合、Flash Player は、ユーザーのメイン接続を許可するかどうかを決める前に、ユーザーのポリシーファイル要求が満たされるのを待ちます。

ソケットサーバーを実装し、ソケットポリシーファイルを提供する必要がある場合は、ポリシーファイルを提供するのに、メイン接続を受け入れるのと同じポートを使用するか、それとも別のポートを使用するかを決めてください。いずれの場合も、サーバーは、クライアントからの最初の送信を待ってから、ポリシーファイルを送信するか、メイン接続を設定するかを決定する必要があります。Flash Player は、ポリシーファイルを要求する場合、接続が確立されると同時に、必ず次のストリングを送信します。

```
<policy-file-request/>
```

ストリングを受信したサーバーは、ポリシーファイルを転送できるようになります。同じ接続をポリシーファイル要求とメイン接続の両方に再使用しないでください。ポリシーファイルの送信後は、接続を閉じてください。そのようにしなかった場合、Flash Player はポリシーファイル接続を閉じてから、再接続してメイン接続を設定します。

詳細については、458 ページの「ソケットポリシーファイル」を参照してください。

データの送信

データ送信が行われるのは、SWF ファイルの ActionScript コードによりサーバーまたはリソースにデータが送信されるときです。データの送信は、ネットワークドメイン SWF ファイルで常に実行できます。ローカル SWF ファイルは、その SWF ファイルが local-trusted サンドボックスまたは local-with-networking サンドボックスに入っている場合にのみ、ネットワークアドレスにデータを送信できます。詳細については、462 ページの「ローカルサンドボックス」を参照してください。

flash.net.sendToURL() 関数を使用すると、URL にデータを送信できます。それ以外のメソッドも、URL に要求を送信します。それらのメソッドとしては、Loader.load() や Sound.load() などのロードメソッドと、URLLoader.load() や URLStream.load() などのデータロードメソッドがあります。

ファイルのアップロードとダウンロード

`FileReference.upload()` メソッドは、ユーザーによって選択されたファイルのリモートサーバーへのアップロードを開始します。`FileReference.upload()` メソッドを呼び出す前に、`FileReference.browse()` または `FileReferenceList.browse()` メソッドを呼び出す必要があります。

`FileReference.download()` メソッドを呼び出すと、ユーザーがリモートサーバーからファイルをダウンロードできるダイアログボックスが開きます。

メモ	サーバーでユーザー認証が必要な場合、ブラウザ内で実行される、つまり、ブラウザプラグインまたは ActiveX コントロールを使用する SWF ファイルでのみ、認証用のユーザー名とパスワードをユーザーが入力できるダイアログボックスを表示できます。ただし、それはダウンロードの場合のみです。Flash Player では、ユーザー認証が必要なサーバーへのアップロードはできません。

呼び出しを行う SWF ファイルが local-with-filesystem サンドボックス内にある場合、アップロードとダウンロードは許可されません。

デフォルトでは、SWF ファイルは、それ自体のサーバー以外のサーバーとの間でアップロードまたはダウンロードを行うことはできません。SWF ファイルが別のサーバーとの間でアップロードまたはダウンロードすることができるのは、そのサーバーがクロスドメインポリシーファイルを提供し、呼び出し側 SWF ファイルのドメインに許可を付与した場合です。

セキュリティドメインにインポートされた SWF ファイルからの埋め込みコンテンツのロード

SWF ファイルをロードするとき、ファイルをロードするために使用する Loader オブジェクトの `load()` メソッドの `content` パラメータを設定できます。このパラメータは、LoaderContext オブジェクトを取得します。この LoaderContext オブジェクトの `securityDomain` プロパティを `Security.currentDomain` に設定すると、Flash Player はロードされた SWF ファイルのサーバー上にクロスドメインポリシーファイルがあるかどうかをチェックします。クロスドメインポリシーファイルが存在し、そのファイルでロードする側の SWF ファイルのドメインが許可されている場合、SWF ファイルを読み込んだメディアとしてロードできます。この方法により、ロードする側のファイルは、SWF ファイルのライブラリにあるオブジェクトにアクセスできるようになります。

SWF ファイルから別のセキュリティサンドボックスにあるロードされた SWF ファイルのクラスにアクセスする代替の方法は、ロードされた SWF ファイルに、`Security.allowDomain()` メソッドを呼び出させ、呼び出す側の SWF ファイルのドメインにアクセスを許可することです。`Security.allowDomain()` メソッドの呼び出しを、ロードされた SWF ファイルのメインクラスのコンストラクタメソッドに追加し、その後、ロードする側の SWF ファイルに、Loader オブジェクトの `contentLoaderInfo` プロパティによって送出された `init` イベントに応答するイベントリスナーを追加させることができます。このイベントが送出される時点で、ロードされた SWF ファイルはコンストラクタメソッド内の `Security.allowDomain()` メソッドの呼び出しを完了しており、ロードされた SWF ファイル内のクラスは、ロードする側の SWF ファイルから使用可能です。ロードする側の SWF ファイルは、`Loader.contentLoaderInfo.applicationDomain.getDefinition()` を呼び出すことにより、ロードされた SWF ファイルからクラスを取得できます。

古いコンテンツの操作

Flash Player 6 では、特定の Flash Player 設定に使用されるドメインは、SWF ファイルのドメインの最後の部分に基づいています。この設定には、カメラとマイクの許可、記憶領域の割り当て、永続共有オブジェクトの記憶領域などがあります。

www.example.com のように SWF ファイルのドメインに 3 つ以上のセグメントが含まれている場合、ドメインの最初のセグメント (www) は削除され、ドメインの残りの部分が使用されます。したがって、Flash Player 6 では、www.example.com と store.example.com はどちらも example.com をこれらの設定のドメインとして使用します。同様に、www.example.co.uk と store.example.co.uk はどちらも example.co.uk をこれらの設定のドメインとして使用します。このため、example1.co.uk と example2.co.uk など、互いに無関係のドメインの SWF ファイルが、同じ共有オブジェクトにアクセスできるという問題が起きる場合があります。

Flash Player 7 以降では、Flash Player 設定は、デフォルトで SWF ファイルの正確なドメインに従って選択されます。たとえば、www.example.com の SWF ファイルは www.example.com の Flash Player 設定を使用し、store.example.com の SWF ファイルはそれとは別の store.example.com の Flash Player 設定を使用します。

ActionScript 3.0 で記述された SWF ファイルでは、`Security.exactSettings` が `true` に設定された場合 (デフォルト)、Flash Player は Player 設定に正確なドメインを使用します。`false` に設定された場合、Flash Player は、Flash Player 6 で使用されていたドメイン設定を使用します。`exactSettings` をデフォルト値から変更する場合は、Flash Player が Player の設定を選択する必要があるイベント (たとえば、カメラやマイクの使用、あるいは永続共有オブジェクトの取得など) が発生する前に変更する必要があります。

バージョン 6 の SWF ファイルをパブリッシュし、そこから永続共有オブジェクトを作成した場合、ActionScript 3.0 を使用する SWF から永続共有オブジェクトを取得するには、`SharedObject.getLocal()` を呼び出す前に、`Security.exactSettings` を `false` に設定する必要があります。

LocalConnection 許可の設定

LocalConnection クラスを使用すると、相互に指示を送ることができる SWF ファイルを作成できます。LocalConnection オブジェクトを使って通信できるのは、同じクライアントコンピュータ上で実行中の SWF ファイルだけです。ただし、これらは異なるアプリケーションで実行されていてもかまいません。たとえば、ブラウザで実行されている SWF ファイルと、プロジェクタで実行されている SWF ファイルとの間で通信することもできます。

どの LocalConnection 通信の場合も、送信側 SWF ファイルと受信側 SWF ファイルがあります。Flash Player では、デフォルトで同じドメイン内の SWF ファイル間で LocalConnection 通信が可能です。別のサンドボックス内の SWF ファイルの場合、受信側は `LocalConnection.allowDomain()` メソッドを使用して送信側に許可を与える必要があります。`LocalConnection.allowDomain()` メソッドにパラメータとして渡すストリングには、正確なドメイン名、IP アドレス、ワイルドカード * を含めることができます。

メモ	`allowDomain()` メソッドは、ActionScript 1.0 および 2.0 の形式から変更されています。これらの以前のバージョンでは、`allowDomain()` は実装するコールバックメソッドでした。ActionScript 3.0 では、`allowDomain()` は、ユーザーが呼び出す LocalConnection クラスのビルトインメソッドです。この変更のため、`allowDomain()` は `Security.allowDomain()` とほとんど同じように動作します。

SWF ファイルでは、LocalConnection クラスの `domain` プロパティを使用してドメインを決定できます。

ホスト Web ページのスクリプトへのアクセス制御

外部スクリプトは、次の ActionScript 3.0 API を使用して実行できます。

- `flash.system.fscommand()` 関数
- `flash.net.navigateToURL()` 関数 (`navigateToURL("javascript: alert('Hello from Flash Player.')"` などのスクリプトステートメントを指定するとき)。
- `flash.net.navigateToURL()` 関数 (`window` パラメータを `"_top"`、`"_self"`、または `"_parent"` に設定したとき)
- `ExternalInterface.call()` メソッド

ローカルで実行する SWF ファイルの場合、SWF ファイルおよび含んでいる Web ページ (存在する場合) が local-trusted セキュリティサンドボックスにある場合のみ、これらのメソッドの呼び出しが成功します。コンテンツが local-with-networking または local-with-filesystem サンドボックスにある場合、これらのメソッドの呼び出しは失敗します。

SWF ファイルをロードする HTML コード内の `AllowScriptAccess` パラメータは、SWF ファイル内から外部スクリプトを実行する機能を制御します。

SWF ファイルをホストする Web ページの HTML コード内でこのパラメータを設定します。このパラメータは、PARAM タグまたは EMBED タグの中で設定します。

AllowScriptAccess パラメータで有効な値は、"always"、"sameDomain"、または "never" の 3 つです。

- AllowScriptAccess が "sameDomain" の場合、外部スクリプトが許可されるのは SWF ファイルと Web ページが同じドメイン内にある場合のみです。これは、AVM2 のコンテンツのデフォルト設定です。
- AllowScriptAccess が "never" の場合、外部スクリプトは常に失敗します。
- AllowScriptAccess が "always" に設定されている場合、外部スクリプトは常に成功します。

HTML ページ内の SWF ファイルで AllowScriptAccess パラメータが指定されていない場合、AVM2 のコンテンツのデフォルトは "sameDomain" です。

次に、HTML ページで AllowScriptAccess タグを設定する例を示します。

```
<object id='MyMovie.swf' classid='clsid:D27CDB6E-AE6D-11cf-96B8-444553540000'
  codebase='http://download.adobe.com/pub/shockwave/cabs/flash/
  swflash.cab#version=9,0,0,0' height='100%' width='100%'>
<param name='AllowScriptAccess' value='never'/>
<param name='src' value=''MyMovie.swf'/>
<embed name='MyMovie.swf' pluginspage='http://www.adobe.com/go/getflashplayer'
  src='MyMovie.swf' height='100%' width='100%' AllowScriptAccess='never'/>
</object>
```

AllowScriptAccess パラメータを使用すると、あるドメインでホストされている SWF ファイルから別のドメインの HTML ページのスクリプトにアクセスすることを禁止できます。別のドメインでホストされているすべての SWF ファイルで AllowScriptAccess="never" を使用すると、HTML ページ内のスクリプトのセキュリティが確保されます。

詳細については、『ActionScript 3.0 リファレンスガイド』で次の項目を参照してください。

- flash.system.fscommand() 関数
- flash.net.navigateToURL() 関数
- ExternalInterface クラスの call() メソッド

共有オブジェクト

Flash Player は、" 共有オブジェクト " (SWF ファイルの外部で存続する ActionScript オブジェクト) をユーザーのローカルファイルシステム上で使用することも、リモート RTMP サーバー上で使用することもできます。共有オブジェクトは、Flash Player 内のその他のメディアと同様に、いくつかのセキュリティサンドボックスに区分されます。しかし、共有オブジェクトのサンドボックスモデルは、少し異なっています。なぜなら、共有オブジェクトはドメインの境界を越えてアクセスできるリソースではないからです。共有オブジェクトは常に、SharedObject クラスのメソッドを呼び出す各 SWF ファイルのドメインに固有の共有オブジェクトストアから取得されます。通常、共有オブジェクトストアは SWF ファイルのドメインよりもさらに細分化されています。デフォルトでは、SWF ファイルごとに、そのオリジン URL 全体に固有の共有オブジェクトストアが使用されます。

SWF ファイルは `SharedObject.getLocal()` および `SharedObject.getRemote()` メソッドの `localPath` パラメータを使用して、その URL の一部分だけに関連付けられた共有オブジェクトストアを使用することができます。この方法により、SWF ファイルは、他の URL からの別の SWF ファイルとの共有を許可できます。`'/'` を `localPath` パラメータとして渡した場合でも、引き続きそれ自体のドメインに固有の共有オブジェクトストアが指定されます。

[Flash Player の設定] ダイアログボックスまたは設定マネージャを使用して、共有オブジェクトへのアクセスを制限できます。デフォルトでは、ドメインごとに最大 100KB のデータサイズまでの共有オブジェクトを作成できます。管理ユーザーおよび一般ユーザーは、ファイルシステムへの書き込み機能に制限を適用することもできます。詳細については、452 ページの「管理ユーザーの管理」および454 ページの「ユーザーの管理」を参照してください。

共有オブジェクトがセキュアであることを指定するには、`SharedObject.getLocal()` メソッドまたは `SharedObject.getRemote()` メソッドの `secure` パラメータに、`true` を指定します。`secure` パラメータについて、次の点に注意してください。

- このパラメータを `true` に設定すると、Flash Player は新しいセキュアな共有オブジェクトを作成するか、既存のセキュアな共有オブジェクトの参照を取得します。このセキュアな共有オブジェクトは、HTTPS 経由で配信される SWF ファイルで、`secure` パラメータを `true` に設定して `SharedObject.getLocal()` を呼び出した場合に限り読み書きできます。
- このパラメータを `false` に設定した場合、Flash Player は新しい共有オブジェクトを作成するか、非 HTTPS 接続で配信された SWF ファイルから読み書きできる既存の共有オブジェクトへの参照を取得します。

呼び出し側の SWF ファイルが HTTPS URL からのものでない場合、`SharedObject.getLocal()` メソッドまたは `SharedObject.getRemote()` メソッドの `secure` パラメータに `true` を指定すると、SecurityError 例外が発生します。

共有オブジェクトストアの選択は、SWF ファイルのオリジン URL が基礎となります。このことは、SWF ファイルが単純な URL から提供されない、読み込みロードとダイナミックロードの 2 つの状況でも当てはまります。読み込みロードとは、`LoaderContext.securityDomain` プロパティを `SecurityDomain.currentDomain` に設定して SWF ファイルをロードする状況のことです。この状況では、ロードされた SWF ファイルは、ロードする側の SWF ファイルのドメインで始まり、その後に実際のオリジン URL を指定した疑似 URL を持ちます。ダイナミックロードとは、`Loader.loadBytes()` メソッドを使用して SWF ファイルをロードすることです。この状況では、ロードされた SWF ファイルは、ロードする側の SWF ファイルの完全 URL で始まり、その後に整数の ID が続く疑似 URL を持ちます。読み込みロードとダイナミックロードのどちらの場合も、`LoaderInfo.url` プロパティを使用して SWF ファイルの疑似 URL を検査できます。疑似 URL は、共有オブジェクトストアを選択する目的で実 URL とまったく同じように扱われます。疑似 URL の一部または全部を使用するには、共有オブジェクトの `localPath` パラメータを指定します。

ユーザーおよび管理者は、" サードパーティ共有オブジェクト " の使用を無効にすることを選択できます。これは、Web ブラウザ内で実行されているすべての SWF ファイルが共有オブジェクトを使用する方法で、その場合、その SWF ファイルのオリジン URL は、ブラウザのアドレスバーに表示される URL と異なるドメインに属しています。ユーザーと管理者は、プライバシーの理由から、クロスドメイントラッキングを避けるために、サードパーティ共有オブジェクトの使用を無効にすることを選択できます。この制限を回避するために、共有オブジェクトを使用するすべての SWF ファイルが、ブラウザのアドレスバーに表示されているのと同じドメインから SWF ファイルが提供される HTML ページ構造内でのみロードされるようにすることもできます。サードパーティ SWF ファイルから共有オブジェクトを使用しようとして、サードパーティ共有オブジェクトの使用が無効にされている場合、`SharedObject.getLocal()` および `SharedObject.getRemote()` メソッドは `null` を返します。詳細については、www.adobe.com/products/flashplayer/articles/thirdpartylso を参照してください。

カメラ、マイク、クリップボード、マウス、キーボードアクセス

SWF ファイルが `Camera.get()` または `Microphone.get()` メソッドを使用してユーザーのカメラまたはマイクにアクセスしようとした場合、Flash Player は [プライバシー] ダイアログボックスを表示します。このダイアログボックスで、ユーザーはカメラおよびマイクへのアクセスを許可または拒否できます。ユーザーおよび管理ユーザーは、カメラへのアクセスをサイトごとに、またはグローバルに無効にすることもでき、その場合は、mms.cfg ファイル内のコントロール、設定 UI、および設定マネージャを使用します (452 ページの「管理ユーザーの管理」、および 454 ページの「ユーザーの管理」参照)。ユーザー制限があると、`Camera.get()` および `Microphone.get()` メソッドは `null` 値を返します。`Capabilities.avHardwareDisable` プロパティを使用すると、カメラおよびマイクが管理上禁止されているか (`true`) または許可されているか (`false`) を決定できます。

`System.setClipboard()` メソッドでは、SWF ファイルでクリップボードの内容をプレーンテキストストリング文字に置き換えることができます。これにより、セキュリティリスクが発生することはありません。クリップボードにカットまたはコピーされるパスワードおよびその他の機密データがもたらすリスクを防ぐために、対応する "getClipboard" (読み取り) メソッドはありません。

Flash アプリケーションが監視できるのは、そのフォーカス内で発生するキーボードイベントとマウスイベントのみです。Flash アプリケーションは、別のアプリケーションでのキーボードイベントまたはマウスイベントを検出できません。

第 18 章

プリント

18

Adobe Flash Player 9 では、OS のプリント機能インターフェイスと通信し、ページをプリントスプーラに送ることができます。Flash Player からスプーラに送信するページには、表示されているコンテンツ、動的なコンテンツ、または画面外にあってユーザーに表示されないコンテンツ (データベースの値や動的なテキストなど) を含めることができます。また、flash.printing.PrintJob クラスのプロパティがユーザーのプリンタ設定に基づいて Flash Player により設定されるため、適切なページ書式を指定できます。

この章では、flash.printing.PrintJob クラスのメソッドとプロパティを使用して、プリントジョブの作成、ユーザーのプリント設定の読み取り、および Flash Player とユーザー OS のフィードバックに基づいたプリントジョブの調整を行う際の指針について詳しく説明します。

目次

ActionScript 3.0 における PrintJob クラスの新機能 488
ページのプリント 488
Flash Player タスクとシステムプリント 489
サイズ、拡大率、用紙の向きの設定. 493
例 : 複数ページのプリント 495
例 : 拡大・縮小、トリミング、および応答 498

ActionScript 3.0 における PrintJob クラスの新機能

ActionScript 3.0 におけるプリント関連の実装では、PrintJob クラスに大きな変更はありません。ただし、次に挙げるとおり、いくつかの重要な差異が生じています。

- `PrintJob.addPage()` メソッドに対する最初の 2 つのパラメータでは、Sprite オブジェクトと Rectangle オブジェクトを指定するようになりました。
- ActionScript 3.0 では、`delete` 演算子を使用してオブジェクト全体を削除 (例 : `delete myPrintJob`) することはできません。ActionScript 2.0 では、オブジェクトまたはオブジェクト内のプロパティを `delete` で削除できました。ActionScript 3.0 の `delete` 演算子は ECMAScript 互換となったため、オブジェクトの動的プロパティを削除する場合にしか `delete` を使用できません。`PrintJob.start()` メソッドを使用すると既存の PrintJob オブジェクトのプロパティをリセットできるので、変数を再利用するために既存の PrintJob オブジェクトを削除する必要はなくなりました。何らかの理由で PrintJob オブジェクトのプロパティを " クリア " する必要がある場合は、`null` を使用してください (例 : `myPrintJob = null`)。
- ActionScript 2.0 におけるグローバル関数の `print()`、`printAsBitmap()`、`printAsBitmapNum()`、`printNum()` は、ActionScript 3.0 ではサポートされていません。代わりに PrintJob クラスと PrintJobOptions クラスを使用してください。
- `PrintJob.addPage()` で特定の条件が発生した場合に Flash Player から例外がスローされるようになったため、コードを記述することで例外をキャッチして処理できます。
- ActionScript 3.0 では、PrintJob オブジェクトが単一フレームに制限されませんが、492 ページの「プリントジョブステートメントのタイミング」に示すように、スクリプトタイムアウトには制限があります。

ページのプリント

ActionScript で Flash Player からページをプリントする際の基本的な手順としては、次に示す 4 つの主要なステートメントを順に使用します。

- `new PrintJob()` : 指定した名前で新しいプリントジョブを作成します。
- `PrintJob.start()` : OS のプリント処理を開始します。これにより、ユーザーに対してプリントダイアログボックスが表示され、また、プリントジョブの読み取り専用プロパティが設定されます。
- `PrintJob.addPage()` : プリントジョブのコンテンツに関する詳細を設定します。これには、Sprite オブジェクト (およびそれに含まれる子) の指定、プリント範囲のサイズ指定、および、イメージをベクターとビットマップのいずれの形式でプリントするかの指定が含まれます。`addPage()` を複数回呼び出せば、複数のページに複数のスプライトをプリントできます。
- `PrintJob.send()` : ページを OS のプリンタに送信します。

したがって、非常に単純なプリントジョブスクリプトは次のようになります (package、import、および class ステートメントはコンパイルのために必要です)。

```
package
{
  import flash.printing.PrintJob;
  import flash.display.Sprite;

  public class BasicPrintExample extends Sprite
  {
    var myPrintJob:PrintJob = new PrintJob();
    var mySprite:Sprite = new Sprite();

    public function BasicPrintExample()
    {
      myPrintJob.start();
      myPrintJob.addPage(mySprite);
      myPrintJob.send();
    }
  }
}
```

メモ	この例は、プリントジョブスクリプトの基本的な要素を示すことを目的としているため、エラー処理は含まれていません。プリントジョブをキャンセルしたユーザーに適切に応答するスクリプトを作成するには、490 ページの「例外および戻り値の処理」を参照してください。

Flash Player タスクとシステムプリント

OS のプリントインターフェイスに対するページの送出処理は Flash Player によって実行されるため、Flash Player が担当するタスクの範囲と、OS 自体のプリントインターフェイスが担当するタスクについて理解しておく必要があります。Flash Player では、プリントジョブの開始、プリンタのページ設定のうち一部の読み込み、プリントジョブのコンテンツの OS への送信、およびユーザーまたはシステムによってプリントジョブがキャンセルされたかどうかの確認を実行します。その他の処理 (プリンタ特有のダイアログボックスの表示、スプールされたプリントジョブのキャンセル、プリンタのステータスに関するレポートなど) は、すべて OS によって実行されます。Flash Player はプリントジョブの開始や書式指定に関する問題に応答しますが、その際に実行できる処理は、OS のプリント機能インターフェイスから取得した特定のプロパティや条件についてレポートを返すことに限られます。アプリケーション開発者は、レポートされたプロパティや条件に応じて何らかの処理を実行するコードを用意する必要があります。

例外および戻り値の処理

ユーザーによってプリントジョブがキャンセルされた場合に対応するため、addPage() および send() を呼び出す前には、PrintJob.start() メソッドが true を返したかどうか確認してください。これらのメソッドに対する呼び出しを次のように if ステートメントで囲むと、続行する前にキャンセルされたかどうかを簡単にチェックできます。

```
if (myPrintJob.start())
{
    // ここに addPage() および send() ステートメントを記述
}
```

PrintJob.start() の戻り値が true の場合は、ユーザーが [印刷] を選択した (または Flash Player が print コマンドを開始した) ことを意味するため、addPage() および send() メソッドの呼び出しができます。

また、プリント処理の管理をしやすくするために、PrintJob.addPage() メソッドに対して Flash Player から例外が送出されるようになりました。これらのエラーをキャッチすることにより、ユーザーに情報と対処方法についての選択肢を示すことができます。PrintJob.addPage() メソッドが失敗した場合は、別の関数を呼び出すか、現在のプリントジョブを停止する方法もあります。例外をキャッチするには、addPage() の呼び出しを次のように try..catch ステートメントで囲みます。例に含まれている [params] の部分には、実際にプリントするコンテンツを指定するパラメータを指定します。

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (e:Error)
    {
        // エラーを処理する
    }
    myPrintJob.send();
}
```

プリントジョブを開始した後は、PrintJob.addPage() でコンテンツを追加し、それによって例外 (ユーザーによるプリントジョブのキャンセルなど) が発生するかどうかを確認します。例外が発生した場合は、catch ステートメントに記述したロジックで、情報と対処方法の選択肢をユーザー (または Flash Player) に示すか、現在のプリントジョブを停止します。ページを正常に追加できた場合は、次に PrintJob.send() でページをプリンタに送信します。

Flash Player がプリントジョブをプリンタに送信する際に問題が発生した場合 (プリンタがオフラインになっていた場合など) も、同様にその例外をキャッチし、情報と対処方法の選択肢をユーザー (または Flash Player) に示すことができます (メッセージテキストと警告を Flash アニメーションで表示することなどが考えられます)。たとえば、次のように if..else ステートメント内でテキストフィールドに新しいテキストを設定するとします。

```
if (myPrintJob.start())
{
    try
    {
        myPrintJob.addPage([params]);
    }
    catch (e:Error)
    {
        // エラーを処理する
    }
    myPrintJob.send();
}
else
{
    myAlert.text = "Print job canceled";
}
```

実際に動作するサンプルについては、498 ページの「例 : 拡大・縮小、トリミング、および応答」を参照してください。

ページプロパティの操作

ユーザーが [印刷] ダイアログボックスで [OK] をクリックし、PrintJob.start() が true を返すと、プリンタの設定で定義されているプロパティにアクセスできるようになります。このプロパティには、用紙の幅と高さ (pageHeight および pageWidth)、用紙の向きが含まれます。これらはプリンタの設定であり、Flash Player の制御下にあるプロパティではありません。したがって設定を変更することはできませんが、これらの情報を使用することで、コンテンツを設定に適した形式に整形してプリンタに送信できます。詳細については、493 ページの「サイズ、拡大率、用紙の向きの設定」を参照してください。

ベクター形式またはビットマップ形式のレンダリング

プリントジョブで各ページをベクターグラフィックとしてスプールするか、ビットマップイメージとしてスプールするかは、手動で設定できます。場合によっては、ベクター形式でプリントするとスプールファイルが小さくなり、画質もビットマップ形式でプリントするより向上することがあります。ただし、コンテンツにビットマップイメージが含まれている場合、アルファ透明度やカラー効果を再現するには、ページをビットマップイメージとしてプリントする必要があります。また、PostScript 非対応のプリンタでは、ベクターグラフィックは自動的にビットマップイメージに変換されます。ビットマップ形式でのプリントを指定するには、次のように、`printAsBitmap` パラメータに `true` を設定した PrintJobOptions オブジェクトを `PrintJob.addPage()` の第 3 パラメータとして渡します。

```
var options:PrintJobOptions = new PrintJobOptions();
options.printAsBitmap = true;
myPrintJob.addPage(mySprite, null, options);
```

第 3 パラメータの値を指定しない場合、そのプリントジョブではベクター形式でのプリントがデフォルトで実行されます。

メモ	ビットマップ形式でのプリントを指定する必要がある場合に、`printArea` (第 2 パラメータ) を指定しないようにするには、`printArea` として `null` を渡します。

プリントジョブステートメントのタイミング

ActionScript 3.0 では、ActionScript の以前のバージョンと同様に、PrintJob オブジェクトが単一フレームに制限されません。ただし、[印刷] ダイアログボックスで [OK] ボタンをクリックすると印刷ステータス情報が表示されるため、ページをスプーラに送信したらただちに `PrintJob.addPage()` および `PrintJob.send()` を呼び出す必要があります。`PrintJob.send()` 呼び出しの格納されたフレームに遅延が到達すると、印刷処理が遅延します。

ActionScript 2.0 には単一フレームの制限があるため、ActionScript 2.0 のすべてのプリントジョブステートメントは 15 秒間でスクリプトタイムアウトの制限に達するよう制限されます。ActionScript 3.0 にも、このスクリプトタイムアウトの制限があります。したがって、プリントジョブのシーケンスにおける主要なステートメント間の時間が 15 秒を超えてはいけません。つまり、15 秒間のスクリプトタイムアウトの制限は、次の間隔に適用されます。

- `PrintJob.start()` と最初の `PrintJob.addPage()` との間
- `PrintJob.addPage()` とその次の `PrintJob.addPage()` との間
- 最後の `PrintJob.addPage()` と `PrintJob.send()` との間

これらの間隔のいずれかが 15 秒を超えていると、その次の PrintJob インスタンスの `PrintJob.start()` の呼び出しで `false` が返され、その次の PrintJob インスタンスの `PrintJob.addPage()` によって、Flash Player がランタイム例外をスローします。

サイズ、拡大率、用紙の向きの設定

488 ページの「ページのプリント」セクションで詳しく説明している基本的なプリントジョブの実行手順の場合は、指定したスプライトの画面サイズおよび位置を直接反映した同等のプリントが出力されます。しかし、プリンタの解像度は環境によって異なり、また、プリンタの設定によってスプライトのプリント結果に悪影響が出る場合もあります。

Flash Player はオペレーティングシステムの印刷設定を読み取ることができますが、これらのプロパティは読み取り専用です。値に応答することはできますが、値を設定することはできません。したがって、プリンタの用紙サイズ設定を調べた後で、コンテンツのサイズなどをプリンタに合わせて調整することになります。また、プリンタの余白や用紙の向きに関する設定を決定することもできます。プリンタの設定に合わせるには、プリント範囲の指定、画面解像度とプリンタのポイント尺度との違いに関する調整、用紙のサイズと向きに応じた幾何学変換などが必要になります。

矩形によるプリント範囲の指定

`PrintJob.addPage()` メソッドでは、スプライトのうちプリントの対象とする範囲を指定できます。それには、第 2 パラメータの `printArea` として Rectangle オブジェクトを指定します。このパラメータの値については、次の 3 つの指定方法があります。

- 特定のプロパティを持つ Rectangle オブジェクトを前もって作成しておく場合は、それを次のように `addPage()` の呼び出しで使用します。

  ```
  private var rect1:Rectangle = new Rectangle(0,0,400,200);
  myPrintJob.addPage(sheet, rect1);
  ```

- Rectangle オブジェクトを前もって設定していない場合は、次の例に示すように、呼び出しの時点で直接指定することもできます。

  ```
  myPrintJob.addPage(sheet, new Rectangle(0, 0, 100, 100));
  ```

- `addPage()` 呼び出しの第 3 パラメータに値を入力する場合に、プリント範囲の矩形を指定しないようにするには、次のように、第 2 パラメータとして `null` を渡します。

  ```
  myPrintJob.addPage(sheet, null, options);
  ```

メモ: プリント寸法を矩形で指定する場合は、`flash.display.Rectangle` クラスを import で読み込んでください。

ポイントとピクセルの比較

矩形の幅と高さはピクセル値です。一方、プリンタで使用する測定単位はポイントです。ポイントの物理サイズは固定 (1/72 インチ) ですが、画面上のピクセルのサイズは解像度によって異なります。したがって、ピクセル値とポイント値の変換比率は、プリンタの設定と、スプライトが拡大または縮小されているかどうかによって変化します。72 ピクセル幅の伸縮されていない Sprite は、用紙上では 1 インチ幅で印刷されます。1 ポイントは 1 ピクセルに相当し、画面の解像度とは無関係です。

インチおよびセンチメートルと、ポイントおよび twip (1/20 ポイント) との間には、次の等式が成り立ちます。

- 1 ポイント = 1/72 インチ = 20 twip
- 1 インチ = 72 ポイント = 1440 twip
- 1 センチメートル = 567 twip

`printArea` パラメータを省略するか、このパラメータの指定が正しくない場合は、スプライト全体がプリントされます。

拡大・縮小

プリントする前に Sprite オブジェクトを拡大または縮小するには、`PrintJob.addPage()` メソッドを呼び出す前に拡大・縮小のプロパティを設定し (ActionScript 3.0 リファレンスガイドの flash.display.DisplayObject クラスの項目を参照)、プリント後にそれらのプロパティを元の値に戻します。Sprite オブジェクトの拡大・縮小は、`printArea` プロパティとは無関係です。つまり、50x50 ピクセルのプリント範囲を指定した場合は常に 2500 個のピクセルがプリントされます。Sprite オブジェクトを拡大または縮小している場合、その設定を反映してプリント結果は伸縮しますが、プリントの対象となる 2500 個のピクセルは変わりません。

例については、498 ページの「例 : 拡大・縮小、トリミング、および応答」を参照してください。

用紙の向きを Landscape (横方向) または Portrait (縦方向) に指定したプリント

Flash Player では用紙の向きに関する設定を検出できるため、次の例に示すように、プリンタ設定に応じてコンテンツのサイズや向きを調整するロジックを ActionScript に組み込むことができます。

```
if (myPrintJob.orientation == PrintJobOrientation.LANDSCAPE)
{
    mySprite.rotation = 90;
}
```

メモ	用紙の向きを調べるためにシステムの設定を読み取る場合は、次のように PrintJobOrientation クラスを import で読み込んでください。 `import flash.printing.PrintJobOrientation;` PrintJobOrientation クラスには、用紙の向きを表す定数値が定義されています。

ページの幅と高さへの対応

用紙の向きに関する設定情報を扱う場合と同様、次のように if ステートメントで囲んだロジックを記述することで、ページの幅と高さの設定を読み取り、それに応じた処理ができます。その例を次のコードに示します。

```
if (mySprite.height > myPrintJob.pageHeight)
{
    mySprite.scaleY = .75;
}
```

また、余白の設定を判断するには、次の例に示すように、用紙の寸法とページの寸法を比較します。

```
margin_height = (myPrintJob.paperHeight - myPrintJob.pageHeight) / 2;
margin_width = (myPrintJob.paperWidth - myPrintJob.pageWidth) / 2;
```

例 : 複数ページのプリント

複数ページのコンテンツをプリントする場合は、各ページのコンテンツを別々のスプライト (下の例では sheet1 および sheet2) に割り当て、それぞれについて PrintJob.addPage( ) を呼び出します。このテクニックを説明するコードを次に示します。

```
package
{
    import flash.display.MovieClip;
    import flash.printing.PrintJob;
    import flash.printing.PrintJobOrientation;
    import flash.display.Stage;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.geom.Rectangle;
```

```
public class PrintMultiplePages extends MovieClip
{
  private var sheet1:Sprite;
  private var sheet2:Sprite;

  public function PrintMultiplePages():void
  {
    init();
    printPages();
  }

  private function init():void
  {
    sheet1 = new Sprite();
    createSheet(sheet1, "Once upon a time...", {x:10, y:50, width:80, 
height:130});
    sheet2 = new Sprite();
    createSheet(sheet2, "There was a great story to tell, and it ended 
quickly.\n\nThe end.", null);
  }

  private function createSheet(sheet:Sprite, str:String, 
imgValue:Object):void
  {
    sheet.graphics.beginFill(0xEEEEEE);
    sheet.graphics.lineStyle(1, 0x000000);
    sheet.graphics.drawRect(0, 0, 100, 200);
    sheet.graphics.endFill();

    var txt:TextField = new TextField();
    txt.height = 200;
    txt.width = 100;
    txt.wordWrap = true;
    txt.text = str;

    if (imgValue != null)
    {
      var img:Sprite = new Sprite();
      img.graphics.beginFill(0xFFFFFF);
      img.graphics.drawRect(imgValue.x, imgValue.y, imgValue.width, 
imgValue.height);
      img.graphics.endFill();
      sheet.addChild(img);
    }
    sheet.addChild(txt);
  }

  private function printPages():void
  {
    var pj:PrintJob = new PrintJob();
```

```
            var pagesToPrint:uint = 0;
            if (pj.start())
            {
                if (pj.orientation == PrintJobOrientation.LANDSCAPE)
                {
                    throw new Error("Page is not set to an orientation of portrait.");
                }

                sheet1.height = pj.pageHeight;
                sheet1.width = pj.pageWidth;
                sheet2.height = pj.pageHeight;
                sheet2.width = pj.pageWidth;

                try
                {
                    pj.addPage(sheet1);
                    pagesToPrint++;
                }
                catch (e:Error)
                {
                // エラーに応答する
                }

                try
                {
                    pj.addPage(sheet2);
                    pagesToPrint++;
                }
                catch (e:Error)
                {
                    // エラーに応答する
                }

                if (pagesToPrint > 0)
                {
                    pj.send();
                }
            }
        }
    }
}
```

例 : 拡大・縮小、トリミング、および応答

場合によっては、画面の表示とプリントとの外観上の違いに対応するために、表示オブジェクトのサイズ (またはその他のプロパティ) をプリント時に調整する必要が生じることがあります。プリントの前に表示オブジェクトのプロパティ (例 : scaleX および scaleY プロパティ) を調整する際には、プリント範囲を定義する矩形よりも大きくオブジェクトを拡大するとトリミングが発生することに注意してください。また、多くの場合には、ページをプリントした後でプロパティを元の値に戻す必要があります。

次のコードは、txt 表示オブジェクトの寸法を拡大・縮小 (ただし、背景にある緑色のボックスはそのまま) していることで、指定した矩形のサイズによってテキストフィールドがトリミングされる結果になる例です。プリントした後は、テキストフィールドを画面表示用の元のサイズに戻します。ユーザーが OS の [印刷] ダイアログボックスでプリントジョブをキャンセルした場合は、Flash Player の表示内容を変更し、ジョブがキャンセルされたことを警告します。

```
package
{
    import flash.printing.PrintJob;
    import flash.display.Sprite;
    import flash.text.TextField;
    import flash.display.Stage;
    import flash.geom.Rectangle;

    public class PrintScaleExample extends Sprite
    {
        private var bg:Sprite;
        private var txt:TextField;

        public function PrintScaleExample():void
        {
            init();
            draw();
            printPage();
        }

        private function printPage():void
        {
            var pj:PrintJob = new PrintJob();
            txt.scaleX = 3;
            txt.scaleY = 2;
            if (pj.start())
            {
                trace(">> pj.orientation: " + pj.orientation);
                trace(">> pj.pageWidth: " + pj.pageWidth);
                trace(">> pj.pageHeight: " + pj.pageHeight);
                trace(">> pj.paperWidth: " + pj.paperWidth);
                trace(">> pj.paperHeight: " + pj.paperHeight);
```

```
            try
            {
                pj.addPage(this, new Rectangle(0, 0, 100, 100));
            }
            catch (e:Error)
            {
                // 何もしない
            }
            pj.send();
        }
        else
        {
            txt.text = "Print job canceled";
        }
        // txt の拡大・縮小プロパティを元に戻す
        txt.scaleX = 1;
        txt.scaleY = 1;
    }

    private function init():void
    {
        bg = new Sprite();
        bg.graphics.beginFill(0x00FF00);
        bg.graphics.drawRect(0, 0, 100, 200);
        bg.graphics.endFill();

        txt = new TextField();
        txt.border = true;
        txt.text = "Hello World";
    }

    private function draw():void
    {
        addChild(bg);
        addChild(txt);
        txt.x = 50;
        txt.y = 50;
    }
  }
}
```

第 19 章

External API の使用

19

ActionScript 3.0 External API を使用すると、Adobe Flash Player 9 の動作している Flash Player コンテナアプリケーションと ActionScript との間で簡単な手順による通信ができます。External API を使用する状況としては、HTML ページ内で SWF ドキュメントと JavaScript との間のインターフェイスを作成する場合や、Flash Player を使用して SWF ファイルを表示するデスクトップアプリケーションを作成する場合があります。

この章では、External API を使用してコンテナアプリケーションを操作する方法、HTML ページ内で ActionScript と JavaScript との間でデータを受け渡す方法、および ActionScript とデスクトップアプリケーションとの間で通信を確立し、データを交換する方法について説明します。

目次

External API について. 502
ExternalInterface クラスの使用. 503
例 : Web ページコンテナに対する External API の使用. 507
例 : ActiveX コンテナに対する External API の使用. 514

External API について

External API は ActionScript の一部で、Flash Player のコンテナとして機能する " 外部アプリケーション " (一般的には Web ブラウザまたはスタンドアロンのプロジェクタアプリケーション) で実行されるコードと、ActionScript との間の通信用メカニズムを提供します。ActionScript 3.0 では、External API の機能が ExternalInterface クラスによって提供されます。Flash Player 8 よりも前のバージョンの Flash Player では、`fscommand()` アクションを使用してコンテナアプリケーション内での通信が実行されていました。ExternalInterface クラスは `fscommand()` に代わるもので、JavaScript と ActionScript との間のすべての通信用に推奨されます。

メモ	以前の `fscommand()` 関数を使用する必要がある場合 (たとえば、以前のバージョンのアプリケーションとの互換性を維持する場合、サードパーティの SWF コンテナアプリケーションやスタンドアロン Flash Player で操作する場合など) は、パッケージレベルの関数として flash.system パッケージに用意されています。

ExternalInterface クラスは、ActionScript と Flash Player から、HTML ページ内の JavaScript や Flash Player を埋め込んだデスクトップアプリケーションへの通信を可能にするサブシステムです。

ExternalInterface クラスは次の条件でのみ使用できます。

- Windows 版 Internet Explorer の全サポートバージョン (5.0 以降)。
- 埋め込みカスタム ActiveX コンテナ (デスクトップアプリケーションに埋め込まれた Flash Player ActiveX コントロールなど)。
- NPRuntime インターフェイスをサポートするすべてのブラウザ。現時点では次のブラウザが該当します。
  - Firefox 1.0 以降
  - Mozilla 1.7.5 以降
  - Netscape 8.0 以降
  - Safari 1.3 以降

その他の環境 (スタンドアローン Player で実行する場合など) では、`ExternalInterface.available` プロパティは `false` を返します。

ActionScript から、HTML ページの JavaScript 関数を呼び出すことができます。External API の機能は、次の点において `fscommand()` よりも強力です。

- `fscommand()` 関数から使用できる機能だけでなく、すべての JavaScript 関数を使用できます。
- 1 つのコマンドと 1 つのストリングパラメータという制限はなく、任意の個数のパラメータを任意の名前で渡すことができます。これにより、External API は `fscommand()` よりも非常に柔軟になっています。
- String パラメータだけでなく、さまざまなデータ型 (Boolean、Number、String など) を渡すことができます。
- 呼び出し結果の値を受け取ることができます。結果は呼び出しに対する戻り値として、直ちに ActionScript に戻されます。

警告

HTML ページの Flash Player インスタンス (`<object>` タグの `id` 属性) に付けられた名前に、ハイフン (-) または JavaScript で演算子として定義されたその他の文字 (+、*、/、\、. など) が含まれていると、コンテナ Web ページを Internet Explorer で表示した場合に、ActionScript からの ExternalInterface 呼び出しが機能しません。
また、Flash Player インスタンスを定義する HTML タグ (`<object>` および `<embed>` タグ) が HTML `<form>` タグ内でネストされていると、ActionScript からの ExternalInterface 呼び出しが機能しません。

ExternalInterface クラスの使用

ActionScript とコンテナアプリケーションとの間の通信には、コンテナで定義されたコード (JavaScript 関数など) を ActionScript から呼び出す方法と、呼び出し可能なように設計された ActionScript 関数をコンテナのコードから呼び出す方法の 2 種類の形式があります。いずれの場合も、呼び出される側のコードに情報が送信され、その結果が呼び出し側コードに返されます。

この通信を容易にするため、ExternalInterface クラスには 2 つの静的プロパティと 2 つの静的メソッドが用意されています。これらのプロパティとメソッドは、外部インターフェイス接続に関する情報を取得し、コンテナ内のコードを ActionScript から実行し、コンテナから ActionScript 関数を呼び出せるようにするために使用されます。

外部コンテナに関する情報の取得

`ExternalInterface.available` プロパティは、現在の Flash Player が含まれているコンテナに外部インターフェイス機能があるかどうかを示します。外部インターフェイスが提供されている場合、このプロパティは `true` になります。利用できない場合には `false` になります。ExternalInterface クラスの他の機能を使用する前に、現在のコンテナにおいて外部インターフェイスによる通信がサポートされているかどうかを、次のようにして必ず確認してください。

```
if (ExternalInterface.available)
{
    // ExternalInterface メソッドをここで呼び出す
}
```

メモ	`ExternalInterface.available` プロパティで確認できるのは、現在のコンテナが ExternalInterface 接続をサポートするタイプなのかどうかです。現在使用しているブラウザの設定で JavaScript が有効化されているかどうかは確認できません。

`ExternalInterface.objectID` プロパティを使用すると、Flash Player インスタンスの一意の識別子 (特に、Internet Explorer における `<object>` タグの `id` 属性や、NPRuntime インターフェイスを使用するブラウザにおける `<embed>` タグの `name` 属性) を確認できます。この一意の ID は、ブラウザ内の現在の SWF ドキュメントを表し、SWF ドキュメントを参照するために使用できます (コンテナ HTML ページ内の JavaScript 関数の呼び出しなど)。Flash Player コンテナが Web ブラウザではない場合、このプロパティは `null` になります。

ActionScript からの外部コードの呼び出し

`ExternalInterface.call()` メソッドでは、コンテナアプリケーション内のコードが実行されます。パラメータは最低 1 つ必要で、コンテナアプリケーション内で呼び出される関数の名前を含んだストリングを指定します。`ExternalInterface.call()` メソッドに渡された追加パラメータは、関数呼び出しのパラメータとしてコンテナに渡されます。

```
// 外部関数 "addNumbers" を呼び出し、
// 2 つのパラメータを渡して、その関数の結果を
// 変数 "result" に割り当てる
var param1:uint = 3;
var param2:uint = 7;
var result:uint = ExternalInterface.call("addNumbers", param1, param2);
```

コンテナが HTML ページの場合は、指定された名前の JavaScript 関数がこのメソッドで呼び出され、その名前はその格納された HTML ページの <script> エレメントで定義されている必要があります。JavaScript 関数の戻り値は ActionScript に返されます。

```
<script language="JavaScript">
  // 2 つの数値を加算し、その結果を ActionScript に返す
  function addNumbers(num1, num2)
  {
    return (num1 + num2);
  }
</script>
```

コンテナがそれ以外の ActiveX コンテナの場合は、このメソッドによって Flash Player ActiveX コントロールからその FlashCall イベントが送出されます。指定された関数名とパラメータは、Flash Player によって直列化されて XML ストリングになります。コンテナは、イベントオブジェクトの request プロパティでその情報にアクセスし、それを使用して自身のコードの実行方法を判定できます。値を ActionScript に返すには、コンテナコードで ActiveX オブジェクトの SetReturnValue() メソッドを呼び出し、その結果 (直列化された XML ストリング) をメソッドのパラメータとして渡します。この通信で使用される XML フォーマットの詳細については、522 ページの「External API の XML フォーマット」を参照してください。

コンテナが Web ブラウザでも、その他の ActiveX コンテナでも、呼び出しが失敗した場合やコンテナ側のメソッドが戻り値を返さなかった場合は、null が返されます。呼び出し側コードによるアクセスが、コンテナ環境の属するセキュリティ Sandbox によって許可されていない場合、ExternalInterface.call() メソッドは SecurityError 例外をスローします。この問題を回避するには、コンテナ環境の allowScriptAccess に適切な値を設定します。たとえば、HTML ページの allowScriptAccess の値を変更するには、<object> および <embed> タグの該当する属性を編集します。

コンテナからの ActionScript コードの呼び出し

コンテナが呼び出すことができるのは関数内の ActionScript コードのみです。それ以外の ActionScript コードは、コンテナから呼び出せません。コンテナアプリケーションから ActionScript 関数を呼び出すには、ExternalInterface クラスに関数を登録し、コンテナのコードからその関数を呼び出すという 2 つの操作が必要です。

最初に、ActionScript 関数を登録し、コンテナから使用可能にすることを示す必要があります。`ExternalInterface.addCallback()` メソッドを次のように使用します。

```
function callMe(name:String):String
{
    return "busy signal =)";
}
ExternalInterface.addCallback("myFunction", callMe);
```

`addCallback()` メソッドには 2 つのパラメータがあります。1 つはストリングの関数名で、コンテナはその名前で関数を認識します。もう 1 つのパラメータは実際の ActionScript 関数で、定義された関数名をコンテナが呼び出すと実行されます。これらの名前は区別されるため、実際の ActionScript 関数が異なる名前であっても、コンテナで使用される関数名を指定できます。このことは、匿名関数が指定された場合や、呼び出される関数が実行時に決定される場合など、関数名が不明な場合に非常に役立ちます。

ActionScript 関数が ExternalInterface クラスに登録されると、コンテナが実際に関数を呼び出せるようになります。その方法は、コンテナのタイプによって異なります。たとえば、Web ブラウザの JavaScript コードの場合、ActionScript 関数は、Flash Player ブラウザオブジェクトのメソッド (`<object>` または `<embed>` タグを表す JavaScript オブジェクトのメソッド) と同じように、登録された関数名を使用して呼び出されます。つまり、パラメータが渡され、呼び出されたローカル関数から結果が返されます。

```
<script language="JavaScript">
    // callResult が値 "busy signal =)" を取得する
    var callResult = flashObject.myFunction("my name");
</script>
...
<object id="flashObject"...>
    ...
    <embed name="flashObject".../>
</object>
```

あるいは、埋め込み ActiveX コントロールを実行している SWF ファイルの関数を呼び出すときに、登録された関数名およびすべてのパラメータが XML フォーマットストリングに直列化されている必要があります。その後、ActiveX コントロールの `CallFunction()` メソッドとパラメータとして XML ストリングを呼び出すことで、呼び出しが実際に実行されます。

いずれの場合も、ActionScript 関数の戻り値はコンテナコードに返されますが、呼び出し側がブラウザの JavaScript コードの場合は値として直接返され、呼び出し側が ActiveX コンテナの場合は XML フォーマットストリングとして直列化して返されます。

例 : Web ページコンテナに対する External API の使用

このアプリケーション例は、ユーザーが自分自身とチャットできる Instant Messaging アプリケーションの場合に、Web ブラウザ内で ActionScript と JavaScript が通信するための方法を示しています (アプリケーション名 : Introvert IM)。メッセージは External API を使用して、Web ページと SWF インターフェイスとの間で HTML 形式で送信されます。この例で示す方法では、次のことを行います。

- ブラウザが通信可能なことを確認してから通信を設定することによる、通信の正しい初期化
- コンテナが External API をサポートしているかどうかの確認
- ActionScript からの JavaScript 関数の呼び出し、パラメータの受け渡し、および応答による値の受け取り
- JavaScript から呼び出される ActionScript メソッドの有効化、および呼び出しの実行

Introvert IM アプリケーションのファイルは、Samples/IntrovertIM_HTML フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
IntrovertIMApp.mxml	MXML ユーザーインターフェイスで構成されるメインアプリケーションファイル。
com/example/programmingas3/introvertIM/IMManager.as	ActionScript とコンテナとの間の通信を確立および管理するクラス。
com/example/programmingas3/introvertIM/IMMessageEvent.as	コンテナからメッセージを受信したときに IMManager クラスから送出されるカスタムイベントタイプ。
com/example/programmingas3/introvertIM/IMStatus.as	アプリケーションで選択可能な、さまざまな “availability” ステータス値を値が表している列挙。
html-template/index.template.html	アプリケーションのコンテナ HTML ページの作成に使用されるテンプレート。このファイルには、アプリケーションのコンテナ部分を構成するすべての JavaScript 関数が格納されています。

ActionScript とブラウザ間の通信の準備

External API の最も一般的な用途の 1 つは、ActionScript アプリケーションと Web ブラウザとの通信です。External API を使用すると、ActionScript メソッドから JavaScript コードへの呼び出し、および JavaScript コードから ActionScript メソッドへの呼び出しを実現できます。ブラウザの仕組みは複雑であり、また、ブラウザ内部で実行されるページレンダリング処理が不明であることから、HTML ページ上の JavaScript コードが最初に実行される前に SWF ドキュメントのコールバックを登録できる保証はありません。したがって、JavaScript から SWF ドキュメントの関数を呼び出す前に、SWF ドキュメント側の接続受け付け準備ができたことを通知するために、必ず SWF ドキュメントから HTML ページへの呼び出しを実行してください。

たとえば、IMManager クラスで実行される一連の手順によって、ブラウザの通信準備ができ、SWF ファイルを通信で使用できることが、Introvert IM で判定されます。ブラウザの通信準備ができたことを判定する最初の手順は、次のように IMManager コンストラクタで実行されます。

```
public function IMManager(initialStatus:IMStatus)
{
    _status = initialStatus;

    // コンテナで External API を使用できることを確認する
    if (ExternalInterface.available)
    {
        try
        {
            // ここで isContainerReady() メソッドを呼び出し、そのメソッドが
            // コンテナを呼び出すことで、Flash Player がロード済みで、コンテナが
            // SWF からの呼び出しを受け付け可能なことを確認する
            var containerReady:Boolean = isContainerReady();
            if (containerReady)
            {
                // コンテナが準備できている場合は、SWF の関数を登録する
                setupCallbacks();
            }
            else
            {
                // コンテナの準備ができていない場合は、Timer を設定して
                // 100ms 間隔でコンテナを呼び出す。コンテナの準備ができたとの応答があれば、
                // タイマーを停止する
                var readyTimer:Timer = new Timer(100);
                readyTimer.addEventListener(TimerEvent.TIMER, timerHandler);
                readyTimer.start();
            }
        }
        ...
    }
    else
    {
        trace("External interface is not available for this container.");
```

```
    }
}
```

コードでは最初に、External API が現在のコンテナでも利用可能かどうかを、ExternalInterface.available プロパティを使用して確認します。利用可能であれば、通信の設定手順が開始されます。外部アプリケーションと通信しようとするとセキュリティ例外やその他のエラーが発生することがあるため、コードは try ブロックでラップされています (簡単にするため、対応する catch ブロックは省略されています)。

次に、ここに示すように isContainerReady() メソッドが呼び出されます。

```
private function isContainerReady():Boolean
{
    var result:Boolean = ExternalInterface.call("isReady");
    return result;
}
```

続いて、isContainerReady() メソッドが ExternalInterface.call() メソッドを使用して、次のように、JavaScript 関数 isReady() を呼び出します。

```
<script language="JavaScript">
<!--
// ------- プライベート変数 -------
var jsReady = false;
...
// ------- ActionScript で呼び出される関数 -------
// ページが初期化され、JavaScript が利用可能かどうかを確認するために呼び出される
function isReady()
{
    return jsReady;
}
...
// <body> タグの onload イベントから呼び出される
function pageInit()
{
    // JavaScript の準備ができたことを記録する
    jsReady = true;
}
...
//-->
</script>
```

isReady() 関数は、単純に jsReady 変数の値を返します。この変数の初期値は false です。Web ページの onload イベントがトリガされると、変数の値が true に変わります。つまり、ページのロード前に ActionScript から isReady() 関数が呼び出されると、JavaScript から ExternalInterface.call("isReady") に false が返され、その結果 ActionScript isContainerReady() メソッドから false が返されます。ページがロードされると、JavaScript isReady() 関数から true が返されるため、ActionScript isContainerReady() メソッドからも true が返されます。

IMManager コンストラクタでは、コンテナの準備状態に応じて 2 つのことが行われます。isContainerReady() から true が返された場合は、コードから単純に setupCallbacks() メソッドが呼び出されて、JavaScript での通信の設定手順が完了します。一方、isContainerReady() から false が返された場合、手順は実質的に保留状態になります。Timer オブジェクトが作成され、次のように 100 ミリ秒間隔で timerHandler() メソッドを呼び出すよう指示されます。

```
private function timerHandler(event:TimerEvent):void
{
    // コンテナの準備ができているかどうかを確認する
    var isReady:Boolean = isContainerReady();
    if (isReady)
    {
        // コンテナの準備ができている場合は、何も調べる必要がなく、
        // タイマーを停止する
        Timer(event.target).stop();
        // コンテナからの呼び出しが可能な ActionScript メソッドを
        // 設定する
        setupCallbacks();
    }
}
```

timerHandler() メソッドが呼び出されるたびに、isContainerReady() メソッドの結果が再び確認されます。コンテナが初期化されると、このメソッドから true が返されます。次に、コードでは Timer を停止して setupCallbacks() メソッドを呼び出し、ブラウザの通信の設定手順を終了します。

JavaScript への ActionScript メソッドの公開

上記の例で示したように、ブラウザの準備ができたことがコードで判定されると、setupCallbacks() メソッドが呼び出されます。このメソッドでは、次のように JavaScript からの呼び出しを受信するよう ActionScript が準備されます。

```
private function setupCallbacks():void
{
    // SWF クライアント関数をコンテナに登録する
    ExternalInterface.addCallback("newMessage", newMessage);
    ExternalInterface.addCallback("getStatus", getStatus);
    // SWF の呼び出し準備ができたことをコンテナに通知する
    ExternalInterface.call("setSWFIsReady");
}
```

setCallBacks() メソッドは、ExternalInterface.addCallback() を呼び出して、JavaScript から呼び出し可能な 2 つのメソッドを登録することで、コンテナとの通信準備のタスクを終了します。このコードでは、メソッドが JavaScript ("newMessage" および "getStatus") から認識されるための名前である第 1 パラメータが、ActionScript でのメソッド名と同じです (この場合、異なる名前を使用することに意味がないため、同じ名前を再利用しました)。最後に、ExternalInterface.call() メソッドを使用して JavaScript 関数 setSWFIsReady() が呼び出され、ActionScript 関数の登録されたコンテナが通知されます。

ActionScript からブラウザへの通信

Introvert IM アプリケーションは、コンテナページでの JavaScript 関数の呼び出しに関するさまざまな例を示しています。最も単純な場合 (setupCallbacks() メソッドの例) には、パラメータを渡したり、返される値を受け取ることなく、JavaScript 関数 setSWFIsReady() が呼び出されます。

```
ExternalInterface.call("setSWFIsReady");
```

isContainerReady() メソッドによる別の例では、ActionScript から isReady() 関数が呼び出され、それに対する応答としてブール値を受け取ります。

```
var result:Boolean = ExternalInterface.call("isReady");
```

JavaScript 関数に値を渡すには、External API を使用する方法もあります。たとえば、IMManager クラスの sendMessage() メソッドについて考えて見ます。このメソッドは、ユーザーが自分の " 交信パートナー " に新しいメッセージを送信するときに呼び出されます。

```
public function sendMessage(message:String):void
{
    ExternalInterface.call("newMessage", message);
}
```

ここでも ExternalInterface.call() を使用して、指定された JavaScript 関数が呼び出され、ブラウザに新しいメッセージが通知されます。また、メッセージ自身は ExternalInterface.call() の追加パラメータとして渡され、結果的に JavaScript 関数 newMessage() のパラメータとして渡されます。

JavaScript からの ActionScript コードの呼び出し

通信は双方向形式でサポートされ、Introvert IM アプリケーションも例外ではありません。Flash Player IM クライアントがメッセージ送信のために JavaScript を呼び出すだけでなく、HTML フォームも JavaScript コードを呼び出して、メッセージを SWF ファイルに送信したり、SWF ファイルに情報を問い合わせます。たとえば、接続が完了し、通信準備ができたことを SWF ファイルがコンテナに通知する場合、ブラウザが最初に行うことは、IMManager クラスの getStatus() メソッドを呼び出して、初期のユーザー availability ステータスを SWF IM クライアントから受信することです。これは、次のように Web ページの updateStatus() 関数で行われます。

```
<script language="JavaScript">
...
function updateStatus()
{
   if (swfReady)
   {
      var currentStatus = getSWF("IntrovertIMApp").getStatus();
      document.forms["imForm"].status.value = currentStatus;
   }
}
...
</script>
```

このコードでは swfReady 変数の値が確認されます。この変数は、SWF ファイルのメソッドが ExternalInterface クラスに登録されたことが、SWF ファイルからブラウザに通知されたかどうかを追跡します。SWF ファイルで通信の受信準備ができている場合、その次の行 (var currentStatus =...) では実際に IMManager クラスの getStatus() メソッドが呼び出されます。コードのこの行では、次の 3 つのことが行われます。

- getSWF() JavaScript 関数が呼び出され、SWF ファイルを表す JavaScript オブジェクトへの参照が返されます。getSWF() に渡されたパラメータによって、HTML ページ内に複数の SWF ファイルが存在する場合にどのブラウザオブジェクトが返されるかが決まります。このパラメータに渡される値は、<object> タグの id 属性、および SWF ファイルの埋め込みに使用される <embed> タグの name 属性と一致している必要があります。
- SWF ファイルへの参照を使用することで、SWF オブジェクトのメソッドであるかのように getStatus() メソッドが呼び出されます。この場合に、関数名 “getStatus” が使用されるのは、ExternalInterface.addCallback() を使用して ActionScript 関数がこの名前で登録されるからです。

- getStatus() ActionScript メソッドでは値が返され、その値が currentStatus 変数に割り当てられて、それがさらに status テキストフィールドのコンテンツ (value プロパティ) として割り当てられます。

メモ	コードを見ていくと、updateStatus() 関数のソースコードで、getSWF() 関数を呼び出しているコード行が実際には次のように記述されていることがわかります。 var currentStatus = getSWF("${application}").getStatus(); ${application} テキストは HTML ページテンプレートのプレースホルダーであり、Adobe Flex Builder 2 でアプリケーションの実際の HTML ページが生成されると、このプレースホルダーテキストが <object> タグの id 属性および <embed> タグの name 属性 (この例では IntrovertIMApp) として使用されているものと同じテキストに置換されます。これは、getSWF() 関数で要求される値です。

sendMessage() JavaScript 関数は、ActionScript 関数へのパラメータの受け渡しを示しています (sendMessage() は、ユーザーが HTML ページの [送信] ボタンを押したときに呼び出される関数です)。

```
<script language="JavaScript">
...
function sendMessage(message)
{
   if (swfReady)
   {
      ...
      getSWF("IntrovertIMApp").newMessage(message);
   }
}
...
</script>
```

newMessage() ActionScript メソッドにはパラメータが 1 つ必要なため、JavaScript コード内の newMessage() メソッド呼び出しで JavaScript message 変数をパラメータとして使用することで、この変数が ActionScript に渡されます。

ブラウザタイプの検出

ブラウザがコンテンツにアクセスする方法はさまざまであるため、この例の `getSWF()` JavaScript 関数に示すように、必ず JavaScript を使用してユーザーが実行しているブラウザを検出すること、およびウィンドウまたはドキュメントオブジェクトを使用して、ブラウザ固有のシンタックスに従ってムービーにアクセスすることが重要です。

```
<script language="JavaScript">
...
function getSWF(movieName)
{
  if (navigator.appName.indexOf("Microsoft") != -1)
{
    return window[movieName];
  }
  else
  {
    return document[movieName];
  }
}
...
</script>
```

スクリプトでユーザーのブラウザタイプを検出していない場合、HTML コンテナの SWF ファイルを再生したときに、ユーザーにとって予期しない動作が生じることがあります。

例 : ActiveX コンテナに対する External API の使用

この例は、External API を使用した、ActiveX コントロールを使用するデスクトップアプリケーションと ActionScript との間の通信を示しています。この例では、ActionScript コードおよび同じ SWF ファイルを含めて Introvert IM を再び使用しているため、ActionScript の External API の使用については説明しません。前述の例をよく理解しておくと、この例を理解しやすくなります。

この例のデスクトップアプリケーションは、Microsoft Visual Studio .NET を使用して C# で記述されています。説明の中心は、ActiveX コントロールを使用した External API の操作方法です。この例は、次のことを示しています。

- Flash Player ActiveX コントロールをホスティングしているデスクトップアプリケーションからの ActionScript 関数の呼び出し
- ActionScript からの関数呼び出しの受信、および ActiveX コンテナでのその処理
- ActiveX コンテナへのメッセージ送信に Flash Player が使用するプロキシクラスを使用した、直列化 XML フォーマットの詳細の非表示

Introvert IM C# ファイルは Samples/IntrovertIM_CSharp フォルダにあります。アプリケーションは、次のファイルで構成されています。

ファイル	説明
AppForm.cs	C# Windows Forms インターフェイスで構成されるメインアプリケーションファイル。
bin/Debug/IntrovertIMApp.swf	アプリケーションにロードされる SWF ファイル。
ExternalInterfaceProxy/ExternalInterfaceProxy.cs	External Interface との通信のための ActiveX コントロールのラッパーとして機能するクラス。ActionScript を呼び出し、ActionScript からの呼び出しを受信するためのメカニズムを提供します。
ExternalInterfaceProxy/ExternalInterfaceSerializer.cs	Flash Player の XML フォーマットメッセージを .NET オブジェクトに変換するタスクを実行するクラス。
ExternalInterfaceProxy/ExternalInterfaceEventArgs.cs	このファイルは、カスタム Deligate クラスとイベント引数クラスという 2 つの C# タイプ (クラス) を定義します。この 2 つは、ActionScript からの関数呼び出しのリスナーを通知するために ExternalInterfaceProxy クラスで使用されます。
ExternalInterfaceProxy/ExternalInterfaceCall.cs	このクラスは、関数名とパラメータのプロパティを使用して、ActionScript から ActiveX コンテナへの関数呼び出しを表す値オブジェクトです。
bin/Debug/IntrovertIMApp.swf	アプリケーションにロードされる SWF ファイル。
obj/AxInterop.ShockwaveFlashObjects.dll、obj/Interop.ShockwaveFlashObjects.dll	Flash Player (Shockwave® Flash) ActiveX コントロールのアクセスに必要な Visual Studio .NET によって、管理コードから作成されたラッパーアセンブリ。

Introvert IM C# アプリケーションの概要

このアプリケーション例は、互いに通信する 2 つのインスタントメッセージクライアントプログラム (1 つは SWF ファイル内、もう 1 つは Windows Forms に埋め込み) を示しています。ユーザーインターフェイスには、Shockwave® Flash ActiveX コントロールのインスタンスが含まれ、ActionScript IM クライアントを含んだ SWF ファイルがロードされます。このインターフェイスには、Windows Forms IM クライアントを構成するいくつかのテキストフィールドも含まれます。1 つはメッセージ入力用フィールド (`MessageText`) で、もう 1 つはクライアント間で送信されるメッセージの複製 (`Transcript`) を表示します。第 3 のフィールド (`Status`) は SWF IM クライアントで設定された availability ステータスを表示します。

Shockwave Flash ActiveX コントロールの包含

Shockwave Flash ActiveX コントロールを独自の Windows Forms アプリケーションに含めるには、最初にそれを Microsoft Visual Studio Toolbox に追加する必要があります。

コントロールをツールボックスに追加するには :

1. Visual Studio Toolbox を開きます。
2. Visual Studio 2003 の [Windows フォーム] セクション、または Visual Studio 2005 の任意のセクションを右クリックします。Visual Studio 2003 ではコンテキストメニューで [項目の追加 / 削除] を選択します。Visual Studio 2005 では [項目の選択 ...] を選択します。

   これにより、[ツールボックスのカスタマイズ] (2003) または [ツールボックス項目の選択] (2005) ダイアログボックスが表示されます。
3. [COM コンポーネント] タブを選択します。このタブには、Flash Player ActiveX コントロールなど、そのコンピュータにあるすべての COM コンポーネントが表示されます。
4. スクロールして Shockwave Flash Object を探し、選択します。

   この項目が表示されない場合は、システムに Flash Player ActiveX がインストールされているか確認してください。

ActionScript から ActiveX コンテナへの通信について

External API を使用した ActiveX コンテナアプリケーションとの通信は、Web ブラウザとの通信と同様になりますが、大きな違いが 1 つあります。前述したように、ActionScript が Web ブラウザと通信する場合、開発者が考慮している限り、関数は直接呼び出されます。関数の呼び出し方法と応答方法の詳細は、プレイヤー間で受け渡しできるようフォーマットされ、ブラウザは認識されません。ただし、ActiveX コンテナアプリケーションとの通信に External API が使用される場合、Flash Player はメッセージ (関数呼び出しおよび戻り値) をある XML フォーマットでアプリケーションに送信し、コンテナアプリケーションからの関数呼び出しと戻り値に同じ XML フォーマットが使用される必要があります。ActiveX コンテナアプリケーションの開発者は、適切なフォーマットの関数呼び出しと応答を認識し、作成可能なコードを作成する必要があります。

サンプルの Introvert IM C# には、メッセージのフォーマッティング防止の可能なクラスが含まれていますが、ActionScript 関数の呼び出しおよび ActionScript からの呼び出しの受信時は、標準のデータ型を使用して操作することができます。ExternalInterfaceProxy クラスを他のヘルパークラスと組み合わせるとこの機能が提供され、任意の .NET プロジェクトで再利用することで External API との通信が容易になります。

次のコードは、メインアプリケーションフォーム (AppForm.cs) から引用したもので、ExternalInterfaceProxy クラスを使用して実現された単純な操作を示しています。

```
public class AppForm : System.Windows.Forms.Form
{
    ...
    private ExternalInterfaceProxy proxy;
    ...
    public AppForm()
    {
        ...
        // このアプリケーションを登録することで、プロキシが ActionScript からの
        // 呼び出しを受信したときに通知を受信する
        proxy = new ExternalInterfaceProxy(IntrovertIMApp);
        proxy.ExternalInterfaceCall += new
    ExternalInterfaceCallEventHandler(proxy_ExternalInterfaceCall);
        ...
    }
    ...
```

このアプリケーションでは、`proxy` という名前の ExternalInterfaceProxy インスタンスが宣言および作成され、その参照がユーザーインターフェイスの Shockwave Flash ActiveX コントロールに渡されます (`IntrovertIMApp`)。次に、プロキシの `ExternalInterfaceCall` イベントを受信するための `proxy_ExternalInterfaceCall()` メソッドが、コードで登録されます。このイベントは、Flash Player から関数呼び出しがあったときに ExternalInterfaceProxy クラスによって送出されます。このイベントへのサブスクライブは、C# コードが ActionScript からの関数呼び出しを受信し、応答する方法になっています。

ActionScript から関数呼び出しがあると、ExternalInterfaceProxy インスタンス (`proxy`) はその呼び出しを受信し、それを XML フォーマットから変換します。そして、プロキシの ExternalInterfaceCall イベントのリスナーとなっているオブジェクトに通知します。AppForm クラスの場合、`proxy_ExternalInterfaceCall()` メソッドでは次のようにイベントが処理されます。

```
/// <summary>
/// SWF で ActionScript ExternalInterface 呼び出しが行われると、
/// プロキシから呼び出される
/// </summary>
private object proxy_ExternalInterfaceCall(object sender,
ExternalInterfaceCallEventArgs e)
{
    switch (e.FunctionCall.FunctionName)
    {
        case "isReady":
            return isReady();
        case "setSWFIsReady":
            setSWFIsReady();
            return null;
        case "newMessage":
            newMessage((string)e.FunctionCall.Arguments[0]);
            return null;
        case "statusChange":
            statusChange();
            return null;
        default:
            return null;
    }
}
...
```

このメソッドは ExternalInterfaceCallEventArgs インスタンスに渡されます。この例ではインスタンス名は `e` です。これにより、このオブジェクトが ExternalInterfaceCall クラスのインスタンスである `FunctionCall` プロパティを取得します。

ExternalInterfaceCall インスタンスが、2 つのプロパティを持つ単純な値オブジェクトです。FunctionName プロパティには、ActionScript ExternalInterface.Call() ステートメントで指定された関数名が格納されています。ActionScript に何らかのパラメータが追加されると、そのパラメータは ExternalInterfaceCall オブジェクトの Arguments プロパティに格納されます。この場合、イベントを処理するメソッドは、トラフィックマネージャのように機能する switch ステートメントです。FunctionName プロパティの値 (e.FunctionCall.FunctionName) によって、AppForm クラスのどのメソッドが呼び出されるかが決まります。

前述のコードの switch ステートメントのブランチは、一般的なメソッド呼び出しシナリオを示しています。たとえば、すべてのメソッドは値を ActionScript に返すか (isReady() メソッド呼び出しなど)、null を返す (他のメソッド呼び出しの場合) 必要があります。ActionScript から渡されたパラメータのアクセスは、newMessage() メソッド呼び出しに示されています (Arguments 配列の第 1 エレメントであるパラメータ e.FunctionCall.Arguments[0] で渡されます)。

ExternalInterfaceProxy クラスを使用した C# からの ActionScript 関数の呼び出しは、ActionScript からの関数呼び出しの受信よりもはるかに単純です。ActionScript 関数を呼び出すには、次のように、ExternalInterfaceProxy インスタンスの Call() メソッドを使用します。

```
    /// <summary>
    /// [ 送信 ] ボタンが押されると呼び出され、
    /// MessageText テキストフィールドの値がパラメータとして渡される
    /// </summary>
    /// <param name="message"> 送信するメッセージ。</param>
    private void sendMessage(string message)
    {
        if (swfReady)
        {
            ...
            // ActionScript の newMessage 関数を呼び出す
            proxy.Call("newMessage", message);
        }
    }
    ...
    /// <summary>
    /// ActionScript 関数を呼び出して現在の "availability" ステータスを
    /// 取得し、それをテキストフィールドに書き込む
    /// </summary>
    private void updateStatus()
    {
        Status.Text = (string)proxy.Call("getStatus");
    }
    ...
}
```

この例に示すように、ExternalInterfaceProxy クラスの Call() メソッドは、ActionScript でそれに相当する ExternalInterface.Call() に非常によく似ています。第 1 パラメータはストリングで、呼び出す関数の名前になります。その他のパラメータ (ここには示されていません) も ActionScript 関数に渡されます。ActionScript 関数から値が返された場合、その値は Call() メソッドで返されます (前述の例)。

ExternalInterfaceProxy クラスの内部

ActiveX コントロールのプロキシラッパーを使用することは必ずしも重要ではなく、独自のプロキシクラス (たとえば、別のプログラミング言語や別のプラットフォーム用) を作成することもできます。プロキシの作成についてはここで詳しく説明しませんが、この例におけるプロキシクラスの内部処理を理解することは有益です。

Shockwave Flash ActiveX コントロールの CallFunction() メソッドを使用すると、External API を使用して ActiveX コンテナから ActionScript 関数を呼び出すことができます。次の ExternalInterfaceProxy クラスの Call() メソッドの一部で、これを示しています。

```
// Shockwave Flash ActiveX コントロールの "_flashControl" で、
// SWF の ActionScript 関数を呼び出す
string response = _flashControl.CallFunction(request);
```

このコード部分で、_flashControl は Shockwave Flash ActiveX コントロールです。ActionScript 関数呼び出しは、CallFunction() メソッドを使用して行われます。このメソッドにはパラメータが 1 つあり (この例では request)、このパラメータは XML フォーマットされた命令の含まれたストリングです。呼び出される ActionScript 関数の名前および任意のパラメータが含まれています。ActionScript から返されたすべての値は XML フォーマットストリングにエンコードされ、CallFunction() 呼び出しの戻り値として送り返されます。この例では、この XML ストリングが response 変数に格納されます。

ActionScript からの関数呼び出しの受信は、段階的な手順で行われます。ActionScript から関数呼び出しがあると、Shockwave Flash ActiveX コントロールは FlashCall イベントを送出するため、SWF ファイルからの呼び出しを受信するためのクラス (ExternalInterfaceProxy クラスなど) ではそのイベント用のハンドラを定義しておく必要があります。ExternalInterfaceProxy クラスでは、このイベントハンドラ関数の名前は `_flashControl_FlashCall()` で、クラスコンストラクタ内のイベントを待ち受けるよう登録されています。

```
private AxShockwaveFlash _flashControl;

public ExternalInterfaceProxy(AxShockwaveFlash flashControl)
{
    _flashControl = flashControl;
    _flashControl.FlashCall += new
    _IShockwaveFlashEvents_FlashCallEventHandler(_flashControl_FlashCall);
}
...
private void _flashControl_FlashCall(object sender,
    _IShockwaveFlashEvents_FlashCallEvent e)
{
    // イベントオブジェクトの request プロパティ ("e.request") を使用して
    // アクションを実行する
    ...
    // 値を ActionScript に返す
    // 返された値は、最初に XML フォーマットストリングとしてエンコードする必要がある
    _flashControl.SetReturnValue(encodedResponse);
}
```

イベントオブジェクト (e) には `request` プロパティ (`e.request`) があります。このプロパティは、関数名やパラメータなど、関数呼び出しに関する情報が XML フォーマットで含まれたストリングです。この情報をコンテナで使用すると、実行するコードを判別できます。ExternalInterfaceProxy クラスでは、要求が XML フォーマットから ExternalInterfaceCall オブジェクトに変換され、同じ情報がアクセスしやすい形式で提供されます。ActiveX コントロールの `SetReturnValue()` メソッドを使用すると、関数の結果を ActionScript の呼び出し側に返すことができます。この場合も、得られたパラメータを External API で使用される XML フォーマットにエンコードする必要があります。

External API の XML フォーマット

Shockwave Flash ActiveX をホストするアプリケーションと ActionScript との間の通信では、関数呼び出しと値をエンコードするために専用の XML フォーマットが使用されます。サンプルの Introvert IM C# では、ExternalInterfaceProxy クラスによって、アプリケーションフォーム内のコードでこのフォーマットを無視し、値を直接操作できるようになっています。これを実現するため、ExternalInterfaceProxy クラスでは ExternalInterfaceSerializer クラスのメソッドを使用して、実際に XML メッセージを .NET オブジェクトに変換しています。ExternalInterfaceSerializer クラスには、次の 4 つのパブリックメソッドがあります。

- `EncodeInvoke()`　引数の関数名と C# ArrayList を適切な XML フォーマットにエンコードします。
- `EncodeResult()`　結果値を適切な XML フォーマットにエンコードします。
- `DecodeInvoke()`　ActionScript からの関数呼び出しをデコードします。FlashCall イベントオブジェクトの `request` プロパティは `DecodeInvoke()` メソッドに渡され、このメソッドで呼び出しが ExternalInterfaceCall オブジェクトに変換されます。
- `DecodeResult()`　ActionScript 関数の呼び出しの結果として受信した XML をデコードします。

External API で使用される XML フォーマットには 2 つの部分があります。1 つのフォーマットは、関数呼び出しを表すために使用されます。もう 1 つのフォーマットは個々の値を表しています。このフォーマットは、関数のパラメータおよび関数の戻り値に使用されます。関数呼び出し用の XML フォーマットは、ActionScript との間の呼び出しに使用されます。ActionScript からの関数呼び出しに対して、Flash Player は XML をコンテナに渡します。コンテナからの呼び出しに対しては、Flash Player はこのフォーマットで XML ストリングを渡すコンテナアプリケーションが必要になります。次の XML は、XML フォーマットされた関数呼び出しの例を示しています。

```
<invoke name="functionName" returntype="xml">
  <arguments>
    ... (individual argument values)
  </arguments>
</invoke>
```

ルートノードは `<invoke>` ノードです。ここには 2 つの属性があり、`name` は関数呼び出しの名前を表し、`returntype` は常に `xml` になります。関数呼び出しにパラメータが含まれている場合、`<invoke>` ノードには `<arguments>` 子ノードができます。さらにその子ノードは、次に説明する個別の値フォーマットを使用してフォーマットされたパラメータになります。

関数のパラメータや関数の戻り値などの個々の値では、実際の値に加えてデータ型情報を含んだフォーマッティングスキームが使用されます。

ActionScript のクラスまたは値	C# のクラスまたは値	フォーマット	コメント
null	null	<null/>	
Boolean true	bool true	<true/>	
Boolean false	bool false	<false/>	
String	string	<string>string value</string>	
Number、int、uint	single、double、int、uint	<number>27.5</number> <number>-12</number>	
Array (エレメントは型の混在可)	ArrayList や object[] など、エレメントの型の混在が可能なコレクション	<array> <property id="0"> <number>27.5</number> </property> <property id="1"> <string>Hello there!</string> </property> ... </array>	<property> ノードは個々のエレメントを定義します。id 属性は 0 から始まるインデックスの数値です。
Object	ストリングキー付きの HashTable など、ストリングキーとオブジェクト値のある辞書	<object> <property id="name"> <string>John Doe</string> </property> <property id="age"> <string>33</string> </property> ... </object>	<property> ノードは個々のプロパティを定義します。id 属性はプロパティ名 (ストリング) です。
その他のビルトインクラスまたはカスタムクラス		<null/> or <object></object>	ActionScript では他のオブジェクトが null または空のオブジェクトとしてエンコードされます。いずれの場合も、すべてのプロパティ値が失われます。