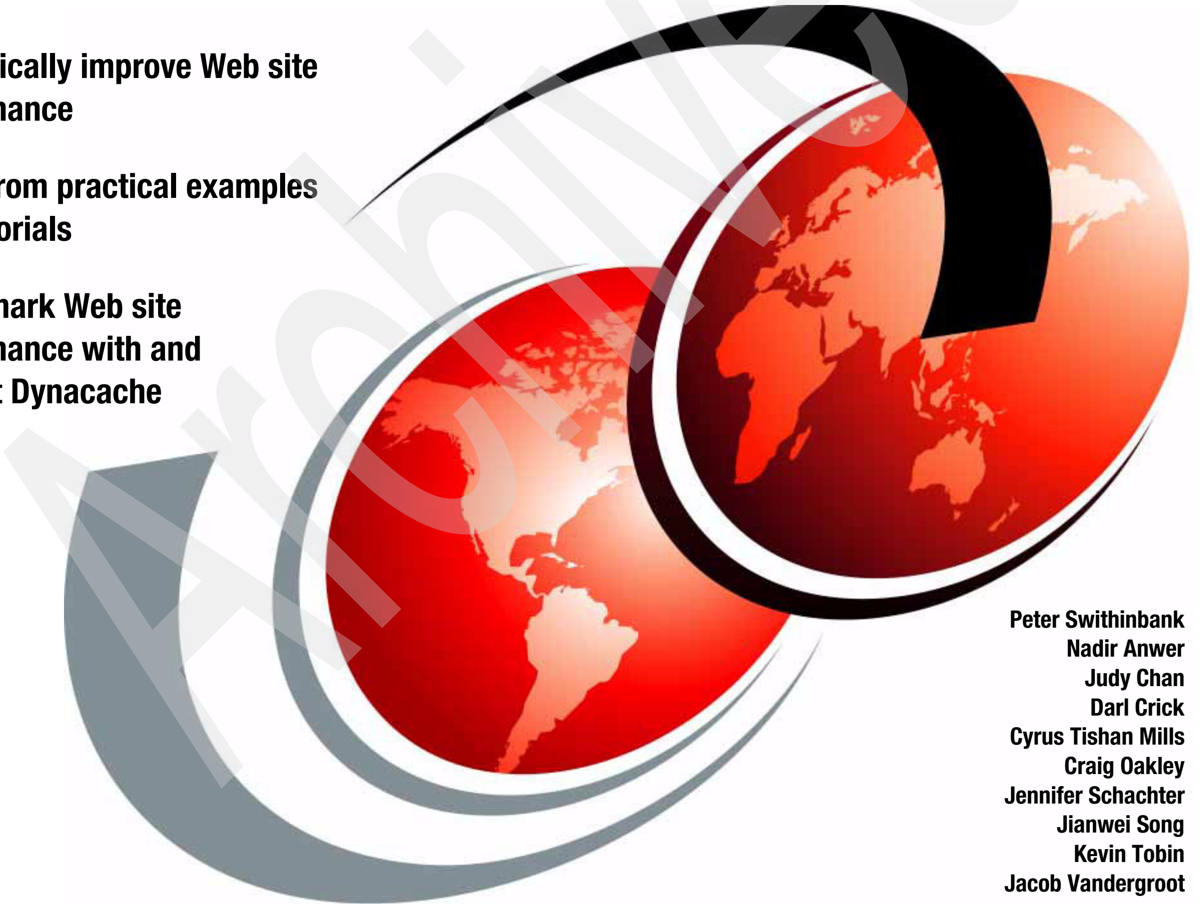
WebSphere® software

IBM

# Mastering DynaCache in WebSphere Commerce



Dramatically improve Web site performance

Learn from practical examples and tutorials

Benchmark Web site performance with and without Dynacache

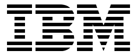
Peter Swithinbank
Nadir Anwer
Judy Chan
Darl Crick
Cyrus Tishan Mills
Craig Oakley
Jennifer Schachter
Jianwei Song
Kevin Tobin
Jacob Vandergroot

Redbooks

ibm.com/redbooks

Archived

Archived



International Technical Support Organization

# Mastering DynaCache in WebSphere Commerce

December 2006

Archived

SG24-7393-00

**Note:** Before using this information and the product it supports, read the information in "Notices" on page ix.

Archived

**First Edition (December 2006)**

This edition applies to Version 6 of WebSphere Commerce.

**© Copyright International Business Machines Corporation 2006. All rights reserved.**
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

**Notices** . . . ix
Trademarks . . . x

**Preface** . . . xi
The team that wrote this redbook . . . xi
Become a published author . . . xiv
Comments welcome . . . xiv

**Part 1. Web site caching** . . . 1

**Chapter 1. Web site performance** . . . 3
1.1 J2EE Web site topologies and caching . . . 4
1.2 Web site performance issues . . . 5
1.2.1 e-Commerce Web site types . . . 5
1.2.2 Unnecessary re-creation of page data . . . 5
1.2.3 Network traffic between tiers . . . 7
1.2.4 WebSphere Funnel model . . . 8
1.2.5 Reduce thread/process switches and CPU load . . . 10
1.3 Project planning . . . 10
1.3.1 Plan for caching early in the design phase . . . 11
1.3.2 Roadmap . . . 11
1.3.3 System performance tuning . . . 12
1.3.4 Performance tuning skills . . . 13
1.3.5 Training . . . 13
1.3.6 IBM Tech-line assistance . . . 14
1.4 Performance terminology . . . 14
1.4.1 Response time . . . 14
1.4.2 Load . . . 16
1.4.3 Throughput . . . 16
1.4.4 Throughput plateau . . . 16
1.4.5 Throughput saturation . . . 17
1.4.6 Path length . . . 20
1.4.7 Bottleneck . . . 20
1.4.8 Scalability . . . 23

**Chapter 2. Caching** . . . 25
2.1 Caching overview . . . 26
2.1.1 How caches work . . . 26
2.1.2 Where caching is performed . . . 29

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

2.1.3 The value of Web caching . . . 31
2.1.4 Static versus dynamic object caching . . . 33
2.1.5 Full Web page versus fragment caching . . . 33
2.1.6 Cache considerations . . . 34
2.2 Introduction to DynaCache . . . 34
2.2.1 DynaCache history . . . 34
2.3 Enabling WebSphere Application Server DynaCache . . . 36
2.4 DynaCache technical overview . . . 38
2.4.1 Features of DynaCache . . . 41
2.5 Servlets and DynaCache . . . 42
2.5.1 Servlet technology . . . 42
2.5.2 Request attributes . . . 42
2.5.3 Servlet filters . . . 43
2.5.4 WebSphere Commerce caching filter . . . 43
2.5.5 JSP includes and forwards . . . 43
2.6 Configuring DynaCache using XML-based policies . . . 44
2.6.1 Basic structure of the cachespec.xml file . . . 45
2.6.2 Cache entry element overview <cache-entry> . . . 48
2.6.3 Cache ID Overview . . . 49
2.6.4 Cache IDs and the cache hit . . . 51
2.6.5 Cache programming support . . . 52
2.6.6 Dependency ID overview <dependency-id> . . . 53
2.6.7 Invalidation rules overview <invalidation> . . . 55
2.6.8 Command-based invalidation . . . 56
2.6.9 Delay-invalidations . . . 58
2.6.10 The effect of updates to the cachespec.xml file . . . 59
2.7 Putting items into the DynaCache . . . 60
2.7.1 Caching servlets and JSPs . . . 60
2.7.2 Java objects and the command cache . . . 61
2.7.3 Command interface . . . 63
2.7.4 DynaCache full page caching . . . 68
2.7.5 DynaCache fragment caching . . . 69
2.8 Invalidation: Getting stale objects out of the cache . . . 73
2.9 The ConsumerDirect cachespec.xml file . . . 75
2.9.1 WebSphere Commerce ECActionServlet explained . . . 76
2.9.2 Cache-id definitions for ConsumerDirect . . . 77
2.10 Impact of memory cache on JVM garbage collection . . . 79
2.11 Configure disk offload . . . 84
2.11.1 Tuning the disk cache . . . 85
2.12 Displaying cache information . . . 87
2.12.1 Install the cacheMonitor.ear application . . . 87
2.12.2 Cache monitor viewing capabilities . . . 88
2.12.3 Cache monitor operational tasks . . . 89

Archived

**Chapter 3. DynaCache invalidation** . . . 91
3.1 DynaCache invalidation defined . . . 92
3.1.1 Invalidation overview . . . 92
3.2 DynaCache invalidation mechanisms and tools . . . 92
3.2.1 The invalidation process . . . 93
3.2.2 Cachespec.xml invalidation policies . . . 93
3.2.3 DynaCache invalidation API . . . 100
3.2.4 Scheduled invalidation . . . 101
3.2.5 Cache Monitor . . . 103
3.3 Invalidation best practices and techniques . . . 104
3.3.1 Time out considerations . . . 105
3.3.2 Cache monitor . . . 105
3.3.3 Dependency IDs . . . 105
3.3.4 Cache instances . . . 105
3.3.5 Warm shutdown . . . 106
3.3.6 Invalidation during the tuning phase . . . 106
3.3.7 Startup – use warm-up to create cache entries . . . 106
3.3.8 Impact of maintenance . . . 106

**Chapter 4. Clustering DynaCache** . . . 107
4.1 Data Replication Service . . . 108
4.1.1 Failover and caching . . . 109
4.1.2 DRS and failover . . . 110
4.1.3 DRS and caching . . . 110
4.2 Replication in DynaCache . . . 114
4.2.1 Specifying the sharing policy declaration in the cachespec.xml . . . 116
4.2.2 Troubleshooting . . . 117
4.3 Best practices . . . 118

**Chapter 5. Caching strategy** . . . 121
5.1 Site requirements . . . 122
5.2 Identifying cache objects . . . 123
5.2.1 Characteristics of cacheable objects . . . 123
5.2.2 Tools and methodology . . . 124
5.3 Cache design . . . 126
5.3.1 Full-page caching and fragment caching . . . 126
5.3.2 Cache instances . . . 127
5.4 Invalidating cached objects . . . 127
5.5 DynaCache and JSP . . . 128

**Chapter 6. Advanced topics** . . . 133
6.1 What is new in Version 6 of DynaCache . . . 134
6.1.1 Disk cache enhancements . . . 134
6.1.2 Cache policy enhancements . . . 135

6.2 Edge Side Include (ESI) caching. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142
6.3 Priming the cache . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 142
6.4 When you must not cache. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 143
6.5 Multiple caching pools and cache instances . . . . . . . . . . . . . . . . . . . . . . . . 143
6.5.1 Cache instance . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 144
6.5.2 Cache instance definition . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 144
6.6 DynaCache tuning. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 145
6.7 Memory caching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146
6.7.1 Cache sizing formula. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 146
6.7.2 Disk caching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 147
6.8 Setting custom system properties . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 148
6.9 Monitoring DynaCache . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 149
6.9.1 DeveloperWorks tooling for monitoring DynaCache . . . . . . . . . . . . 152
6.10 Reference section . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153
6.10.1 Class element . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153
6.10.2 Name element. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 153
6.10.3 Sharing policy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 154
6.10.4 Property. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 155
6.10.5 Cache entry IDs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 157
6.10.6 Cache servlet filtering and Commerce DC_ variables . . . . . . . . . 168
6.10.7 ConsumerDirect jspStoreDir issue . . . . . . . . . . . . . . . . . . . . . . . . 170

**Chapter 7. FAQs**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 171
7.1 DynaCache FAQs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 172
7.2 Clustering FAQs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 175

**Part 2. DynaCache implementation**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 177

**Chapter 8. DynaCache tutorial**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 179
8.1 Environment setup . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180
8.1.1 Software stack. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180
8.1.2 WebSphere Commerce setup. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 180
8.1.3 Enable DynaCache service. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 182
8.2 Installing the Cache Monitor . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 184
8.3 Caching ConsumerDirect store . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 189
8.3.1 Catalog subsystem URLs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 189
8.3.2 TopCategoriesDisplay. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 191
8.3.3 CategoryDisplay . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 201
8.3.4 ProductDisplay . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 204

**Chapter 9. Benchmarking DynaCache** . . . . . . . . . . . . . . . . . . . . . . . . . . . . 209
9.1 Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
9.1.1 Benchmarking benefits . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210
9.1.2 Benchmarking considerations. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 210

Archived

9.1.3 Benchmarking DynaCache . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 213
9.2 Benchmark creation process. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 214
9.2.1 Setting up benchmark-creation tests. . . . . . . . . . . . . . . . . . . . . . . . 214
9.2.2 Executing tests and recording results . . . . . . . . . . . . . . . . . . . . . . . 216
9.2.3 Interpreting and analyzing the test results . . . . . . . . . . . . . . . . . . . 217
9.3 Benchmarking example. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 218
9.3.1 Test environment . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 218
9.3.2 Test data set and scenario . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 220
9.3.3 Execution and results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 223
9.4 Conclusion. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 233

**Chapter 10. Case study: A DynaCache anti-pattern**. . . . . . . . . . . . . . . . . 235
10.1 Online shop project brief . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 236
10.2 Issues encountered . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 236
10.2.1 DynaCache not enabled . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 236
10.2.2 Inability to cache page fragments . . . . . . . . . . . . . . . . . . . . . . . . . 237
10.2.3 Cache invalidations causing severe performance impacts . . . . . . 238
10.2.4 Cached page sizes greater than 200Kb . . . . . . . . . . . . . . . . . . . . 238
10.2.5 Large numbers of duplicated similar cache areas. . . . . . . . . . . . . 239
10.3 Lessons learned from the exercise . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 239
10.3.1 Include DynaCache in the design of applications . . . . . . . . . . . . . 240
10.3.2 Retrofitting DynaCache will only be a limited success . . . . . . . . . 240
10.3.3 Use accurate workload traffic for simulation . . . . . . . . . . . . . . . . . 240
10.3.4 Invalidate as little as possible . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 240
10.3.5 Warm up the cache . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 240
10.4 Changes in the next version of the online shop . . . . . . . . . . . . . . . . . . 241
10.4.1 Break all pages into cacheable fragments . . . . . . . . . . . . . . . . . . . 241
10.4.2 Reduce the number of dependency IDs . . . . . . . . . . . . . . . . . . . . 241
10.4.3 Remove cache page expiries . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 242
10.4.4 Incorporate DB triggers to update the CACHEIVL table . . . . . . . . 242
10.4.5 Write a scheduled task to clean the CACHEIVL table . . . . . . . . . 242
10.4.6 Fix inefficiencies in the search fragments . . . . . . . . . . . . . . . . . . . 242
10.5 Conclusion. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 243

**Chapter 11. Seven steps to get started caching your WebSphere Commerce Web site** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 245
11.1 Servlet caching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 246
11.2 Caching personalized fragments. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 246
11.3 Excluding self-executing fragments from the cache . . . . . . . . . . . . . . . 247
11.4 Fragment caching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247
11.5 Command caching . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 247
11.6 Invalidation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 248
11.7 Replication. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 249

Archived

**Appendix A. Web services caching** . . . . . . . . . . 251
WebSphere Web service caching support . . . . . . . . . . 252
WebSphere Commerce Web service caching . . . . . . . . . . 252
  Overview of the WebSphere Commerce Web services framework . . . . . 252
Caching the business logic. . . . . . . . . . 254
Caching the response . . . . . . . . . . 255

**Appendix B. Caching in WebSphere Extended Deployment**. . . . . . . . . . 257
Introduction to WebSphere XD. . . . . . . . . . 258
  Dynamic operations . . . . . . . . . . 258
  High performance computing . . . . . . . . . . 258
  Extended manageability . . . . . . . . . . 259
  On-demand router. . . . . . . . . . 259
WebSphere Commerce and WebSphere XD . . . . . . . . . . 260
References. . . . . . . . . . 263

**Appendix C. Sales Center caching** . . . . . . . . . . 265
IBM Sales Center . . . . . . . . . . 266
Caching the response in Sales Center. . . . . . . . . . 267

**Abbreviations and acronyms** . . . . . . . . . . 269

**Related publications** . . . . . . . . . . 271
IBM Redbooks . . . . . . . . . . 271
Online resources . . . . . . . . . . 271
How to get IBM Redbooks . . . . . . . . . . 273
Help from IBM . . . . . . . . . . 273

**Index** . . . . . . . . . . 275

Archived

# Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
*IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.*

*The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law*: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

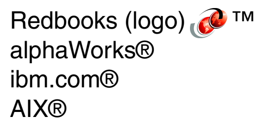
This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Archived

# Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

| | | |
|---|---|---|
| Redbooks (logo)  ™ | DB2® | Tivoli® |
| alphaWorks® | IBM® | WebSphere® |
| ibm.com® | Rational® | |
| AIX® | Redbooks™ | |

The following terms are trademarks of other companies:

EJB, Java, Java Naming and Directory Interface, JDBC, JDK, JMX, JSP, JVM, J2EE, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Excel, Microsoft, Windows Server, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Archived

# Preface

This IBM® Redbook describes how to use WebSphere® DynaCache to improve the performance of WebSphere Commerce Web sites.

Today's Web sites are a demanding mixture of static images surrounded by mini-shopping carts, e-marketing spots, and other eye-catching fragments, all of which change from view to view and user to user. Sites must be richly featured and personalized to attract customers – and they must deliver this content at a high level of performance as well. But the richness and personalization customers want is often the enemy of good Web site performance.

DynaCache technology gives Web site developers a robust tool for achieving excellent Web site performance. It can be applied retrospectively to existing Web sites whose performance is not meeting the owning company's requirements. It is even better applied from the beginning of a J2EE™ Web project, and will yield performance gains well beyond those achieved at a comparable cost by adding more hardware or rewriting the solution.

This book leads you through an explanation of what caching is, and what is special about caching Web sites. It then describes the capabilities offered by WebSphere DynaCache and how to most effectively make use of those capabilities. The discussion is enhanced by practical examples and tutorials to help you configure DynaCache and implement a sample WebSphere Commerce store. Finally, the book describes how to approach benchmarking for an online store, and how to quantify the effectiveness of a dynamic caching policy on site performance. It also presents a case study of a real-world Web site problem that was turned around by an IBM team applying DynaCache technology.

## The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Raleigh Center.

**Peter Swithinbank** is a project leader at the ITSO, Hursley Center. He writes IBM Redbooks™ and teaches IBM classes worldwide on Web services and building business integration solutions. Peter has worked for IBM for 28 years and has been with the ITSO for two years. He has a diploma in software engineering from Oxford University and an MA in Geography from the University of Cambridge.

Archived

**Nadir Anwer** is a Staff Software Engineer for the WebSphere Commerce Advanced Technical Services group based in IBM Toronto Lab, Canada. He has over 13 years of combined software development experience in Web technologies. He holds a Master degree in Computer Science from the Nottingham Trent University, UK. His areas of expertise include Web site performance and emerging open-source technologies.

**Judy Chan** is a Advisory Software Developer in IBM Canada Lab. She has eight years of experience in DB2® development and six years of experience in WebSphere Commerce development. She holds a Bachelor degree in Computer Science and Mathematics, and a Master degree in Mathematics from York University. Her areas of expertise include DB2 development on CLP, SQLJ and SQL procedure, WebSphere Commerce development on Admin Console, inventory, RFQ, tickler and Sales Center development.

**Darl Crick** is a Senior Technical Staff member. Darl gave the team a great deal of assistance and advice, as well as extensively reviewing the book. Many of Darl's insights and experiences with DynaCache have been incorporated into the Redbook. Darl's experience comes from working in Commerce Development for four years and DB2 Development for six years.

**Cyrus Tishan Mills** is an Application Solution Architect working with RBC Dexia Investor Services. Previously, Mr. Mills worked as a WebSphere Commerce Consultant for IBM Software Services for WebSphere. He holds a Master of Applied Science degree in Computer Engineering and Bachelor of Mathematics in Computer Science from the University of Waterloo. Mr. Mills also worked with the WebSphere Commerce Suite Performance Team while completing his research at IBM's Center for Advanced Studies (CAS).

**Craig Oakley** is a Technical Specialist in Online Technologies in Australia. He has 12 years experience in the online technologies. He holds a Bachelors Degree in Applied Science from Monash University, Melbourne. His areas of expertise include WebSphere, WebSphere Commerce, Web Server Technology, IP Networks and Web Server Performance.

**Jennifer Schachter** is a Software Developer in Toronto. She has two years of experience with WebSphere Commerce, both as part of the Performance team and as a member of the Marketing Team. She holds an Honours Bachelor of Mathematics in Computer Science from the University of Waterloo. Her areas of expertise include WebSphere Extended Deployment and WebSphere Commerce, Dynamic Caching. She has written extensively on best practices with WebSphere Commerce and Dynamic Caching.

**Jianwei Song** is a software developer in IBM Toronto Lab. He has five years of experience in WebSphere Commerce System Testing. He holds a Ph.D. in Mathematics and M.S. in Computer science from University of Saskatchewan,

Archived

Canada. His areas of expertise include software quality assurance, testing automation and performance analysis.

**Kevin Tobin** is a Senior IT Specialist based in Sydney, Australia. He has 25 years of experience in developing real-time computer software systems. He holds a degree in Computing and Information Systems from Monash University in Melbourne. Kevin's principle areas of expertise include WebSphere Performance Tuning and J2EE architecture, deployment and design. Kevin is a qualified Education Center for IBM Software instructor and regularly runs courses on WebSphere Application Server administration, development and other J2EE-related technologies.

**Jacob Vandergoot** is a WebSphere Commerce developer in Toronto, Canada. He has 6 years experience with the WebSphere Commerce product, mainly focusing on the framework and integration. He holds a Bachelor of Science degree from Ryerson University and is pursuing a Masters Degree in Software Engineering at the University of Waterloo.

Thanks are due to the WebSphere Application Server performance teams who assisted us enormously, and provided some of the charts used in the Redbook. And special thanks are also due to the following colleagues from IBM:

**Alex Budanitsky**
IBM Toronto. Alex helped to write the chapter on benchmarking.

**Stan Cox**
IBM Raleigh. Thanks to Stan for permission to use Figure 1-5 on page 9.

**Stacy Joines**
IBM Raleigh. Stacy Joines is an STSM with IBM Software Services. Stacy helped the team with information from her SW612 WebSphere Performance class, provided great charts for use in the book, and helped to review drafts.

**Rohit Kelapure**
IBM Toronto. Rohit Kelapure is the team lead for WebSphere DynaCache. Rohit gave us lots of help with the intricacies of DynaCache and the new features shipped with WebSphere Application Server v6. He provided us with some performance charts and reviewed the book thoroughly for us.

**Andy Kovacs**
IBM Toronto. Andy was our local manager in Toronto. This Redbook was not written in an ITSO Center, as is usually the case. Andy made our residency in Toronto possible and also a pleasant experience.

**Brian Nolan**
IBM Raleigh. Brian helped with the planning and resourcing of the book. Without his assistance and support, it would not have happened.

## Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

**ibm.com**/redbooks/residencies.html

## Comments welcome

Your comments are important to us!

We want our Redbooks to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  **ibm.com**/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept HYJ; Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

Archived

# Part 1

# Web site caching

In the first part of this Redbook we introduce Web site caching. If DynaCache is new to you, we recommend you read the first four chapters and then skip chapters 4 and 6 and go to the second part of the Redbook.

Chapter 1, Web site performance, is important especially if you haven't thought about the underlying strategies you need adopt to build web sites with good performance. Performance is not something that can be compartmentalized into a tuning task right at the end of testing a web site. It should be a main consideration before, during and after development, as no web site ever stands still.

Chapter 2, Caching, starts and the beginning - how do caches work? By the end of the chapter you should understand how to define a cache for WebSphere Commerce and how to store and retrieve items from the cache.

© Copyright IBM Corp. 2006. All rights reserved.

In Chapter 3, DynaCache invalidation, we discuss how to evict items from the cache. This is just as import as building and using the cache. If stale items are left in the cache then the web site will display wrong, out of date information.

Chapter 5, Caching strategy steps back from the mechanics of defining a cache and putting items in and getting them out. We invite you to think about what to cachem and how to design a good caching policy for a web site.

Chapter 4, Clustering DynaCache shows how to use the symmetric clustering in WebSphere Network Deployment, with DynaCache. It explains how to control cache replication across a cluster. How do you achieve the best balance between using multiple processors and moving cached items to different servers on the cluster. Appendix B, “Caching in WebSphere Extended Deployment” on page 257 takes the use of clustering in WebSphere Commerce a further step and shows how to partition a cache to make the most use of in memory caching.

Chapter 6, Advanced topics covers a number of topics including what’s new in WebSphere Commerce v6, and tuning the cache. There is also a short reference section which adds some additional usage notes to the descriptions in the Infocenter.

Chapter 7, FAQs answers some commonly asked questions about WebSphere Commerce and DynaCache.

Archived

# 1

# Web site performance

In this chapter, we describe common performance related issues encountered in typical Web-based projects and discuss the importance of planning for performance.

The following topics are discussed:

- Java 2 Enterprise Edition Web site topologies
- Common performance problems and solutions
- The importance of planning
- Performance terminology

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

## 1.1 J2EE Web site topologies and caching

J2EE-based systems commonly incorporate the Model-View-Controller (MVC) design pattern, where the responsibilities of an application are assigned to layers of code that support the user interface (View), business logic (Controller), and data access (Model). Similarly, the physical topology of the hardware that serves the application is often split into tiers:

- An edge tier made up of caching proxies and/or load balancers
- A Web tier consisting of Web servers such as Apache or IBM HTTP Server
- An application tier for managing data
- The data or persistence tier for storing and retrieving data

In these types of distributed architectures, such as the one shown in Figure 1-1, caching can be used to maximize the performance and minimize the workload of each tier, resulting in substantial performance improvements. Cached items fall into two groups:

- Static data, such as HTML, images, and Java™ Script files
- Dynamically created data that contains the output results from the runtime program execution of code components, such as servlets and JSPs.

.

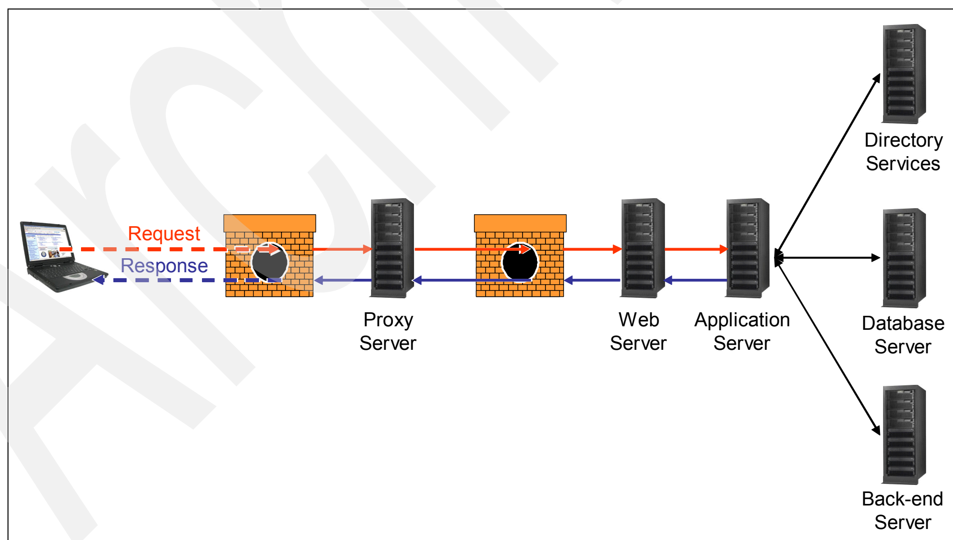



*Figure 1-1 Request/response path through various J2EE layers.*

Archived

## 1.2 Web site performance issues

Several factors can influence the performance capabilities of a Web site under stress. The following sections take you through some of the important contributing factors to consider and how caching can sometimes reduce the contributing stresses.

### 1.2.1 e-Commerce Web site types

Understand your Web site's characteristics. In general, Business to Consumer (B2C) sites derive greater performance benefits from the caching of entire HTML pages and/or HTML page fragments than do Business to Business (B2B) sites because the number of clients requesting services from a B2C site is generally much larger than that of a B2B. There are more successful cache hits per client on a well structured B2C site than on a B2B site.

B2C sites typically have many requests that target the same data and the benefits for caching that data are much higher when compared to the more personalized, often highly client-specific B2B page requests. Despite this, caching can still play an important role in a B2B site. For B2B sites, it is a case of adjusting the caching strategy to better suit the data request models and interactions occurring between the client and the site. The common caching strategy for a B2B site takes into account the fact that there are often many page fragments or parts of an HTML page that can be cached, rather than whole pages.

Having a good understanding of your type of Web site and its inherent dynamics is important when assessing the potential benefits of caching.

### 1.2.2 Unnecessary re-creation of page data

When analyzing a Web site, you will generally find pages that rarely change. Often only a small part of the page may change. Caching pages or page fragments that change on an infrequent basis improves performance significantly.

The process of identifying the cacheable fragments and developing the correct strategy will be one of the mandatory activities during your caching analysis phase.

Figure 1-2 on page 6 shows an example Java Server Page (JSP™) made up of several fragments.

Archived

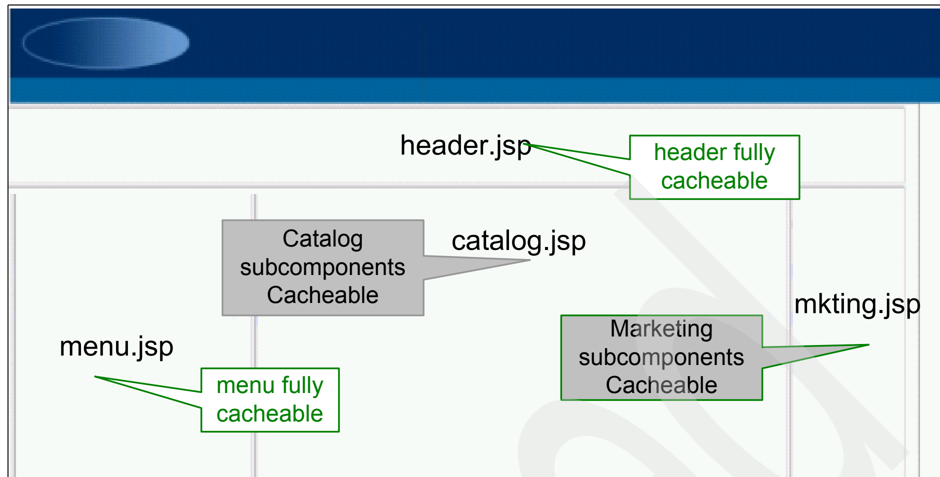



*Figure 1-2 An example parent JSP with its child JSPs fragments*

We could analyze each of the individual JSPs and assess their cacheability based on:

- What each child JSP fragment actually does
- How often the fragment changes
- The cost of generating each fragment
- Reducing the number of queries executed on the database server even if this does not reduce response times directly

Characteristically, the header, menu, and footer JSPs rarely change in most applications and therefore are ideal for caching.

Your analysis of the catalog and marketing page in Figure 1-2 should involve assessing the volatility and processing overhead required to create the data for those components. However, before dismissing these fragments as uncacheable, you should still check whether there are subcomponents within them that could be cached, such as inventory numbers displayed by the catalog.jsp or marketing information targeting specific groups in the marketing.jsp.

It may be possible to reduce the volatility of data on a page without significantly reducing the effectiveness of the page. For example, rather than displaying how may items are in stock, just display whether the item is in stock or not.

Although inventory list items change over time, they may change infrequently enough that they warrant consideration for caching. If you keep recreating parts of pages that do not change you are wasting valuable system resources.

Archived

## 1.2.3 Network traffic between tiers

The next performance item we consider is the network. As illustrated in Figure 1-3, the system network connects each tier. By caching as close to the client as we can, significant gains can be achieved by reducing the number of network hops taken while traversing through each of the remaining tiers to the database server and back to the client.

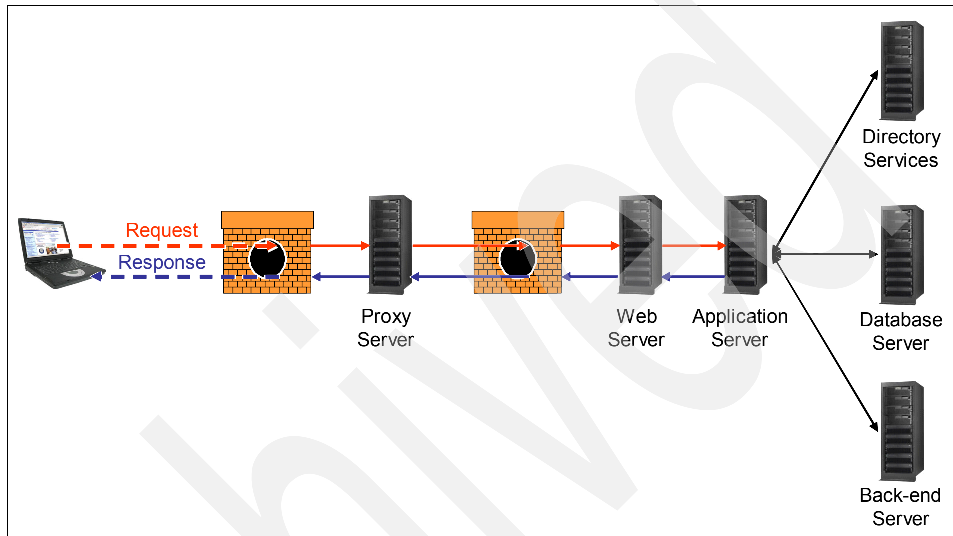



*Figure 1-3 Network of servers*

Apart from the caching aspects, it is also important that your network design is well thought out and the network is maintained in a healthy state. Many performance problems are caused directly by poorly designed network configurations. Before you engage in any DynaCache tuning exercise, ensure there are no bottlenecks in the network. You should check that:

- Server network cards are configured correctly, meaning they are set to a fixed address, and not auto detect.
- All components (routers, switches, IP sprayers, and so forth) are configured and capable of running at the correct network speed.
- TCP/IP has been configured correctly, for example, with appropriate time-out and keep-alive settings.
- You are not competing with other traffic, such as a WAN, on this network.
- The servers are on a separate subnet.

### TCP/IP keep-alive and time-out

If your site has a lot of objects (for instance images) that are sent to a user's browser during a single page request, and these objects are going to be cached, then give careful consideration to the TCP/IP keep-alive setting.

Using keep-alive can help in certain situations where you do not want to be continually tearing down the connection and then reestablishing it for each client request. Turning keep-alive on can give you a performance improvement for that individual client; this is especially important for SSL sessions. The downside is you may lose scalability in terms of the number of connections available for your other clients wanting site access. TCP/IP time-outs can be problematic and are used to determine how long it takes to detect a failed server. Correct settings may be crucial in keeping your site running smoothly during fail-over scenarios.

## 1.2.4 WebSphere Funnel model

.

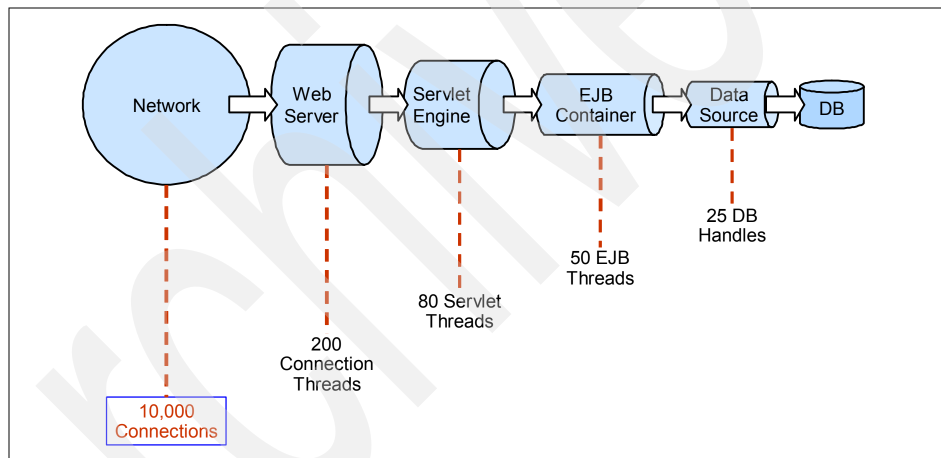



*Figure 1-4 Queue tuning points in a J2EE Web site: the Funnel model*

In Figure 1-4 decreasing values are assigned to the tuning parameters for each of the displayed Web site components. This deliberate reduction in the maximum number of available resources assigned to each successive layer is called the *Funnel model* methodology. The benefit of the funnel tuning model is that we want as many clients to connect to our system as possible, but without overwhelming the resources in each of the layers downstream (for example, database connections). The funnel helps us place these requests into various queues at each layer, where they will wait until the next layer has the capacity to process them. In summary, the funnel model helps us handle bursts of client



Archived

requests without inundating the back-end application or database servers. Figure 1-5 on page 9 provides an example of some queue tuning parameters.

> **Note:** When configuring WebSphere Commerce we recommend that the Datasource connection pool is equal to the servlet thread pool plus the number of scheduler threads plus one. Every incoming request to a Commerce system requires a connection to the database, even if caching is being used, to avoid deadlock.

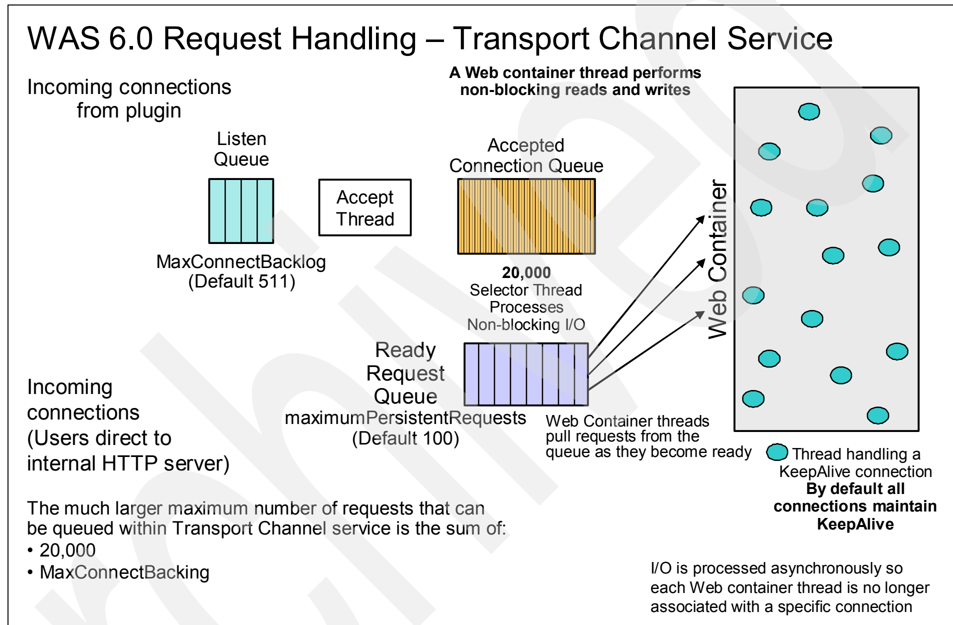



*Figure 1-5 IBM HTTP Web server request handling, queue settings, and architecture*

As shown in figure Figure 1-5, Web requests arriving at the Web server plugin are initially stored in a *Listen* queue, with a maximum depth set to MaxConnectBacklog. The MaxConnectBacklog value is a custom property configured in the application server. The Web server will hold up to MaxConnectBacklog pending connection requests for the application server, and then will reject any further requests until the application server has processed some of the outstanding connection requests.

In the application server an *Accepting* thread pulls one request at a time from the Listen queue, and stores it in the Accepted Connection Queue. The request waits in the Accepted Connection Queue until it is processed by the *Selector* thread.

Archived

In WebSphere Application Server v6 the selector threads unblock the Web container threads from the connection threads.

### *Why so many queues?*

Queues handle overflow requests. Their primary functions are to:

- Allow the HTTP requestor to make a connection
- Enable the request to wait in the queue to receive service without being lost

Queuing is especially useful for handling sudden, major increases in traffic conditions, such as when a larger than normal number of clients hit the Web site simultaneously. The downside of queuing is that it may impact your failover/outage detection capabilities. Failover detection is usually triggered by a *failure to connect*, and large queues can mask outage situations; hence delay the failover. The WebSphere Application Server infocenter describes in great detail the various queues and other parameter settings that you may need to tune. You can access this information at:

```
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.
websphere.nd.doc/info/ae/ae/tprf_tuneprf.html
```

## 1.2.5 Reduce thread/process switches and CPU load

Caching can effectively contribute to large reductions in CPU usage, particularly in the application servers and back-end subsystems. Caching can significantly free up worker threads running within the Web, application, and database tiers depending on where the caching has been implemented. By off loading work activities from these servers, the entire system copes better with much larger volumes of requests. As demand increases, cached applications generally scale better since they only have to deal with new data requests or provide update type services.

# 1.3 Project planning

The inclusion of appropriately defined caching tasks in the project planning phase is key to ensuring that your system will meet its service level agreements.

### Timing

There are a number of times during the development cycle when the capabilities and performance of DynaCache should be considered. Among those times are the following project milestones:

- At project inception when determining performance requirements

- During system design when laying out the Web pages
- At each stage of component building by the Java developers
- During regression testing to capture any introduced performance issues
- At system completion when all known defects have been corrected
- After any code changes in a mature system

### 1.3.1 Plan for caching early in the design phase

Caching is best done as a planned activity, not as something that is retro-fitted into an existing application. You do not want to be faced with having to make an inflexible, sluggish system perform better.

In the worst cases, we have seen newly developed systems that are not only frustratingly slow but start to disintegrate when placed under anything approaching projected production system loads. When this happens, it can then become a major exercise to change the application in order to remedy the problems.

Be certain to spend sufficient time analyzing the performance requirements in the early stages of development. This will help you to meet the project deadline with the delivery of a well performing system, not just a functioning system, and will ensure that you create a Web site that satisfies your end users.

The best candidates for caching are operations that are large, slow, or resource-intensive to produce. When evaluating potential caching candidates, make sure that they are publicly accessible. The more users that can take advantage of a given cache entry the better. Obviously, do not cache pages that are never reused.

Make sure developers and testers know the service level requirements of the Web pages they are designing and testing. Have them consider caching aspects at the outset, rather than retro-fitting caching considerations later in the project.

### 1.3.2 Roadmap

It is critically important to get your project off to a good start and lay down the correct performance foundation from day one.

In our experience, the lack of a performance roadmap indicates a J2EE-based project heading for trouble. Your performance roadmap must cover the salient points of performance best practice. Key aspects of this roadmap include:

- Understanding your performance objectives

Archived

- Studying existing systems or modelling proposed systems
- Planning the key milestones within the project
- Communicating the objectives and ensuring that all participants are clear on the meaning of any performance-related terminology
- Identifying the tooling that you are going to use for testing
- Identifying key personnel and skills
- Identifying skills gaps and providing for training
- Including a mandatory performance and caching section in design documents
- Mentoring developers on the plan and insisting that they do performance tests early and often
- Developing and testing caching as part of development activity

> **Note:** The development environment now includes a full test server (Rational® Application Developer v6).

### 1.3.3 System performance tuning

Performance tuning is best carried out by a subject matter expert. The important thing to cover in the planning phase is the allocation of sufficient time for this critical task. Include in your plan enough time for the execution of long running tests.

A long running test establishes, among other things, the stability of the entire system. Long running tests execute over several days and often need to be repeated. In some cases, they can only be run on weekends when systems become available, so do not overlook that possibility when determining constraints, budget, project plans, personnel, and so forth.

Full system performance tuning cannot commence until the application is complete: you cannot tune the performance of an incomplete or broken system.

System performance tuning includes the following steps:

- Performance test the completed system and establish a baseline.
- In an n-way, multi-node system, benchmark and tune the performance of an individual node using $1/n^{th}$ of the workload; then benchmark and tune the complete cluster.
- Locate bottlenecks in your system.
- List bottlenecks in order of priority and remove them in that order.
- Minimize the impact of those bottlenecks that you cannot remove.

- Once theperformance objectives have been obtained, measure the system to establish a new baseline.

### 1.3.4 Performance tuning skills

Developers need to know how to derive performance numbers from the components they are building and the importance of code path analysis.

Performance testers need to know the various tuning points within a WebSphere Application Server and Commerce site, including:

- JVM™ tuning - Minimum and maximum heap memory allocations, garbage collection, class loader options, parameter passing options, and so forth.
- Queue settings - WebSphere has many queues for buffering waiting requests, such as Servlet container thread pools, EJB™ pools, JDBC™ connection pools, RMI/IIOP buffer pools, prepared statement caches, and so forth.
- Network settings - For example time-outs, keep-alive intervals, and connection sizing.
- Database tuning.
- JMS message tuning.
- DynaCache configuration, analysis, and tuning.
- Operating system tuning.
- CPU, hard disk, and general resource monitoring.
- Thread dump analysis for troubleshooting thread contention problems.
- Memory usage/dump analysis for detecting leaks.
- Performance test tool scripting and operation.

### 1.3.5 Training

An individual or group of individuals may require training to develop the necessary skills to make configuration or application changes. Several companies offer performance training classes on WebSphere Application Server and WebSphere Commerce, and it is highly recommended that individuals in need of training attend them.

Performance tuning is a specialist profession in its own right and training is no substitute for real world experience. It takes experience just to know what “normal” performance looks like when looking at a Web site’s performance characteristics. If your company does not have a specialist performance practitioner on its staff, we highly recommend that you engage a consultant throughout the project life cycle.

If a consultant is engaged, ensure that someone from your organization shadows them as much as possible to help develop performance tuning skills in-house. Performance tuning is an on-going activity that often lasts the lifetime of an application or system.

### 1.3.6 IBM Tech-line assistance

IBM Tech-line provides a highly skilled, readily accessible team of IT specialists who can provide quality remote technical sales support to help with:

- Solution design
- Sales strategies
- Server sizing
- Technical recommendations
- Product research and positioning
- Configuration and pricing
- Upgrades, research, and configuration
- Performance and benchmark data from modelling tools

## 1.4 Performance terminology

By *performance terminology* we simply mean the fundamental vocabulary used by performance experts.

The key terms that we discuss here are:

- Response time
- Load
- Throughput
- Path length
- Bottleneck
- Scalability
- Capacity

### 1.4.1 Response time

*Response time* measures an individual user's wait for a request and is usually expressed as an average, 95th or 99th percentile.

Archived

## Average

In statistics, an average is calculated by adding together all the values in a sample, then dividing by the number of members in the sample.

## Percentile

A *percentile* is a ratio. There are a hundred percentiles in a sample. The 95th percentile is the value that 95 percent of the sample lies below. The 99th percentile is the value that 99 percent of the sample lies below. So in terms of response times, the 95th percentile is a time period, by which 95 percent of requestors have all received a response.

What is an acceptable value for a response time is generally set by de facto industry standards.

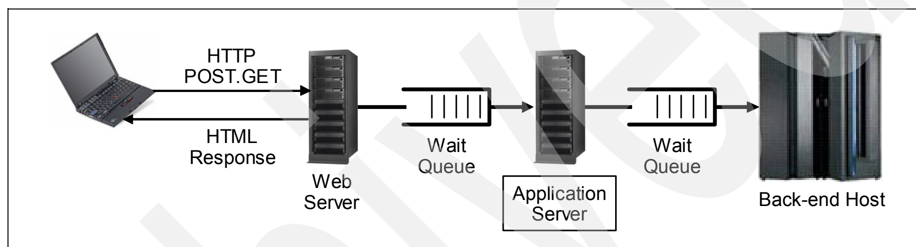



*Figure 1-6 Example of requests queuing on a Web site*

Response time is measured from the time a request is made until the HTML response is received.

Web site response time is a function of:

- Raw processing time
- Wait time at any number of queues (see Figure 1-6)
- Transfer time between multiple components

Response time is a critical measurement and poor response times result in dissatisfied customers. Many Web sites fail simply because of response time issues.

A performance practitioner considers response time:

- Under peak loading
- Under extreme loading (seasons such as Christmas, Easter, Thanksgiving, and so forth)
- Over modest dial-up connections

Archived

### 1.4.2 Load

*Load* is the pressure against the Web site and is expressed as:

- User activity
- Users arriving
- Users logging in
- Users sending requests
- Request activity
- Requests/second, pages/hour, and so forth

### 1.4.3 Throughput

*Throughput* measures things completed in a unit of time. For a Web site the most common measurement of interest is the number of HTML pages served per second.

Throughput applies to lots of concepts, not just Web sites. For example:

- Restaurants: Customers served/hour
- Bridges: Cars across/minute

### 1.4.4 Throughput plateau

*Throughput plateau* is a capacity measurement and is the maximum obtainable system output in a unit of time. In other words, it is the maximum processing rate that a system is capable of.

Consider a restaurant example where:

- The restaurant only has one server.
- It takes one minute to serve a customer.

The maximum throughput is one customer/minute.

Archived

*Figure 1-7 Throughput plateau and the use of queuing*

The throughput plateau is not a measurement of arriving requests, but how many are fulfilled. Excess requests may:

- Queue: Requests start to line up behind one another waiting to be serviced.
- Leave: The requestor gives up waiting and goes elsewhere.
- Be discarded: The request is thrown away because there are insufficient resources to process it.

## 1.4.5 Throughput saturation

When a Web site reaches maximum throughput, additional load does not yield additional throughput. In other words, the maximum throughput is a saturation point and can often be characterized as reaching a bottleneck, such as 100% CPU utilization. Figure 1-8 shows the throughput saturation graph of transactions versus users.

Archived

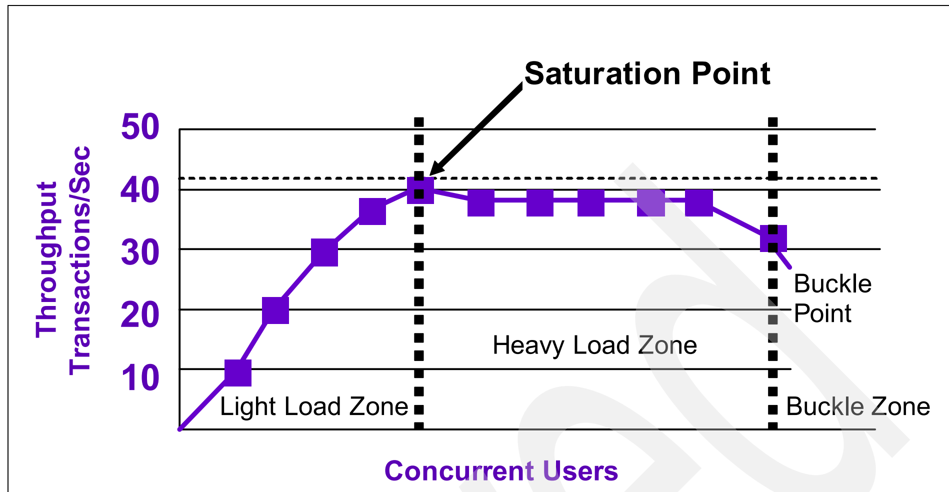



*Figure 1-8 Throughput saturation curve*

Response time is closely tied to maximum throughput. Once we reach maximum throughput:

- New arrivals begin to queue.
- "Time in queue" must be added to compute the overall response time.

A server or system may support more load beyond maximum throughput. Consider the example of a hamburger store. As shown in Figure 1-9, the hamburger customers are probably being served faster than they arrive, and only a short queue has built up as three customers arrive in a group. Queues only exist for brief bursts of activity, and the response time is stable.

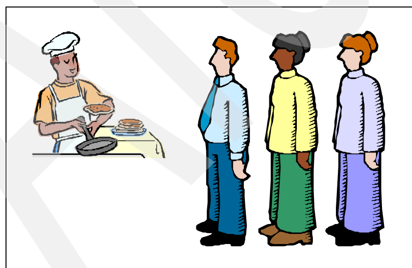


*Figure 1-9 Arrival time greater than cooking time, stable response time*

If customers arrive slightly faster than the attendant can produce the hamburgers then a queue will rapidly build up, as illustrated in Figure 1-10.

Archived

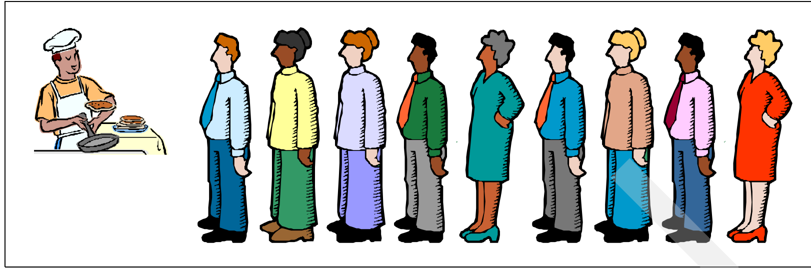


*Figure 1-10 Arrival time less than cooking time: Response time increases as wait time increases*

A server or system may support more load beyond maximum throughput by queuing requests.

Figure 1-11 shows that as the number of customers arriving increases, but can be handled, the throughput increases linearly and the response time remains stable. But when the maximum throughput is reached the response time climbs rapidly.

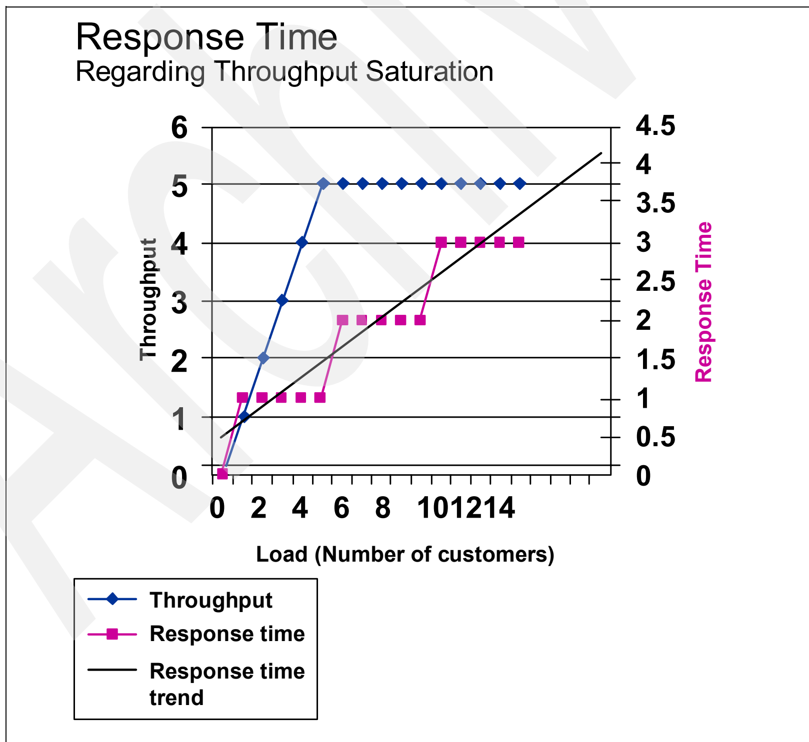



*Figure 1-11 Throughput versus response time*

### 1.4.6 Path length

*Path length* refers to the number of steps that an action takes.

Reducing the path length to speed up a Web site or application can consist of:

- Speeding up steps
- Reducing the number of steps an activity takes

For example, one optimization is called "loop *n* invariation," where you move static statements out of a loop to reduce statement execution *n* times.

The following sections present some examples of how to improve performance on Web sites by decreasing path length.

#### Consolidating Web site pages

If a site has a "greeting" page, followed by a "logon" page, then a transaction page, you could consolidate the first two pages into a greeting/logon page. The consolidated pages can then transition to the transaction page. Having fewer pages equates to less work for the servers and faster transition to the transaction page.

#### Refactoring

Another technique involves reducing the application activity path length. For example, if a particular operation consisted of:

1. Obtain connection
2. Store user information
3. Obtain connection
4. Store sale

You could refactor this into:

1. Obtain connection
2. Store user information and sale

### 1.4.7 Bottleneck

A *bottleneck* is a restriction or choke point in the system. All sites have bottlenecks of one form or another. Bottlenecks impede the performance of your site and can be caused by many things. Bottlenecks are often associated with:

- Multi-threaded and multi-user programs
- Shared resources such as a CPU, a pooled resource, or disk I/O
- Indiscriminate use of thread synchronization

Archived

- A badly tuned JVM heap
- Poor network configuration
- Poorly tuned databases, missing indexes
- Insufficient use of caching

The rule of thumb, adopted by most experienced performance practitioners, is to resolve bottlenecks in descending order of their severity. The performance mantra is that your system is a fast as your slowest component. Removing the worst blockages first yields the biggest performance benefits. It is not uncommon for the removal of one blockage to eradicate other bottlenecks entirely, simply because they were really only symptomatic of the larger bottleneck.

It is important to identify bottlenecks and resolve them, rather than building more slack into the system to cope with the effects of the bottlenecks. The slack will hide the problem for a while before it remanifests itself, perhaps more seriously.

A simple analogy of a bottleneck is a sink filling up with water. It is self evident that the plug covering the drain hole is the cause of rising water levels in a sink. Building a bigger sink will not solve this problem, yet that is what some performance novices initially attempt. They will notice that the system is running out of some resource and will react by increasing the resources' pool settings hoping that the problem with go away. As in the sink analogy, all that does is delay the inevitable, that is, we cause a larger queue of waiting requests to form, which then exacerbates the performance issue further. Finding and removing the true cause of the bottleneck is the only way forward. Then target the next biggest remaining bottleneck and apply trade-offs as to how much effort versus potential improvement gains when deciding whether to continue working on the next bottleneck or to stop the process.

In WebSphere Commerce the biggest bottleneck is almost always the database. Therefore, tuning the database and caching results can be critical factors in the success of your Commerce applications.

Archived

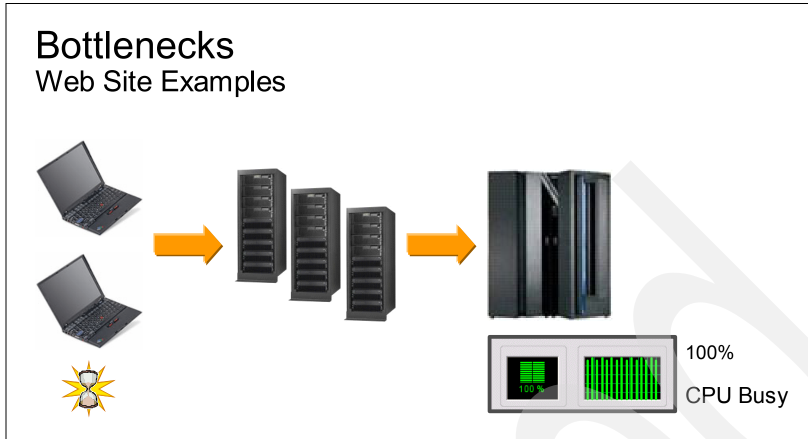



*Figure 1-12 Bottleneck caused by performance of the database*

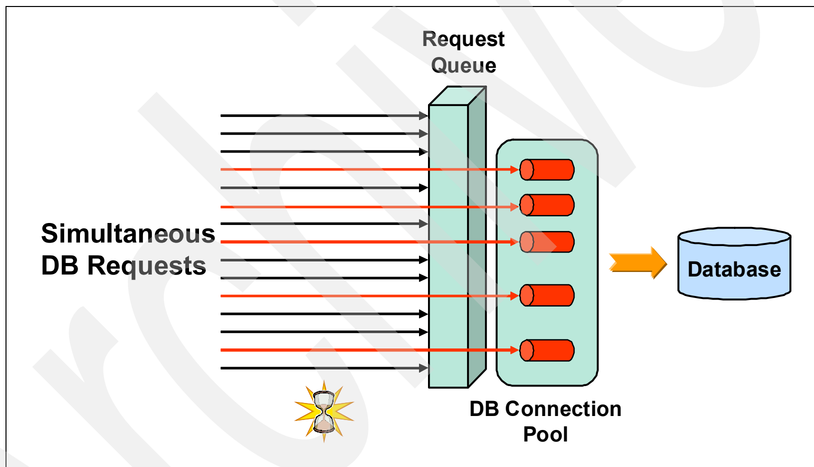



*Figure 1-13 Bottleneck caused by insufficient database connection pool size*

In summary:

- Almost every system you encounter has bottlenecks
- Eliminating every bottleneck is usually not feasible
- Not every bottleneck has the same performance impact
- Eliminate bottlenecks according to severity

The removal of significant bottlenecks is your primary activity in performance analysis.

### 1.4.8 Scalability

*Scalability* defines how easily a site can be expanded to accommodate increasing levels of service requirements. You should consider the following points:

- Sites must often expand, sometimes with little prior warning.
- Sites must be able to grow to support increased load.
- Increased loads can come from many sources, such as new markets, normal growth, and extreme peaks such as Christmas, Easter, Thanksgiving, and so forth.

Good scalability makes site growth possible and easy.

How can you grow a Web site to handle more load? You have a couple of solutions based on either vertical or horizontal scaling.

#### Vertical scaling

Vertical scaling involves adding more processors, or in our case, more WebSphere Commerces to a single copy of an operating system. However, before doing this you should ask yourself:

- Can our JVM use these CPUs?
- Can we fully utilize the CPUs we already have?
- Do we need more JVMs?

#### Horizontal scaling

Horizontal scaling involves adding more machines hosting WebSphere Commerce to the site cluster. The question posed then is “How many do we need?”. Each machine hosts new JVMs and you need to consider the impact on other parts of the infrastructure, such as:

- Network
- HTTP Servers
- Databases
- License costs

Archived

Archived

2

# Caching

In this chapter we review generic caching concepts and explain in detail what DynaCache is and how you should expect to work with it.

We discuss how caches work and how DynaCache evolved. Then we explain what items are cacheable in DynaCache and how you configure the DynaCache policy to put items into the cache and later remove them.

Finally, we introduce the Cache Monitor application and provide a tutorial that includes practical examples of using the Cache Monitor.

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

# 2.1 Caching overview

In the computer science world, a cache is a special high-speed mechanism for storing and retrieving data. Two types of caching are commonly used:

- Memory-based caching
- Disk-based caching

Caching techniques are implemented by system designers to improve application response times and reduce system load. Caching methodologies have long been used to improve the performance of Internet applications. In caching parlance, any request that can be satisfied directly by data held in a cache is termed a *cache hit*. The effectiveness of a cache is judged by its *hit rate*.

The strategies for determining what information should be kept in the cache constitute some of the more interesting problems that a J2EE architect may encounter.

Principally, the concerns of a cache designer are:

- How to get data into the cache.
- How to retrieve cached items quickly.
- What happens if cached data becomes invalid?
- How to remove invalid cache items.

## 2.1.1 How caches work

Caches have a number of working parts. The following sections describe the important components that you need to be familiar with.

### Cache identifiers

In simple terms, a caching service stores and retrieves objects from high-speed devices such as memory or hard disk. In order to be able to quickly retrieve the object, the caching system annotates each cached entry with a unique identifying string called a cache identifier or a *cache-id.* The cache-id is then stored in an index. Cache identifiers are like primary keys in a database, where the cache-id "key" is composed from one or more parts of the data being cached.

A common technique for producing a cache key is to use a hashing algorithm, where an algorithm is executed against the data and a key is produced. The Java Hash Map (java.util.HashMap) utility used for storing and retrieving Java objects is based on this concept.

Archived

In the Web world, caches often follow user-defined rules on how to construct cache-ids from information associated with an application server request (to execute a servlet, JSP, or Java command).

## Cache hits and misses

Once an object with a particular cache-id is in the cache, a subsequent request for an object with the same cache-id is served from the cache (a cache "hit"). If the object is not in the cache, then the object will need to be created, and served back to the client, as well as stored in the cache to service future requests for the object.

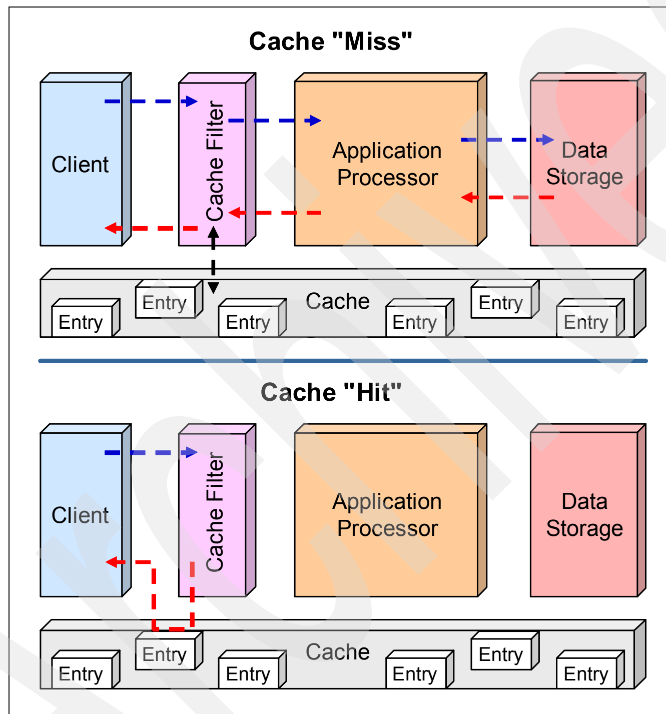



*Figure 2-1 A cache "hit" and a cache "miss"*

## Cache invalidation

Changes to your business or persistence models may impact the accuracy of related items held in cache. Items that become stale or "invalid" need to be evicted from the cache. The process of removing stale items from the cache is called *invalidation.* Note that caches generally support a number of types of invalidation mechanisms, such as rule based, time based and dependency based.

You will find that putting data objects into the cache is a fairly straightforward process. However, determining how to remove them, including managing the potentially complex web of dependencies held between related data components, will challenge most implementers. A solution architect needs to choose wisely which mechanism best fits; we discuss benefits and pitfalls later.

It is critical to understand how the various invalidation mechanisms work and the success of your caching project will be measured by how well you analyze, design, and implement your strategy. The general rule to follow is to invalidate as little as possible, without compromising the accuracy of requested data. This topic is covered in more detail in Chapter 3, “DynaCache invalidation” on page 91.

### Cache item dependencies

To complicate matters, cached items may have dependent relationships with other components held in the cache. A group of dependent items may all become invalid if any of the associated components are changed.

For example, the contents of a shopping cart held in a cache may be invalidated by changes made to an order by a customer. Furthermore, order item changes could potentially invalidate any related, cached, inventory components.

Invalid related items present us with a tricky problem. We need to discard them all at the same time. We therefore need a mechanism to evict all parties affected by change.

One solution involves grouping related items with a *dependency ID* attribute. When an item with a particular dependency ID is invalidated, then all related items that share that same dependency ID are also discarded.

Refer to Figure 2-2 on page 29 for an example of invalidation based on using dependency IDs to group objects together.

Archived

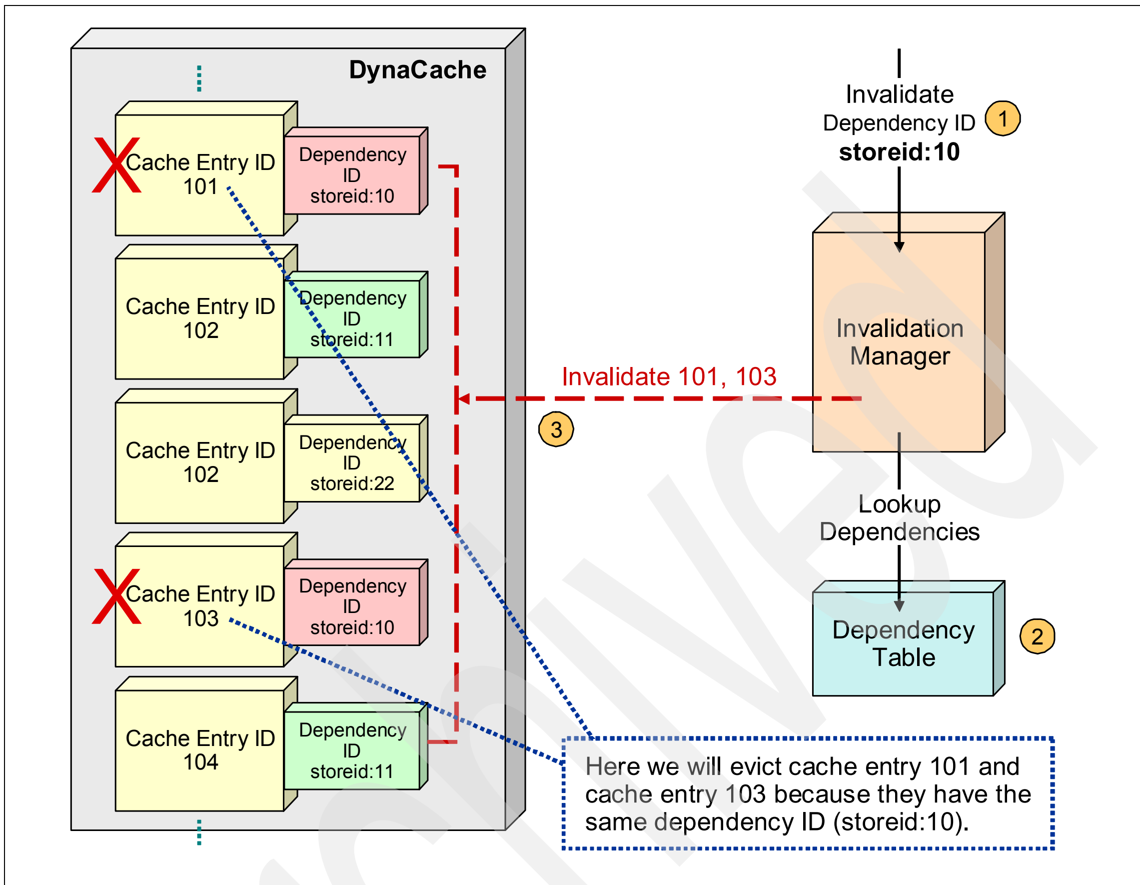



*Figure 2-2 Related items can be invalidated through the use of dependency ID*

## 2.1.2 Where caching is performed

In a typical IBM WebSphere topology, caching can be performed at several places. Some of the most notable caching locations are:

- At the Web client or browser
- At the Internet Service Provider (Akamai is an example)
- In a caching proxy server located in front of the application servers
- In the HTTP Web server (for example, static content and edge side includes)
- At the application server in DynaCache
- In the back-end database caching buffer pools

### *Client-side caching*

Caching capabilities are built in to most Web browsers today and in that case, the cache works only for a single user. For example, the browser checks if a local copy of a home page is available and if this is true, the timestamp of the local copy in the browser cache is recorded.

This timestamp will be sent to the Web server in the following HTTP GET request. The browser might request the home page by specifying the requested URI as "/". In that same home page request, the browser can use the HTTP header request field `If-Modified-Since` to indicate to the Web server that it already has a cached version that is timestamped "Sat, 10 July 200X 10:00:00 GMT."

The Web server checks the page modification time against the time specified in the HTTP request. The Web server determines that the page has not been modified since "Thurs, 6 May 200X 09:50:00 GMT," so it replies back to the browser that the page has not been modified. A return `HTTP 304` response code is used to notify that a page has not changed since the specified date.

In this example, the Web server has indicated that the page has not been modified, so the browser cache entry is current. Therefore the browser displays the page from the cache. In this case, the browser also assumes that none of the images contained in the page has been modified.

> **Note:** DynaCache does not do any processing of cache control headers.

### *Server-side caching*

Purpose built caching systems can be implemented between the client and the application server and are known as proxy servers or proxy caches. Ideally, caches are placed as close to the client as possible without compromising security. DynaCache is an example of a server-side caching mechanism.

Proxy server caches are strategically placed near network gateways in order to reduce traffic, increase network bandwidth, and lower the overall costs of internet connections. A single proxy server can easily manage many users simultaneously while maintaining cached objects derived from many sources.

Most of the benefits are derived from caching objects requested by one client for later retrieval by another client. Several proxy servers can also be joined together into a cluster or hierarchy such that any cache can request items from a neighboring cache member, the assumption being that in doing so, we can reduce the need to fetch the object directly from the source of origin.

### *Reverse proxy*

Proxy caches can be placed directly in front of a particular server. The rationale behind this is to reduce the number of requests that the server must handle, thereby leaving it free to process new requests that it has not serviced before. A front-end server such as this is called a *reverse proxy* server to reflect the fact that it caches objects for many clients, but only on behalf of one server.

## 2.1.3 The value of Web caching

Web caching involves storing HTML pages, images, servlet responses, and other Web-based objects for later retrieval. There are three significant advantages to Web caching:

- Reduced bandwidth consumption (fewer requests and responses that need to go over the network).
- Reduced server load (fewer requests for a server to handle).
- Reduced latency (since responses for cached requests are available immediately, and closer to the client being served). Together, they make the Web less expensive and better performing.

These advantages add up to a better performing Web site and therefore a better user experience.

The advantages of using DynaCache (DC) can be seen in performance tests run by the WebSphere performance team using the Trade application. Figure 2-3 on page 32 shows comparative results for using DynaCache in an EJB and JDBC application running on WebSphere Application Server 6.0.2, and Figure 2-4 on page 33 shows the comparable results for WebSphere Application Server 6.1. The Trade application used the Dynamic Mapping (DMAP) JSP interface to DynaCache.

Archived

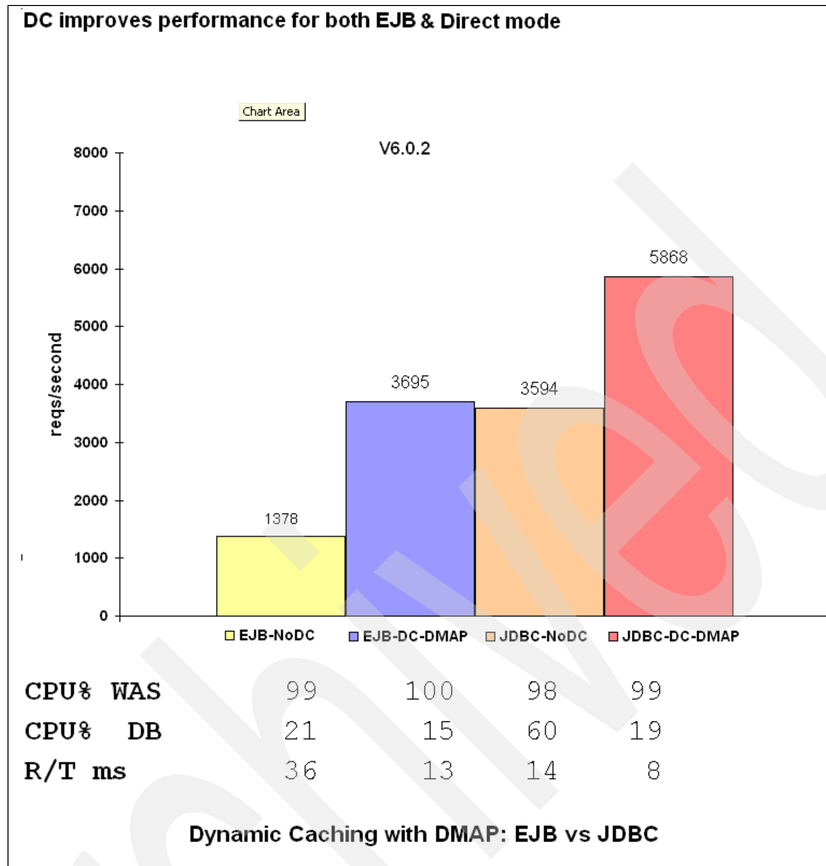



*Figure 2-3 Impact of DynaCache in WebSphere V6.02*

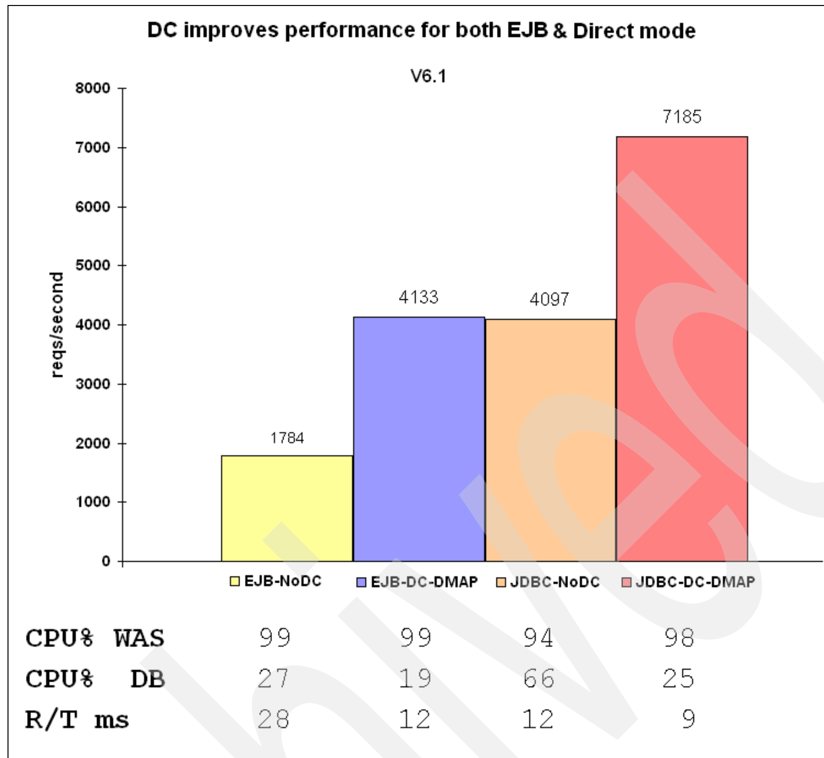



*Figure 2-4 Impact of DynaCache in WebSphere V6.1*

## 2.1.4 Static versus dynamic object caching

Most caching strategies target Web site content that rarely changes, such as graphical images and text files. However, many sites serve dynamic content, containing personalized information and data that changes more frequently.

Caching dynamic content requires more sophisticated caching techniques. The IBM WebSphere Application Server DynaCache system provides an elegant solution in terms of caching dynamic content.

## 2.1.5 Full Web page versus fragment caching

A fragment is a part or all of a rendered HTML page. Although a whole page may not be “cacheable,” it may contain sections of varying cacheability that can be separated into fragments and then cached independently. Any content that can be independently requested can be cached. This means that a fragment cannot depend on any information placed in its request scope by its parent or from other

fragments. That means, if the child object gets invalidated, the child object can be executed on its own.

### 2.1.6 Cache considerations

The DynaCache caching policy that you have or will set up for your application is critical in contributing to reduced response time and providing better end user experience. It is critically important that you carefully specify the policies so that the correct content is served out of the cache. For example, use language as part of the cache key, so that an English fragment is not served out of the cache for a French page. Not all Web site content should be cached.

You need to consider the cost of regenerating a response within a given time frame. Calculate the number of CPU cycles that are needed and the critical resources that are accessed (such as the number of database queries that are executed), and then weigh that against the reusability of the response within the window of time that a cached response would be valid. Heavy calculations that stay valid for long periods are ideal cache candidates. Heavy calculations that live for shorter periods are less beneficial as a cacheable item.

You may want to save Database Server CPU cycles at the expense of executing more Application Server cycles since it is easier to add another Application Server to a cluster than add another database server.

The reusability of the object should also be considered in terms of whether the object is specific to a user, session, store, or if it is a site-wide or publicly reusable object that is reusable across requests.

## 2.2 Introduction to DynaCache

In this section we provide a brief history of DynaCache and follow that up with a high-level overview of the technology. We then spend the rest of the chapter exploring the technical aspects of storing items in the cache. We introduce some of the cache invalidation components but leave detailed explanations for the specific chapter that deals with this important topic.

### 2.2.1 DynaCache history

For several years, IBM Research has developed and refined technologies that enable the caching of dynamic content. These technologies were implemented, deployed, and verified at various high-volume sporting event sites such as the 1998 Winter Olympic Games in Nagano. The success of the sports sites

Archived

demonstrated the feasibility of caching dynamic content and confirmed the scalability and reliability of the caching technologies.

DynaCache has evolved from this research into a feature rich, cross platform, WebSphere Application Server based caching system. DynaCache has been built into all editions of WebSphere Application Server, from Express through to Enterprise, and has existed with various degrees of capability since WebSphere Application Server version 3.5. DynaCache can easily cache several types of Web objects regardless of whether they are static or dynamic.

## WebSphere Commerce caching history

Table 2-1 shows the evolution in caching technologies used by WebSphere Commerce Business Edition (WCBE). Up until R4, Commerce used its own proprietary caching engine and did not adopt DynaCache technology until WebSphere Application Server V5.0.2 was released.

*Table 2-1 Evolution of caching in WebSphere Commerce*

| Release | R1, R2, R3 | R4 | R5 |
|---|---|---|---|
| WC Level | WCBE 5.4 | WCBE 5.6 | WCBE 5.7 |
| WebSphere Application Server Level | V 4.0.5 (R3) | V 5.0.2 | V 5.1 |
| Implementation | Proprietary WCBE | DynaCache | DynaCache |
| Storage | Disk Based | Memory based + Disk offload | Memory based + Disk offload |
| Scope | Full pages | Full pages + Fragments | Extended full pages + Fragments |
| Invalidation | Manual + Database Triggers | Manual + Database Triggers WAS API WCBE invalidation command + Time based proxy invalidation | Manual + Database Triggers WAS API WCBE invalidation command + Dynamic proxy invalidation |
| External cache | None | Edge Server/ Caching Proxy | Edge Server/ Caching Proxy |
| Configuration | <instance>.xml | cachespec.xml | cachespec.xml |
| Runtime Management | None | Cache Monitor | Cache Monitor |



Archived

## 2.3 Enabling WebSphere Application Server DynaCache

DynaCache is ready-to-go straight out of the box. It is available immediately after you have installed WebSphere Application Server, although it needs to be enabled by a simple administration switch setting. By default the service is enabled. (See Figure 2-5 and Figure 2-6 on page 37)

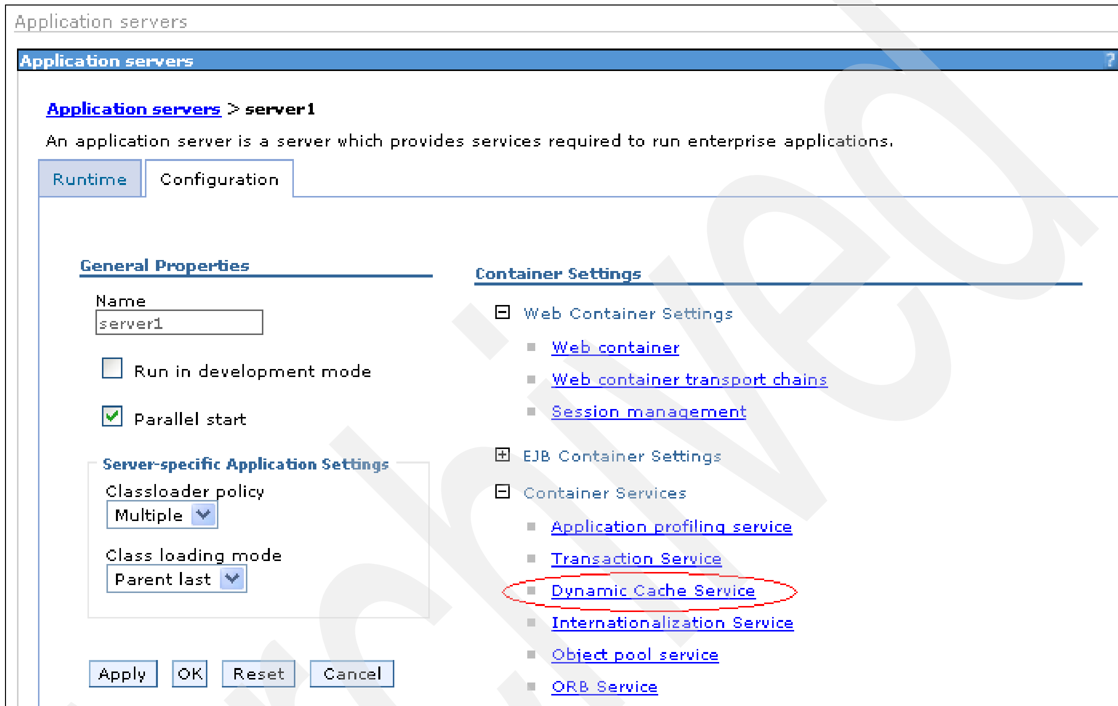



*Figure 2-5 DynaCache Service in the administration console*

Archived

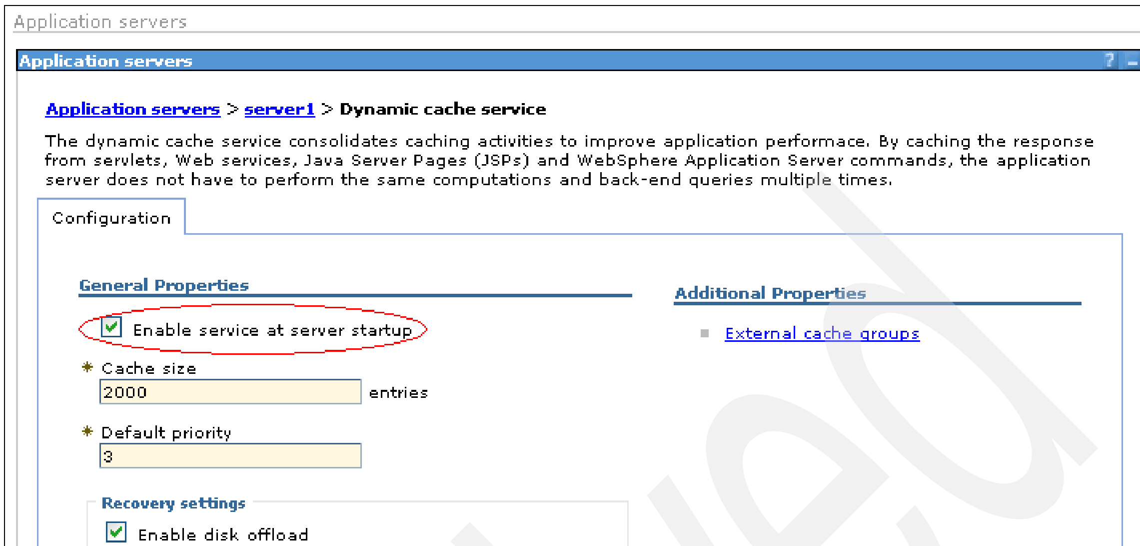



Figure 2-6 Enabling the dynamic caching service

You also need to enable servlet caching in the Web container to cache the output from servlets and JSPs. In a base installation, navigate from the home page in the Administration console to enable servlet caching, as shown in Figure 2-7, by selecting:

**Application Servers → Server1 → Web Container Settings → Web Container**

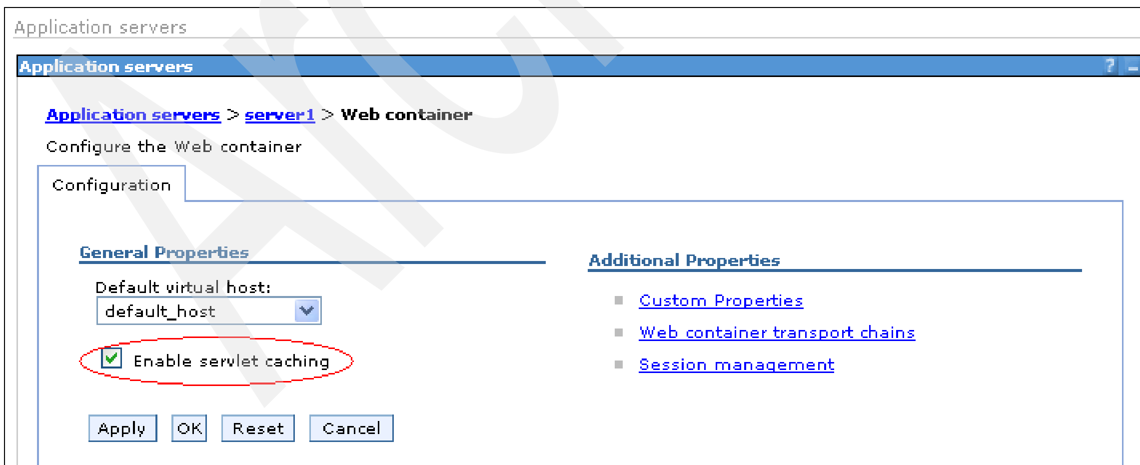



Figure 2-7 Enabling servlet caching on the Web container

In a clustered environment you need to do this for every member of the cluster. Perform the following steps:

1. **Application Servers** → ***cluster member*** → **Web Container Settings** → **Web Container**
2. Perform the enablement step.
3. Save the configuration and restart the server to make the changes take effect.

Apart from the a few administration enablement operations, there are no time-consuming installation or integration efforts required. Java classes providing support for the DynaCache subsystem are found in the DynaCache.jar file, which is part of the standard WebSphere Application Server runtime library.

## 2.4 DynaCache technical overview

Think of DynaCache as a sophisticated Java hashtable. The code used to provide the in-memory cache services extends the Java Dictionary class, which is the abstract parent of Hashtable. You configure DynaCache to store objects, and later, based on some data matching rules, DynaCache retrieves those objects and serves them from its cache. Caching rules are stored in a configuration file called cachespec.xml. Single or multiple caches are supported. Caches are stored in the JVM heap memory and DynaCache supports overflow to disk if enabled and when required.

The system administrator has some degree of control over what is placed in memory and what (if anything) overflows to disk via configuration settings. DynaCache also calls a least recently used (LRU) algorithm during the item selection process for memory-based item eviction or overflow. The LRU algorithm consults with a user-specified cache entry priority number before the evictee is chosen. Higher numbered items stay in memory longer.

Disk offload of cache entries from memory occurs when the memory cache fills up or when the server is in the process of performing a normal, administrator-directed shut down and the "FlushToDiskOnStop" property is enabled. Upon a server restart, requests for cache entries are fulfilled by reloading the saved disk data into memory.

DynaCache removes stale cache items, both as individuals or dependent, related groups. The process of removing these items is called *invalidation*. DynaCache creates a user-defined, unique "key" to store and retrieve cache items and a second, optional, shared group key for group invalidation. DynaCache also provides an API for developers to call invalidation as a runtime function.



Archived

DynaCache is supported in WebSphere clustered environments using the Distributed Replication Service.

A purpose-built DynaCache monitoring application can be installed from the <WAS 6 Root>/installableapps subdirectory found in every installation of the application server. You can use this application for:

- Cache administration
- Cachespec.xml rule debugging
- Statistics gathering
- Monitoring of the cache
- Invalidating the cache

Each cache instance is independent of and not affected by any other cache instances.

Applications running on an application server can access cache instances on other application servers as long as they are part of the same replication domain. WebSphere Application Server V5.1 DynaCache provided a feature called *cache instance*. In V6, this feature was extended and now provides two types of cache instances: *servlet cache instance* and *object cache instance*.

The *servlet cache instance* stores servlets, JSPs, Struts, Tiles, command objects and SOAP requests. It allows applications like WebSphere Portal Server and WebSphere Commerce to store data in separate caches.

The *object cache instance* is used to store, distribute, and share Java objects. The DistributedObjectCache and DistributedMap APIs are provided so the applications can interact with the object cache instances.

The DistributedMap and DistributedObjectCache interfaces are simple interfaces for the DynaCache. Using these interfaces, J2EE applications and system components can cache and share Java objects by storing a reference to the object in the cache. The default DynaCache instance is created if the DynaCache service is enabled in the Administrative Console. This default instance is bound to the global Java Naming and Directory Interface™ (JNDI) namespace using the name services/cache/distributedmap.

Archived

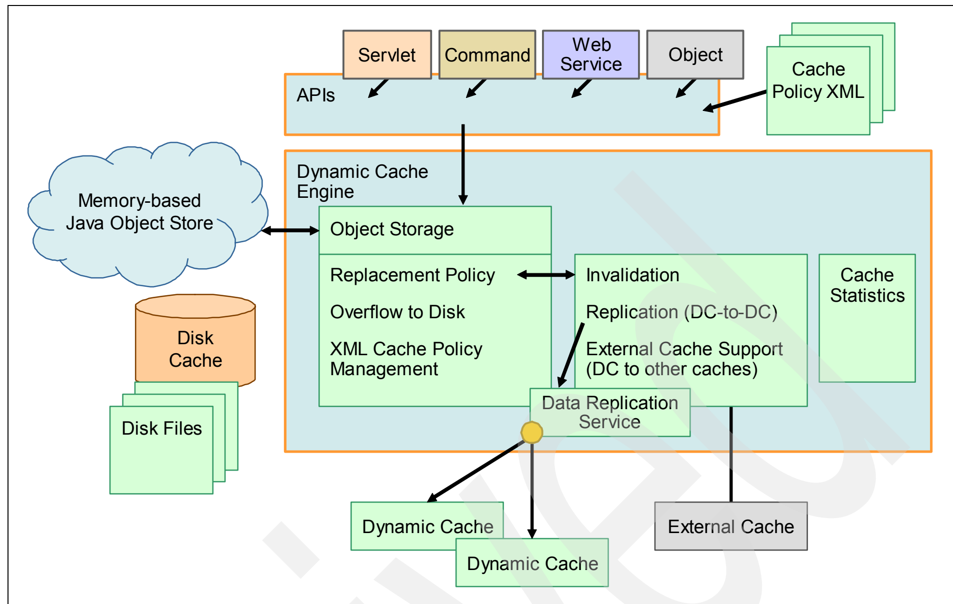


*Figure 2-8 WebSphere Application Server DynaCache overview*

The DynaCache service works within an application server Java Virtual Machine (JVM), intercepting calls to cacheable objects. For example, it intercepts calls to the servlet service() method or a Java command performExecute() method, and either stores the output of the object to the cache or serves the content of the object from the DynaCache. For a servlet, the resulting cache entry contains the output or the side effects of the invocation, like calls to other servlets or JSP files, or both.

The DynaCache loads and builds a caching policy for each cacheable object from its configuration cachespec.xml file located under the WEB-INF directory. This policy defines a set of rules specifying when and how to cache an object (that is, based on certain parameters and arguments), and how to set up dependency relationships for individual or group removal of entries in the cache.

Each data request (meaning any invocation of a cacheable servlet, JSP, Web service, or other object) is associated with a set of input parameters that are combined to form a unique key, called a *cache identifier* or *cache-id*. A *key policy* defines which cache identifiers result in a cacheable response. If a subsequent request generates the same key, the response is served from the cache. If a unique key does not match any of the rules, the response for the request is not cached.

## 2.4.1 Features of DynaCache

The main features of DynaCache are described in Table 2-2.

*Table 2-2 DynaCache features*

| Cache object | Description |
|---|---|
| Servlet/JSP results | DynaCache can cache any existing whole page or fragment generated by a servlet or JSP. DynaCache matches data found in the HTTP request header with the cache rules specified in the XML configuration file - cachespec.xml. |
| Command cache (Web Services, POJOs and EJBs) | Used to cache dynamic data before it is transformed into a presentable format (that is, HTML). These include EJB and Web service responses. A cacheable command object must inherit from the class com.ibm.websphere.command.cachableCommandImpl for it to work with DynaCache. |
| Replication support | Enables cache sharing and replication in a clustered environment. |
| Invalidation support | Includes rules-based, time-based, group-based, and programmatic cache invalidation techniques to ensure the freshness, consistency, and accuracy of the content. |
| Edge of network caching support | Extends the WebSphere Application Server caches into network-based caches, through the implementation of external cache control and distributed-fragment caching and assembly support. |
| JSR168 compliant portlets | DynaCache can cache any existing whole page or fragment generated by a servlet, JSP or a JSR 168 compliant Portlet[a]. DynaCache matches data found in the HTTP request header with the cache rules specified in the XML configuration file, cachespec.xml. |
| Cache monitor | Cached results can be viewed and managed via a cache monitor J2EE application. The cache monitor provides a GUI interface to change the cached data manually. You have to install the application manually using the CacheMonitor.ear file (located in the <WAS_HOME>/installableapps directory) using the WAS Admin GUI.<br>This installable Web application provides a real-time view of the state of the DynaCache. The only way to externally manipulate the data in the cache is by using the cache monitor.<br>After installation, the cache monitor can be accessed as: http://your host_name:your port_number/cachemonitor. |

a. Support added in WebSphere Application Server v6.1.0.0

Archived

**Note:** We recommend taking great care if the cache monitor is enabled in production not to clear the cache using the monitor. If the cache is cleared in a production environment, it can lead to instability or poor performance until the caches are repopulated. If environment access is not controlled, we recommend not installing the cache monitor in production to eliminate any inadvertent use of cache clearing.

## 2.5 Servlets and DynaCache

In this section we discuss

- Servlet technology
- Requests and their attributes
- Servlet filters
- The caching filter in WebSphere Commerce
- JSP includes and forwards

### 2.5.1 Servlet technology

*Servlets* are Java classes that specifically handle HTTP requests and responses. A request object is delivered to the service method of a servlet. The service method determines which HTTP operation is required and calls the servlet method that handles that operation. The most common HTTP operations are GET and POST, which are handled by the equivalent servlet methods doGet() and doPost().

Both doGet() and doPost() accept the same two parameters:

- An HTTP request object
- An HTTP response object

### 2.5.2 Request attributes

A servlet determines what processing is required by examining the contents of the request object. The data sent as part of the request is called the *request attributes*. After processing the request, a servlet generates the HTTP response object (HTML, cookies, and so forth) and sends it back to the client.

Servlets also store request attributes or other state information in a memory storage area called the HTTP Session object. This data can be used between client calls to track the current status of an individual client’s transactions.



Archived

## 2.5.3 Servlet filters

Servlets are preprocessed by filter objects when pre-pended at runtime or declaratively. Servlet filters are chained together to perform multiple operations on requests or responses associated with any targeted servlet. Each servlet filter normally has a specific task, such as providing security checks, auditing, or performing some kind of data validation or transformation. For details, see:

`http://java.sun.com/blueprints/corej2eepatterns/Patterns/InterceptingFilter.html`

The WebSphere Web container hooks the inbound request and outbound responses from servlets that you have selected for caching and presents them to DynaCache. The container does not use the servlet filter mechanism but rather uses an internal mechanism to do this.

## 2.5.4 WebSphere Commerce caching filter

Commerce provides a servlet cache filter to construct the cache ID from session information. The servlet cache filter sets up Http request attributes from stored session data. The DynaCache uses the request attributes as component elements in constructing the cache ID. The caching filter is not a part of DynaCache technology and its primary purpose is to enable WebSphere Commerce specific data to be processed by the DynaCache filter.

Technically, the "caching filter" extracts data variables from the HTTP session object and places them into the HTTP request prior to delivery to the DynaCache filter. Doing so allows these variables to be referenced in the cachespec and therefore used to augment the level of granularity for caching Commerce data objects.

## 2.5.5 JSP includes and forwards

Suppose you have some common HTML code that you want to appear on every page, such as a navigator or header. You could copy the code into each HTML and JSP file, but if it changed, you would have to find all the files that used it and update each of them. It would be much easier to have one copy and include it everywhere you need it – and JSP developers do exactly that.

### JSP include

The basic mechanism is to use a <jsp:include> with a PAGE attribute naming the page to be included, and end with </jsp:include>.

A `flush` attribute is also required, and it must have the value `"true"`. Once you use an include in your JSP, the contents of the output are written. Therefore, you

can no longer do anything that involves sending HTTP headers, such as changing content type or transferring control using an HTTP redirect request.

Examples of jsp:include and includes in JSTL and in Struts are provided in section 5.5 “DynaCache and JSP” on page 128.

### JSP forward

The jsp:forward request is similar to a jsp:include, but you cannot get control back afterwards. The attribute `flush="true"` is required because once you execute the include, you have committed your output. Prior to the include, the output might all be in a buffer; therefore, you can no longer do anything that might generate headers, including setContentType(), sendRedirect( ), and so on.

An alternate include mechanism is <%@include file="filename"%>. This mechanism is slightly more efficient (the inclusion is done at the time the JSP is being compiled), but is limited to including text files (the file is read, rather than being processed as an HTTP URL. The <jsp:include> can include a URL of any type (HTML, servlet, JSP, CGI, even PHP or ASP).

## 2.6 Configuring DynaCache using XML-based policies

The sample WebSphere Commerce application ConsumerDirect is used throughout this Redbook to help explain various aspects of DynaCache. Figure 2-9 on page 45 shows the ConsumerDirect store catalog display home page.

Archived

*Figure 2-9 Section of the store catalog display home page in the ConsumerDirect application.*

The process of caching Commerce Applications involves studying the application thoroughly and determining which pages are the best candidates for caching. Other chapters of this Redbook guide you through the process of how to select the best candidates for caching. In this section we focus on how to cache once you have identified your target components.

### 2.6.1 Basic structure of the cachespec.xml file.

To start caching the Consumer Direct application you first need to create a cache specification file called *cachespec.xml.* A schematic view of cachespec.xml is shown in Figure 2-10 on page 46.

Archived

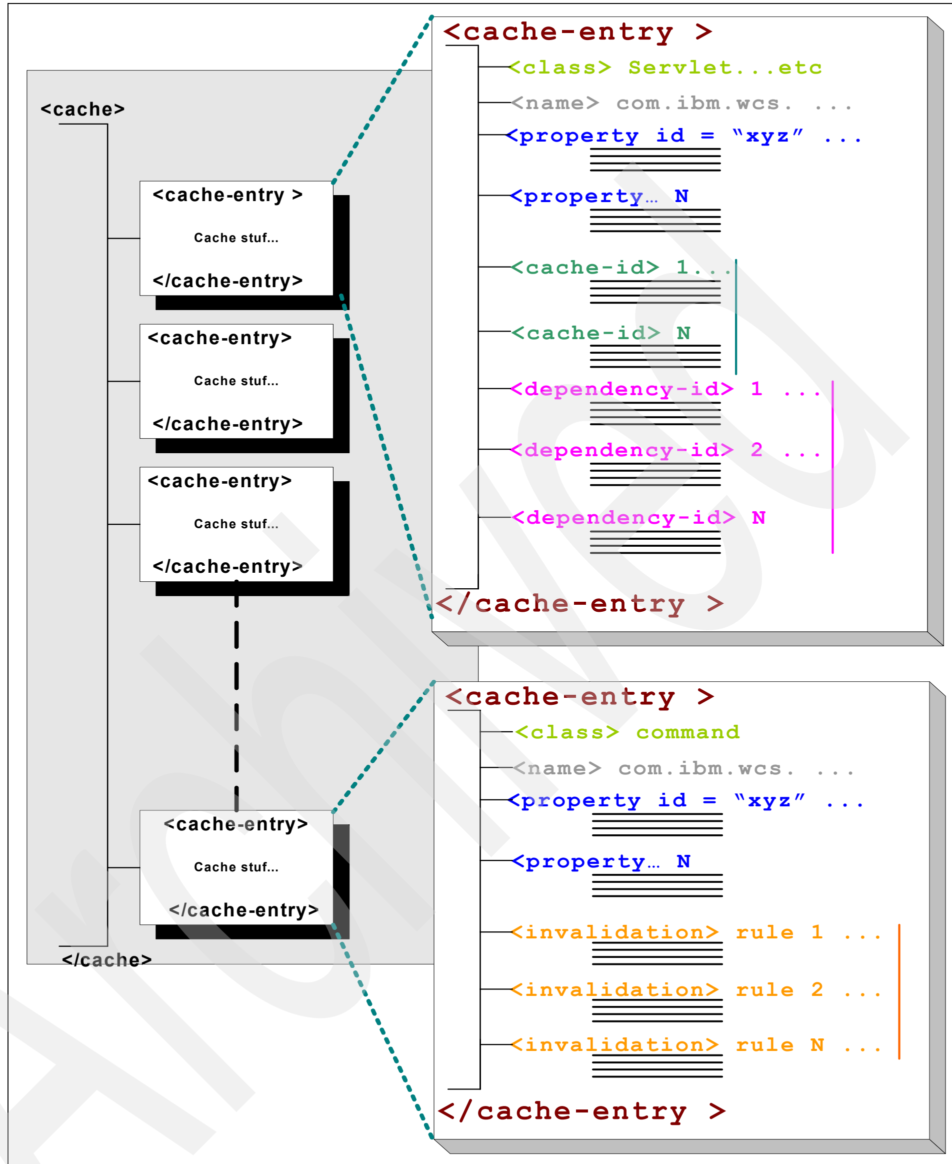



*Figure 2-10 General structure of the cachespec.xml file*

The cachespec.xml file contains configuration entries for your caching definitions and rules and is often referred to as the *cachespec*. Example 2-1 on page 47 shows a typical cachespec.

*Example 2-1 A simple cachespec.xml file with one JSP cache entry*

```
<cache>
   <cache-entry>
      <class>servlet</class>
      <name>/StoreCatalogDisplay.jsp</name>
      <property name="save-attributes">false</property>
      <property name="store-cookies">false</property>
      <timeout>3600</timeout>
      <priority>3</priority>
      <cache-id>
         <component id="storeId" type="parameter">
            <required>true</required>
         </component>
         <component id="catalogId" type="parameter">
            <required>true</required>
         </component>
      </cache-id>
      <dependency-id>storeId
         <component id="storeId" type="parameter">
            <required>true</required>
         </component>
      </dependency-id>
      <invalidation>storeId
         <component id="action" type="parameter" ignore-value="true">
            <value>update</value>
            <required>true</required>
         </component>
         <component id="storeId" type="parameter">
            <required>true</required>
         </component>
      </invalidation>
   </cache-entry>
</cache>
```

In WebSphere vernacular, the cachespec is a deployable, XML policy configuration file that allows you to specify:

- What is going to be cached (Servlets, JSP, Java commands, and so forth)
- Where it is going to be cached (memory or disk)
- When cache items are to be evicted (invalidation)
- How cache entries are related (invalidation dependencies)

Store the cachespec.xml file in the WEB-INF directory of your Web module. Changes to the cachespec are automatically picked up by DynaCache.

Cachespec.xml can be stored in the WebSphere Application Server properties directory. A cachespec in the property directory defines rules that are global to all applications. It is uncommon to do this because caching policies are normally application specific.

In the cachespec there a few important high-level XML elements that you will need to master. They are:

- The cache entry element: <cache-entry>
- The cache ID element: <cache-id>
- The dependency ID element: <dependency-id>
- The invalidation rule element: <invalidation>

To introduce you briefly to these XML artifacts, we start off at a high level and then work our way through the general cachespec structure. Later, we build on that knowledge by taking each of these XML components in turn and exposing more of the detail.

## 2.6.2 Cache entry element overview <cache-entry>

When enabled by an administrator, the WebSphere Application Server DynaCache service parses the cachespec.xml file during startup, and extracts a set of configuration parameters from each <cache-entry> element.

A cachespec.xml file can have one or several cache entries placed in it. Each <cache-entry> element defines how we are going to cache some object on our Web site. An object could be a servlet, a JSP, or some other Java-based object such as an EJB or Web service.

Each <cache-id> element defines a rule for caching an object and is composed of the following sub-elements:

- Component
- Timeout
- Inactivity
- Priority
- Property
- Idgenerator
- Metadatagenerator

The <component> sub-element can appear many times within the <cache-id> element. Each time it specifies how to generate a component of a cache ID. There are several different types of component elements, such as:

- Parameter
- Session
- Attribute

Archived

- Locale
- Method
- Field

The <timeout>, <priority>, and <property> sub-elements can be used to control the cache entry expiry, cache eviction policy, and other generic properties for a cached object with an identifier generated by its enclosing <cache-id> element.

Each cache entry has its own properties, including things like its:

- Sharing policy for clustered environments (which takes priority over the sharing policy set for the replication domain)
- Invalidation Time-To-Live (TTL) value
- Single or multiple cache ID rules

Therefore, in each cache entry, you will find one or more cache identifiers that uniquely identify what is to be cached. Beyond the list of properties, you may encounter dependency or invalidation rules that tell DynaCache how and when to remove an item and any group of dependent items from the cache.

The following pattern, expressed in pseudo XML, typically repeats for each cache-entry you encounter:

```
<cache-entry>
   < name and properties>
   < list of cache-id‘s>
   < list dependency-id’s> optional
   < list of invalidation rules> optional
</cache-entry>
```

### 2.6.3 Cache ID Overview

As the DynaCache service places objects in the cache, it labels them with unique identifying strings (cache IDs) constructed according to rules that you specify inside your cache entry <cache-id> elements and sub-elements. Example 2-2 shows an actual cache ID generated by DynaCache.

*Example 2-2 An internal cache ID generated by DynaCache.*

```
/webapp/wcs/stores/ConsumerDirectATP/include/styles/style1/CachedFooter
Display.jsp:storeId=511:DC_userType=G:DC_lang=-1:UTF-8:requestType=GET
```

For now, ignore the Commerce-specific, DC_x type parameters that appear in Example 2-2; they are covered later. Think how DynaCache built this ID.

Archived

You can probably deduce what happened: the ID generation logic appends the full JSP path, selected JSP parameters and their values, and finally the HTTP request type GET into a single text string. You also probably noticed that the appended string items are separated from one another by a colon. Together, these aggregated strings form the JSP's unique cache lookup key.

How does the ID generator know how to do this, meaning which parts to include when building cache IDs? It is the cachespec.xml file definitions that tell DynaCache what parts or "components" are to be used in the assembly process when forming various ID strings. Figure 2-11 shows the CachedFooterDisplay.jsp output content as stored in the DynaCache[1].

*Figure 2-11 Cached JSP fragment CachedFooterDisplay.jsp*

Figure 2-12 on page 51 shows diagramatically how a cachespec entry is constructed from filtering a URL.

[1] Note: the image is not displayed because it is stored as a separate fragment.

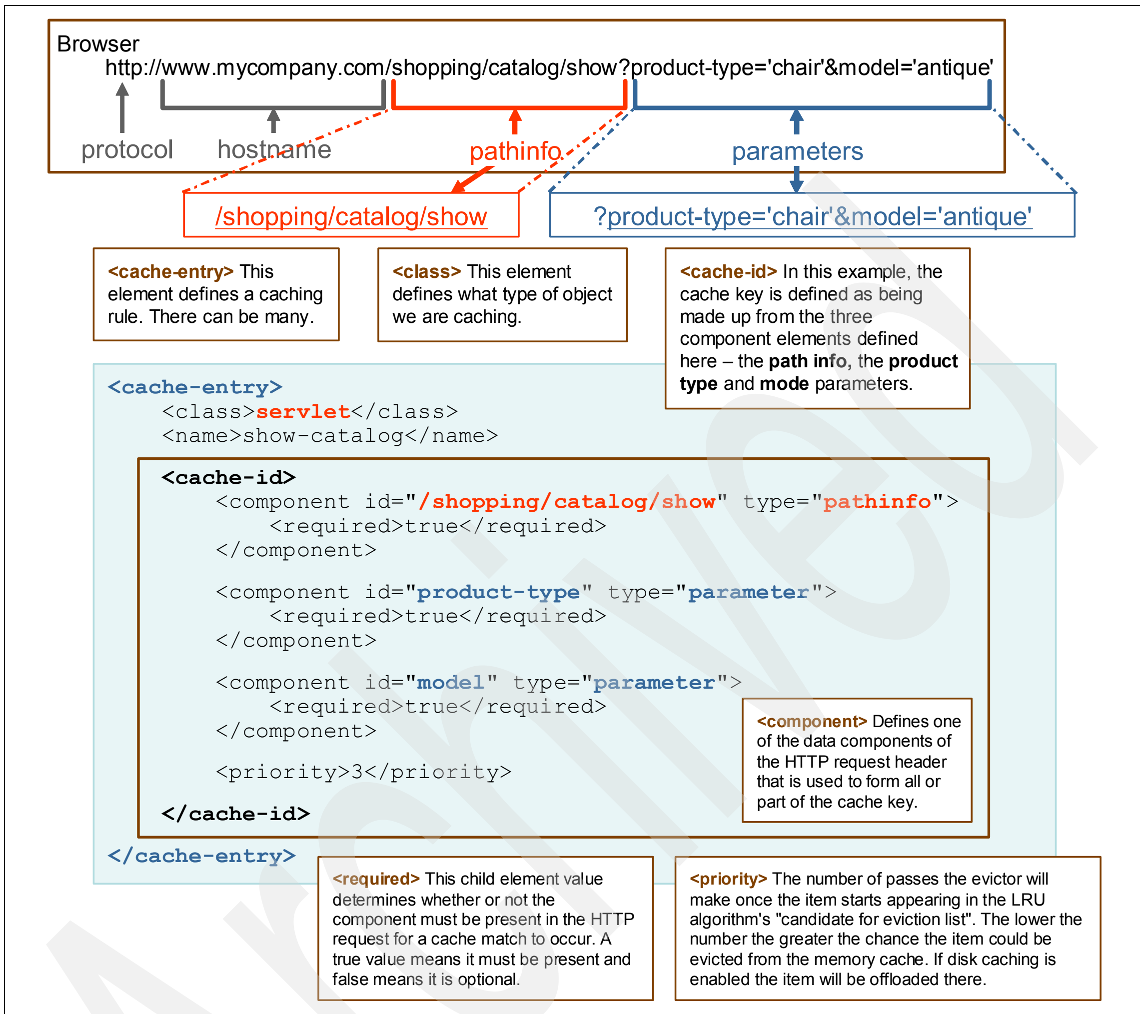



*Figure 2-12 Defining cachespec entries (JSP example)*

## 2.6.4 Cache IDs and the cache hit

Once a cache ID is safely stored in the cache, any subsequent requests that match with the cache ID are served from the cache (a cache “hit”). The <cache-id> rules define how to construct cache IDs from information associated with a client HTTP request.

In Example 2-3, we provide an example of a cache entry that will cache the output from the StoreCatalogDisplay JSP. The example shows two parameter values (storeid and catalogId) that together with the name of the JSP will form

the unique cache ID that is used to store and retrieve the object from cache. Requests made to the StoreCatalogDisplay JSP containing previously unseen storeid or catalogId parameter values will result in the creation of a new cache entry.

*Example 2-3 Cache entry defining a cache id*

```
<cache-entry>
   <class>servlet</class>
   <name>/StoreCatalogDisplay.jsp</name>
   <property name="save-attributes">false</property>
   <property name="store-cookies">false</property>
   <timeout>3600</timeout>
   <priority>3</priority>
   <cache-id>
      <component id="storeId" type="parameter">
         <required>true</required>
      </component>
      <component id="catalogId" type="parameter">
         <required>true</required>
      </component>
   </cache-id>
</cache-entry>
```

## 2.6.5 Cache programming support

DynaCache provides other capabilities in addition to servlet and JSP caching. Cache IDs also define how information is obtained programmatically from Cacheable Command objects, that is, objects that implement the DynaCache Command interface. There is more coverage on Command objects later.

In summary, for an object to be cached, DynaCache must know how to generate a unique ID for different invocations of that object.

Cache IDs can be developed in one of the following ways:

- Using your XML <cache-id> element definitions inserted into the cache policy of a cache entry.
- Writing custom Java code to build the ID from input variables and system state. The custom Java code is placed in a WebSphere shared library so that it can be accessed by the runtime. Custom ID generators, although not common, are useful when application processing “state” is a factor in creating the identifiers.

Although each cache entry may have multiple cache-ID rules, only one of the entries in the list of rules is executed at any one time. DynaCache searches through the list of cache IDs in the exact order you defined them in the cachespec.xml file. It keeps looking for a rule until it finds one that matches, or it exhausts the list.

In the case where none of the <cache-id> generation rules produce a matching cache ID, then the object is not cached.

## 2.6.6 Dependency ID overview <dependency-id>

Dependency ID elements are used to ensure related cache items that become out-of-date as a group are all evicted as a group. This process is known as invalidation. Each related cache item shares the same dependency ID, so it only takes one member of the dependency group to get invalidated, for the rest of the group to be evicted.

The dependency ID can be as simple as just a name, such as “storeId,” for example, `<dependency-id>storeId</dependency-id>`. In this example, `storeId` can be referred to as the *base string*.

Dependency ID definitions are often more complex. You can specify additional sub-components that are appended one-by-one to the dependency to create a more refined invalidation key.

Suppose you want to invalidate only selected pages that apply to a particular store. The dependency ID will need some way of identifying that store. You implement this by using <component id> tags. Each <component id> tag declares additional string data that is appended to the base name string to form the final, fully qualified dependency ID string. Example 2-4 on page 54 shows an actual dependency ID taken from the ConsumerDirect cachespec.xml file. Figure 2-13 on page 55 shows a snapshot of dependency IDs viewed with the cache monitor.

Archived

*Example 2-4 Dependency ID: Sample cachespec definition*

```
<dependency-id>storeId
   <component id="" ignore-value="true" type="pathinfo">
      <required>true</required>
      <value>/StoreCatalogDisplay</value>
      <value>/TopCategoriesDisplay</value>
      <value>/CategoryDisplay</value>
      <value>/ProductDisplay</value>
   </component>
   <component id="storeId" type="parameter">
      <required>true</required>
   </component>
</dependency-id>
```

Note that if any of the required <component> elements are missing, (meaning the parameters that you specified as required) then the dependency ID rule is discarded by the runtime engine. Multiple <dependency-id> rules can exist per cache entry. All dependency ID rules execute separately. We cover dependency IDs in greater detail in the invalidation chapter.

Archived

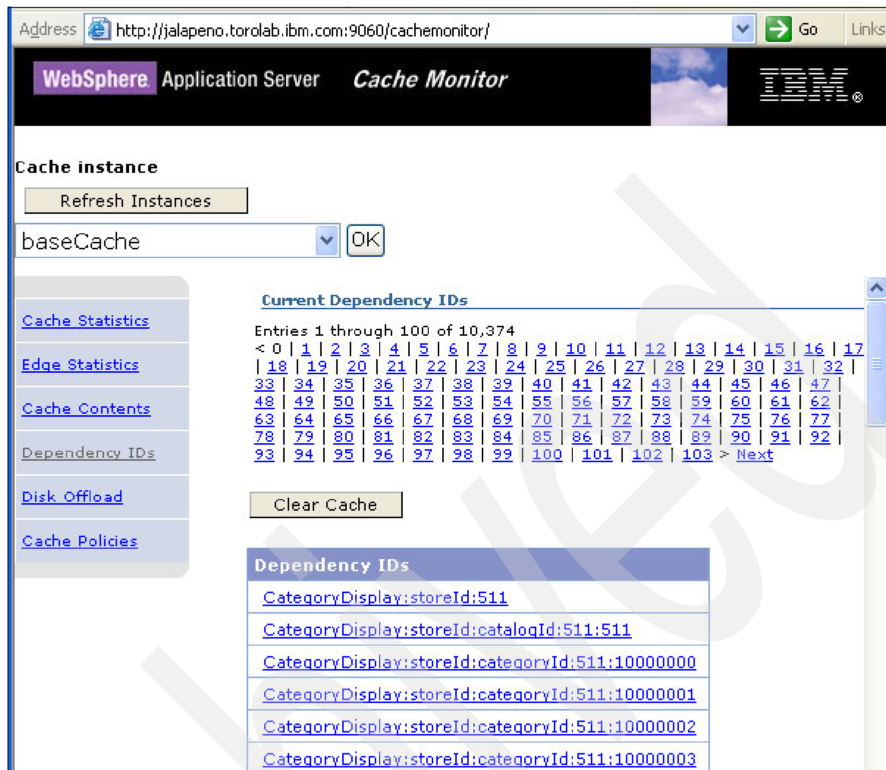



*Figure 2-13 Sample dependency IDs generated from cachespec definitions*

## 2.6.7 Invalidation rules overview <invalidation>

*Invalidation rules* inform DynaCache when and how to remove objects from the cache. They are defined in exactly the same manner as dependency IDs and use the <dependency-id> definitions to work out which objects to evict.

Example 2-5 shows a snippet of the command-based invalidation rules used in the sample ConsumerDirect application.

*Example 2-5 Extracted samples of invalidation rules used in ConsumerDirect application*

```
<cache-entry>
<class>command</class>
<sharing-policy>not-shared</sharing-policy>
<name>com.ibm.commerce.catalogmanagement.commands.AddCatalogDescCmdImpl
</name>
<name>com.ibm.commerce.catalogmanagement.commands.UpdateCatalogDescCmdI
mpl</name>
```

Archived

```
    <!-- StoreCatalogDisplay Invalidation -->
    <!-- ******************************* -->
    <invalidation>StoreCatalogDisplay:storeId
       <component id="getStoreId" type="method">
          <required>true</required>
       </component>
    </invalidation>
    <!-- ******************************** -->
    <!-- TopCategoriesDisplay Invalidation -->
    <!-- ******************************** -->
    <invalidation>TopCategoriesDisplay:storeId:catalogId
       <component  id="getStoreId" type="method">
          <required>true</required>
       </component>
       <component  id="getCatalogId" type="method">
          <required>true</required>
       </component>
    </invalidation>
    <!-- ******************************** -->
    <!-- CategoryDisplay Invalidation      -->
    <!-- ******************************** -->
    <invalidation>CategoryDisplay:storeId:catalogId
       <component  id="getStoreId" type="method">
          <required>true</required>
       </component>
       <component  id="getCatalogId" type="method">
          <required>true</required>
       </component>
    </invalidation>
...
```

### 2.6.8 Command-based invalidation

Invalidation rules are activated by the execution of a Java command, which you can write yourself. A Java command class extends from the WebSphere Command Framework API. Invalidation IDs for command-based invalidation are constructed based on methods and fields provided by the commands.

The values of the <name> elements in Example 2-5 are two classes that belong to the Java package: com.ibm.commerce.catalogmanagement.commands.

The two classes are:

- AddCatalogDescCmdImpl
- UpdateCatalogDescCmdImpl

Both of these catalog management classes implement the command-based interfaces defined in the command framework. DynaCache hooks into the methods defined in the command interface. Command execution is carried out in a method called performExecute(), which the latter classes must implement.

Both of these catalog management classes update the ConsumerDirect application's catalog descriptions. In this example, the cache designer has created invalidation rules that run based on either of these classes having their performExecute() methods called. By default, the invalidations occur prior to the performExecute() command actually being called; however, you can change it to occur after performExecute() returns if necessary using the delay-invalidations property which we discuss later.

Example 2-5 on page 55 specifies a policy to invalidate any cache entries that are identified by the same storeId and catalogId. DynaCache intercepts calls to the AddCatalogDescCmdImp and the UpdateCatalogDescCmdImpl commands (which add or update a catalog description entry) and generates the invalidation ID based on the value returned by the getStoreId() and getCatalogId() methods. Upon execution of the commands, the DynaCache compares the invalidation IDs with each of the dependency IDs. Any cache entry for which any of its dependency IDs matches with any of the invalidation IDs is removed.

The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules execute separately.

If you take a look at the snippet of XML that performs the CategoryDisplay invalidation in Example 2-6, you can see that base ID (defined by the <invalidation> tag) is CategoryDisplay:storeId:catalogId and there are two further components making up the ID. The two components are the returned string results obtained by calling the methods getStoreId() and getCatalogId().

With a store ID of 511 and a catalog ID of 8003, an invalidation ID of `CategoryDisplay:storeId:catalogId:511:8003` is created.

*Example 2-6* CategoryDisplay invalidation example.

```
<!-- ********************************* -->
<!-- CategoryDisplay Invalidation      -->
<!-- ********************************* -->
```

Archived

```
<invalidation>CategoryDisplay:storeId:catalogId
   <component  id="getStoreId" type="method">
      <required>true</required>
   </component>
   <component  id="getCatalogId" type="method">
      <required>true</required>
   </component>
</invalidation>
```

### 2.6.9 Delay-invalidations

For performance reasons DynaCache is not transactional. In terms of data integrity, DynaCache is intentionally “best-effort” so that it can execute quickly. As a consequence, the caching of incorrect data can occur under certain rare conditions involving command-based invalidation.

In command-based invalidation, any data update command will call invalidations before the performExecute() method is called. Since DynaCache is multithreaded, it is possible that another data fetch command could potentially read data immediately after an update command finishes its invalidation calls, but before the update command commits its changes to the back-end database.

This is a problem. The fetch command would see the data that it has just read (which is actually stale) is not in the cache, and so it places it back in there. The scenario results in cache entries with stale data that do not get invalidated as intended.

To circumvent this problem, the delay-invalidations property is set in the cache policy. The delay-invalidations is used to delay command invalidations until after the performExecute() method. You use delay-invalidations at the cache-entry level in a policy for a command resource to delay all invalidations done by it. Example 2-7 shows a cache entry using the delay-invalidations property.

*Example 2-7 delay-invalidations sample definition*

```
<cache-entry>
      <class>command</class>
      <name>UpdateCommand</name>
      <cache-id>
         <component id="userGroup" type="field" />
         <component id="getUserNumber" type="method" />
      </cache-id>
      <invalidation>USER
         <component id="userGroup" type="field" />
         <component id="getUserNumber" type="method" />
```

```
      </invalidation>
      <invalidation>GROUP
         <component id="userGroup" type="field" />
      </invalidation>
      <property name="delay-invalidations">true</property>
</cache-entry>
```

In the example, invalidation IDs for both "USER" and "GROUP" basenames are created and called after UpdateCommand executes. If the delay-invalidations property is not set, the invalidation IDs are created and invalidations occur before the command executes.

## 2.6.10 The effect of updates to the cachespec.xml file

Modify the cachespec.xml at any time and the caching service responds to changes in the cachespec.xml file without the need for a server restart.

When new versions of the file are detected, the old policies are replaced. Objects cached through the old policy file are not automatically invalidated from the cache; they are either reused with the new policy or eliminated from the cache through the replacement algorithm.

If you are caching static content and you are adding the cache policy to an application for the first time, you must restart the application. You do not need to restart the application server to activate the new cache policy.

### Cache entry order is important

For each of the three IDs (cache, dependency, invalidation) generated by cache entries, a <cache-entry> contains multiple elements. The DynaCache runs the <cache-id> rules in order, and the first one that successfully generates an ID is used to cache that output.

If the object is to be cached, each one of the <dependency-id> elements is run to build a set of dependency IDs for that cache entry.

Finally, each of the <invalidation> elements is run, building a list of IDs that the DynaCache invalidates, whether or not this object is stored in the cache at the time the invalidation rule is matched.



Archived

## 2.7 Putting items into the DynaCache

In this section we explain the basics of how DynaCache can:

- Cache the output of a servlet or JSP
- Cache Java objects using command caching – for example, Java Beans, EJBs, and Web Services

### 2.7.1 Caching servlets and JSPs

When the application server starts and a targeted servlet or JSP is called for the first time, no cached entry is found and so the service() method of the servlet is called. The DynaCache Web container intercepts the response from the service() invocation and caches the results. This process is illustrated in Figure 2-14.

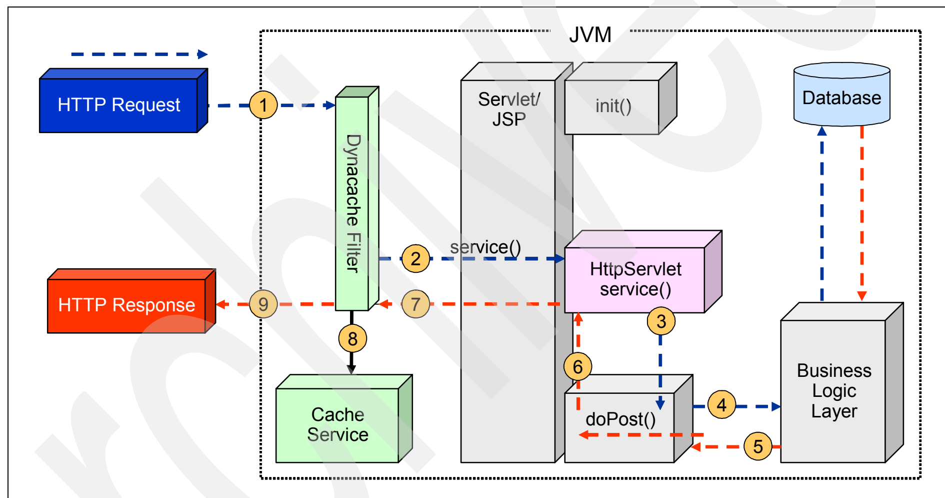



*Figure 2-14 Before caching: A request must traverses all layers*

The next time the servlet is called and a match is found in the caching rules engine, the cached results are returned and the service() method is not called. This avoids all of the processing that would have been done by the Servlet, resulting in a substantial performance boost.

If, however, there isn't a cache entry for this ID, the service() method is executed as normal, and the results are caught and placed in the cache before they are returned to the requestor.

Figure 2-15 illustrates how DynaCache increases performance by bypassing the service() method call entirely whenever a cache hit occurs.

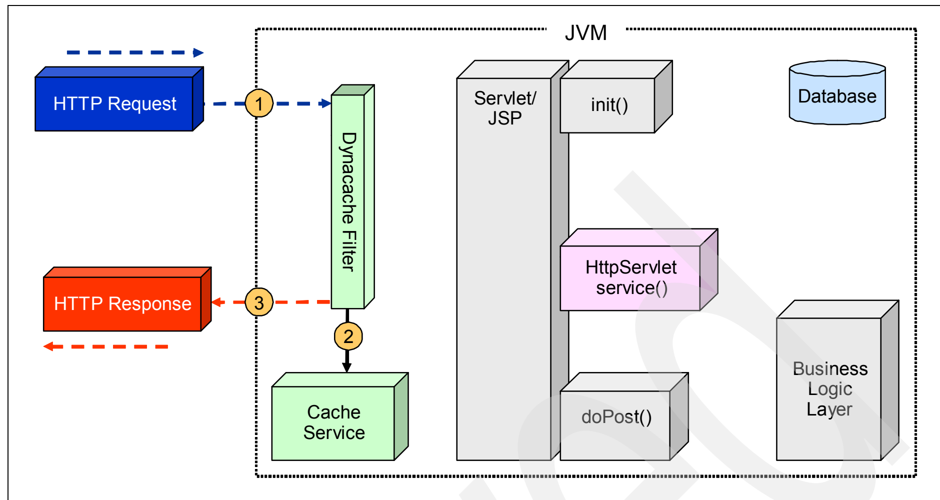



*Figure 2-15   A cache hit means the servlet and business logic is not called*

Servlets and JSPs are configured for caching via entries in the cachespec.xml. A servlet cachespec entry is designated by its URI path or by its class name. The classname option is more inclusive because it will catch any invocation of the servlet, regardless of any servlet alias mappings defined. So which option should you use? The answer is "it depends."

Usually, the servlet is cached by its alias, since different aliases often imply different processing operations. Determining whether to cache on URI or classname depends entirely on the application. In most cases, the cache entry for the servlet needs to be further qualified by additional inputs, such as the request parameters or values from the user session information. This is explained in the section on specifying cache entries.

## 2.7.2 Java objects and the command cache

DynaCache provides support for caching the returned results from Java object calls. DynaCache provides a very straightforward mechanism that allows you to cache the results from these Java command objects; it is called *command caching*.

Commands written to the WebSphere command architecture access databases, file systems, and connectors to perform business logic, and often execute remotely on another server. Significant performance gains can be achieved by caching command results and avoiding their repeated execution.

To take advantage of command caching, applications must be written to the WebSphere command framework API. This API is based on a common programming model, and uses a series of getters(), setters() and execute() methods to retrieve its results. To use a command, the client creates an instance of the command and calls the execute() method for the command. Depending on the command, calling other methods could be necessary. The specifics will vary with the application.

Implementing command caching involves software development. However, the development requirements for command caching are not difficult. You can cache the output from calls to:

- Java classes (POJO)
- Java Beans
- EJBs
- Web services

As an application architect you should establish a coding pattern that will allow applications to use caching at a later stage in development without having to go back and re-implement the code again.

For example, say you have a system that uses a Java Bean to query a database and that bean reads in all of the names of the states, counties, or provinces within your country, including tax rates per region. Rather than have each call to the bean hit the database, you could do the following:

- Modify the bean by adding the necessary TargetableCommand methods.
- Update your cachespec.xml and add a new cache entry that includes this new command.

By implementing the previous steps, DynaCache is now aware of your bean, and can intercept calls made to that bean and serve responses from the cache instead. Example 2-8 shows how this might look in the cachespec.xml file. Note that the <sharing-policy> tag is for enabling clustered data replication support.

*Example 2-8 Example of specifying a command cache entry*

```
<?xml version="1.0"?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
   <cache-entry>
      <class>command</class>
      <sharing-policy>shared</sharing-policy>

<name>com.ibm.myapp.statetaxation.StateTaxCacheCommand.class</name>
      <cache-id>
         <component type="method" id="getStateTaxList">
```

```
            <required>true</required>
         </component>
         <priority>1</priority>
      </cache-id>
   </cache-entry>
</cache>
```

Commands cache the data before it is transformed into HTML. In order to take advantage of command caching, you must make simple code changes to your Java objects so that DynaCache can call them to retrieve the data. Command objects inherit from com.ibm.websphere.command.cachableCommandImpl and must provide certain methods to function properly.

### 2.7.3 Command interface

Commands are Java objects that follow a usage pattern that consists of three operational methods. They are:

- Set: Initialize the input properties of the command object.
- Execute: Perform the specific business logic for which the command was written.
- Get: Retrieve the output properties that are set by the execution.

Each command can be in one of three states based on which methods have been executed:

- New: The command has been created but the input properties have not been set.
- Initialized: The input properties have been set.
- Executed: The execute method has been called and the output properties have been set and can be retrieved.

Executed command objects can be stored in the cache so that subsequent instances of that command object's execute method can be served by copying output properties of the cached command to the current command for retrieval by its get methods.

DynaCache supports caching of command objects for later reuse by servlets, JSPs, EJBs, or other business logic programming. To identify these cached command objects, a unique cache ID is generated based on the fields and methods that represent or are used to retrieve input properties of the command.

To cache a command in an application, a cache ID creation rule must be written in the cache policy file, and the command must be changed to extend the

Archived

“CacheableCommandImpl” class instead of implementing the standard “TargetableCommand” interface.

CacheableCommandImpl is an abstract class that implements the methods necessary for the command to interact with the caching framework. Because the CacheableCommand interface extends the TargetableCommand interface, the command in the application must continue to implement the methods needed for the TargetableCommand interface.

The standard TargetableCommand interface provides only the client-side interface for generic commands and declares three basic methods:

- IsReadyToCallExecute()
  This method is called on the client side before the command passes to the server for execution.
- Execute()
  This method passes the command to the target and returns any data.
- Reset()
  This method resets any output properties to the values they had before the execute method was called so that the object can be reused.

When a command is called to execute, the request is intercepted by the cache, and a cache ID is generated based on the values of the input properties specified in its cache policy. If a cache entry exists for this cache ID, the output properties are copied from the cached object to this instance of the command, and the state of this instance is changed to “executed” without actually executing the business logic. If an entry with the generated ID is not found, the execute method is called, and the executed command object is stored in the cache. If an ID is not generated, this is considered not to be an instance of cacheableCommand caching.

Archived

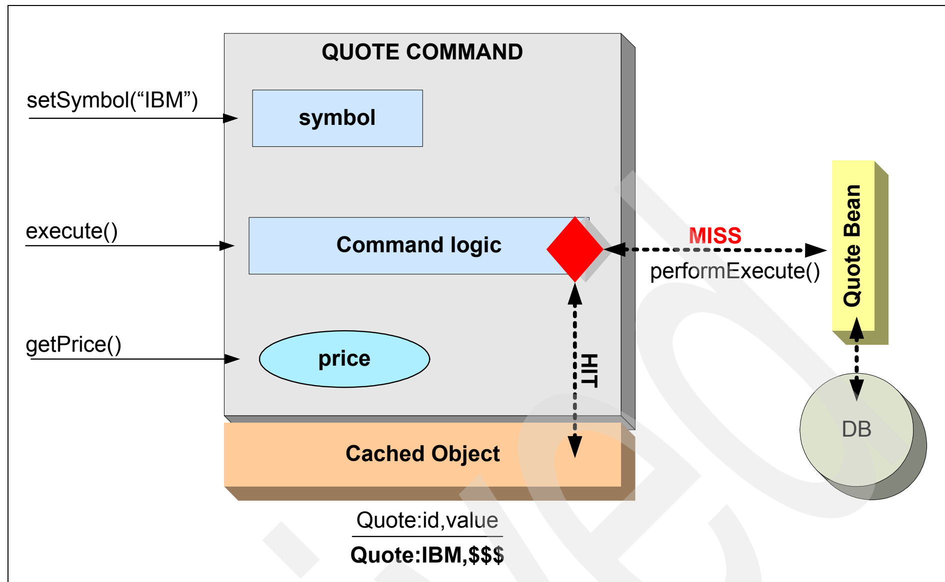



*Figure 2-16 Implementing a quote command used in a stock quote application*

Figure 2-16 shows the logic of the sample code in Example 2-9. The code has been taken from a StockQuote application. In order to execute this command, first invoke all of the setter methods. Once this is done, the isReadyToCallExecute() method returns true, then the command is executed and sets the output parameter which, in this case, is the QuoteDataBean variable called quoteData. The getPrice() method returns the QuoteData bean object.

*Example 2-9 QuoteCommand.java - A sample DynaCache command class*

```
package com.ibm.cache.sample.command;
import javax.naming.InitialContext;
import com.ibm.cache.sample.QuoteDataBean;
import com.ibm.cache.sample.ejb.*;
import com.ibm.websphere.command.CacheableCommandImpl;
public class QuoteCommand extends CacheableCommandImpl {
   public StockSession stockHome = null;
   public QuoteDataBean quoteData = null;
   public String symbol = null;

   public QuoteCommand() {
        try {
            StockSessionHome StockSessionHome = null;
            if (StockSessionHome == null) {
                InitialContext ic = new InitialContext();
```

```
                try {
                    StockSessionHome = (StockSessionHome)
                    (javax.rmi.PortableRemoteObject
                        .narrow(ic.lookup("ejb/StockSession"),
                        StockSessionHome.class));
                }
                catch (Exception e) {
                    StockSessionHome = (StockSessionHome)
                    (javax.rmi.PortableRemoteObject
                        .narrow(ic.lookup("ejb/StockSession"),
                        StockSessionHome.class));
                }
            }
            stockHome = StockSessionHome.create();
        }
        catch (Exception e) {
            System.out.println("Error on StockSession Lookup");
            e.printStackTrace();
        }
    }

   public QuoteCommand(String symbol) {
        this();
        this.symbol = symbol;
    }
////////
// CacheableCommand interface methods
////////
   public boolean isReadyToCallExecute() {
        return stockHome != null && symbol != null;
    }

   public void performExecute() throws Exception {
        quoteData = stockHome.getPrice(symbol);
    }

////////////
//end Cacheable command methods
////////////
   public void setSymbol(String symbol) {
        this.symbol = symbol;
    }
   public String getSymbol() {
        return symbol;
    }
```

```
    public QuoteDataBean getPrice() {
         return quoteData;
     }
}
```

Example 2-9 is a simple example of a Java Command class in action. The example shows how to cache the response to a call made against a stock quote session EJB.

Example 2-10 shows the cachespec.xml file entry to get the previous Java command to work with DynaCache.

*Example 2-10 Cachespec entry that matches QuoteCommand Java code*

```
<?xml version="1.0"?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
   <cache-entry>
      <class>command</class>
      <sharing-policy>not-shared</sharing-policy>
      <name>com.ibm.cache.sample.command.QuoteCommand.class</name>
      <cache-id>
         <component type="method" id="getSymbol">
            <required>true</required>
         </component>
         <priority>1</priority>
         <timeout>180</timeout>
      </cache-id>
   </cache-entry>
</cache>
```

The Java snippet in Example 2-11 shows how the previous example, QuoteCommand class, is called by client Java code.

*Example 2-11 Calling QuoteCommand class*

```
String id = “IBM”;
QuoteDataBean quoteData = new QuoteDataBean(id);
QuoteCommand cmd = new QuoteCommand(id);
cmd.execute();
quoteData = cmd.getPrice();
```

The cmd.execute() method call, as you can see from the code, is implemented in QuoteCommand’s inherited class (CacheableCommandImpl), which is where the

caching management takes place. If there is no cache entry, the CacheableCommandImpl parent class then calls the executeCommand() method of a CommandTarget proxy class, which in turn calls the performExecute() method of the target class (QuoteCommand). Once the performExecute() method has finished, the parent class caches the results. Clients can then retrieve information via getter methods on the Command object.

Under the covers, the CacheableCommandImpl.execute() implements the execute() method defined in the Command interface, thereby overriding the implementation provided in the TargetableCommandImpl class. During execution, the execute() method does the following:

- It throws an UnsetInputPropertiesException if this command's isReadyToCallExecute() method returns false.
- It retrieves the CommandTarget object for this command from the TargetPolicy.
- If the target is already cached, it returns the cached value.
- If it is not cached, it calls the proxy CommandTarget.executeCommand() method to execute the command, which in turn calls the TargetableCommand.performExecute() method, and then caches the entire object.

The command may be cached after execution, depending on the sharing policy. If the hasOutputProperties() method returns true and the returned command is not the same instance as this command, it calls the setOutputProperties method so that the results will be copied into this command. Finally, it sets the time of execution of the command.

**Note:** It is important to remember that any objects referenced in a command must be serializable.

## 2.7.4 DynaCache full page caching

Full page caching is exactly what it sounds like: the HTML output of an entire page is cached as a single entity. You configure full page caching by specifying the consume-subfragments property and setting its value to true in your <cache-entry> definition for the servlet or JSP.

The advantage of using full page caching with the Commerce controller servlet is performance. The disadvantage is that if this mechanism is used, then the page output cannot contain any personalized information. If the cached page did contain personalized information, then users see another user’s cached information and not their own. In the following sections we show you how to use

the consume-subfragments tag and the do-not-consume tag to allow you to cache a full page except the portions that are cached as fragments.

### Consume-subfragments

In DynaCache, full page caching is enabled by setting the <cache-entry> property consume-subfragments to true for the main targeted parent servlet or JSP, for example, <property name="consume-subfragments">true</property>. When set to true, the consume-subfragments property tells DynaCache to cache the targeted JSP and any of its children's JSPs as a single page.

When a servlet is cached, only the content of that servlet is stored, with placeholders for any other fragments it includes or to which it forwards. Consume-subfragments(CSF) tells the cache to continue saving content when the parent servlet includes a child servlet.

The parent entry (the one marked CSF) will include all the content from all fragments in its cache entry (in the cache monitor you will see the include statements being consumed), resulting in one large cache entry that has no includes or forwards, but rather the content from the whole tree of entries. This saves a significant amount of application server processing, but is typically only useful when the external HTTP request contains all the information needed to determine the entire tree of included fragments.

## 2.7.5 DynaCache fragment caching

In DynaCache, fragment caching generally means caching a part of a rendered HTML page. In practice, a page may not be "cacheable" in its entirety, but it may contain sections of varying cacheability that can be separated into fragments and cached independently. Any content that can be independently requested can make up a cacheable fragment.

For fragment (JSP) caching, WebSphere Commerce has to execute the command (controller, task, and view commands) to identify which JSP is to be executed before DynaCache can determine whether the JSP can be served from the cache or not. The advantage of this method is flexibility, because different cache entries could be reassembled to form a page based on user information.

In order for a fragment to be cacheable, the fragment has to be executable on its own. That is, it must be able to execute independently of any other fragments of the Web page. Because the fragment being excluded must be self-executing, it cannot depend on any attributes set by the parent JSP fragment – unless an attribute has been defined as being cached with the parent.

This can be a problematic because not all fragments in WebSphere Commerce are self-executing. To test if the fragment is self-executing, pass the fragment's

Archived

URL to a Web browser, including the necessary parameters. For example, the mini shopping cart contained in the ConsumerDirect sample store is accessed by the following URL:

`http://localhost/webapp/wcs/stores/servlet/ConsumerDirect/include/MiniShopCartDisplay.jsp?storedId=10001`.

The following method describes an alternative way of determining if the fragment is self-executing. After the cachespec.xml file is configured, click the servlet's page.

- Use the cache monitor to invalidate the child fragment.
- Execute the page request to confirm the execution of the fragment is successful.

If the execution is successful, that fragment is self-executing.

*Figure 2-17 Each of these three eMarketing Spots is self-executing and can therefore be cached individually*

In Figure 2-17, each eMarketing spot can be cached because it is self-executing, so it can be cached as a fragment. By caching it separately it can be invalidated without invalidating the parent.

You could cache the eMarketing spot by using the cache entry shown in Example 2-12.

*Example 2-12 Caching the* eMarketing spot.

```
<class>servlet</class>
      <name>/ConsumerDirect/include/eMarketingSpotDisplay.jsp</name>
      <property name="store-cookies">false</property>
      <property name="save-attributes">false</property>
      <property name="do-not-consume">true</property>
      <cache-id>
         <component id="emsName" type="parameter">
            <required>true</required>
         </component>
         <component id="maxNumDisp" type="parameter">
            <required>true</required>
         </component>
```

Archived

```
        <component id="catalogId" type="parameter">
           <required>true</required>
        </component>
        <component id="maxItemsInRow" type="parameter">
           <required>true</required>
        </component>
        <component id="maxColInRow" type="parameter">
           <required>true</required>
        </component>
        <priority>1</priority>
        <timeout>3600</timeout>
     </cache-id>
```

The cache-entry for the parent servlet contains values for those parameters passed into the fragment using the jsp:include or c:import tags. This means that on a cache hit, the parameters passed from the parent to the fragment are not regenerated. To have new values passed to the fragment, invalidate or re-execute the parent JSP

## do-not-cache

This property, when set to “true,” instructs DynaCache to *not cache* a fragment and *not consume* it in any parent cache item that incorporates it. The implication of this is that the JSP will be re-executed *every time* the page is drawn.

Example 2-13 provides example cachespec definitions. In previous versions of DynaCache the child cache entry had to be declared before any parent cache entry for the property to work. This is not the case with DynaCache today, but the convention has remained in common practice.

*Example 2-13 do-not-cache used to prevent the caching of a child JSP*

```
<cache-entry>
    <class>servlet</class>
    <name>/DNCChild.jsp</name>
    <property name="do-not-cache">true</property>
    <cache-id>
        <timeout>0</timeout>
    </cache-id>
</cache-entry>

<cache-entry>
    <class>servlet</class>
    <name>/DNCParent.jsp</name>
    <property name="consume-subfragments">true</property>
    <cache-id>
```

```
        <timeout>0</timeout>
    </cache-id>
</cache-entry>
```

The do-not-cache property also works with ESI servers.

### do-not-consume

The purpose of this property is often misunderstood by the DynaCache novice. When set to “true,” the do-not-consume property instructs DynaCache to cache the item independently of any consuming parent that it belongs to. It does *not* mean “do-not-cache.” A simple analogy to explain it is “do not make me live with my all-consuming parent in the cache, I want to be independent, thank you.”

The do-not-consume property works particularly well for situations where only a small portion of a candidate Web page contains personalized information, for example, a personalized welcome message or a mini shopping cart. In this scenario, you would want to cache the majority of the page, but separate the personalized bit. By using the do-not-consume property, most of a page can be rendered from the cache and completed with a portion that has been cached elsewhere, or not cached at all.

So, for the latter scenario to work properly, the parent object’s <cache-entry> would be marked with the property consume-subfragments and the child fragment that contains the personalization area would be marked with the do-not-consume property.

With this combination, you can achieve performance gains that approach those of whole page caching and still provide personalized content.

Example 2-14 includes sample cachespec entries that show how to use the two properties “do-not-consume” and “consume-subfragments.” Notice that the “do-not-consume” child component (that is, the mini shopping cart) must be declared before the parent object, which in the example is StoreCatalogDisplay.

*Example 2-14 do-not-consume and consume-subfragments properties in action*

```
<cache-entry>
    <class>servlet</class>
    <name>/MiniCurrentOrderDisplay.jsp</name>
    <property name="do-not-consume">true</property>
    <property name="save-attributes">false</property>
    <cache-id>
        <component id="DC_userId" type="attribute">
            <required>true</required>
        </component>
```

```
    </cache-id>
</cache-entry>

<cache-entry>
    <class>servlet</class>
    <name>com.ibm.commerce.server.EACActionServlet.class</name>
    <property name="store-cookies">false</property>
    <property name="save-attributes">false</property>
    <property name="consume-subfragments">true</property>
    <cache-id>
        <component id="" type="pathinfo">
            <required>true</required>
            <value>/StoreCatalogDisplay</value>
        </component>
        <component id="storeId" type="parameter">
            <required>true</required>
        </component>
        <component id="catalogId" type="parameter">
            <required>true</required>
        </component>
    </cache-id>
</cache-entry>
```

## 2.8 Invalidation: Getting stale objects out of the cache

A second, critical invalidation policy defines when to evict cache items. The strategy for removing expired cache entries is an essential part of cache design. Cache entries might:

- Expire after a given amount of time
- Be removed from the cache based on dependency rules
- Be removed on the basis of a Least Recently Used algorithm when the cache needs space for new entries

The operation of removing entries from the cache is called *invalidation*. The easiest method for cache invalidation is to set a *time-to-live* (TTL) value on the cache entry, though we do not encourage this, as we explain later in the invalidation best practices section.

Wherever possible, you should create an explicit invalidation policy for each cache entry in the cachespec.xml file. This policy defines an invalidation rule very similar to the cache ID rule, and an invalidation ID is generated based on that rule. When an invalidation rule is specified for an object, DynaCache generates

an invalidation ID during the execution of that object and checks it against the cache IDs of all entries currently in the cache. If a cache ID matching the invalidation ID is found in the cache, the cache entry associated with this cache ID is removed from the cache (meaning it is invalidated).

DynaCache also provides a group-based invalidation mechanism based on dependency IDs. A dependency ID defines a cache entry's dependency rule. Different objects, based on their defined rules, might generate the same dependency ID. If an invalidation ID is generated for a request that matches a given dependency ID, all the cache entries associated with that dependency ID are removed from cache.

The invalidation ID is generated by concatenating the invalidation ID base string with the values returned by its component element. If a required component returns a null value, then the entire invalidation ID is not generated and no invalidation occurs. Multiple invalidation rules can exist per cache-entry. All invalidation rules run separately.

We encourage defining invalidation rules to automate the invalidation of cache entries. If you do not have invalidation rules configured, then you need to invalidate the entire cache when content changes. This means that everything is removed from the cache, and not just the changed pages. The invalidation of the complete cache is not recommended and should be avoided whenever possible since it leads to a poor user experience.

### Time-based invalidation

The simplest way to invalidate cache entries is with time-based elements, although it is not necessarily the best way. This method is useful for user-specific objects when you cannot invalidate the cache entries by any other mechanism.

You can accomplish time-based invalidation by specifying the <timeout>value</timeout> sub-element within a cache-entry in the cachespec.xml file. Value is the amount of time, in seconds, the cache entry is kept in the cache. The default value for this element is 0 (zero), which indicates this entry never expires.

There is another time-based invalidation technique in WebSphere Application Server 5.1 and later. You can use the <inactivity>value</inactivity> sub-element to specify a time-to-live (TTL) value for the cache entry based on the last time the cache entry was accessed. Value is the amount of time, in seconds, to keep the cache entry in the cache after the last cache hit. The mini shopping cart should use this type of invalidation rule so that your cache does not fill up with user-specific data that is no longer being accessed.

### Command-based invalidation

You can also invalidate entries in the cache through command-based invalidation using dependency and invalidation IDs. A cache-entry needs dependency IDs defined for each component that it relies on. When an object is cached for this entry, the dependency IDs are generated and associated with it. You can build a dependency tree, so that when object x is refreshed, so are objects y and z, and so on.

These entries are invalidated using command-based invalidation rules. Command-based invalidation means invalidation IDs are generated upon execution of a command. These invalidation IDs are constructed based on the methods and fields provided by the commands. When a request causes invalidation IDs to be generated, the entries in the cache that have a dependency ID matching the generated invalidation ID are invalidated.

To have cache invalidation triggered by a command, you must first declare the dependency-id components for the cache-entries you want to invalidate. The dependency-id components represent the components that, should they change, would invalidate the content of this cache entry. See Example 8-4 on page 197 for an illustration of command-based invalidation.

### Triggered invalidation

A final way to configure automatic cache invalidation is using the WebSphere Commerce CACHEIVL table in combination with database triggers.

## 2.9 The ConsumerDirect cachespec.xml file

WebSphere Commerce provides a number of sample applications, and the ConsumerDirect store is one of them. The ConsumerDirect sample application is also provided with a ready-to-use cachespec.xml file and in this section, we are going to put that file under the microscope. Let's take a look at the first few lines of the sample ConsumerDirect cachespec.xml file as shown in Example 2-15.

*Example 2-15 Targeting a struts-based application for caching*

```
<?xml version="1.0"?>
<!DOCTYPE cache SYSTEM "cachespec.dtd">
<cache>
   <cache-entry>
      <class>servlet</class>
      <name>com.ibm.commerce.struts.ECActionServlet.class</name>
      <property name="consume-subfragments">true</property>
      <property name="save-attributes">false
         <exclude>jspStoreDir</exclude>
```

```
    </property>
```

Perhaps in examining the XML in Example 2-15, you have already deduced that the `<cache-entry>` child elements, `<class>` and `<name>`, are part of a definition that describes a servlet type of cache entry.

### Element <class>

The `<class>` tag specifies the type of object (servlet) to cache and the <name> tag tells us which: com.ibm.commerce.struts.ECActionServlet.class. The <class> element is required and specifies how the application server interprets the remaining cache policy definition. For more information on the supported <class> types, see Table 6-2 on page 153.

### Element <name>

The `<name>` element is an important part of the cache entry. Take another look at the name element in the first ConsumerDirect cache entry in Example 2-16.

*Example 2-16 Specifying the Struts controller Servlet in a cachespec entry*

```
<cache-entry>
   <class>servlet</class>
   <name>com.ibm.commerce.struts.ECActionServlet.class</name>
```

To cache the output of the Commerce main action servlet we need the servlet's fully qualified package name so that the runtime DynaCache engine can locate it in the Web container and then start to filter it. In the sample cachespec.xml, this is provided as:

```
<name>com.ibm.commerce.struts.ECActionServlet.class</name>
```

The com.ibm.commerce.struts.ECActionServlet[2] is an important servlet in the context of the Commerce runtime architecture and is explained next.

## 2.9.1 WebSphere Commerce ECActionServlet explained

WebSphere Commerce V6 store applications use the Struts framework. In the case of Struts, every request URI ending in `.do` maps to the same ActionServlet.class. However, prior to WebSphere Application Server V6.0, only one cache policy is supported per servlet in releases. Therefore, to cache Struts responses, the cache policy has to be written for the ActionServlet servlet based on its servlet path.

[2] This was <name>com.ibm.commerce.server.RequestServlet.class</name> in earlier versions of WebSphere Commerce.

Archived

For Commerce, the name of the Struts ActionServlet handling all inbound store-based HTTP requests is com.ibm.commerce.struts.ECActionServlet.class. In Version 5 of Commerce, the non-Struts-based store applications also funneled requests through a single servlet, the RequestServlet.

In Example 2-15 you can see that ECActionServlet is the servlet that creates the output we want to cache. It is the only servlet to be configured in the entire cachespec file because all the Web traffic passes through it.

The meanings of the tags shown in Example 2-15 are given in Table 2-3.

*Table 2-3 Explanation of the initial cache tags in ConsumerDirect*

| Sample content | Description |
|---|---|
| <cache-entry> | We are defining a cache entry, the child elements of which will define the properties. |
| <class>servlet<class> | We are going to cache a servlet. |
| `<name>...`**`ECActionServlet`**`.class</name>` | ECActionServlet is the main servlet in Commerce applications through which all HTTP requests pass. |
| `<property name="consume-subfragments...` | Cache the entire page. Any subfragments or child JSPs must be included in the cached output. |
| `<property name="save-attributes">false` | Don't save any of the HTTP request attributes in the cached output. |
| `<exclude>jspStoreDir</exclude>` | Do not save the jspStoreDir attribute. Refer to "ConsumerDirect jspStoreDir issue" on page 170 for more details. |

## 2.9.2 Cache-id definitions for ConsumerDirect

Appearing immediately after the cache entry definitions in the ConsumerDirect cachespec.xml file, there is a list of <cache-id> definitions. The first <cache-id> defines what is required to properly cache the ConsumerDirect "Store Catalog Display" page shown in Figure 2-9 on page 45.

The XML fragment that specifically caches the "Store Catalog Display" page is shown in Figure 2-18 on page 78.

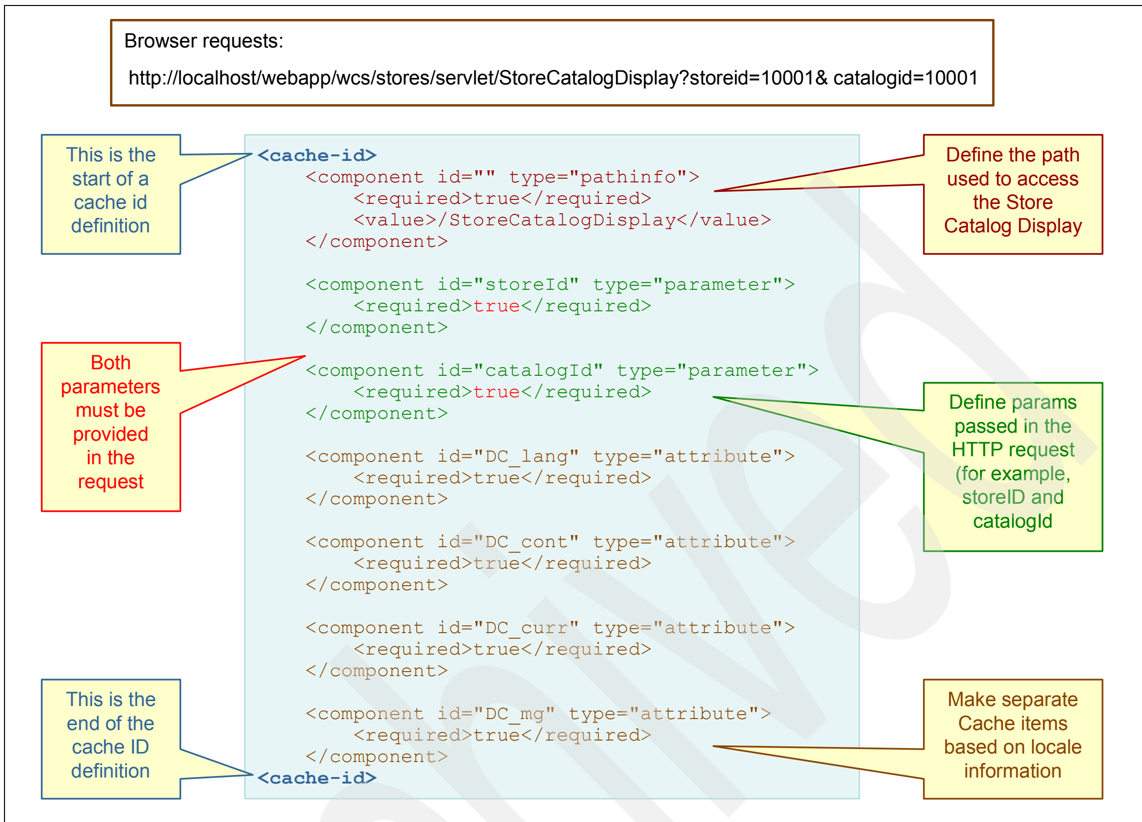



*Figure 2-18 Cache ID definitions for caching the StoreCatalogDisplay page*

Refer to Table 6-5 on page 169 for an explanation of the Commerce attributes DC_lang, DC_curr, DC_cont, and DC_mg, and how they are useful in Commerce specific cachespec.xml files. Be aware, however, that if an application does not have a dependency on language, there is no reason for your cache-id to contain the DC_lang component.

## TopCategoriesDisplay

Following the StoreCatalogDisplay is the next cache-id entry, shown in Example 2-17, for the TopCategoriesDisplay page.

*Example 2-17 Cache ID definitions for caching the TopCategoriesDisplay page*

```
<cache-id>
   <component id="" type="pathinfo">
      <required>true</required>
      <value>/TopCategoriesDisplay</value>
```

Archived

```
   </component>
   <component id="storeId" type="parameter">
      <required>true</required>
   </component>
   <component id="catalogId" type="parameter">
      <required>true</required>
   </component>
   <component id="categoryId" type="parameter">
      <required>false</required>
   </component>
   <component id="DynaCache" type="attribute">
      <required>true</required>
   </component>
   <component id="DC_curr" type="attribute">
      <required>true</required>
   </component>
   <component id="DC_cont" type="attribute">
      <required>true</required>
   </component>
   <component id="DC_mg" type="attribute">
      <required>true</required>
   </component>
</cache-id>
```

The elements in this entry are very similar to the StoreCatalogDisplay entry. The URL this cache ID executes against looks like:

```
/TopCategoriesDisplay?storeId=<storeId>&catalogId=<catalogId>
```

Notice that the categoryId parameter did not appear in this URL. This is because categoryId is declared optional in the cachespec (see Example 2-17). CategoryId is not required to be present for the rule to fire because its <required> property is set to `false`. If the categoryId were present in the request URL, then a different cache-id is generated because the DynaCache internal id-generator class would then include the value of the categoryId parameter when generating the unique cache ID.

## 2.10 Impact of memory cache on JVM garbage collection

For large Web sites that cache many thousands of items, the cache designer needs to be aware of the potential impact these objects can have on JVM memory utilization. Implementing DynaCache without proper consideration can

easily lead to a large increase in memory resource consumption and therefore cause serious side effects.

Some of the symptoms you may observe are *OutOfMemory* exceptions due to *heap fragmentation* type problems. One of the root causes of this problem is configurations that result in greater numbers of DynaCache objects accumulating in the JVM heap than can be handled by existing resources.

For example, if the average page size is 50 KB and you set the in-memory cache pool size to 10,000 objects, you end up needing 500,000 KB (half a gigabyte) of JVM heap to accommodate the cache. DynaCache does not compress its cache entries.

If the maximum heap size is set to one gigabyte, then this represents a significant amount of the heap being allocated to DynaCache. It also means that the rest of your application must operate within the remaining 500,000 KB of heap space, which may not be what you intended.

In an effort to try to prevent some of these fragmentation issues, DynaCache will reuse cached item memory objects via a technique called object pooling. Nonetheless, incorrect caching and invalidation strategies eventually lead to increased memory fragmentation and poorer garbage collection performance.

Some of the possible solutions are as follows:

- Applying iFixes related to diskoffload, replication (DRS), and external cache management memory utilization might need to be done to your system depending on its version.
- Changing replication policy to *Push & Pull* from the *Push* usage pattern *might* need to be done. (The appropriate DRS replication policy is a difficult issue. Push & Pull will actually end up using more memory, and is discussed at length in Chapter 4, “Clustering DynaCache” on page 107).
- Applying a JDK™ upgrade to allow tuning of class block clusters.
- Creating additional JVMs to spread the load.

Archived

We now discuss the tuning of the Java heap and review some of the options available to you to assist with your DynaCache design. Refer to Figure 2-19, which illustrates the runtime layout of JVM memory segments, as you read through the next few topics.

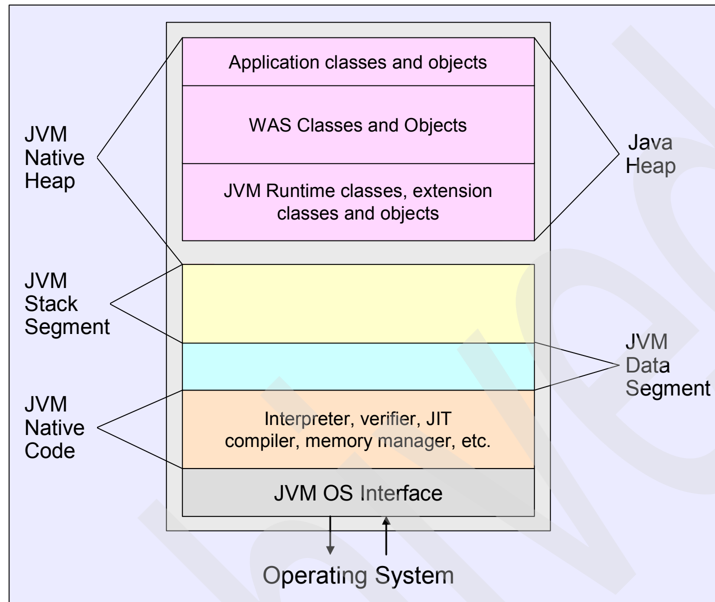



*Figure 2-19 JVM memory segments*

## kCluster, pCluster, and fragmentation

Java objects located in the Java heap are usually mobile; that is, the garbage collector (GC) can move them around if it decides to re-sequence the heap. Some objects, however, cannot be moved either permanently or temporarily. Such immovable objects are known as *pinned objects*. One kind of situation to look out for, which is associated with pinning memory, is the use of the JNI to call external programs. Use of JDBC-2 drivers is a case in point.

In the Java SDK Release 1.3.1, Service Refresh 7 and above, the garbage collector (GC) allocates what is called a *kCluster* as the first region at the bottom of the heap. A kCluster is an area of storage that is used exclusively for class blocks. It is large enough to hold 1280 entries and each class block is 256 bytes long.

The GC then allocates a *pCluster* as the second object on the heap. A pCluster is an area of storage that is used to allocate any pinned objects. It is 16 KB long.

When the kCluster is full, the GC allocates class blocks in the pCluster. When the pCluster is full, the GC allocates a new pCluster of 2 KB. Because this new pCluster can be allocated anywhere in the heap and must be pinned, it can lead to fragmentation problems.

### How fragmentation occurs

The pinned objects effectively deny the GC the ability to combine free space during heap compaction; this can result in a heap that contains a lot of free space but in relatively small, discrete amounts, so that an allocation that appears to be well below the total free heap space fails. When the request fails, we need to run a full GC compaction to free up memory. During the compaction, processing in the JVM comes to a halt. The more frequently we run compactions the larger the degradation of performance.

### How to avoid fragmentation

Java SDK Release 1.3.1 at SR7, and later, provides command-line options to specify the size of the JVM kCluster and pCluster regions. Refer to Table 2-4 for a summary of the switches. For additional information see the IBM Java technology Web page "*Diagnosis Documentation,*" which has a section on garbage collection and performance tuning. The Web address is:

```
http://www-128.ibm.com/developerworks/java/jdk/diagnosis/142.html
```

*Table 2-4 JVM kCluster and pCluster sizing switches*

| JVM region | JVM switch | |
|---|---|---|
| kCluster | **-Xk** | |
| pCluster overflowsize | **-Xp sz,ovfl** | The -Xp switch has two parameters:<br>**sz** = the pCluster size parameter in KB<br>**ovfl** = the overflow size parameter in KB |

Set the initial sizes of the clusters large enough to help avoid fragmentation issues occurring on your Web site. It is not unusual in a large Java application, such as WebSphere Application Server, that the default kCluster space might not be sufficient to allocate all class blocks.

*Example 2-18 Sample garbage collection output using -verbosegc switch*

```
<GC(VFY-SUM): pinned=4265(classes=3955/freeclasses=0) dosed=10388
movable=1233792 free=5658>
```

In Example 2-18 on page 82, the pinned size value(4265) and classes size value(3955) are about the right size needed for the -Xk parameter. However, we recommend that you add 10% to the reported value (3955)

In the preceding example, -Xk4200 is probably a reasonable setting. The difference between pinned (=4265) and classes (=3955) provides a guide for the initial size of pCluster; however, because each object might be a different size, it is difficult to predict the requirements for the pCluster and pCluster overflow options.

### Configuring the kCluster

Set -Xk to handle objects up to the specified size using the -Xk option:

```
-Xk maxNumClass
```

Here, maxNumClass specifies the maximum number of classes the kCluster can contain.

-Xk instructs the JVM to allocate space for maxNumClass class blocks in kCluster. The GC trace data obtained by setting -Xtgc2 can help provide a guide for the optimum value of the maxNumClasses parameter. You must keep -Xtgc2 enabled until memory fragmentation is satisfactory.

### Configuring the pCluster

Specify the pCluster and pCluster overflow sizes using the -Xp command-line option:

```
-Xp sizeClusterKB[,sizeOverflowKB]
```

Here, sizeClusterKB specifies the size of the initial pCluster in KB and sizeOverflowKB optionally specifies the size of overflow (subsequent) pClusters in KB. The default values of sizeCluster and sizeOverflow are 16 KB and 2 KB, respectively. If your application suffers from heap fragmentation, turn on the GC trace (-Xtgc2) and specify the -Xk option. If the problem persists, experiment with higher initial pCluster settings and overflow pCluster sizes.

### Configuring the Large Object Area

IBM Sovereign 1.4.2 SDK SR1 and later (build date of 20050209 and later) supports the configuration of Large Object Area to reserve the Java heap for allocating large objects (>=64 KB). For details, refer to Technote 1236509 at:

```
http://www.ibm.com/support/docview.wss?rs=180&context=SSEQTP&q1=large+o
bjects&uid=swg21236509&loc=en_US&cs=utf-8&lang=en
```



Archived

## 2.11 Configure disk offload

If the estimated total size of all cached objects is bigger than the available memory, you can enable the disk offload option in WebSphere Application Server. We would expect most WebSphere Commerce sites to use disk offload, unless a Web site has a small catalog.

Priority weighting is used in conjunction with the least recently used (LRU) algorithm to move objects from the cache onto disk. You configure priorities for the cached objects in the cachespec.xml file. A higher priority entry is less likely to be moved to disk.

If you decide to use the disk offload option, you need to also configure your file system for fast I/O access. Depending on your hardware and software, you can use various disk striping or caching techniques. Disk offload is configured on the same panel where you enable the DynaCache service. You should investigate using a high-speed Storage Area Network with memory.

The three settings related to the disk offload configuration are the following:

- **Enable disk offload**: Specifies whether disk offload is enabled. If a cache entry that was moved to disk is needed again, it is moved back to memory from the file system.
- **Offload location**: Specifies the location on the disk to save cache entries when disk offload is enabled. If disk offload location is not specified, the default location is used:

  `$install_root/temp/node/servername/_DynaCache/cacheJNDIname`

  If a disk offload location is specified, the node, server name, and cache instance name are appended.

  For example, `$install_root/diskoffload` generates the location as `$install_root/diskoffload/node/servername/cacheJNDIname`. This value is ignored if disk offload is not enabled.
- **Flush to disk**: Specifies if in-memory cached objects are saved to disk when the server is stopped. This value is ignored if `Enable disk offload` is not selected.

Archived

## 2.11.1 Tuning the disk cache

There are several custom properties for the JVM available to tune the disk cache[3].

All custom properties can be set using the following steps:

1. In the Administrative Console, select **Servers → Application servers → <AppServer_Name> → Java and Process Management → Process Definition → Java Virtual Machine → Custom Properties → New**.
2. Enter the Name of the custom property.
3. Enter a valid value for the custom property.
4. Save your changes and restart the application server.

### Custom JVM properties

There are three custom properties available:

1. htodCleanupFrequency
2. htodDelayOffloadEntriesLimit
3. htodDelayOffload

The first, htodCleanupFrequency, is related to the disk cache cleanup time The other two properties, described in the following sections, are related to tuning the delay offload function.

#### *htodCleanupFrequency*

Tune the disk cache cleanup time using the com.ibm.ws.cache.CacheConfig. htodCleanupFrequency custom property. This property defines the amount of time between disk cache cleanups. By default, the disk cache cleanup is scheduled to run at midnight to remove expired cache entries and cache entries that have not been accessed in the past 24 hours. However, if you have thousands of cache entries that might expire within one or two hours, the files that are in the disk cache can grow large and become unmanageable.

Use com.ibm.ws.cache.CacheConfig.htodCleanupFrequency to change the time interval between disk cache cleanups. The value is set in minutes, that is, a value of 60 means 60 minutes between each disk cache cleanup. The default is 0 which means that the disk cache cleanup occurs at midnight every 24 hours.

[3] WebSphere Application Server Version 5.0.2.18, 5.1.1.12 and 6.0.2.17 include APAR PK13460: “DISK CACHE PERFORMANCE, GARBAGE COLLECTION, ADDITIONAL PMI MATRIX AND CACHE POLICY ENHANCEMENTS” which should be installed. The APAR is found at http://www.ibm.com/support/docview.wss?rs=180&uid=swg1PK13460

Archived

**Tip:** If you are using objects that use inactivity or timeout to clean up, then you need to configure this parameter to execute more frequently. If you use a 15 minute timeout we suggest you use a 15 minute cleanup interval.

### *htodDelayOffloadEntriesLimit and htodDelayOffload*

Tune the delay offload function using these custom properties:

- com.ibm.ws.cache.CacheConfig. htodDelayOffloadEntriesLimit
- com.ibm.ws.cache.CacheConfig. htodDelayOffload

The delay offload function uses extra memory buffers for dependency IDs and templates to delay the disk offload and minimize the input and output operations. However, if most of your cache IDs are longer than 100 bytes, the delay offload function might use too much memory.

To increase or decrease the in-memory limit, use com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntriesLimit. This custom property specifies the number of different cache IDs that can be saved in memory for the dependency ID and template buffers. Consider increasing this value if you have a lot of memory in your server and you want to increase the performance of your disk cache. The default value in WebSphere Commerce is 100,000,[4] which means that each dependency ID or template ID can have up to 100,000 different cache IDs in memory. Specify a value suitable for your environment, but it must be a number higher than 100 because this is the minimum setting allowed.

The com.ibm.ws.cache.CacheConfig.htodDelayOffloadcustom property specifies whether extra memory buffers are used in memory for dependency IDs and templates to delay disk offload and to minimize input and output operations to the disk. The default value is *true*, which means enabled. With the property enabled, there is a considerable performance boost in disk cache throughput. Consider disabling it if your cache IDs are larger than 100 bytes because this option might use too much memory when it buffers your data. If you set this property to *false*, all the cache entries are copied to disk immediately after they are removed from the memory cache.

[4] WebSphere Commerce already modifies the WebSphere Application Server from its default of 1000.

# 2.12 Displaying cache information

The DynaCache monitor is an installable Web application that displays simple cache statistics, cache entries, and cache policy information for servlet cache instances. Figure 2-20 is an example monitor screen display.

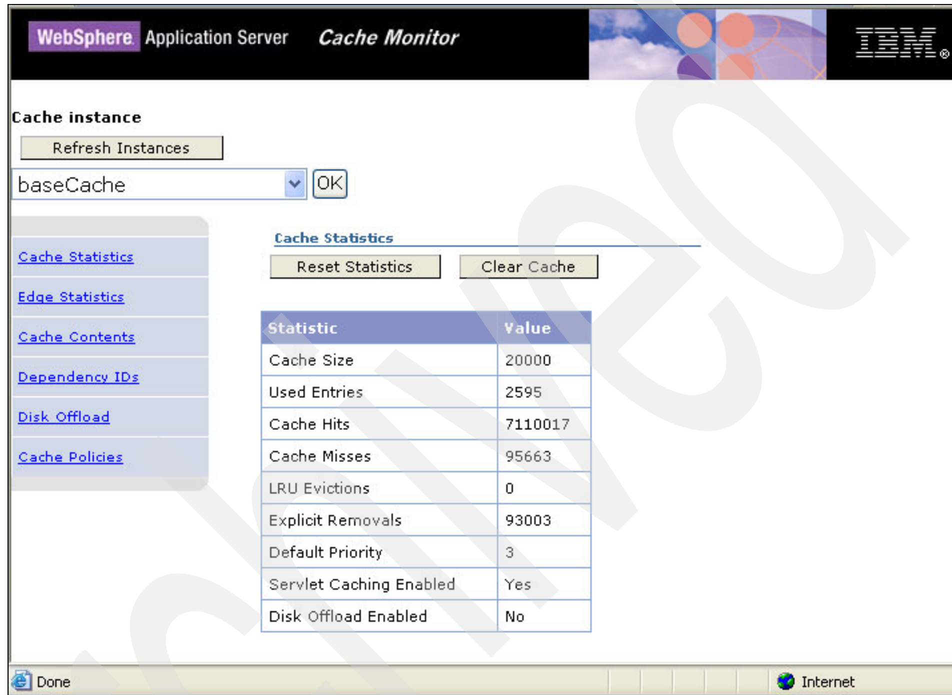



*Figure 2-20 Example Cache Monitor application window*

## 2.12.1 Install the cacheMonitor.ear application

The cache monitor application is not installed automatically, so use the administrative console or an equivalent wsadmin script to install the cache monitor application from the <app_server_root>/installableApps directory. The name of the application is cacheMonitor.ear.

Install the cache monitor into the application server you want to monitor. Refer to Chapter 8, “DynaCache tutorial” on page 179 to see exactly how to perform the cache monitor installation. The cache monitor has to be applied to each server you want to monitor. It does not provide a unified view of the cache for all the servers in the cluster.

Archived

**Attention:** Recall our recommendation that you only use the cache monitor on production servers with security and access control applied to reduce the chances of someone invalidating the whole cache and causing a massive performance degradation of a Web site with potentially serious business impact.

Once installed, you can access the cache monitor using the following url: `http://host_name:port/cachemonitor` where your port is the port associated with the host on which you installed the cache monitor application. Once the monitor is up and running you can verify the list of all cache instances that you have configured.

If you did not configure any cache instances the Cache Monitor will only display the cache entries stored in the default, base Cache instance.

### 2.12.2 Cache monitor viewing capabilities

For each cache instance configured, you can perform the following actions:

- View the Cache Statistics page and verify the cache configuration and cache data. Click **Reset Statistics** to reset the counters.
- View the Edge Statistics page to view data about the current ESI processors configured for caching. Click **Refresh Statistics** to see the latest statistics or content from the ESI processors. Click **Reset Statistics** to reset the counters.[5]
- View the Cache Contents page to examine the contents that are currently cached in memory.
- View the Disk Offload page to view content that is currently offloaded from memory to disk.
- View the Cache Policies page to see which cache policies are currently loaded in the DynaCache. Click a template to view the cache ID rules for the template.

[5] This has been the source of a lot of APARS. Prior to WebSphere Application Server v6.1, any and only one server in a cluster would show the status and contents of all WSI caches configured to work with DynaCache.



Archived

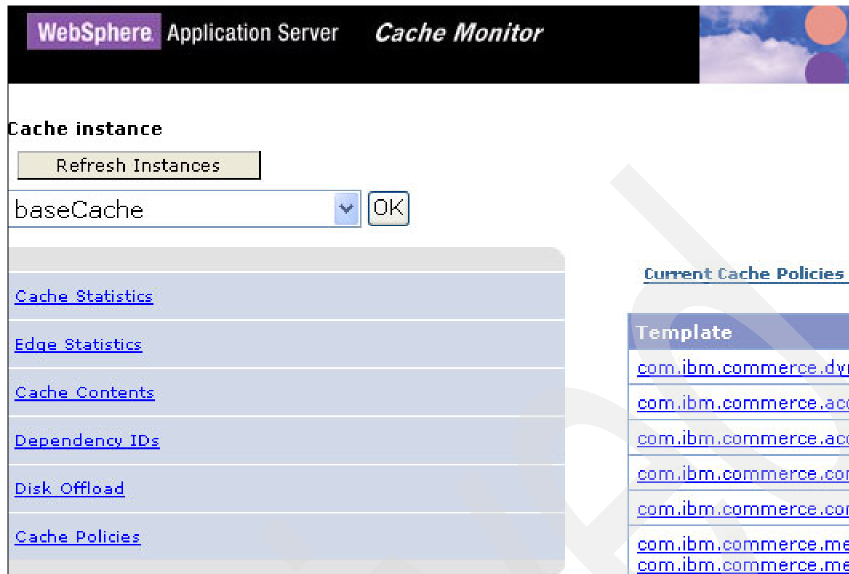



*Figure 2-21 Cache monitor viewing capabilities*

When you are viewing contents on memory or disk, you can click:

- A template to view all entries for that template
- A dependency ID to view all entries for the ID
- The cache ID to view all the data that is cached for that entry

## 2.12.3 Cache monitor operational tasks

You can use the cache monitor to perform basic operations on the data stored in a cache instance as shown in Table 2-5.

*Table 2-5 Cache Monitor operational tasks*

| Monitor operation | Task execution |
|---|---|
| Remove an entry from cache. | Invalidate when viewing a cache entry. |
| Remove all entries for a certain dependency ID. | Invalidate when viewing entries for a dependency ID. |
| Remove all entries for a template. | Invalidate when viewing entries for a template. |
| Move an entry to the front of the LRU queue to avoid eviction. | Refresh when viewing a cache entry. |

Archived

| Monitor operation | Task execution |
|---|---|
| Move an entry from disk to cache. | Send to Memory when viewing a cache entry on disk. |
| Clear the entire contents of the cache. | Clear Cache while viewing statistics or contents. |
| Clear the contents on the ESI processors. | Clear Cache while viewing ESI statistics or contents. |
| Clear the contents of the disk cache. | Clear Disk while viewing disk contents. |

In addition to using the cache monitor to track the cache, you can also use the Performance Monitoring Infrastructure (PMI) and the mBean JMX™ API, as explained in 6.9 “Monitoring DynaCache” on page 149.

Archived

3

# DynaCache invalidation

This chapter discusses the mechanics and rationale behind removing items from the DynaCache. The topics covered are:

- What invalidation is and why it is it important
- Invalidation techniques and capabilities
- Invalidation best practices

Archived

© Copyright IBM Corp. 2006. All rights reserved.

# 3.1 DynaCache invalidation defined

For each item cached by your Web site, you need to determine when the item is no longer valid, and remove its cache entry. This process is known as invalidation.

This section is a high-level overview of DynaCache invalidation in WebSphere.

## 3.1.1 Invalidation overview

Invalidation ensures that users of your e-commerce Web site are not browsing stale or invalid content from your site as a result of caching items that have changed.

In many cases, the validity of a cache entry is governed by business requirements. For example, when a catalog is updated and a new product description or price is created, the cache entry that contains the old product information will be wrong. The entry is removed from the cache so it will not be served up to a user as part of an outdated product page. Another scenario is a marketing campaign. The cache entries will become invalid at the end of the campaign.

Invalid cache entries are also known as *stale* entries. They must be evicted from the cache. In most cases the invalid cache entry is regenerated and added as a new cache entry.

A correct and efficient invalidation strategy will allow your Web site to react appropriately to change and ensures your customers or business clients get the most accurate information possible and in a timely manner.

# 3.2 DynaCache invalidation mechanisms and tools

This section provides some guidelines for the process of invalidation, as well as describing how to use the following DynaCache invalidation mechanisms:

- Invalidation policies defined within the cachespec.xml
- Programmatic invalidation using the DynaCache API
- Scheduled invalidation
- Cache Monitor Web application



Archived

## 3.2.1 The invalidation process

To work out when a cached page is no longer valid, you need to know what might make the cached page out of date. For example, a cached shopping cart page is invalidated when a customer adds a new item to the cart. Cached items may also be invalidated when an administrator updates the store with the WebSphere Commerce Accelerator or when new catalog data is added using other WebSphere Commerce tooling.

After you have listed the events or actions that make a cached page or fragment invalid, you can use those events to track down which components are responsible for building those pages. In cases where commands are invoked you should define invalidation rules in the cachespec.xml to invalidate dependent cache entries. In short, the steps are:

1. Identify the events that cause a page to become invalid.
2. Track down the components responsible for building those pages.
3. Build invalidation rules based on the components you have identified.

You can also define invalidation rules based on request parameters.

In some cases, it is not events or actions, but rather an elapsed amount of time that invalidates a cache item. For these cases you define invalidation rules based on the elapsed time since a cache entry was last used, or since a cache entry was created. You can also configure the WebSphere Commerce scheduler to invalidate cache entries at a scheduled time interval.

It is essential to have a solid understanding of the business logic and business requirements of your application to know when and how to invalidate cache items.

## 3.2.2 Cachespec.xml invalidation policies

What invalidation mechanisms are used to define invalidation policies in the cachespec.xml?

### Dependency identifiers

The *dependency identifier* is the key to configuring invalidation in the cachespec.xml. The dependency identifier <dependency-id> is defined after the <cache-id> element within the <cache-entry> element. A sample dependency identifier is shown in Example 3-1 on page 94.

Archived

*Example 3-1 Sample dependency identifier*

```
<cache-entry>
   <class>servlet</class>

<name>/ConsumerDirect/include/styles/style1/CachedHeaderDisplay.jsp</name>

   <property name="do-not-consume">true</property>
   <property name="save-attributes">false</property>

   <cache-id>
      <component  id="storeId" type="parameter">
         <required>true</required>
      </component>
      <component  id="catalogId" type="parameter">
         <required>true</required>
      </component>
      <component  id="DC_userType" type="attribute">
         <required>false</required>
         <not-value>-1002</not-value>
      </component>
      <component  id="DC_lang" type="attribute">
         <required>true</required>
      </component>
   </cache-id>

   <dependency-id>storeIdPages
      <component id="storeId" type="parameter">
         <required>true</required>
      </component>
   </dependency-id>
```

The dependency ID is a label that is used to identify which cache entries to invalidate. The same label may be attached to one, or more than one, cache entry, creating a group of cache entries. It is used in an invalidation rule to invalidate a whole group of entries.

In Example 3-1, the dependency ID base string name is `storeIdPages`. In the cache ID, the component sub-elements have component ID values. The invalidation rule compares the component IDs with the dependency ID to identify cache entries to invalidate. In the example, the component ID `storeId` matches the dependency ID `storeIdPages`. The type attribute specifies that storeId is a request parameter.



Archived

The value of the dependency-id element is generated by concatenating the dependency ID base string with the values that are returned by its component elements. An example of a dependency ID is:

```
storeIdPages:storeId:10001
```

Each cache entry may have multiple dependency IDs with one or more component identifiers.

The <require> sub element indicates that storeId must have a value. If a required component returns a null value, the entire dependency does not generate and is not used.

In summary, dependency IDs label cache entries and are used by invalidation rules to invalidate one or more cache entries at a time. Dependency IDs and invalidation rules are defined in cachespec.xml. Figure 3-1shows a high-level view of how two invalidation rules affect cache entries defined in the cachespec.xml. These rules invalidate cache objects defined by the first cache entry (dotted line) or cache objects defined by the second cache entry (solid line) when their full dependency ID strings match. Note that in the cachespec.xml, only the dependency ID base string is shown. This string is part of the dependency ID but does not represent the full dependency ID.

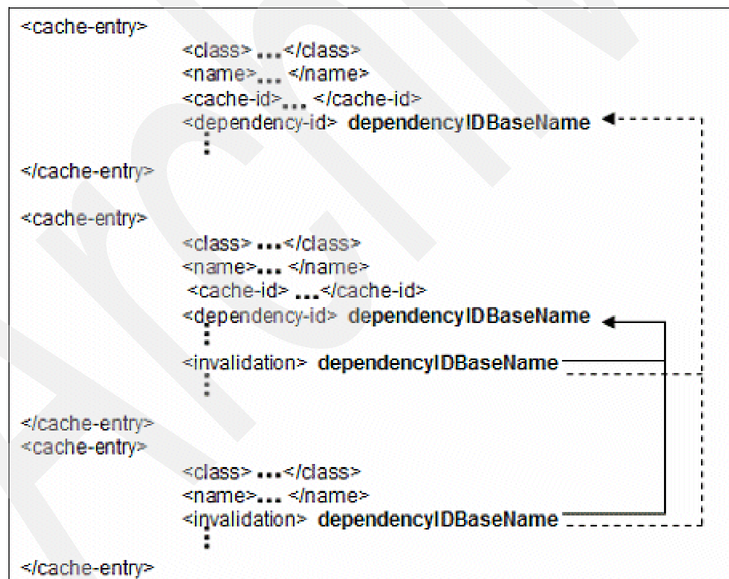



*Figure 3-1 Dependency ID: Invalidation rule relationship*

In the following section, we describe how to define invalidation policies and how dependency IDs are used by those invalidation policies to invalidate cache entries in the cachespec.xml.

## Command-based invalidation

Step one in the process of defining invalidation rules is to identify the events or actions that will invalidate a cache entry. Step two, when using command-based invalidation, is to identify the controller or task commands that are invoked when those events occur. For example, when you use the WebSphere Commerce Accelerator to update a product attribute, the ProductAttributeUpdateCmdImpl command is invoked. You can use this command to trigger an invalidation of product-related fragments or pages when the command is invoked.

Example 3-2 shows a portion of the sample invalidation cachespec.xml that can be found at WC_installdir/samples/DynaCache/invalidation.

The cache entry in Example 3-2 shows how to invalidate product-related cache entries after the product-related attributes have been updated. When the product has been updated, one of the commands listed in the cache entry, such as ProductAttributeUpdateCmdImpl or AttributeValueUpdateCmdImpl, is invoked. These command names are specified within the <name> element. As a result, the subsequent invalidation rules, which are defined within the <invalidation> tags, are executed.

In the first invalidation rule all cache entries having the dependency ID productId are invalidated if their product ID matches the ID returned by a call to the method getCatentryId as specified in the component sub-element.

*Example 3-2 Invalidation using commands*

```
<cache-entry>
   <class>command</class>
   <sharing-policy>not-shared</sharing-policy>

<name>com.ibm.commerce.catalogmanagement.commands.ProductAttributeUpdateCm
dImpl</name>

<name>com.ibm.commerce.catalogmanagement.commands.AttributeValueUpdateCmdI
mpl</name>

<name>com.ibm.commerce.catalogmanagement.commands.UpdateAttributeCmdImpl</
name>

<name>com.ibm.commerce.catalogmanagement.commands.UpdateAttributeValueCmdI
mpl</name>
```



Archived

```
   <invalidation>productId
      <component id="getCatentryId" type="method">
         <required>true</required>
      </component>
   </invalidation>

   <invalidation>MiniCart:DC_storeId
      <component id="getStoreId" type="method">
         <required>true</required>
      </component>
   </invalidation>

   <invalidation>storeId:productId
      <component id="getStoreId" type="method">
         <required>true</required>
      </component>
      <component id="getCatentryId" type="method">
         <required>true</required>
      </component>
   </invalidation>

   <invalidation>CategoryDisplay:storeId
      <component id="getStoreId" type="method">
         <required>true</required>
      </component>
   </invalidation>

</cache-entry>
```

The other invalidation elements describe how cache entries should be invalidated based on different dependency IDs.

## Rule-based invalidation

Another form of invalidation, less commonly used in WebSphere Commerce, but similar to command-based invalidation, is rule-based invalidation. Example 3-3 on page 98 demonstrates this concept by invalidating a cache entry based on a request parameter.

When the parameter with the name `action` has a value of `update`, the cache entries having the dependency ID `category` are made invalid. They are removed from the cache. Like command-based caching, the invalidation rule in rule-based invalidation is specified inside the <invalidation> tags. The component sub-element indicates the condition, that is, the action parameter must have the string value `update`.



Archived

*Example 3-3 Rule-based invalidation*

```
<cache-entry>
   <name>newscontroller </name>
   <class>servlet</class>

   <cache-id>
      <component id="action" type="parameter">
         <value>view</value>
         <required>true</required>
      </component>
      <component id="category" type="parameter">
         <required>true</required>
      </component>
      <component id="layout" type="session">
         <required>false</required>
      </component>
   </cache-id>

   <dependency-id>category
      <component id="category" type="parameter">
         <required>true</required>
      </component>
   </dependency-id>

   <invalidation>category
      <component id="action" type="parameter" ignore-value="true">
         <value>update</value>
         <required>true</required>
      </component>
      <component id="category" type="parameter">
         <required>true</required>
     </component>
   </invalidation>

</cache-entry>
```

## Timeout

A timeout rule specifies an absolute time-to-live (TTL) value for the cache entry. When a timeout element is used, the cache entry will be removed from the cache after the specified amount of time has elapsed, regardless of any other invalidation rules.

Archived

*Example 3-4 Using timeouts to invalidate*

```
<cache-id>
   <component id="storeId" type="parameter">
      <required>true</required>
   </component>
   <component id="DC_userId" type="attribute">
      <required>true</required>
   </component>
   <timeout>3600</timeout>
</cache-id>
```

The timeout value is specified in seconds, with a default value of zero. Zero means that the entry will never expire. In Example 3-4, there is a a timeout of 3600 seconds or 60 minutes, so the cache entry is invalidated after 1 hour.

## Inactivity

An *inactivity* sub-element specifies a time-to-live (TTL) value for the cache entry based on the last time that the cache entry was accessed. In Example 3-5, the inactivity value, given in seconds, indicates the amount of time to keep the cache entry in the cache after the last cache hit. Note that if a timeout value is less than the inactivity value, then the inactivity value has no meaning and the cache entry is always invalidated before the inactivity time is reached.

*Example 3-5 Using inactivity to invalidate*

```
<cache-id>
   <component id="storeId" type="parameter">
      <required>true</required>
   </component>
   <component id="DC_userId" type="attribute">
      <required>true</required>
   </component>
   <inactivity>240</inactivity>
   <timeout>3600</timeout>
</cache-id>
```

## Priority

The *priority* sub-element is used to keep some cache entries in memory while evicting others when in memory cache is full. The priority value, a positive integer between 1 and 255, delays removing cache entries from the in memory cache. The least recently used (LRU) algorithm only removes items from the cache that have a priority of zero. When the cache runs out of storage space the LRU algorithm searches through the cache entries for those with a priority of zero, and evicts them from the cache to make space for new items. Each iteration of the

LRU algorithm decrements the priority of currently cached elements by one, so that eventually even the highest priority item has a priority of zero and is evicted.

*Example 3-6 Prioritizing cache entries*

```
<cache-id>
   <component id="storeId" type="parameter">
      <required>true</required>
   </component>
   <component id="DC_userId" type="attribute">
      <required>true</required>
   </component>
   <priority>1</priority>
   <inactivity>240</inactivity>
   <timeout>3600</timeout>
</cache-id>
```

### 3.2.3 DynaCache invalidation API

Cache objects are also invalidated programmatically. This is done using the DynamicCacheAccessor object in the DynaCache API. Example 3-7 shows how to use the API.

*Example 3-7 Invalidation using the DynaCache API*

```
import com.ibm.websphere.cache.DynamicCacheAccessor;
import com.ibm.websphere.cache.DistributedMap;
   .
   .
   .
   if (DynamicCacheAccessor.isCacheEnabled()) {
      DistributedMap map = DynamicCacheAccessor.getDistributedMap();
      map.invalidate(cacheKey);
   }
```

In Example 3-7, the static method isCacheEnabled() of the DynamicCacheAccessor object is used to find out if caching has been enabled. getDistributedMap() retrieves the cache. The invalidate method using a cacheKey invalidates cache entries. The cacheKey is either a dependency ID or a cache entry ID. If the cacheKey is a dependency ID, then all cache entries having a matching dependency ID are invalidated. If the cacheKey is a cache ID, then only that cache entry is invalidated.

For more information, refer to the WebSphere Application Server v6 Infocenter. Navigate: **Reference** → **Developer** → **API Documentation** → **Application**

Archived

**programming interfaces**. Select com.ibm.websphere.cache to view DynaCache-related APIs including the DynamicCacheAccessor.

## 3.2.4 Scheduled invalidation

We have discussed invalidating cache entries with time-based elements or through a combination of command-based invalidation using dependency and invalidation IDs. These entries are invalidated using command-based invalidation rules.

Another way to invalidate cache content is by making use of the WebSphere Commerce scheduler, which periodically invalidates cache content based on entries in the CACHEIVL table, shown in Figure 3-2, using the DynaCacheInvalidation command. After installing WebSphere Commerce APAR IY88656[1] it is possible to invalidate cache entries after some future point in time. That is, you can specify a future time to begin to apply invalidation rules. For example, if you are planning to run a time-limited e-Marketing campaign, at the time you develop the e-Marketing campaign Web pages, you can also specify when the pages are to be evicted from the cache at the end of the campaign.

| Key | Name | Data type | Length | Nullable |
|---|---|---|---|---|
| | TEMPLATE | VARCHAR | 1024 | Yes |
| | DATAID | VARCHAR | 2960 | Yes |
| | INSERTTIME | TIMESTAMP | 10 | No |
| | OPTCOUNTER | SMALLINT | 2 | Yes |

*Figure 3-2 CACHEIVL table definition*

The frequency at which the DynaCacheInvalidation command is called is set by the WebSphere Commerce scheduler. To modify the frequency, launch the WebSphere Commerce Administration Console and select **Site** → **Configuration** → **Scheduler**. Figure 3-3 shows how this is done. For more information, refer to the section on the scheduler in the WebSphere Commerce Administration Guide.

[1] Obtainable as an efix from IBM service - likely to be shipped in 6.0.0.2, 5.6.0.7 and 5.6.1.3.

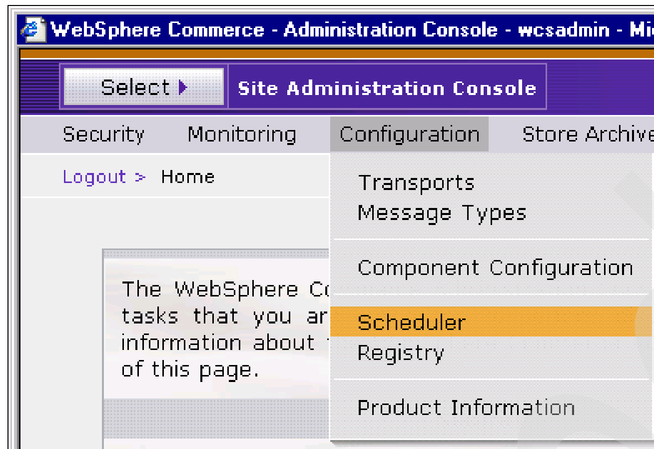



*Figure 3-3 Accessing the scheduler*

The WebSphere Commerce scheduler runs the DynaCacheInvalidation command at a set interval. This command processes the entries in the CACHEIVL table as follows:

- The clearall string value in the TEMPLATE or DATA_ID columns of the CACHEIVL table is used by DynaCacheInvalidation to clear the cache using the DynaCache invalidation API (clear).
- If the TEMPLATE column is set, then the DynaCacheInvalidation command calls the DynaCache invalidation API (invalidateByTemplate) and uses the name as the template ID. If the clearall string value (which is case insensitive) is found in the TEMPLATE column, then the DATA_ID column is ignored and the DynaCacheInvalidation command clears the cache. If the TEMPLATE column is not empty, the command invalidates using the template ID, ignoring the DATA_ID column.
- If the DATA_ID column is set and the template name is not set, then the DynaCacheInvalidation command calls the DynaCache invalidation API (invalidateById) and uses the DATA_ID as the dependency ID of the cache entries to invalidate. If the TEMPLATE column is empty and the clearall string value is found in the DATA_ID column, then the command clears the cache.
- When the DynaCache invalidation API is called, it invalidates the cache entries.

Create database triggers to populate the CACHEIVL table. Use the sample triggers in Example 3-8 to detect and react to changes in the Store and Catalog pages. This list is only a subset of several possible triggers, and is not comprehensive. Ensure that the invalidation IDs have matching dependency ID values for the cache entries you want to invalidate.

*Example 3-8 Catalog and store triggers*

```
CREATE TRIGGER cache_1
AFTER UPDATE ON catalog
REFERENCING OLD AS N FOR EACH ROW MODE DB2SQL
INSERT INTO cacheivl (template, dataid, inserttime)
(SELECT NULLIF('A', 'A'),  'storeId:' ||
RTRIM(CHAR(storecat.storeent_id)), CURRENT TIMESTAMP
   FROM storecat
  WHERE storecat.catalog_id = N.catalog_id);

CREATE TRIGGER cache_2
AFTER UPDATE ON storecat
REFERENCING NEW AS N FOR EACH ROW MODE DB2SQL
INSERT INTO cacheivl (template, dataid, inserttime)
(SELECT NULLIF('A', 'A'),  'storeId:' || RTRIM(CHAR(N.storeent_id)),
CURRENT TIMESTAMP
   FROM catalog
  WHERE catalog.catalog_id = N.catalog_id);

CREATE TRIGGER cache_3
AFTER UPDATE ON cattogrp
REFERENCING NEW AS N FOR EACH ROW MODE DB2SQL
INSERT INTO cacheivl (template, dataid, inserttime)
(SELECT NULLIF('A', 'A'),  'storeId:' ||
RTRIM(CHAR(storecat.storeent_id)), CURRENT TIMESTAMP
   FROM storecat
  WHERE storecat.catalog_id = N.catalog_id);

CREATE TRIGGER cache_4
AFTER UPDATE ON staddress
REFERENCING NEW AS N FOR EACH ROW MODE DB2SQL
INSERT INTO cacheivl (template, dataid, inserttime)
(SELECT NULLIF('A', 'A'),  'storeId:' ||
RTRIM(CHAR(storeentds.storeent_id)), CURRENT TIMESTAMP
   FROM storeentds
  WHERE storeentds.staddress_id_cont = N.staddress_id);
```

### 3.2.5 Cache Monitor

The WebSphere Application Server Cache Monitor has options to manually invalidate cache entries. Once installed, you can launch the cache monitor using one of the following methods:

- Use a Web browser with the Web address:
  `http://host_name:port/cachemonitor`

- For more secure access the Administration host machine:
`https://admin_host_name:port/cachemonitor`

If the virtual host VH_instance_name_admin was used to install the Cache Monitor, then the Cache Monitor can be accessed as:
`https://admin_host_name:8002/cachemonitor`

### Capabilities

The Cache Monitor performs the following invalidation operations:

- Remove an entry from cache. Select **Invalidate** when viewing a cache entry.
- Remove all entries for a certain dependency ID. Select **Invalidate** when viewing entries for a dependency ID or select **Invalidate** when viewing entries for a template.
- Clear the entire contents of the cache. Select **Clear Cache** while viewing statistics or contents.
- Clear the contents on the ESI processors. Select **Clear Cache** while viewing ESI statistics or contents.
- Clear the contents of the disk cache. Select **Clear Disk**.

### Installation

For details on how to install the Cache Monitor application refer to Chapter 8, “DynaCache tutorial” on page 179, which covers the process in great detail.

## 3.3 Invalidation best practices and techniques

The best practice is to invalidate as little as possible! There is no value in discarding a cache entry that is still valid just because a time-out has been reached. You should adopt an attitude that it is best if you only invalidate when you absolutely have to.

Define invalidation rules to automate the invalidation of cache entries. Without invalidation rules, the whole cache is invalidated when content changes. Everything will be removed from the cache, not just the changed pages. Invalidation of the whole cache is not recommended and you should avoid it whenever possible. It may be common for the business user to change the campaigns that they are running, and once that has been done, they will expect the system to change immediately. Establish a schedule for making changes to Web pages with the result that multiple changes are made at one time and, if some of the same pages are hit, the amount of cache invalidation required is smaller.

Archived

Be very careful with operations that invalidate the whole cache. In a production environment, accidently invalidating an entire cache may prove disastrous, so consider removing that Clear Cache button from your production cache monitor application. There are many sites that follow this policy.

It is important that you thoroughly check and test changes that require a large part of the cache to be invalidated before you push the changes into production. You don't want to incur the hit of rebuilding a large part of the cache during peak production times.

## 3.3.1 Time out considerations

The simplest way to invalidate cache entries is with time-based elements. You should think of this is a last resort. Only apply this method to user-specific objects when you cannot find any other way to invalidate the objects.

## 3.3.2 Cache monitor

From experience, we generally consider the cache monitor application to be too dangerous for use in a production environment. The main problem is the administrator may make mistakes, such as accidently pressing the Clear Cache button, and the resulting performance impact could bring down a Web site for many hours. We recommend using cache monitor for debugging activities, viewing relationships, and testing.

## 3.3.3 Dependency IDs

Keep dependency names short. Doing so lowers the amount of memory required to hold each identifier in the internal hashmap tables and can provide a performance boost because it shortens processing time.

## 3.3.4 Cache instances

Multiple cache instances help to decrease invalidation overhead. A cache performs better the smaller the cache size involved. Whenever DynaCache invalidates a cached item it places a lock on the internal cache. Therefore, the bigger the cache the longer the lock hold time and the slower the performance.

By spreading out cache entries into separate cache instances (also called cache pools) you minimize the performance impact of these locks.

Archived

### 3.3.5 Warm shutdown

Prevent unnecessary invalidation at server shutdown by configuring disk offload of all cache entries from the memory cache when the server is in the process of performing a normal shutdown. On server restart, requests for cache entries are satisfied by reloading the disk cache into memory.

> **Note:** Several minutes are added to the shutdown of a server if this option is used. The caching engine serializes Java objects to disk.
>
> Ensure that your cached objects are serializable. Test this out beforehand.

### 3.3.6 Invalidation during the tuning phase

It is considered a best practice to test invalidations during the tuning phase and measure the impact. Measure the benefit of dynacaching, as illustrated in Chapter 9, “Benchmarking DynaCache” on page 209.

### 3.3.7 Startup – use warm-up to create cache entries

Most WebSphere Commerce sites depend on DynaCache to deliver satisfactory performance. Before promoting the staging environment to production, run automated scripts to cause the cache or caches to be created. The Web site is then ready for production-level user request volumes. Warming the cache removes unnecessary stress from the business logic layers and improves stability and performance of the Web site. Be aware that warming the cache may generate a lot of traffic in a replication domain.

### 3.3.8 Impact of maintenance

Don’t forget to take into account the impact of application and system maintenance on your system caches when implementing your DynaCache strategy.

Archived

# 4

# Clustering DynaCache

In this chapter we discuss how to use DynaCache effectively in a clustered environment and explain how Data Replication Service (DRS) is used to copy data from one WebSphere node to another.

We explore some of the issues you must deal with using DynaCache in a clustered environment and, drawing upon best practices recommended by the development team, explain how to configure the Data Replication Service.

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

## 4.1 Data Replication Service

The Data Replication Service (DRS) is an internal, proprietary component of the WebSphere Application Server. The DRS is used by other components to move data from node to node within a network of clustered application servers.

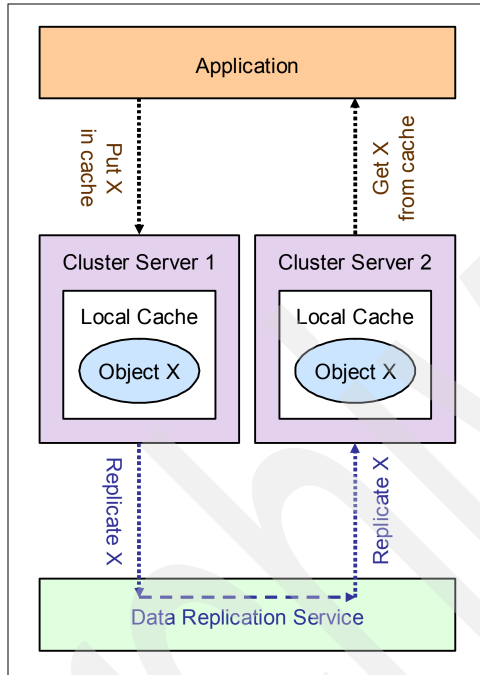



*Figure 4-1 Moving a cache entry across cluster members using DRS*

In Figure 4-2, DRS is used to share data between the four cluster members 1 through 4.

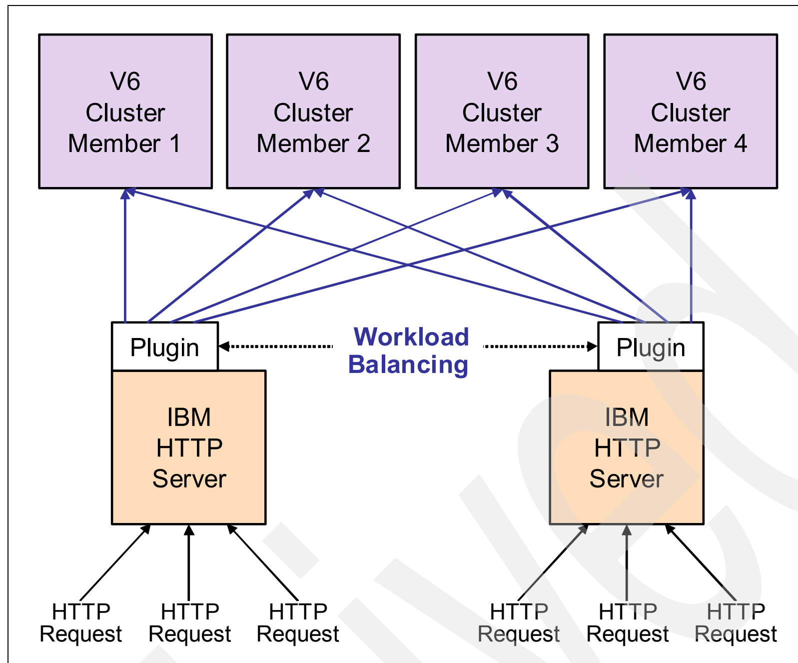



*Figure 4-2 Example of a clustered WebSphere topology*

## 4.1.1 Failover and caching

The most well known use of DRS is to replicate persistent HTTP Session data. If an application server fails, the request is routed to another application server, and the session data will be available there.

In order to minimize the impact of a failure, DRS coordinates with the WebSphere Cluster Workload Management routing algorithm to ensure requests and data end up in the same place. For example, a new capability of WebSphere Application Server v6 is to capture a stateful session bean and enable failover to another instance of that bean in a different WebSphere Application Server.

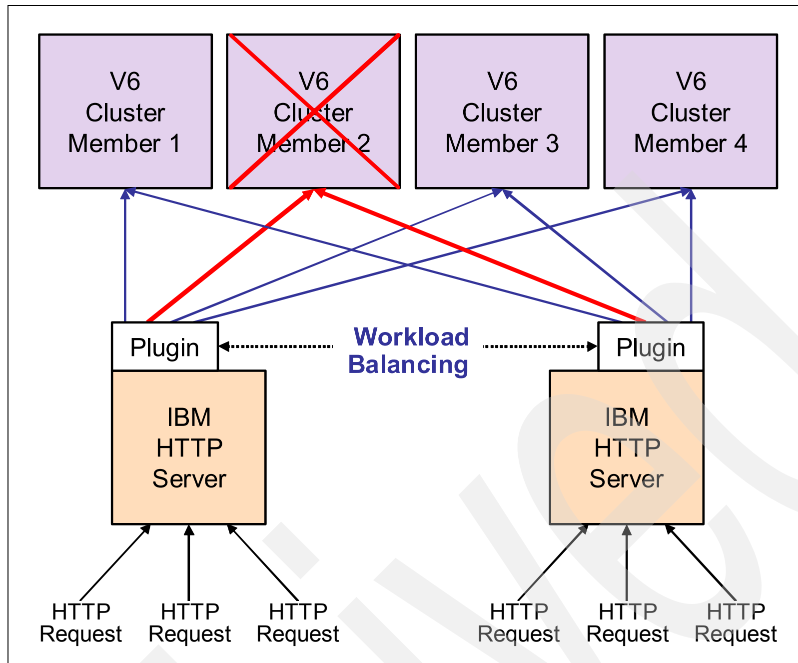



*Figure 4-3 WebSphere supports failover in a clustered environment*

The Data Replication Service caters to two scenarios: failover and caching.

## 4.1.2 DRS and failover

Failover support ensures that HTTP Sessions and EJBs can be moved to another application server. This is transparent to the user but it is important that the you understand how this is implemented internally to design the best configuration to work with HTTP session and caching scenarios. The best practice, as explained at the end of this chapter, is to implement separate replication domains for each service.

## 4.1.3 DRS and caching

The second use of Data Replication Service is for caching. From the previous chapters we know that once an object is available in the cache, repeated requests for the same information are handled faster. Since the data in the cache must be the same, no matter which application server the request arrived at, it makes sense to cache the same data in of all the application servers. Data Replication Service handles moving cache entries from server to server.

## Replicators and replication domains

*Replicators* are the producers and consumers that are responsible for moving data from one location to another in a clustered environment. In Version 5.x of WebSphere Application Server, the underlying transport used is Java Message Service (JMS) messages.

More configuration is required in WebSphere Application Server v5 than in v6. Administrators need to create replicators within a replication domain. The default configuration is for all the application servers in a domain to connect to all the other application servers, as shown in Figure 4-4.

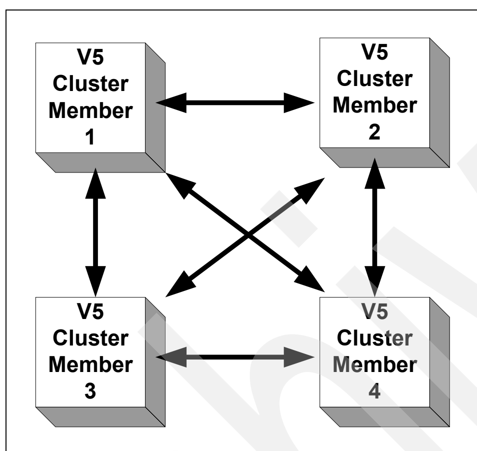



*Figure 4-4 Default configuration for Version 5 DRS.*

It is possible to reduce the overhead by limiting which application servers talk to which by using *partitions*. Those that are configured to talk to each other are a partition and also part of a *Multi-Broker Replication Domain*.

In WebSphere Application Server v6 it is not necessary to create and configure replicators, and the concept of partitioning is masked. It is still possible to limit the number of copies of the data, but it is not necessary to expose the details of the configuration.

## N-way peer to peer

Look at the configuration in Figure 4-4. The cluster members are application servers that have been created as members of a cluster. Data Replication Service is used to copy session data between the clustered application servers.

The default topology for Data Replication Service in Version 5 is for all replicators to service all channels in the replicator domain. This means for four application

Archived

servers, there will be three backups and the original. You can see how this could quickly consume a lot of memory.

To reduce the memory overhead without losing the backup, you can set the *Single Replica* flag to limit the replicators to one backup copy. In the event of a failover, the Session Manager will find the backup and make it available to the application server where the request lands.

### Single replica

The single replica configuration scales much better in terms of memory consumption. In the event of a failure of an application server, the request will be routed to the next clustered application server in the rotation. If the session information exists in the new server, it is used; if not, the session manager retrieves it from wherever it was backed up. As soon as the request hits the new server, its updated session information is sent to another server, so there is always a backup. (Unless the wrong two servers happen to fail at the same time.)

Client Server to Client Server topology is one where another application server is configured to store backup session data in the local memory space.

This topology reduces the overhead on application servers handling requests, but introduces a single point of failure (SPOF). It is useful to configure two independent application servers to store backup session information. By introducing a second server on a second machine, the single point of failure is eliminated.

A replication domain consists of servers and cluster members that have the capability to replicate information from one cluster member to any other cluster member.

Some changes in WebSphere Application Server v6 include cooperation between the Data Replication Service and the Workload Management subsystem to coordinate which cluster members serve as backups for other cluster members.

Ideally, session failover data and stateful session bean failover data should end up in the same place – the place that a failed-over session will arrive in the event it needs to be served by a cluster member other than the one that originated the session.

Another improvement is that the underlying mechanism has been rewritten using a proprietary transport to move data. This reduces overhead and improves overall system performance.

## WebSphere Application Server v5 to v6 migration

The change in the underlying communication mechanism removes the need for replicators. The only configuration decision you make is how many backup copies of the data will be needed. The default is one.

Version 6 benefits from a faster transport mechanism, the channel framework, which eliminates the one-thread-per-queue limitation. Sitting on top of a more robust transport also removes the need for manual partitioning.

Creating a replication domain is as easy as selecting a checkbox in the Administrative console, or you can manually create a domain. Because Data Replication Service is used for both cache replication and session data, you can configure cache replication under Server, then Container Service, then select DynaCache Replication.

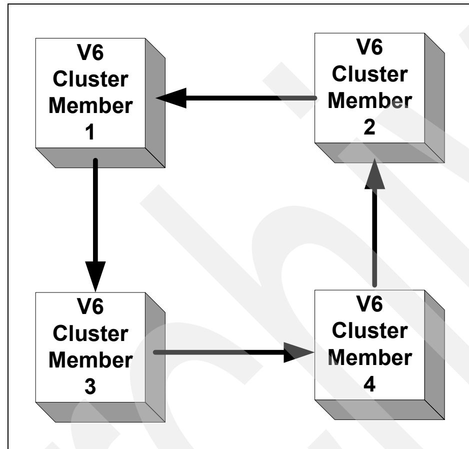



*Figure 4-5 Default topology configuration for V6 Distributed Replication Service*

Archived

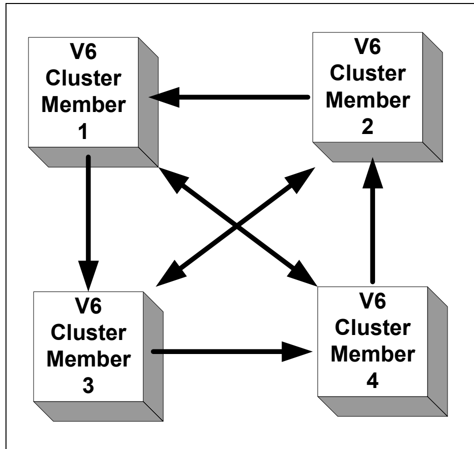



*Figure 4-6 Topology required for DynaCache DRS to function properly*

In the default topology (see Figure 4-5 on page 113), each server in the domain holds a replica of the data from one other server. In the second example (see Figure 4-6), the double headed arrows mean that data flows from each process to every other process, so that for each replicated object, there are three remote copies and the local original.

This topology would probably be more than what is needed for HTTP Session replication, but in our DynaCache environment, it is the only allowable configuration for cache replication because when caching dynamic content, the cache is only useful if it is available on all the machines where a request could arrive.

## 4.2 Replication in DynaCache

Highly available, load-balanced, WebSphere-based production environments will have several application servers joined together in a cluster. With appropriate HTTP session replication configurations in place, any arriving HTTP client request is serviced by any member of the cluster.

DynaCache also supports data replication. Certain cache entries are highly reusable across users, and shared between servers in a cluster. Configure cache replication to accomplish this. Cache replication is also necessary to ensure that invalidation messages are shared between the servers in a cluster.

Archived

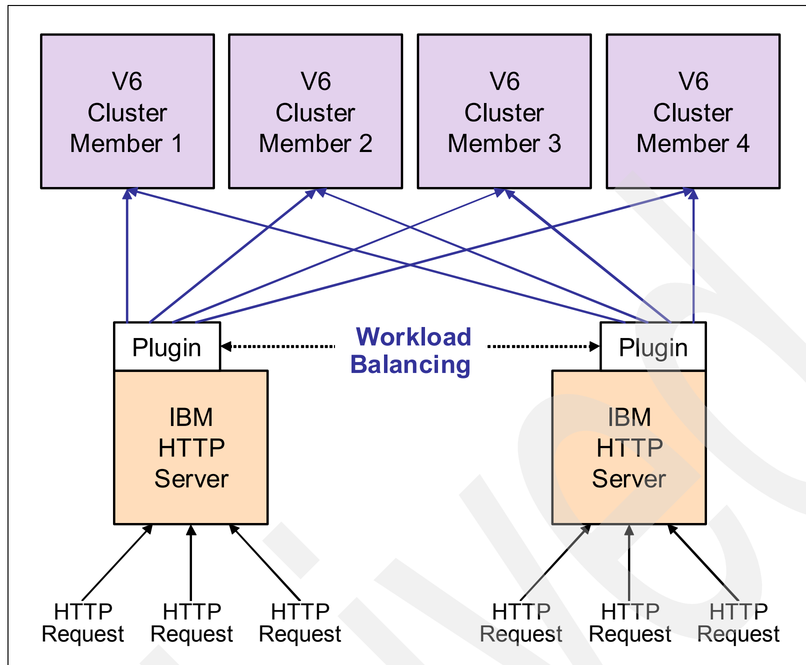



*Figure 4-7 Workload balancing between multiple servers*

Sharing cache content creates some network traffic. To minimize the impact, only replicate valuable content. Controlling what is replicated in the sharing policy section is discussed next.

## Sharing policy

When you are preparing your Web site to operate within a cluster and planning to use DynaCache distributed capabilities, you should be aware of particular operational aspects that you can control. The first thing to be aware of is the *sharing policy.*

Specifying which cache entries are shared between the servers controls which cache entries are replicated. Specify in each cache entry the *cluster sharing* characteristics that control if a cached item is replicated to other the cluster members.

The most likely sharing candidates are objects whose content is cached at the servlet level since that content generally does not contain personalized data. Those kinds of entries are a good starting point for your selection process.

One good practice to work with concerns the sharing of JSP fragments. Only replicate cached JSP fragments that have a long life expectancy in cache, and

are reusable across users. The only exception to this rule is if the cost to create the JSP fragment is very high and it would be more economical to replicate to the other cluster members than have them recreate it. Remember, if you have chosen to use the Push policy, then once any cache item is replicated, the other cluster members use that cache entry and do not recreate a local copy from scratch.

### 4.2.1 Specifying the sharing policy declaration in the cachespec.xml

This sharing policy defines how data is replicated from server to server. By default all cache entries are not shared across a cluster. You override that by specifying the sharing policy for each cache entry you want replicated. You also need to configure replication in your environment via the administration console.

*Example 4-1 Sample cache-entry showing <sharing-policy> being used*

```
<cache-entry>
<class>servlet</class>
<name>/ConsumerDirect/include/MiniShopCartDisplay.jsp </name>
<property name="save-attributes">false</property>
<property name="store-cookies">false</property>
<property name="save-attributes">false</property>
<property name="do-not-consume">true</property>
<sharing-policy>not-shared</sharing-policy>
<cache-id>
        <timeout>3600</timeout>
            <component  id="storeId" type="parameter">
                <required>true</required>
            </component>
            <component  id="catalogId" type="parameter">
                <required>true</required>
            </component>
            <component  id="DC_userId" type="attribute">
                <required>true</required>
            </component>
 </cache-id>
```

Table 4-1 on page 117 explains in detail what options you have when setting the value for the <sharing-policy> property. Note that if the <sharing-policy> element is not present, a not-shared value is assumed.

In single server environments, not-shared is the only valid value. When enabling replication, the default value is shared-push only. This property does not affect distribution to Edge Side Include processors through the Edge fragment caching property.

Archived

*Table 4-1 Sharing policy options*

| Value | Description |
|---|---|
| not-shared | Cache entries for this object are not shared among different application servers. These entries can contain non-serializable data. For example, a cached servlet can place non-serializable objects into the request attributes, if the <class> type supports it. |
| shared-push | Cache entries for this object are automatically distributed to the DynaCaches in other application servers or cooperating Java virtual machines (JVMs). Each cache has a copy of the entry at the time it is created. These entries cannot store non-serializable data. |
| shared-pull[a] | Cache entries for this object are shared between application servers on demand. If an application server gets a cache miss for this object, it queries the cooperating application servers to see if they have the object. If no application server has a cached copy of the object, the original application server executes the request and generates the object. These entries cannot store non-serializable data. This mode of sharing is not recommended. |
| shared-push-pull | Cache entries for this object are shared between application servers on demand. When an application server generates a cache entry, it broadcasts the cache ID of the created entry to all cooperating application servers. Each server then knows whether an entry exists for any given cache ID. On a given request for that entry, the application server knows whether to generate the entry or pull it from somewhere else. These entries cannot store non-serializable data. |

a. Shared-pull is not normally recommended, consider shared-push-pull instead.

## 4.2.2 Troubleshooting

To troubleshoot a DRS problem, examine the SystemOut.log for all servers in the cluster, and their server.xml files, which contain the DRS configuration information.

To search deeper, trace the group named DRS, or the com.ibm.ws.drs.* component to find more detailed information.

### Replication problem: Message size

The replication service has a default value of 5 MB for the maximum message size transmitted over the wire. The maximum message size can be changed in WebSphere Commerce v6. When the consumer service (Http session memory to memory replication or WebSphere DynaCache service) tries to send messages larger than the maximum message size, the replicator that is instructed to send this message disconnects the client connection to the

replication service, causing broken pipe exceptions. The exceptions happen either immediately, or on subsequent message sends. The exceptions cause the replication service instance to reset its connectivity to the current replicator and it attempts to connect to an alternate replicator in the domain. Thus, this problem may also manifest itself in the form of frequent reset attempts in the logs, as shown in Example 4-2.

*Example 4-2 Connect reset log*

```
[6/25/04 11:32:57:888 EDT]    3d2b3 DRSResetJMS   A DRSW0005I:
WebSphere internal replication has recovered from a previous connection
failure.
   [6/25/04 11:32:58:038 EDT]    3d2b3 DRSCacheApp   E DRSW0001E: A error
   occured communicating over WebSphere internal replication. The
   exception is: com.ibm.disthub.impl.jms.JMSWrappedException:
   {800870265|java.io.IOException: Broken pipe|at
   com.ibm.disthub.impl.jms.TopicPublisherImpl.publishInternal
   (TopicPublisherImpl.java:511)
           at com.ibm.disthub.impl.jms.TopicPublisherImpl.publish
   (TopicPublisherImpl.java:450)
           at com.ibm.disthub.impl.jms.TopicPublisherImpl.publish
   (TopicPublisherImpl.java:375)
           at com.ibm.ws.drs.DRSJMS.jmsPubUpd(DRSJMS.java:200)
           at com.ibm.ws.drs.DRSCacheApp.jmsPubUpd(DRSCacheApp.java:3290)
           at com.ibm.ws.drs.DRSJMS.jmsPubUpd(DRSJMS.java:272)
           at com.ibm.ws.drs.DRSCacheApp.jmsPubUpd(DRSCacheApp.java:3290)
           at com.ibm.ws.drs.DRSAPI.updateEntryProp(DRSAPI.java:803)
           at com.ibm.ws.drs.DRSCacheApp.updateEntryProp(DRSCacheApp.java:
```

## 4.3 Best practices

The following recommendations are based on advice the authors received from the development team.

- Do not replicate mini carts or sticky sessions.
- Do not use wsadmin scripts that are intended for version 5 because they use multi-broker domains, which do not have the performance benefits of v6 replication domains.
- Create a distinct domain for HTTP and EJB session data, and another domain for Cache replication.
- Use only a small number of replicas to improve performance. Increasing the number of replicas may reduce the time it takes a session to move to another

server, but it does so at the cost of overall performance. One, two, or three replicas should be sufficient in most cases.

- Use dedicated DRS servers for replication.
- When considering which replication policy to use, consider first using the simplest, SHARED-PUSH, and perhaps only for the most expensive cache entries.

Archived

Archived

5

# Caching strategy

In this chapter we describe important design considerations when creating a cache policy for new and existing Web sites. We begin by discussing the non-functional requirements of a Web site that may impact performance and help to drive the need for and definition of a cache policy. Next, we discuss the process of identifying objects that are suitable for caching, recommended strategies for creating a cache, and the tools used to find cacheable objects. We then analyze our cache objects to identify the policies that will determine when we will invalidate those objects. Finally, we discuss important mechanisms used to ensure that JSPs are correctly cached using DynaCache.

In particular, we discuss the following topics in this chapter:

- Site requirements
- Identifying cache objects
- Invalidating cache objects
- DynaCache and JSP

© Copyright IBM Corp. 2006. All rights reserved. 

## 5.1 Site requirements

Requirements are key to any software project and it is no different with creating a WebSphere Commerce site that can benefit from the caching capabilities of DynaCache. These requirements can help you drive the design of your WebSphere Commerce implementation so that its components can be cached at the correct level of granularity, and can also help to create an effective cache policy.

Listed here are some of the key performance requirements:

- *Concurrent users:* The total number of active users that are expected to be using your e-commerce site. A user may visit your site many times but can have only one session at a time. When considering a requirement for concurrent users, an important metric is *think-time*, which refers the time between requests for any given user. As you increase think-time, the number of supported concurrent users will increase and think-time will vary by browse scenario.
- *Peak versus average usage:* The variance in the load that is being placed on your e-commerce site. This can depend on a variety of conditions, for example, time of day, time of the year, or special events. Performance requirements have to be maintained during peak usage as well as during average usage.
- *Page views per second*: Throughput in terms of the total number of times a user visits or views a page, measured per second and scaled over multiple users. A page view includes requests for all the files that are contained on a page as well as the page itself. A page view can include one or more hits, where a *hit* is any type of request to the server.
- *Response time:* The amount of time required to complete a single page view or page hit, and measured in seconds. Response time will depend heavily on other requirements and caching needs to be taken into account to optimize response time.
- *Kilobytes per second:* Throughput in terms of the total number of bytes that can be transferred to a user per unit of time, usually seconds. Caching can help to meet this type of requirement, especially when bandwidth is limited.
- *Browse-buy ratio:* The ratio of the number of users who are visiting your e-commerce site but not buying products versus the number of users who are buying products, meaning checking out from their shopping cart. Typically, a B2B e-commerce site will have a smaller browse-buy ratio than a B2C site. For example, a B2C browse-buy ratio might be 100:2 or 2 percent buyers while a B2B site might have a browse-buy ration of 100:35 or 35 percent buyers. In scenarios where the browse-buy ratio is extremely high, for example in B2C sites, caching can play an especially crucial role in improving

Archived

performance since it is more likely that we can cache pages related to browsing versus pages related to buying. In WebSphere Commerce, we cache the catalog pages that many users will browse through rather than the checkout pages, which are unique to a single users at a single instance.

- *Locale:* The location from which users will be browsing your Web site. This is particularly significant if your Web site must support a variety of locales and the amount of traffic from each locale may be known. For example, if your site must support both English and French, but you know that 95% of your visitors are English speaking, then clearly caching the English pages will yield a greater benefit than caching the French ones.

When creating an e-commerce site, there are a variety of performance requirements that the site must meet. Each of these requirements demand that performance be fine tuned, including a strategy for caching to improve performance. Furthermore, capacity planning models created and used by IBM assume that any WebSphere Commerce implementation will implement caching using DynaCache.

E-commerce applications must always be concerned with performance, especially in the case of B2C WebSphere applications, and as a result the use of DynaCache in these scenarios is not optional.

## 5.2 Identifying cache objects

When planning to use DynaCache as part of your caching strategy, first identify the objects that are available to be cached and then identify which of those objects have the qualities that make them good candidates for caching. A variety of tools and methods are available to help you do this.

### 5.2.1 Characteristics of cacheable objects

Several types of objects, such as Java Server Pages (JSPs), servlets, and WebSphere Commerce commands are cached in DynaCache. Among these objects, identify those that have cacheable characteristics.

Good candidates for caching are usually:

- Long lived and more static so that they are less likely to be invalidated
- Reusable by many users and with little personalization or custom content
- Highly reusable across many parts of your site, for example, reusable JSP fragments

- Are large enough to have an impact on performance, for example, full-page caching
- Self-executable even as fragments
- Free of security-sensitive dynamic data

Poorer candidates for caching are usually:

- Locale-sensitive
- Highly variable based on their request parameters
- Highly dynamic, that is, they are invalidated frequently
- Have security considerations that require finer grained access to the cache

Consider carefully:

- User-specific objects- these usually give a good return, but not as good as generic candidates

Keep in mind that the greatest benefit from caching is achieved when objects are cached closer to the client and earlier in the processing cycle of the request. In the WebSphere Commerce architecture this means we should cache JSPs and servlets, and consider caching commands within our application server, as shown in Figure 5-1. Caching a request from a client at a later stage of processing lowers the performance benefit since the earlier layers introduce additional processing time and latency. Cached commands are processed by the Web container. They improve performance by avoiding both EJB invocation and requests to the database server.

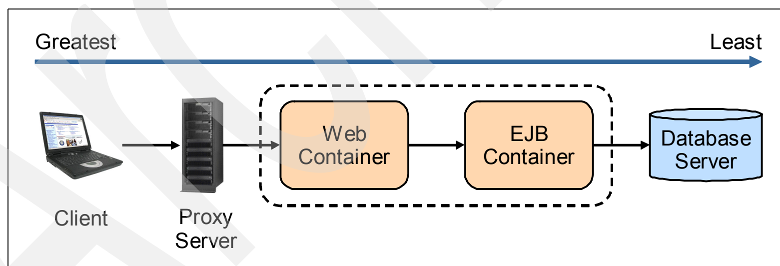



*Figure 5-1 Benefit of caching*

## 5.2.2 Tools and methodology

How do we find objects of the types we have chosen to cache in our Web application? There are many objects in a large Web application: how do we select the ones we will work on to make them part of the cache policy? In this

section we describe some of the tools that are available to identify cacheable objects as well as some of the recommended methods of using the tools.

## Analytics

Analyze Web server logs for existing Web sites to identify traffic patterns using tools such as Perl scripts. This helps to identify important servlets to cache. Count the number of hits for each unique servlet URL, ignoring any request parameters. Servlets with a large number of hits are good candidates for caching. Group the hits by hour to find out which servlets were accessed the most at what time. These servlets must be in the cache at the right time in order to reap the greatest performance benefit.

Web server logs also show the content length, or size of the servlet request. When a servlet is cached, knowledge of its size helps us to fine tune the cache by setting an appropriate cache size in terms of cache entries. JVM performance is improved by reducing the amount of time spent doing compaction.

Use third-party analytics tools to identify servlets that should be cached. For example, use such analytics reports to determine which pages are hit most frequently by visitors to the site. If those pages are also suitable for caching, then modify the cache policy to cache those pages and improve performance.

## Tools on alphaWorks

The alphaWorks® Web site publishes new technologies and research for early adopter developers to use and test. It contains many resources, including the DynaCache Policy Editor (DCPE). The DCPE was built for WebSphere Studio Application Developer 5.1 and may not be compatible with the latest WebSphere Commerce toolkit, which runs on Rational Application Developer (RAD).

### *DynaCache Policy Editor*

The DynaCache Policy Editor (DCPE) is an Eclipse plug-in that is used to generate, create, or edit cache policies for DynaCache from within WebSphere Studio Application Developer (WSAD).

The editor validates cache policies against an XML schema, assists in completing cache policy XML, and ensures that invalid changes are flagged immediately. The plug-in also includes a simple tool that analyzes servlets and JSPs and then generates cache policies and adds them to the cache policy file.

For more information see:
http://www.alphaworks.ibm.com/tech/cachepolicyeditor

Archived

### Code review

Try to identify commands that are cacheable. Review the source code, JavaDocs or other API documentation, and identify commands that conform to the WebSphere Command Framework and extend the CacheableCommandImpl class, as well as those that do not. You may be able to identify new commands or commands that, if modified, can be made cacheable. You may also identify ways to implement commands that lead to more efficient invalidation policies.

# 5.3 Cache design

In this section we recommend some approaches to defining an efficient cache policy, including full-page and fragment caching as well as cache instances.

## 5.3.1 Full-page caching and fragment caching

Full-page caching must be performed by a servlet. In the cachespec.xml, we specify that all fragments of a servlet are cached along with the servlet that includes them, using the *consume-subfragments* property. The cache entry for a full cached page contains:

- The servlet
- The content from the servlet's fragments that have no includes or forwards

Full-page caching is encouraged when many, if not all, users share the page. Good candidates in WebSphere Commerce for full-page caching are the catalog pages. Even pages containing personalized information can be cached at the this level.

Full-page caching maximizes what is contained in a given cache entry by caching the largest possible result of any request to your servers, and minimizes the time required to return the cached information on its subsequent requests. For this reason, use full-page caching whenever possible.

Sometimes full-page caching is not possible, because of user-specific fragments that need to be created for each user-page-view. Exclude those fragments from the full page cache entry using the *do-not-consume* property. As a result, DynaCache will achieve near full-page caching performance. The user-specific fragments are cached using separate cache entries. All of the pieces are reassembled from the cache when there is a request for the page. As the number of fragments increases, so does the time required to reassemble them.

In the event that fragment caching is not possible, you can mark a fragment as uncacheable. For example, a page is marked for full-page caching but has one or more fragments that are not cacheable. By setting the *do-not-cache* property to

true for a fragment in the cachespec.xml, you ensure that the fragment will not be cached or consumed by its parent. This way, you can still reap the benefit of caching a page and most of its fragments.

When possible, instead of marking a fragment as uncacheable, first consider spending some time to rewrite it or create the appropriate invalidation policies in order to make it more suitable for caching.

### 5.3.2 Cache instances

Multiple cache instances are supported in WebSphere Application Server version 6 and later. The pools are configured using the WebSphere Application Server administration console, properties files, or resource references.

Create multiple cache instances to segregate cache entries into different pools based on their types, namely servlets, JSPs, and commands. This prevents a frequently occurring scenario where several small cache entries for commands will cause a larger servlet cache entry to be removed from the cache.

Another use of cache instances is to segregate objects that are very costly to build but are also infrequently used, and thus likely to be evicted.

Cache instances also improve performance by reducing synchronization time. When under load, DynaCache will synchronize the cache in memory with the cache on disk. During this time, nothing can be read from or written to the cache, effectively disabling the cache. Creating multiple smaller caches will help to reduce synchronization time.

## 5.4 Invalidating cached objects

As described in the previous chapter, invalidation of cache objects is initiated declaratively through the cachespec.xml, programmatically using the DynaCache API, or administratively, for example, using the WebSphere Commerce scheduler or Cache Monitor. When possible, invalidation policies are declared in the cachespec.xml. This allows you to remove only a stale subset of objects in the cache, rather than clearing the entire cache. Invalidating the entire cache is not recommended and should be avoided whenever possible.

Although invalidation policies are not included in the default cachespec.xml, several invalidation examples can be found here:

WC_installdir/samples/DynaCache/invalidation

Archived

In general avoid the use of timeouts since the timeout will override other invalidation triggers and may invalidate a cached object that may not be stale. Instead, use commands, database triggers, or inactivity to initiate invalidation of cached objects.

To use command-based invalidation, your commands must conform to the WebSphere Command Framework and extend the CacheableCommandImpl class.

## 5.5 DynaCache and JSP

There are some important techniques and features of DynaCache for correctly caching JSPs. They are discussed here with reference to the inventory count example where applicable.

### JSP dynamic includes

When using the <jsp:include> tag to dynamically include a JSP fragment into another JSP, always set the `flush` attribute to `true` so that those fragments will be cached correctly by DynaCache. In Example 5-1, we include the JSP fragment to display in-stock inventory while ensuring that the flush attribute is correctly set.

*Example 5-1 JSP dynamic include with flush*

```
<jsp:include
path="../../../Snippets/ReusableObjects/InventoryCountDisplay.jsp"
flush="true">
   <jsp:param name="fulfillCentreId"
value="${CommandContext.store.fulfillmentCenterId}" />
   <jsp:param name="itemSpcId" value="${catalogEntry.itemspc_id}" />
   <jsp:param name="storeId" value="${WCParam.storeId}" />
</jsp:include>
```

The JSPWriter buffers the data rather than flushing it to the cache's writer when interpreting the include tag. The buffering prevents DynaCache from knowing when the application stopped writing the data to the parent writer. As a result, the child fragment's content might also be cached as part of the parent's cache entry, causing the child fragment to appear twice.

### Using the Java Server Tag Library (JSTL)

WebSphere Commerce also uses the Java Server Tag Library (JSTL) as an alternate mechanism to dynamically include JSP pages. This is done through the use of the <c:import> tag, which supports relative and absolute URLs, while the



Archived

standard <jsp:include> tag only supports relative URLs. If <% out.flush(); %> does not surround the <c:import> tag, DynaCache will introduce problematic behavior. The reason and result of this behavior is the same as what happens when the flush attribute on the <jsp:include> tag is not set to true, which is described in the previous section. To prevent this problem, surround the <c:import> tags with <% out.flush(); %>. Flush the buffer before the import begins, and after it ends, to ensure that none of the child content is written to the parent's cache entry.

Example 5-2 shows how the inventory count JSP fragment is dynamically included into another JSP using the <c:import> surrounded with the appropriate out.flush tags.

*Example 5-2 JSTL import with flush*

```
   <%out.flush();%>
   <c:import
url="../../../Snippets/ReusableObjects/InventoryCountDisplay.jsp">
      <c:param name="fulfillmentCenterId"
value="${CommandContext.store.fulfillmentCenterId}" />
      <c:param name="itemSpcId" value="${catalogEntry.itemspc_id}"/>
      <c:param name="storeId" value="${WCParam.storeId}"/>
   </c:import>
   <%out.flush();%>
```

The out.flush statement is required for all <c:import> statements, not just those for fragments. Even if the page is being cached using full-page caching, the out.flush tags are still required to surround the <c:import> tags.

## Caching with Struts

Another important technology framework for WebSphere Commerce is Struts. As of version 6.0, WebSphere Commerce has moved from its proprietary model-view-controller implementation to the Struts open source implementation, developed by the Apache Software Foundation. Struts is a well-documented framework for J2EE Web application development, and has become an industry standard for deploying model-view-controller applications.

For more information on Struts, see:
http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.developer.doc/concepts/csdstrutskeycompons.htm

### *Using WebSphere Application Server prior to version 6.0*

In the previous version of WebSphere Application Server, only one cache entry per servlet was supported, as shown in Example 5-3 on page 130. However, when you are using Struts, every request that ends in `.do` maps to the same



Archived

*ActionServlet* servlet. To cache Struts responses, write a cache policy for the ActionServlet servlet based on its servlet path.

For example, consider two Struts actions: /HelloParam.do and /HelloAttr.do. To cache the responses based on the `id` request parameter and the `arg` request attribute respectively, use the cache policy shown in Example 5-3.

*Example 5-3 Single cache entry for multiple Struts actions*

```
<cache-entry>
   <class>servlet</class>
   <name>org.apache.struts.action.ActionServlet.class</name>
   <cache-id>
      <component id="" type="servletpath">
         <value>/HelloParm.do</value>
      </component>
   </cache-id>
   <cache-id>
      <component id="" type="servletpath">
         <value>/HelloAttr.do</value>
      </component>
      <component id="arg" type="attribute">
         <required>true</required>
      </component>
   </cache-id>
</cache-entry>
```

### Using WebSphere Application Server, Version 6.0 or later

The current version of WebSphere Application Server is capable of mapping multiple cache entries per servlet. The cache policy in Example 5-3 has been rewritten in Example 5-4 with more than one cache-entry.

*Example 5-4 Multiple cache entries for each Struts action*

```
<cache-entry>
   <class>servlet</class>
   <name>/HelloParam.do</name>
   <cache-id>
      <component id="id" type="parameter">
         <required>true</required>
      </component>
</cache-entry>
<cache-entry>
   <class>servlet</class>
   <name>/HelloAttr.do</name>
   <cache-id>
```



Archived

```
      <component id="arg" type="attribute">
         <required>true</required>
      </component>
   </cache-id>
</cache-entry>
```

## Caching with Tiles

Among the key benefits of the Struts framework in WebSphere Commerce is its support for Tiles. *Tiles* is a templating system that is used to define layouts into which content, called Tiles, can be dynamically inserted. Tiles allow developers to create reusable content, layouts that are easy to manage, and a consistent look and feel in a Web application.

For more information on Tiles, see:
http://struts.apache.org/1.x/struts-tiles/

The Tiles framework is built on the <jsp:include> tag, so everything that applies to JSP caching also applies to Tiles. The `flush` attribute of the <tiles:insert> tag must be set to `true`, as shown in Example 5-5, if any fragments that are included using the <tiles:insert> tag are to be cached correctly.

*Example 5-5 A JSP using Tiles*

```
<html>
<body>
<tiles:insert page="layout.jsp" flush="true">
   <tiles:put name="header" value="/header.jsp" />
   <tiles:put name="body" value="/body.jsp" />
   <tiles:put name="footer" value="/footer.jsp" />
</tiles:insert>
</body>
</html>
```

DynaCache also provides support for Tiles attributes. A layout template may include or insert a page that requires attributes from its parent, as shown in Example 5-6. In this example, the *header* Tile requires the *userType* attribute. This attribute is defined in the layout.jsp template file and passed to the header Tile using the nested <tiles:put> tag.

*Example 5-6 layout.jsp*

```
   <%String userType = "123"; %>
   <tiles:insert attribute="header">
      <tiles:put name="userType" value="<%= userType %>" />
   </tile:insert>
```

```
   <table>
      <tr>
         <td> <tiles:insert attribute="body"> </td>
      </tr>
      <tr>
         <td> <tiles:insert attribute="footer"> </td>
      </tr>
   </table>
</body>
</html>
```

To cache the header Tile based on the userType attribute we must define a cache entry for it in the cachespec where userType is marked as a Tiles attribute in the cache-id. See Example 5-7.

*Example 5-7 Cache entry for the header Tile*

```
<cache-entry>
   <class>servlet</class>
   <name>/header.jsp</name>
   <cache-id>
      <component id="userType" type="tiles_attribute">
         <required>true</required>
      </component>
   </cache-id>
</cache-entry>
```

6

# Advanced topics

This section discusses a number of more advanced topics concerning caching, which are best studied after you have read the chapters on implementation.

The first section describes the improvements introduced in Version 6 of WebSphere Commerce to the management of disk caching, and enhancements to cache policies made in response to lessons learned from experiences using DynaCache.

The chapter also includes tuning advice and guidance on how to monitor Edge Side Includes with the Cache Monitor.

The last part of the chapter is devoted to a brief reference section largely derived from the InfoCenter. This is primarily included for convenience when reading this Redbook, and partly to add some usage notes to help you interpret the reference information given in the Infocenter.

© Copyright IBM Corp. 2006. All rights reserved.

## 6.1 What is new in Version 6 of DynaCache

WebSphere Application Server provides Struts and Tiles caching, Web Services client and server caching, edge caching using Edge Side Includes, JSR 168 Portlet caching, servlet/JSP fragment caching, distributed map, and command caching. In WebSphere Application Server v6.1, WebSphere Dynamic Caching can be configured as an in-memory or as a disk cache, or both. The majority of the DynaCache improvements in WebSphere Application Server v6.1 are in the disk cache infrastructure.

The DynaCache service allows persisting objects to disk (specified by a file system location) so that objects evicted from the memory cache do not have to be regenerated by the application server. The disk caching technology was integrated as a overflow or spill-over cache.

However, the DynaCache designers have observed that the majority of commercial implementations treat the disk cache as the primary cache and the memory cache as a buffer. This has brought about a redesign in the disk cache and memory cache integration services and it is in this area where most of the infrastructure development changes occurred.

### 6.1.1 Disk cache enhancements

As more objects get stored on the disk, we need to limit the size or number of entries that are on disk, and we need to have greater control over what gets put on disk and what the behavior is when the disk is full.

The total number of objects that need to be cached and size of these objects often imply that the primary store for these objects is the disk. The changes are the following:

- A new option has been provided to use the disk cache as primary object store (per cache instance and per cache entry).
- You can specify objects to be pinned in memory per cache entry based on the expected frequency of use.
- In the past, administrators have been concerned about the impact of DynaCache on memory usage, and its effect on the performance of their systems. A new feature provides a mechanism to allow the administrator to tune the amount of metadata that is kept in memory for more efficient cache access and general book-keeping in order to reduce perceived outages during background DynaCache activity.
- The programmer or solution architect can specify advanced caching criteria in the application's cache policy.

- You can exclude a child fragment from full page caching and that child fragment is not to be cached as a separate fragment.
- You can selectively cache cookies along with the response. The existing 'store-cookies' property saves all the cookies.
- You can define a cache policy based on the range of values for a given parameter or attribute.
- You can choose to not use the cache under certain conditions (`skip-cache`), for example, to support the retrieval of preview content.
- Disk cache size is configurable, to limit the offload to disk. This has meant implementing changes to the disk eviction algorithm so that it kicks in at some threshold value before the disk cache fills up and keeps space available.
- Administrators are allowed to specify an upper bound on the size of a single entry that is stored in the cache. Larger sized objects will still be managed for invalidation propagation.
- Contention for cache was reduced by re-evaluating synchronization of resources in cache.
- Deletion time for entries on disk cache was reduced by reorganizing disk layout to separate metadata (dependencies) that group objects together for invalidation from the actual serialized data.
- Disk scan times for expired entries on disk were reduced.

The following sections describe the latest developments and changes made to DynaCache. Be aware, however, that the majority of these new features have been backported to WebSphere Application Server v5.0.2.18, v5.1.1.13, and v6.0.2.17 and above.

## 6.1.2 Cache policy enhancements

A number of caching policy enhancements have been made, including the following:

- Do-not-cache property
- Skip-cache-attribute
- Value/not-value ranges
- Store-cookies property
- Consume-subfragments exclude
- Portlet caching support
- Disk Cache Enhancements

Archived

## Property do-not-cache

Use this property when you want DynaCache to totally ignore the fragment. This means that DynaCache will neither cache nor consume the fragment, meaning no part of that fragment will come from the cache and the fragment must be processed by the application server runtime.

*Example 6-1 Using do-not-cache to prevent the caching of a child JSP*

```
<cache-entry>
    <class>servlet</class>
    <name>/AChildOfParentMainPage.jsp</name>
    <property name="do-not-cache">true</property>
    <cache-id>
        <timeout>0</timeout>
    </cache-id>
</cache-entry>

<cache-entry>
    <class>servlet</class>
    <name>/ParentMainPage.jsp</name>
    <property name="consume-subfragments">true</property>
    <cache-id>
        <timeout>0</timeout>
    </cache-id>
</cache-entry>
```

Even though ParentMainPag.jsp is consuming subfragments and includes AChildOfParentMainPage, AChildOfParentMainPage.jsp will not be cached or consumed.

Do-not-cache can also be extended for use in edge side includes and therefore cached on the edge of your network.

*Example 6-2 do-not-cache and edge-cacheable properties*

```
<cache-entry>
    <class>servlet</class>
    <name>/DoNotCache.jsp</name>
    <property name="do-not-cache">true</property>
    <property name="edge-cacheable">true</property>
    <cache-id>
        <timeout>0</timeout>
    </cache-id>
</cache-entry>
```



Archived

Notice the presence of the edge-cacheable property in Example 6-2. This property and the “do-not-cache” property must be present to define a fragment that is not cacheable on the edge.

### *When to use do-not-cache*

A fragment must be self-executing to be cacheable; however, not every self-executing fragment is cacheable. Instead of rendering this entire page uncacheable, you can elect to cache the full page and leave some fragments uncached.

There are very few fragments that are not cacheable. Before designating something as not cacheable, determine the value of caching the fragment versus the possible changes required to make the fragment cacheable. We recommend that you use this technique infrequently.

Before you decide a fragment is uncacheable, consult the different invalidation techniques available for cache content.

### *do-not-cache anti-pattern*

We have seen cases where the do-not-cache property has been incorrectly used to compensate for poor page design. For example, don’t use do-not-cache to bypass rarely used page fragments like personalized e-mail address details. If ninety-five percent of your clients don’t use that e-mail fragment then it should really have been placed on a separate page and a link inserted in its place.

A poor design forces everyone to have to absorb the performance costs of generating that e-mail detail for the sake of the few who actually will use it. If the site is a high volume Web site with many hundreds of thousands of hits per hour, we will hit the database for every singe page request in order to provide that e-mail address, even though the vast majority of the users don’t want it. Attention to small details like that can make big performance differences.

## Skip-cache-attribute

This defines an attribute that, when present in a request, specifies that the response cannot be retrieved from or stored in the specific cache instance. The following three examples illustrate the use of this attribute.

Example 6-3 applies to the base cache; Example 6-4 applies to cacheinstance1. To invoke the skip-cache functionality, simply set the request attribute previewRequest to true as shown in Example 6-5. SkipCache.jsp is then not retrieved from cache regardless of its cache policy.

*Example 6-3 cachespec.xml (Base Cache)*

```
<cache>
    <skip-cache-attribute>previewRequest</skip-cache-attribute>
   .
   .
   .
</cache>
```

*Example 6-4 cachespec.xml (cacheinstance1)*

```
<cache-instance name="cacheinstance1">
    <skip-cache-attribute>previewRequest</skip-cache-attribute>
   .
   .
   .
</cache-instance>
```

*Example 6-5 Invoking skip-cache from a filter*

```
public void doFilter (ServletRequest request, ServletResponse response,
FilterChain chain)    {
    try       {
        String parm = request.getParameter("previewRequest");
        if (parm == null)
            request.setAttribute("previewRequest",null);
        else if (parm.equals("true"))
            request.setAttribute("previewRequest","true");
        else if (parm.equals("false"))
            request.setAttribute("previewRequest","false");

        chain.doFilter(request, response);
    }
    catch (Throwable t) {}}
```

## Value/not-value ranges

Values and Not-values ranges are used to define ranges of integer values that specify whether or not a fragment is cached. They are specified using the `<range` tag and specifying `low` and `high` values.

*Example 6-6 Specifying value range*

```
<value>
        <range low="1" high="100" />
</value>
```



Archived

*Example 6-7 Case 1: Specifying value range*

```
<component id="parm1" type="parameter">
    <required>true</required>
    <value>
        <range low="10" high="20" />
    </value>
</component>
```

*Example 6-8 Case 1 results: Value range*

```
Request = ../jspname.jsp?parm1=10Result: Cached
Request = ../jspname.jsp?parm1=-1Result: Not cached
```

*Example 6-9 Case 2: Specifying not-value range*

```
<component id="parm1" type="parameter">
    <required>true</required>
    <not-value>
        <range low="-10" high="25" />
    </not-value>
</component>
```

*Example 6-10 Case 2 results: Not-value range*

```
Request = ../jspname.jsp?parm1=5Result: Not cached
Request = ../jspname.jsp?parm1=85Result Cached
```

### Store-cookies property

Specify any number of cookies to save in the cache object. The value defaults to false, saving no cookies. The following examples are from cachespec.

*Example 6-11 Store all cookies except cookie1 and cookie2*

```
<property name="store-cookies">true
    <exclude>cookie1</exclude>
    <exclude>cookie2</exclude>
</property>
```

*Example 6-12 Store only cookie1*

```
<property name="store-cookies">false
    <exclude>cookie1</exclude>
</property>
```

## Consume-subfragments exclude

Consume-subfragments exclude allows you to mark an included fragment to not be consumed by its parent.

The cache entry shown in Example 6-13 does not consume the subfragment. It caches only if it has its own entry in the cache policy.

*Example 6-13 Consume all subfragments except child.jsp*

```
<property name="consume-subfragments">true
    <exclude>/child.jsp</exclude>
</property>
```

In Example 6-14 the cache entry also does not consume child.jsp, but caches it since it has a cache policy.

*Example 6-14*

```
<property name="consume-subfragments">true</property>
…
<cache-entry>
    <class>servlet</class>
    <name>/child.jsp</name>
    <property name=“do-not-consume">true</property>
    <cache-id>
        <timeout>0</timeout>
    </cache-id>
</cache-entry>
```

## Portlet caching

Portlet caching is essentially the same as servlet caching. There are new request components available and some servlet request components are unavailable.

Note that there are interactions between enabling portlet and servlet caching such that if portlet caching gets enabled, servlet caching does, too.

Archived

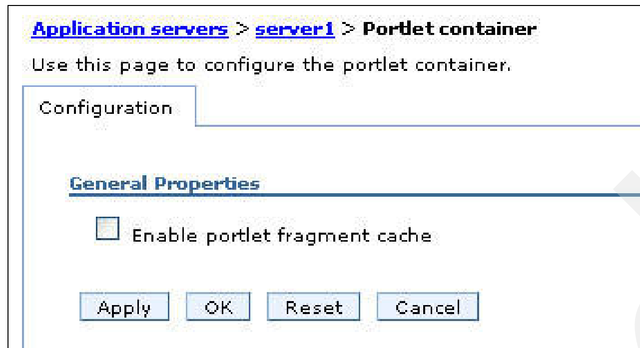



*Figure 6-1 Enabling portlet caching*

If servlet caching is disabled, so is portlet caching.

*Example 6-15 Portlet caching in cachespec.xml*

```
<cache-entry>
  <class>portlet</class>
  <name>WorldClock</name>
  <cache-id>
     <component id="*" type="parameter">
        <required>false</required>
     </component>
     <component id="currentTimeZone" type="portletSession">
        <required>false</required>
     </component>
     <timeout>180</timeout>
  </cache-id>
</cache-entry>
```

Table 6-1 show a list of the available new portlet components (all servlet components are also valid for portlets unless specified).

*Table 6-1 Portlet components for the cachespec.xml file*

| component | description |
|---|---|
| portletSession | Only applicable for portlets. Retrieves the portlet scoped (scoped by portletwindow) values from the HTTPSession. |
| timeout | The value in portlet.xml overrides the one in cachespec.xml. |
| portletWindow | New; uses the portlet window identifier as part of the cacheId. |

Archived

| component | description |
|---|---|
| portletMode, windowState | New; represents the portlet mode or window state of the portlet. |
| sessionId | New; applicable to portlets and servlet. Uses the session ID as cache property. |
| Not applicable | Cannot be used for portlets: cookie, pathInfo, servletpath. |

## 6.2 Edge Side Include (ESI) caching

Edge Side Include (ESI) is a simple markup language used to define Web page components for dynamic assembly and delivery of Web applications at the edge of the Internet.

For details of how to configure ESI caching, refer to the “Configuring Edge Side Include caching” section in the WebSphere Application Server Information Center.

The ESI processor's cache can be monitored through the cache monitor application. In order for the ESI processor's cache to be visible in the cache monitor, the DynaCacheEsi application `esiInvalidationMonitor` property must be set to `true` in the plugin-cfg.xml file, as shown in Example 6-16.

*Example 6-16 Monitoring Edge Side Include caches*

```
<?xml version-"1.0"?>
<Config>
<Property Name="esiEnable" Value="true"/>
<Property Name="esiMaxCacheSize" Value="1024"/>
<Property Name="esiInvalidationMonitor" Value="true"/>
```

## 6.3 Priming the cache

Some business Web sites, such as those of stock exchanges and event ticketing companies, are regularly subject to extreme load pressures at a particular time of day. For example, at a stock exchange, loads can peak at the start of the business day when trading commences. Other sites may simply change their product catalog or update prices on a daily basis. Either way, the cache will need to be rebuilt in order to accommodate the new changes.



Archived

We recommend that you run automatic scripts to ensure that your site's most frequently accessed pages and data are loaded into cache well before they are needed by your clients. Then, when a large number of requests come flooding in, your site is ready to cope.

This caching pre-loading activity is often referred to as *warming up* or *priming* the cache. Again, the objective of cache warming is to be as responsive as possible before the commencement of very heavy processing periods by having the application server primed to cope with these loads.

Priming the cache can be very effective in clustered environments, where the distributed replication service may take quite a while to update all the cluster members, especially if the number of cache entries to distribute is large.

## 6.4 When you must not cache

When creating your high-level caching strategy, you first need to determine what pages in your store should be cached. Pages that are good candidates for caching are pages that are accessed frequently, but are also stable for a period of time, and contain content that can be reused by a variety of users. You should beware of caching security information and cookies.

### *Personalized fragments*

Any servlet and JSP file content that is private, requires authentication, or uses SSL should not be cached externally. The authentication required for those servlet or JSP file fragments cannot be performed on the external cache.

### *Cookies and attributes*

By default, dynamic caching caches the cookies (when caching by servlet class) and all request attributes (servlet and JSP pages) along with the cache entries. However, WebSphere Commerce cookies and request attributes contain user-specific information that should not be cached. As a result, the following property names and values are mandatory when caching full pages:

```
<property name="save-attributes">false</property>
<property name="store-cookies">false</property>
```

## 6.5 Multiple caching pools and cache instances

DynaCache supports the configuration of multiple separate caching pools to store, retrieve, and share data objects within the DynaCache. Each pool is called a *cache instance*.

### 6.5.1 Cache instance

Properties such as cache size, priority, and disk offload are configured independently for each cache instance. Objects that are stored in a particular cache instance are not affected by other cache instances.

As a system designer, you now have more granularity of control in terms of what happens to your cached objects. For example, one cache pool may not allow disk offload while the other does – allowing you to pin certain objects in a high speed in-memory cache and place other objects into a second cache, where there is greater potential of being offloaded to disk.

You may even separate objects by size, keeping smaller objects in one cache and larger objects in another. This helps prevent situations where a small object squeezes out a larger valuable object from the memory cache and onto the disk.

Objects that are stored in a particular cache instance are available to applications on other servers by accessing a cache instance of the same name. The two servers must be within the same replication domain to share data.

### 6.5.2 Cache instance definition

The root element of the cachespec.xml file is `cache` and contains `cache-instance` and `cache-entry` elements. The cache-entry elements can also be placed inside cache-instance elements to make that cache entry part of a cache instance that is different from the default.

Each cache-instance element must contain at least one cache-entry element. A cache entry that is matched within a cache-instance element is cached in the servlet cache instance that is specified by the name attribute. If identical cache-entry elements exist across cache-instance elements, the first cache-entry element that is matched is used.

*Example 6-17 Defining multiple cache instances*

```
<cache>
   <cache-instance name="cache_instance_1">
      <cache-entry>
      ...
      </cache-entry>
   </cache-instance>

   <cache-instance name="cache_instance_2">
      <cache-entry>
      ...
      </cache-entry>
```



Archived

```
    ...
    </cache-instance>

</cache>
```

## 6.6 DynaCache tuning

Tuning DynaCache is much like tuning any other performance-enhancing component: it is an iterative process. It should begin at application design, with guidance from the application architect on what should and should not be cached based on input from the requirement stage and knowledge of the application scenarios.

This process is further refined through the development, validation, and production phase of the project. It is invaluable during validation and pre-production phases of development to monitor performance changes so as to understand and rectify the impact of cache policies and tuning on system behavior.

You need to run some projected workload without caching to determine values such as the cost of generating the object, although you may choose to rely on the intuition and experience of the application architect.

Guidelines for determining the effectiveness of caching should take into account the following:

- The cost of generating a response should be greater than the maximum cache access time, where the maximum cache access time should factor in overhead for disk access, distribution policy, and so on.
- The lower the validity of the object and response, the more likely that it will not be reused. This can result in larger latencies due to cache misses and cleanup overhead, than simply not caching the object.
- The objects with more popularity and business value should be assigned a higher relative priority.
- The higher the degree of connectivity of an object, the more costly it is to invalidate and evict the object from the cache. Take this into account when determining where to cache the object in terms of keeping it in the memory cache, disk cache, or distributing the object across the cluster.
- The DynaCache specification provides attributes that can be used to declare properties of the cached object such as timeout (in seconds), priority (LOWEST PRIORITY = 0, DEFAULT PRIOIRTY = 3, HIGHEST PRIORITY = 16), and inactivity, to affect the treatment of these cached objects.

Archived

# 6.7 Memory caching

DynaCache accesses and retrieves objects primarily from the memory cache. This cache keeps references to the cached objects and can be configured with limits on the number of entries that are cached in memory. After the limit of entries that is specified for the memory cache is reached, adding additional entries in the cache will require that entries be evicted out of memory, based on how recently the evicted entry was last accessed, and the priority of the object that is inserted into the cache.

## 6.7.1 Cache sizing formula

Choosing the size of the memory cache, in terms of the number of entries, should be done based on how much memory is available for caching.

The average memory, in bytes, that is used by the system to reference a cached object with its dependency IDs can be computed as:

```
size = o + c + ( k* ( dp + tm + 128) )
      o = the average size of the object
      c = the average size of the cache ID
      k = is 4 for 32-bit platforms and 8 for 64-bit platforms
      dp = the number of templates
      tm = dependency IDs that are associated with this object
```

The number of entries that is specified should be large enough to hold the cache entries that are associated with the popular or more frequently used categories. The memory cache and therefore memory dedicated for the cache should be large enough not only to cache content belonging to categories that have higher business value, but also enough additional entries to form a working set in order to minimize the amount of thrashing due to Least Recent Used (LRU) eviction.

The Java Virtual Machine (JVM) heap settings should also be set. The recommended setting for the JVM heap is to have 40% of free heap after caching. This tuning involves either increasing the size of JVM or reducing the size of the in-memory cache (or cache objects that require less memory). There are lots of trade-offs here, such as higher JVM causing longer garbage collection. It is a fine balance that can only be determined with proper testing.

DynaCache cleans expired entries from the memory cache in the background. The daemon responsible for this cleanup will wake up every five seconds. This is sufficient for most deployments. On the other hand, for deployments that do infrequent invalidation and possibly invalidate entries only once a day, this can be set higher. If the deployment has a lot of automated or trigger-driven invalidation, the cleanup interval should be set lower.

## 6.7.2 Disk caching

DynaCache has the option to cache content in disk when the content is evicted from the memory cache. It is highly recommended that the offload directory be located on a high speed, separate disk or partition that is dedicated for caching only.

A dedicated disk drive enables better response times for the disk cache by reduced contention for disk space with application data and code on the file system where WebSphere Application Server is installed. We recommend a minimum partition size of twice the expected volume of cached content.

### Controlling offload with the persist-to-disk property

The storage and access of objects from disk involves serialization and de-serialization of objects. This feature comes at a higher cost, and should be taken into consideration when deciding what content should be persisted to disk.

It is possible for you to selectively cache content to the disk through cache policies that are defined in the cachespec.xml file, in particular the persist-to-disk property. The persist-to-disk property has two values: true or false. The default value is true. When this property is set to false, the cache entry is not written to the disk when overflow or a server shutdown occurs.

### Disk cache cleanup and tuning

Objects that are in the disk cache are cleaned up when they are explicitly invalidated through either programmatic or policy-based invalidations, or when the objects expire. When the disk cache is cleaned up the tables that host the dependency ID to cache ID mappings and template ID to cache ID mappings are updated and the disk space occupied by the cache entries is returned to the internal storage manager. The available space on the file system does not increase after the objects are deleted from the cache because the space is claimed back by the internal storage manager and reused by other objects that are cached to the disk.

The cleanup is done in the background as a low priority thread to reduce contention for the disk from active request and response threads. The time to perform this cleanup, as reported in the logs, tracks the duration of the scan. With the low priority of the scan, it can take several minutes.

Archived

# 6.8 Setting custom system properties

> **Important:** You should use the administrative console to set the following three custom properties in V6.1. Setting them manually is not recommended.

### Disk cache cleanup

You can activate the disk cache cleanup once a day at a specified time by using the com.ibm.ws.cache.CacheConfig.htodCleanupHour system property. In the Administrative Console:

1. Select **Application servers** → <your server> → **Process Definition** → **Java Virtual Machine** → **custom properties**
2. Click the **New** button and declare the system property as the key and its value in the value field, which defaults to 0 (= 12:00 midnight). Or you can specify the cleanup to run at a specific frequency by setting the `com.ibm.ws.cacheCacheConfig.htodCleanupFrequency` system property. Its value is expressed in minutes.

The disk cache cleanup occurs in two phases: *scan* and *delete*. In the scan phase, the algorithm identifies objects that have expired on disk. Since the cleanup algorithm is only looking for expired entries, cached objects without an expiration value (an expiration value of 0) will always remain on disk until explicitly invalidated.

The policy of never expiring objects should be reconsidered if disk space is an issue in the deployment. The delete phase returns disk space to the internal storage manager and ensures that all references to the object are correctly purged. Most large deployments that have a large amount of content on the disk typically choose to specify that cleanup occurs at a frequency that ranges from 30 minutes to a couple of hours, depending on the average expiration time of content in the cache.

You can optimize the disk cache cleanup for disk I/O by buffering the metadata that is associated with cached objects in memory. These auxiliary buffers can hold the dependency and template information for the objects so that the object deletion time is decreased.

### com.ibm.ws.cache.CacheConfig.htodDelayOffload

By setting the com.ibm.ws.cache.CacheConfig.htodDelayOffload system property to true you will enable this optimization.

Archived

### com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntiresLimit

You can tune the memory that is utilized by this optimization by setting the com.ibm.ws.cache.CacheConfig.htodDelayOffloadEntiresLimit system property to a value that specifies the maximum number of cache IDs that any dependency ID can map to in the auxiliary buffer.

Any dependency that maps to more cache IDs than those specified using the htodDelayOffloadEntriesLimit are not buffered and are written to disk. Administrators managing large deployments prefer to set this to a value that approximates the total number of entries in the entire cache for optimal performance.

## 6.9 Monitoring DynaCache

If the Performance Monitoring Infrastructure (PMI) service is enabled, then there are two ways to collect DynaCache PMI statistics without a server restart:

- Perform all the steps listed in the following discussion on the Runtime tab instead of the Configuration tab.
- Write a client program that will issue MBean JMX API calls to enable the DynaCache PMI module and collect DynaCache PMI statistics.

The PMI service is enabled by default on server startup.

Starting from WebSphere Application Server v5.0.2.18, v5.1.1.13, v6.0.2.17 and v6.1.0.0, Dynacache has introduced a number of statistics exposed via the DynaCache MBean for monitoring and tuning DynaCache.

About 50 statistics, such as ExplicitInvalidationsFromDisk and ObjectsReadFromDisk400K, have been exposed to provide a comprehensive view of the cache. For more details refer to the technote for APAR PK13460 found at `http://www-1.ibm.com/support/docview.wss?uid=swg27007969.`

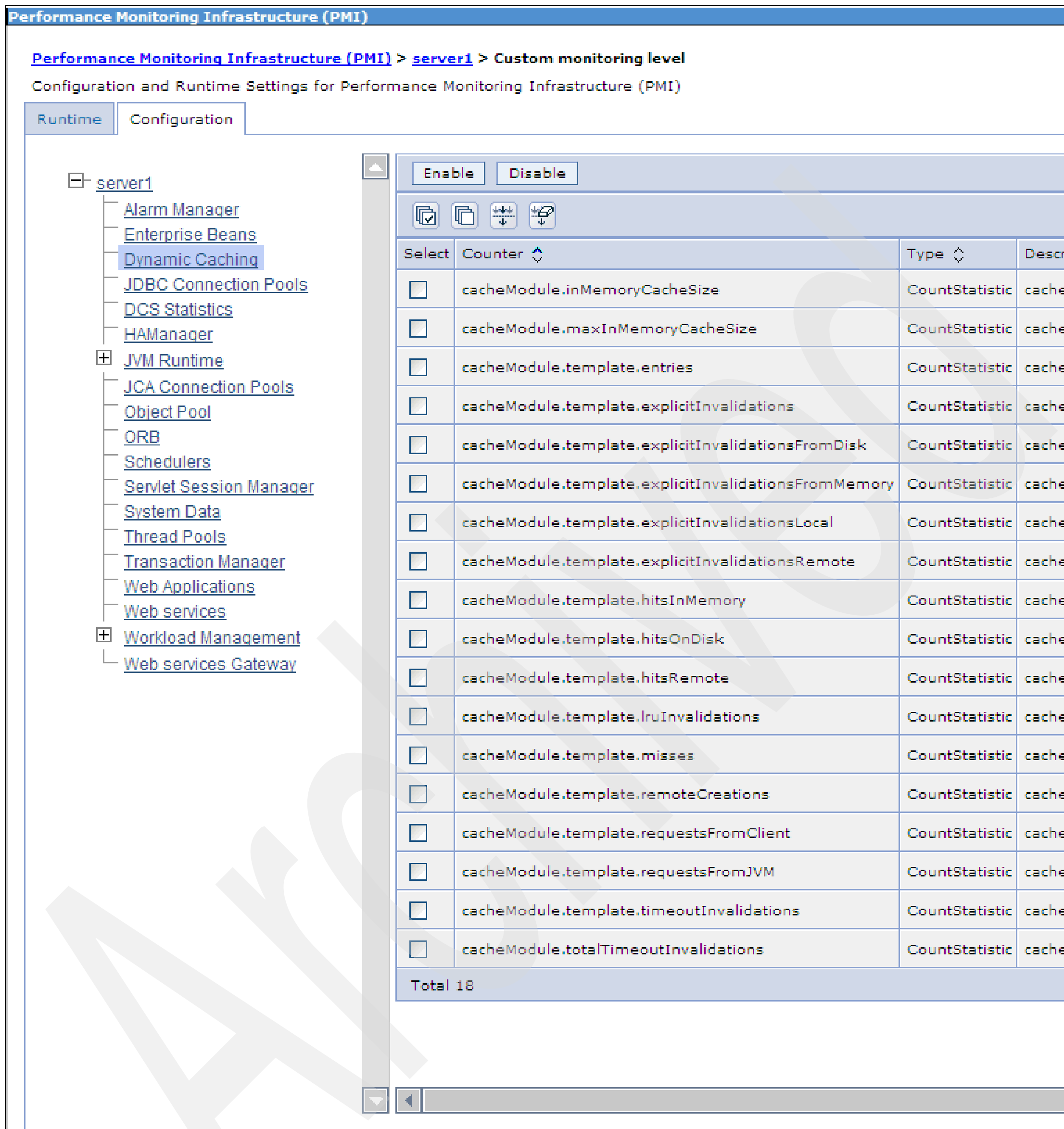



*Figure 6-2 Selectable custom monitoring performance metric modules in DynaCache*

You can select the “Custom monitoring level” and select and enable all of modules listed under DynaCache. This will require server restart.

Monitoring will not impact performance significantly. You can collect metrics from a few servers as a sample in order to establish a pattern.

Any data collected can be viewed with WebSphere's internal Tivoli® Performance Viewer (TPV), or you can collect the data in TPV logs.

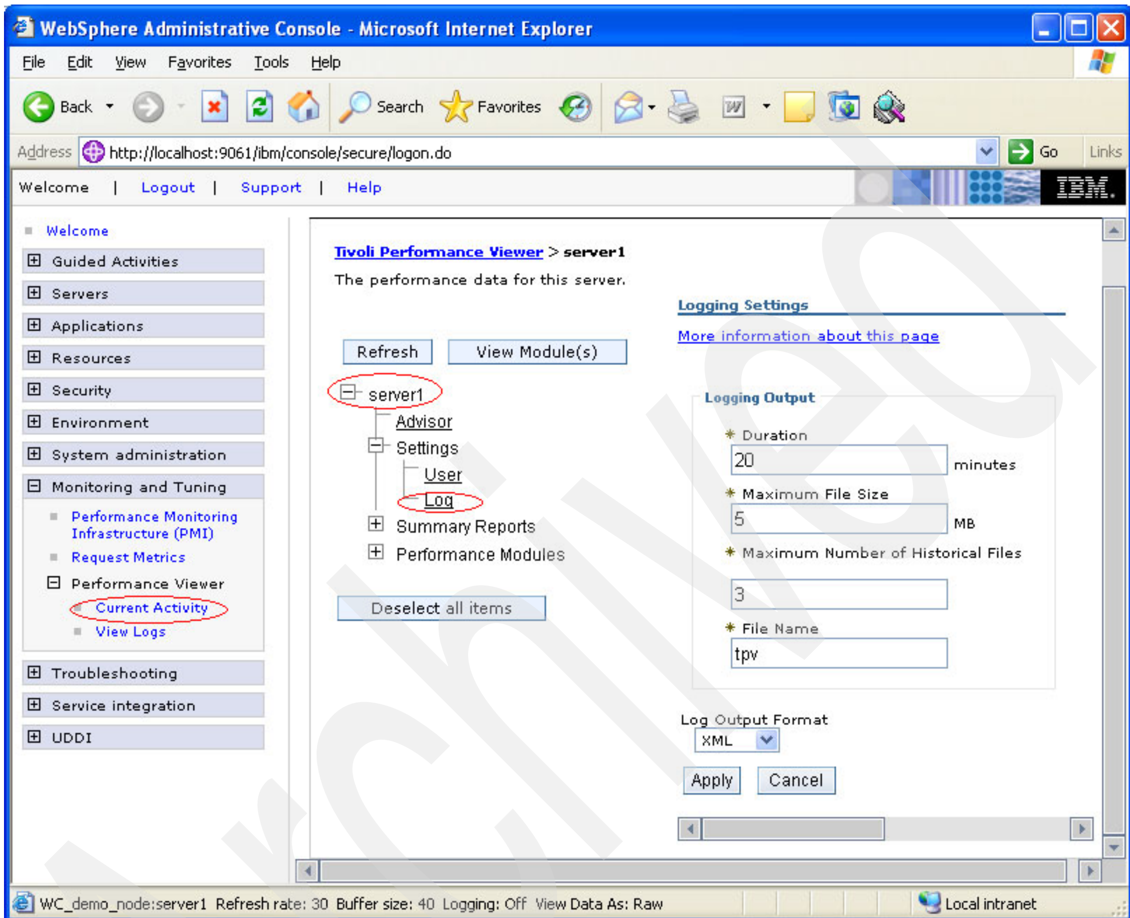



*Figure 6-3 Monitoring and Tuning PMI main menu*

To creating TPV logs:

1. Select **Monitoring and Tuning** → **Performance Viewer** → **Current Activity** → ***server_name*** → **Settings** → **Log** in the console navigation tree.
2. To see the Log link on the Tivoli Performance Viewer page, expand the Settings node of the TPV navigation tree on the left side of the page. After selecting **Log**, the TPV log settings are displayed on the right side of the page.
3. Select **Start Logging** when viewing summary reports or performance modules.

4. When finished, select **Stop Logging**.

Note that logging stops when any one of the following events occurs:

- The logging duration expires
- Stop Logging is clicked
- The file size and number limits are reached

The settings are adjustable in the Log Settings panel, described previously in step one. By default, the log files are stored in the profile_root/logs/tpv directory on the node on which the server is running. To conserve space, TPV automatically compresses the log file when it has finished writing to it. There is a single log file in each .zip file and the file will have the same name as the .zip file.

View the logs locally or remotely using a standalone TPV.

### 6.9.1 DeveloperWorks tooling for monitoring DynaCache

The DynaCache team has produced a cache statistics script and parser that is to be released on DeveloperWorks. The script is written in Jython and collects the DynaCache statistics collected by the DynaCache mbean.

The Java parser (DynaCacheStatisticsParser.class) imports the results into a Microsoft® Excel® file. The DynaCache team is currently working on a visualizer to graph the mbean counter values over time. The tool is extremely useful to see what the cache is doing and identify potential performance improvements.

Archived

# 6.10 Reference section

This section reproduces information from the WebSphere Application Server infocenter, together with some usage notes we have added for your assistance. The information will not be updated to reflect changes to WebSphere Application Server and the infocenter. You should always refer to the infocenter if there is any doubt about the currency of the information reproduced in this section.

## 6.10.1 Class element

*Table 6-2 DynaCache class types supported*

| Value[a] | Description |
|---|---|
| servlet | Refers to both servlets and Java Server Pages (JSP) files that are deployed in the WebSphere Application Server servlet engine |
| webservice | Extends the servlet with special component types for Web services requests |
| JAXRPCClient | Used to define a cache entry for the Web services client cache |
| command | Refers to classes using the WebSphere Application Server command programming model |

a. DynaCache is not recommended for caching static pages, so Static has been omitted from the DynaCache class types. See 7.2, “Clustering FAQs” on page 175 for a discussion about caching static pages.

Example 6-18 illustrates the class element.

*Example 6-18 Class element examples*

```
<class>command</class>
<class>servlet</class>
<class>webservice</class>
<class>JAXRPCClient</class>
```

## 6.10.2 Name element

<name>name</name>

The following guidelines help you specify a cacheable object in DynaCache.



Archived

### <name>Command</name>

For the required element `<name>`, you must include the fully qualified Java package name, including a trailing .class, of the configured object, for example:

```
com.ibm.commerce.wcs.DoSomeCommand.class
```

### <name>Servlet</name>

If you placed the cachespec.xml file in the WebSphere Application Server properties directory, then this required element must include the full URI of the JSP file or servlet you want cached.

However, if you placed the cachespec.xml file in the Web application, which is where we recommend that you put it, then this required element can be relative to the specific Web application context root.

Remember that placing the file in the properties directory makes it global to all applications.

### <name>webservices</name>

For Web services entries, you must include the URI of the SOAP router that is associated with the Web service that you want to cache.

For the Web services client cache, the name is the target end point of the cacheable Web service or the URI of the SOAP router that is associated with the cacheable Web service. You can use the SOAP address location in the Web Services Description Language (WSDL) file to define the name for the Web services client cache.

## 6.10.3 Sharing policy

When working within a cluster with a distributed cache, these values determine the sharing characteristics of entries that are created from this object.

*Example 6-19 Sharing policy*

```
<sharing-policy>
   not-shared | shared-push | shared-pull | shared-push-pull
</sharing-policy>
```

If this element is not present, a not-shared value is assumed. In single server environments, not-shared is the only valid value. When enabling a replication, the default value is not-shared. This property does not affect distribution to Edge Side Include processors through the Edge fragment caching property. See Table 4-1 on page 117 for more information.

Example 6-20 shows the sharing policy list.

*Example 6-20 Sharing policy possible values*

```
<sharing-policy>not-shared</sharing-policy>
<sharing-policy>shared-push</sharing-policy>
<sharing-policy>shared-pull</sharing-policy>
<sharing-policy>shared-push-pull</sharing-policy>
```

## 6.10.4 Property

You can set optional properties on a cacheable object, such as a description of the configured servlet. The following syntax is used:

```
property name="key">value</property>
```

`Key` is the name of the property for this cache entry element, and `value` is the corresponding value.

The class determines valid properties of the cache entry. Table 6-3 identifies the properties that are currently defined.

*Table 6-3 Cachespec.xml property definitions*

| Property | Valid classes | Value |
|---|---|---|
| ApplicationName | All | Overrides the J2EEName application ID so that multiple applications can share a common cache ID namespace. |
| EdgeCacheable | Servlet | True or false. The default is false. If the property is true, then the given servlet or JSP file is externally requested from an Edge Side Include processor. Whether or not the servlet or JSP file is cacheable depends on the rest of the cache specification. |
| ExternalCache | Servlet | Specifies the external cache name. The external cache name needs to match the external cache group name. |

Archived

| Property | Valid classes | Value |
|---|---|---|
| consume-subfragments | Servlet,<br>Web service | True or false. The default is false. When a servlet is cached, only the content of that servlet is stored, and includes placeholders for any other fragments to which it includes or forwards. Consume-subfragments (CSF) tells the cache not to stop saving content when it includes a child servlet. The parent entry, the one marked CSF, includes all the content from all fragments in its cache entry, resulting in one big cache entry that has no includes or forwards, but the content from the whole tree of entries. Consume-subfragments can save a significant amount of application server processing, but is typically only useful when the external HTTPrequest contains all the information needed to determine the entire tree of included fragments.Use the <exclude> element to tell the cache to stop consuming for the excluded fragment and instead, create a placeholder for the include or forward. For example, exclude A.jsp from the consume-subfragment, as follows:<br><property name="consume-sbufragments">true<br><exclude>/A.jsp<exclude><br></property> |
| do-not-consume | Servlet,<br>Web service | True or false. The default is false. When a fragment parent has the consume-subfragment property set to true the child fragment content is saved in the cache entry of the parent. Do-not-consume (DNC) tells the cache to stop saving the content for this fragment in the parent cache-entry and create a placeholder instead for the include or forward. |
| do-not-cache | Servlet and Portlet | Defines a fragment that is neither cached nor consumed by its parent.<br><cache-entry><br>...<br><property name="do-not-cache">true</property><br>or<br><cache-id><br><property name="do-not-cache">true</property><br></cache-id><br></cache-entry> |
| alternate_url | Servlet | Specifies the alternate URL that is used to invoke the servlet or JSP file. The property is valid only if the EdgeCacheable property also is set for the cache entry. |
| persist-to-disk | All | True or false. The default is true. When this property is set to false, the cache entry is not written to the disk when overflow or server stopping occurs. |

Archived

| Property | Valid classes | Value |
|---|---|---|
| save-attributes | Servlet | True or false. The default is true. When this property is set to false, the request attributes are not saved with the cache entry.<br>Use the <exclude> element to specify the request attributes that do not apply to the save-attributes property. For example, to save only the attr1 attribute with the cache entry:<br><property name="save-attributes">false<br><exclude>attr1</exclude><br></property><br>To save all attributes except the attr1 attribute in the cache entry, set the property to true in the preceding sample. If you do not use the <exclude> element, either all or no request attributes are saved with the cache entry. |
| delay-invalidations | Command | True or false. When this property is set to true, the commands that are invalidating cached objects based on the invalidation rules in this cache entry invalidate the cache entries after running. By default, the invalidation occurs before the command runs. |
| store-cookies | Servlet | Takes one or more cookie name as its argument which is saved along with the cache object and restored by the servlet cache in the response with a set-cookie header.<br>Save all cookies except cookie1 as part of the cache-entry as follows:<br><property name="store-cookies">true<br><exclude>cookie</exclude><br></property><br>Save only cookie1 as part of the cache-entry, as follows:<br><property name="store-cookies">false<br><exclude><cookie1</exclude><br></property> |
| ignore-get-post | Servlet | True or false. The default is false. When the property is set to true the request type is not appended to the cache-id for GET and POST requests unless the requestType component sub-element is defined. By default the request type is automatically appended to the cache-id for GET and POST requests. |

### 6.10.5 Cache entry IDs

To cache an object, the application server must know how to generate a unique ID for different invocations of that object. These IDs are built either from user-written custom Java code or from rules that are defined in the cache policy of each cache entry. (Note that custom Java code needs to be placed into a shared library.)

Each cache entry can have multiple cache ID rules that run in order until either a rule returns a non-empty cache ID, or no more rules are left to run. If none of the cache ID generation rules produce a valid cache ID, the object is not cached.

Each cache-id element defines a rule for caching an object and is composed of the following sub-elements:

- Component
- Timeout
- Inactivity
- Priority
- Property
- Idgenerator
- Metadatagenerator

Example 6-21 on page 158 illustrates a cache-id element.

*Example 6-21 cache-id definition grammar*

```
<cache-id>
   component* | timeout? | inactivity? | priority? | property* | idgenerator? |
metadatagenerator?
</cache-id>
```

In this example, * = zero or more, | = or, and ? = value.

## Component sub-element

Use the component sub-element to generate a portion of the cache ID. The component sub-element consists of the following attributes and elements:

- Attributes
  - Id
  - Type
  - Ignore-value
- Elements
  - Index
  - Method
  - Field
  - Required
  - Value
  - Not-value

Use the ID attribute to identify the component.

Use the type attribute to identify the type of component. Table 6-4 lists the values for type.

*Table 6-4 Cachespec.xml component types*

| Type | Object | Meaning |
|---|---|---|
| method | Command | Calls the indicated method on the command or object. |
| field | Command | Retrieves the named field in the command or object. |
| parameter | Servlet | Retrieves the named parameter value from the request object. |
| parameter-list | Servlet | Retrieves a list of values for the named parameter. |
| session | Servlet | Retrieves the named value from the HTTP session. |
| cookie | Servlet | Retrieves the named cookie value. |
| attribute | Servlet | Retrieves the named request attribute. |
| header | Servlet and Web service | Retrieves the named request header. |
| pathInfo | Servlet | Retrieves the pathInfo element from the request. Dynacache pulls out the pathinfo using: `(String) request.getAttribute("javax.servlet.include.path_info");` |
| servletpath | Servlet | Retrieves the servlet path. Dynacache pulls out the servlet path using: `(String) request.getAttribute("javax.servlet.include.servlet_path");` |
| locale | Servlet | Retrieves the request locale. |
| requestType | Servlet | Retrieves the HTTP request method from the request. |
| tiles_attribute | Servlet | Retrieves the value of an attribute from a tile. |
| SOAPEnvelope | Web service and client | Retrieves the SOAPEnvelope element from a Web services request. An ID attribute of Hash uses a Hash of the SOAPEnvelope element, while Literal uses the SOAPEnvelope element as received. |
| SOAPAction | Web service | Retrieves the SOAPAction header, if available, for a Web services request. |
| serviceOperation | Web service | Retrieves the service operation for a Web services request. |
| serviceOperation Parameter | Web service | Retrieves the specified parameter from a Web services request. |

Archived

| Type | Object | Meaning |
|---|---|---|
| operation | Web services client cache | Indicates an operation type in the Web Services Description Language (WSDL) file. The id attribute is ignored and the value is the operation or method name. If the namespace of the operation is specified, format the value as namespaceOfOperation:nameOfOperation. |
| part | Web services client cache | Indicates an input message part in the WSDL file or a request parameter. Its ID attribute is the part or parameter name, and the value is the part or parameter value. |
| SOAPHeaderEntry | Web services client cache | Retrieves special information in the Simple Object Access Protocol (SOAP) header of the Web services request. The id attribute specifies the name of the entry. In addition, the entry of the SOAP header in the SOAP request must have the actor attribute, which contains com.ibm.websphere.cache. For example: `<soapenv:Header>` `<getQuote soapenv:actor="com.ibm.websphere.cache">IBM</getQuote>` `</soapenv:Header>` |
| sessionID | Servlet | Retrieves the HTTP session ID. |

### *Ignore-value*

Use the ignore-value attribute to specify whether or not to use the value that is returned by this component in cache ID formation. This attribute is optional, with a default value of false. If the value is true, only the ID of the component is used when creating a cache ID, or no output is used when creating a dependency or invalidation ID.

### *Index element*

Use the index element with the previous component type to add the value of the element at the specified index position in the collection or array to the ID that is being created.

*Example 6-22 Use of Index element*

```
<cache-entry>
   <class>servlet</class>
   <name>xxx.jsp</name>
   <cache-id>
      .
      .
         <component id="users" type="attribute">
            <required>true</required>
```

```
            <index>1</index>
         </component>
      .
      .
   </cache-id>
   <dependency-id>dep
      <component id="users" type="attribute" multipleIDs="true">
         <required>true</required>
      </component>
   </dependency-id>
</cache-entry>
```

The previous cache policy generates the following component to use in the cache ID: `users: b`. Use the <method> element to call a void method on a returned object.

### *Method: Calling Java methods*

Use the method element to call a method on a returned object. Method and field objects are infinitely nestable in any combination. The method must be public and is not valid for edge-cacheable components. For example:

```
<component id="getUser" type="method"><method>getUserInfo
<method>getName</method></method></component>
```

This method is equivalent to getUser().getUserInfo().getName().

Component types attribute, method, or field can return an object. When the object returned is a collection or array, the index ID is created with a comma separated list of the elements in the collection or array. For example, if the request attribute `users` returns an array [a, b] and the cache entry is defined like Example 6-23, then the cache ID will contain the string `users: a,b`. The dependency ID will be `dep: a,b`.

*Example 6-23 cachespec.xml where attribute users returns an array [a, b]*

```
<cache-entry>
   <class>servlet</class>
   <name>xxx.jsp</name>
   <cache-id>
      .
      .
      <component id="users" type="attribute">
         <required>true</required>
      </component>
      .
      .
```

```
   </cache-id>
   <dependency-id>dep
      <component id="users" type="attribute">
         <required>true</required>
      </component>
   </dependency-id>
</cache-entry>
```

Use the multipleIDs attribute with the component types to specify and generate multiple dependency IDs (or invalidation IDs), based on the items in the collection or array (see Example 6-24).

*Example 6-24 Using multipleID attribute in generating a dependency ID*

```
<cache-entry>
   <class>servlet</class>
   <name>xxx.jsp</name>
   <cache-id>
      ...
            <component id="users" type="attribute">
            <required>true</required>
         </component>
      ...
   </cache-id>
   <dependency-id>dep
      <component id="users" type="attribute" multipleIDs="true">
         <required>true</required>
      </component>
   </dependency-id>
</cache-entry>
```

Based on Example 6-24, the cache policy will generate the following dependency IDs:

```
dep:a,b
dep:a
dep:b
```

### Field element

Use the field element to access a field in a returned object. Method and field objects are infinitely nestable in any combination. The field must be public. This field is not valid for edge-cacheable components. For example:

```
<component id="getUser" type="method"><method>getUserInfo
<field>name</field></method></component>
```



Archived

This method is equivalent to the getUser().getUserInfo().name method.

### Required element

Use the required element to specify whether or not this component must return a non-null value for this cache ID to represent a valid cache. If set to true, this component must return a non-null value for this cache ID to represent a valid cache ID. If set to false, the default, a non-null value is used in the formation of the cache ID and a null value means that this component is not used at all in the ID formation. For example:

```
<required>true</required>
```

### Value element

Use the value element to specify values that *must* match to use this component in cache ID formation. For example:

*Example 6-25 Use of value*

```
<component id="getColor" type="method">
   <required>true</required>
   <value>blue</value>
   <value>red</value>
</component>
```

### Not-value

Use the not-value element to specify values that *must not* match in order to use the component in cache ID formation. This method is similar to value element, but instead prescribes the defined values from caching. You can use multiple not-value elements when more than one value that is not valid exists. This is shown in Example 6-26.

*Example 6-26 Use of not-value*

```
<component id="getColor" type="method">
   <required>true</required>
   <not-value>blue</not-value>
   <not-value>red</not-value>
</component>
```

The component sub-element has either a method and a field element, a value element, or a not-value element. The method and field elements apply to commands only. The following example illustrates the attributes of a component sub-element:

```
<component id="isValid" type="method" ignore-value="true"><component>
```

Archived

## Timeout

The timeout sub-element is used to specify an absolute time-to-live (TTL) value for the cache entry. For example,

```
<timeout>value</timeout>
```

`Value` is the amount of time, in seconds, to keep the cache entry. Cache entries that are in memory are kept indefinitely, as long as the entries remain in memory. Cache entries that are stored on disk are evicted if they are not accessed for 24 hours.

## Inactivity

The inactivity sub-element is used to specify a time-to-live (TTL) value for the cache entry based on the last time that the cache entry was accessed. It is a sub-element of the cache-id element.

```
<inactivity>value</inactivity>
```

Here `value` is the amount of time, in seconds, to keep the cache entry in the cache after the last cache hit.

## Priority

Use the priority sub-element to specify the priority of a cache entry in a cache. The priority weighting is used by the least recently used (LRU) algorithm of the cache to decide which entries to remove from the cache if the cache runs out of storage space. For example,

```
<priority>value</priority>
```

`Value` is a positive integer between 1 and 255, inclusive.

Example 6-27 keeps the cache entry in the cache for a minimum of 35 seconds and a maximum of 180 seconds. If the cache entry is accessed within each 35 second inactivity period, the inactivity period is extended for another 35 seconds. However, because the timeout element is also configured, the cache entry is always invalidated after 180 seconds. If the cache entry is not accessed within the 35 second period, the entry is removed from the cache.

*Example 6-27 inactivity and timeout usage - Sample 1*

```
<cache-id>
   <component id="timeout" type="parameter">
      <required>true</required>
   </component>
   <timeout>180</timeout>
   <inactivity>35</inactivity>
   <priority>1</priority>
```

Archived

```
</cache-id>
```

Example 6-28 keeps the cache entry in the cache for a minimum of 600 seconds. If the cache entry is accessed within each 600 second period, the inactivity period is extended for another 600 seconds. If the cache entry is not accessed within the 600 second period, the cache entry is removed from the cache.

*Example 6-28 Another inactivity and timeout - Sample 2*

```
<cache-id>
   <component id="timeout" type="parameter">
      <required>true</required>
   </component>
   <inactivity>600</inactivity>
   <priority>1</priority>
</cache-id>
```

In Example 6-29 the value for inactivity has no meaning because the timeout period is less than the inactivity period. The cache entry is always invalidated after 180 seconds, no matter how often the cache entry is accessed.

*Example 6-29 Timeout value < inactivity period*

```
<cache-id>
   <component id="timeout" type="parameter">
      <required>true</required>
   </component>
   <timeout>180</timeout>
   <inactivity>600</inactivity>
   <priority>1</priority>
</cache-id>
```

### Property sub-element

Use the property sub-element to specify generic properties for the cache entry. For example:

```
<property name="key">value</property>
```

`Key` is the name of the property to define, and `value` is the corresponding value.

For example:

```
<property name="description">The Snoop Servlet</property>
```

## Idgenerator and metadatagenerator sub-elements

Use the Idgenerator element to specify the class name that is loaded for the generation of the cache ID. The Idgenerator element must implement the com.ibm.websphere.servlet.cache.Idgenerator interface for a servlet or the com.ibm.websphere.webservices.Idgenerator interface for the Web services client cache. An example of the Idgenerator element follows:

```
<Idgenerator> class name </Idgenerator>
```

`Class name` is the fully qualified name of the class to use. Define this generator class in a shared library. Example 6-30 shows an example of a custom written command cache ID generator.

*Example 6-30 Cache ID generator for a command object*

```
package com.ibm.ws.cache.command;
import com.ibm.websphere.command.*;
import java.util.*;
public class QuoteIdgenerator implements CommandIdgenerator {
   public String getId(CacheableCommand command, ArrayList groupIds) {
     QuoteCommand cs = (QuoteCommand)command;
     // add dependency ids for quotecommand the ticker for this command
     groupIds.add("QuoteCommandIDGen");
     groupIds.add("ticker:"+cs.getTicker());
     return "QuoteCommmandIdGen Ticker:"+cs.getTicker();
   }
}
```

Use the metadatagenerator element inside the cache-id element to specify the class name loaded for the metadata generation. The MetaDataGenerator class must implement the com.ibm.websphere.servlet.cache.MetaDataGenerator interface for a servlet or the com.ibm.websphere.cache.webservices.MetaDataGenerator interface for Web services client cache. The MetaDataGenerator class defines properties like timeout, inactivity, external caching properties or dependencies. An example of the metadatagenerator element is:

```
<metadatagenerator> class name </metadatagenerator>
```

In this example, `class name` is the fully qualified name of the class to use. Define this generator class in a shared library.

*Example 6-31 Sample metadata generator source code*

```
package com.ibm.ws.cache.servlet;
import javax.servlet.http.*;
import com.ibm.websphere.servlet.cache.*;
```

Archived

```
public class MyMetaDataGenerator implements MetaDataGenerator {
   private int timeout=0;
   private int priority=0;
   public void setMetaData(ServletCacheRequest req, HttpServletResponse 
resp) {
      System.out.println("**** setMetaData***");
       FragmentInfo fragmentInfo =  (FragmentInfo) 
req.getFragmentInfo();
       String tout = req.getParameter("metaDataTimeout");
       if (tout != null) {
            timeout = new Integer(tout).intValue();
            if (timeout != 0) {
               fragmentInfo.setTimeLimit(timeout);
            }
       }
       String pri = req.getParameter("metaDataPriority");
       if (pri != null) {
            priority = new Integer(pri).intValue();
            if (priority!=0) {
               fragmentInfo.setPriority(priority);
            }
       }
   }
   public void initialize(CacheConfig cc) {
   }
}
```

## Dependency-id element

Use the dependency-id element to specify additional cache identifiers that associate multiple cache entries to the same group identifier.

The value of the dependency-id element is generated by concatenating the dependency ID base string with the values that are returned by its component elements. If a required component returns a null value, the entire dependency does not generate and is not used.

Validate the dependency IDs explicitly through the DynaCache API, or use the invalidation element. Multiple dependency ID rules can exist in one cache-entry element. All dependency rules run separately.

### Invalidation element

To invalidate cached objects, the application server must generate unique invalidation IDs. Build invalidation IDs by writing custom Java code or through rules that are defined in the cache policy of each cache entry. The following example illustrates an invalidation in the cache policy:

```
<invalidation>component* | invalidationgenerator? </invalidation>
```

### Invalidationgenerator sub-element

The invalidationgenerator element is used with the Web Services client cache only. Use the invalidationgenerator element to specify the class name to load for generating invalidation IDs. The InvalidationGenerator class must implement the com.ibm.websphere.cache.webservices.InvalidationGenerator interface. An example of the invalidationgenerator element is:

```
<invalidationgenerator>class name</invalidationgenerator>
```

In this example, `class name` is the fully qualified name of the class that implements the com.ibm.websphere.cache.webservices.InvalidationGenerator interface. Define this generator class in a shared library.

## 6.10.6 Cache servlet filtering and Commerce DC_ variables

Prior to WebSphere Commerce Version 5.5, WebSphere Commerce provided its own caching mechanism. Using the previous mechanism, Web pages could be cached based on either of two methods:

- Session-independent (SI) caching: Pages were cached based on URL parameters.
- Session-dependent (SD) caching: Pages were cached based on URL parameters, user's language, preferred currency, parent organization, contract IDs, and member groups.

Cache IDs for SI caching were generated based on the URL parameters; for SD caching, the cache IDs were created with the URL parameters plus the session information.

In order to provide the same functionality as the previous session-dependent caching, but using the WebSphere Application Server dynamic caching mechanism, WebSphere Commerce has introduced the servlet filter known as the *cache filter*. This cache filter is designed to set up the request attributes from the session information to be used by the DynaCache to construct the cache ID. Since the session information is set by the WebSphere Commerce Server runtime, the cache filter will not be able to set all of the request attributes until the second request against the Web site.

*Table 6-5 WebSphere Commerce-specific attributes*

| Request attribute | Description |
|---|---|
| DC_curr | User's preferred currency |
| DC_lang | User's preferred language |
| DC_porg | User's parent organization |
| DC_cont | User's current contract |
| DC_mg | User's explicit member groups |
| DC_storeId | Store identifier |
| DC_userId | User's identifier |
| DC_portal | WebSphere Portal's adapter identifier |
| DC_buyCont | Buyer's eligible contracts (only valid for Supply Chain business model) |
| DC_userType | Type of logged on user (G/R/S) |

Since a user can be eligible for multiple contracts and can belong to multiple member groups, the request attributes DC_cont and DC_mg might contain multiple values. For such a user, the values are sorted and concatenated together with a semicolon (;) as a separator.

In addition, multiple contract and member group request attributes are defined (for example, DC_cont0, DC_cont1, ... DC_contN where N is the number of contracts to which the user is entitled).

For example, if a user is eligible for contracts 10004 and 10005, then the following request attributes are set up: DC_cont is 10004;10005, DC_cont0 is 10004, DC_cont1 is 10005.

The purpose of setting request attribute DC_cont is to allow construction of a cache ID that has a limited number of components.

The purpose of setting individual request attributes DC_cont0, DC_cont1, ..., DC_contN is to allow construction of dependency IDs for more granular cache invalidations.

Since the member group information is not part of the session data, the cache filter has to retrieve this information from the database based on the user ID. In order to prevent performance degradation due to repeating database queries, the cache filter uses WebSphere command caching to accomplish this task. A command class called MemberGroupsCacheCmdImpl extends directly from the

Archived

WebSphere command framework, and is used to cache the member groups to which users belong, based on user IDs.

### 6.10.7 ConsumerDirect jspStoreDir issue

At the time of writing the jspStoreDir cachespec entry definition is a temporary work around for an internal JSP caching problem that you may run into with multiple hosted sites running ConsumerDirect.

For stand-alone sites, the problem will not occur.

Archived

7

# FAQs

This chapter answers some commonly asked questions about DynaCache. The first set of questions are about general DynaCache topics. The second set deal specifically with DynaCache and clustering.

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

## 7.1 DynaCache FAQs

### *Do I have to write any Java code to start caching objects?*

*No.* DynaCache loads and processes the cachespec.xml file and starts caching without the need for any developer code. Many objects in WebSphere Commerce will automatically appear in the cache without any effort from the system builder. Developers can use DynaCache APIs if they wish, but it is not necessary.

### *Do I have to restart the server if I change the cachespec.xml?*

*No*. DynaCache continues to monitor the file and automatically implements updates. It will work out what to do with the previous cache entries and remove them if required.

### *Can I still cache parts of a page and avoid caching the personalized bits so that I get at least some benefit from caching?*

*Yes.* You can nominate an entire page for caching with exclusion instructions that omit any parts you do not want (using the **`do-not-cache`** command). The parts that are excluded are always executed by the application server and DynaCache will assemble the response page from both the cached and executed parts. The design of personalized fragments and the impact on caching is an important consideration for the Web site designer.

### *What is the best way to serve static content in WebSphere Commerce?*

WebSphere Commerce serves static content directly through the Web server. An alias is created in the httpd.conf file during the instance creation to point to the stores directory in the following fashion:

```
Alias /wcsstore C:\WebSphere\AppServer\profiles\demo\installedApps\
WC_demo_cell\WC_demo.ear/Stores.war
```

All static content gets picked up directly on the Web server, and the application server only handles dynamic requests. If the Web server resides on a separate machine, the assets are copied over to the Web server.

### *What types of servlets and JSPs does DynaCache support most effectively?*

- Caching a simple presentation JSP file gives moderate performance gains.
- Caching a servlet that requests large amounts of information from EJBs or databases reduces WebSphere Application Server and database loading and the number of network interactions.

- Caching servlets that pull information from outside WebSphere Application Server produce the biggest performance gains.

### DynaCache caches the JSP and servlet output. Does this mean it caches the HttpServletResponse object only?

*Not quite.* DynaCache caches the output of the servlet, that is, what is written to the response.getWriter() method. Unless your cachespec.xml file expressly prohibits this, DynaCache also caches “side effects” of the servlet’s execution, like setting cookies and headers, including and forwarding to other servlets, and setting content type and character encoding.

### Is this an “in memory” cache, or an “on disk” cache? Does it use the Java heap?

*Both.* DynaCache resides primarily on the Java heap. This keeps it in memory. However, DynaCache also supports the use of virtual memory, which we call Hash Table On Disk or HTOD.

The HTOD subsystem is used for overflow situations and to provide support for very large caches. It is an optional capability. Cache entries will also be offloaded to disk upon server shutdown and can be reused when the server is restarted.

Note that the effect of heap fragmentation and garbage collection is always a consideration when caching memory objects.

### Can I influence what goes into the memory cache and what will overflow to disk?

*Yes.* Each cachespec entry has a `priority` value. DynaCache shunts lower priority items to the disk cache if the memory cache becomes full.

DynaCache also uses a LRU (Least Recently Used) algorithm to assist in selecting candidates to move. The priority is essentially the number of free passes an entry can have to stay in the cache when the LRU algorithm is looking for cache entries to evict. The bigger the number, the higher the priority to remain in memory. Recent changes have been made to provide even greater control over the memory and disk. Refer to the section “What is new in Version 6 of DynaCache” on page 134.

### Does the DynaCache need an external cache for caching?

*No.* DynaCache does not require an external cache to be present for caching. It will, however, extend the abilities of such caches to include caching certain servlet and JSP files.

### What are the security implications of using the DynaCache?

DynaCache does all the processing within the Web container after the security processing completes. Within the application server, there are no extra security problems to be considered.

However, when using an external cache, security risks change dramatically. Caches outside of WebSphere Application Server do not undergo security processing. It is important *not* to store sensitive data in an external cache. Anyone with JNDI namespace access can look up a cache instance and examine the contents of the cache.

### What happens if cache data becomes stale?

You can use dependency IDs and invalidation rules that will automatically evict stale cache entries. This book describes in detail how to go about planning, configuring and implementing invalidation.

### What about clusters? Does DynaCache work in a clustered environment?

*Yes.* DynaCache provides Distributed Replication Support (DRS) for distributing cached entries across a cluster.

### Does each node in a cluster have its own disk cache? Can I use network attached storage devices to share the disk cache for all members in the cluster?

*No*. You cannot do this. DynaCache keeps a per node, “in memory” index of all items that are located on the disk, which would break if you attempted to share disk cache files across a cluster. The index would have to be shared across all cluster members, but for now there is no support for such a concept.

### I have a very large catalog that I am thinking of caching. I can’t fit it all into memory so how much disk space will I need?

The approximate guide for computing your disk cache space is:

```
page size x number of entries = disk size
```

### Would a SAN improve the performance for a disk cache?

*Yes.*

Archived

***I have read that there is an option to allow the persisting of the memory cache to disk on an application server shutdown. What impact will saving the cache have on shutting down the server?***

*Significant.* Several minutes are added to the shutdown of a server if this option is used. DynaCache serializes Java objects to disk. Your cached objects must be serializable. Test this out beforehand.

***Should I always use timeouts on my cache entries for invalidation?***

*No.* Use timeouts sparingly! The best practice for invalidating a cached item and any dependent items is to invalidate only when you need to.

We have found many examples of using timeouts throughout the cachespec. This often forces objects out of the cache that are still perfectly valid. The server wastes considerable time recreating the same objects in the cache, completely unnecessarily.

Use timeouts as a last resort, or if you know that your object will change within every timeout period.

## 7.2 Clustering FAQs

***I don't really see how having a separate replicator for each domain can provide increased availability in the case where the other replicator goes down, since each of application instances would have a replicator defined and can't jump over to use the next replicator.***

The assignment of application servers to replicators is to be viewed only as an initial startup mapping. If in the course of operations, one of the replicators in the domain goes down or becomes unavailable, the server will fail over to one of the remaining replicators in the domain.

***If an application server goes down for thirty minutes, when it comes back up, would it receive all the invalidation requests from DRS? Or would it be out of sync with the rest of the application servers? If it is the first, is the only way out to disable flush to disk on startup?***

When it comes back up, if flush to disk is off, it will be bootstrapped. If push/pull, it will only push metadata.

Archived

### *What happens if a replicator goes down? Would the second replicator pick up any invalidations that didn't get pushed by the first one?*

*Yes.* Invalidations will be picked up automatically because both replicators see all invalidations.

### *Is there a recommended hardware guideline for DRS? One CPU per replicator? The size of the JVM?*

512 MB is a reasonable size for the JVM. Updates are buffered in memory so memory is at more of a premium than the CPU.

### *What protocol does DRS use for communication in WebSphere Application Server 5.1?*

JMS

Archived

# Part 2

# DynaCache implementation

In this part we provide a tutorial to help you set up DynaCache and perform benchmarking.

The last chapter, “Case study: A DynaCache anti-pattern” is a real example, provided by IBM Software Services, of how they used DynaCache to bring a Web site back from premature extinction. On delivery from test the Web site turned out to have completely unacceptable performance. With a tight schedule, and limited access to the original developers (who had left the project), the service team implemented a caching scheme and were able to achieve a just barely acceptable level of performance, but sufficient to publish the Web site. With the time this bought them, they were able to fix some of the underlying problems in the design of the Web site, and then apply DynaCache more effectively with even better results.

© Copyright IBM Corp. 2006. All rights reserved.

Archived

# 8

# DynaCache tutorial

Part 1 of this book explained the theory and practice of DynaCache. This chapter demonstrates how you can put that knowledge and skill into practice by giving you a tutorial in doing several basic DynaCache tasks in WebSphere Commerce V6.0 to improve performance.

This chapter shows how to do the following:

- Configure DynaCache
- Set up Cache Monitor
- Configure the cachespec.xml file
- Verify DynaCache setup

Upon completion of this tutorial, you should be able to:

- Implement full-page caching
- Separately cache page fragments, such as eSpots and mini-cart
- Implement cache command invalidation for mini-cart fragments
- Build cache-ids from different components
- Understand the additional requirements to use DynaCache in a production environment

© Copyright IBM Corp. 2006. All rights reserved.

# 8.1 Environment setup

Before starting the tutorial, you must set up the environment correctly as explained in this section.

## 8.1.1 Software stack

This tutorial is based on single-tier installation of WebSphere Commerce on Microsoft Windows® Server 2003. If you are using a platform other than Microsoft Windows, you need to substitute the platform-specific tasks in this tutorial accordingly.

The WebSphere Commerce software versions we used were the following:

- IBM HTTP Server 6.0
- IBM WebSphere Application Server 6.0.2.5
- IBM WebSphere Commerce 6.0
- IBM DB2 Enterprise Server Edition 8.1 Fix Pack 10

## 8.1.2 WebSphere Commerce setup

After installing the identified software stack, create a WebSphere Commerce instance as explained in Part 7 of the WebSphere Commerce installation guide.

This tutorial guides you through publishing and using the B2C ConsumerDirect sample store included in WebSphere Commerce. The approaches illustrated in this tutorial are general in nature and can be applied to any WebSphere Commerce store.

The names shown in Table 8-1 are used in this tutorial.

*Table 8-1 Tutorial setup configuration*

| Setup variable | Tutorial-specific name |
|---|---|
| Web server instance name | webserver1 |
| WC instance name | demo |
| DB2 instance name | DB2-0 |

### Modify the WebSphere Commerce instance to handle "double clicks"

When a user clicks on cacheable link A and then cacheable link B, DynaCache creates two placeholder entries with different cache keys (one for A and one for



Archived

B). By default, in WebSphere Commerce, if the user clicks on B before A can return, then a feature called "DoubleClickHandler" will drop request B and process only request A. The result is that both the DynaCache placeholders wrongly end up being filled with the response to A.

The solution is to selectively disable DoubleClickHandler for certain commands in the WebSphere Commerce instance_name.xml file. Make a backup copy before the file is edited.

When switching between non-SSL and SSL requests, the parameters are encrypted. Encryption must be suppressed so that the parameters in the cachespec.xml file are used. To suppress the encryption, make the following changes to the file:

1. Open the <wc_instance_name>.xml file (in our case demo.xml) in a text editor.
2. Search for the following text:

```
</ProtectedParameters>
```

3. Add the following text immediately below it:

```
<NonEncryptedParameters>
   <Parameter name="storeId" />
   <Parameter name="langId" />
   <Parameter name="catalogId" />
   <Parameter name="productId" />
</NonEncryptedParameters>
```

The default instance creation enables an option named DoubleClickHandler. This option is used to handle multiple requests for the same command from the same user. This code does not work well with dynamic caching. It must be selectively disabled on a command basis. For the purposes of this example it has been disabled for the commands CategoryDisplay, ProductDisplay, TopCategoryDisplay, and StoreCatalogDisplay, as shown in the next steps.

4. Add the following text below `</NonEncryptedParameters>`:

```
<DoubleClickMonitoredCommands>
   <excludeCommands>
      <command name="CategoryDisplay" />
      <command name="ProductDisplay" />
      <command name="TopCategoryDisplay" />
      <command name="StoreCatalogDisplay" />
   </excludeCommands>
</DoubleClickMonitoredCommands>
```

5. Save your changes and close the text editor.

### 8.1.3 Enable DynaCache service

Although the DynaCache service is enabled by default, it is important to make sure that the service is enabled correctly.

Perform the following steps to enable DynaCache service:

1. Start WebSphere Commerce server.
   a. Ensure that your database management system is started by selecting **Control Panel** → **Adminis**trative Tools → Services → DB2 - DB2-0 → Start**.**
   b. Ensure that the Web server configured for WC is started by selecting **Control Panel** → **Administrative Tools** → **Services** → **IBM HTTP Server for WebSphere Commerce (demo)** → **Start**.
   c. Start a WebSphere Commerce instance by selecting **Control Panel** → **Administrative Tools** → **Services** → **IBM WebSphere Application Server V6 - WC_demo** → **Start**.
2. Configure DynaCache using the WebSphere Administrative Console.
   a. Expand Servers from the left navigation menu (Figure 8-1).

*Figure 8-1 WAS Servers menu*

   b. Navigate to the DynaCache service screen (Figure 8-2) by selecting **Application servers** → **server1** → **Container Settings** → **Container Services** → **DynaCache Service**.
   c. Ensure that “Enable service at server startup” is checked.
   d. Ensure that “Enable disk offload” is checked.
   e. Optionally, you can specify the Disk Offload location on your file system.
   f. Ensure that “Flush to disk” is checked.
   g. After you have made any necessary changes, click **Apply** and then **Save all changes**.

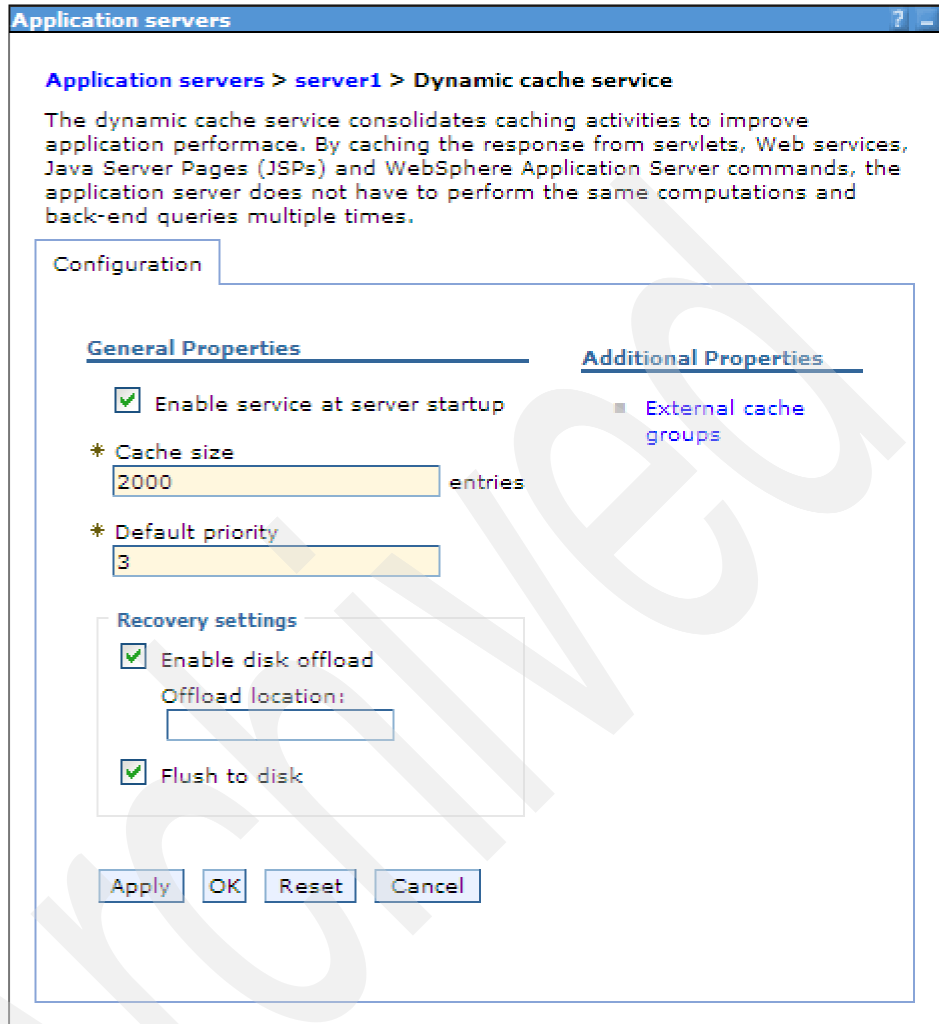



*Figure 8-2 DynaCache service*

h. Restart the WebSphere Commerce instance by selecting **Control Panel** → **Administrative Tools** → **Services** → **IBM WebSphere Application Server V6 - WC_demo** → **Restart**.

## 8.2 Installing the Cache Monitor

The DynaCache monitor is an installable Web application that displays simple cache statistics, cache entries, and cache policy information. We use Cache Monitor to verify the cachespec.xml configuration for the ConsumerDirect store in this tutorial.

1. Use the WebSphere Application Server Administrative console to install CacheMonitor.ear

   a. Expand **Applications** from the left navigation menu (Figure 8-3).

*Figure 8-3 Applications*

   b. Navigate to the Install New Application screen by selecting **Applications** → **Install New Application**.

   c. Click **Browse** and locate the CacheMonitor.ear under the WAS_install_root/installableApps directory.

   d. Click **Next** to accept the default settings (Figure 8-4).

Archived

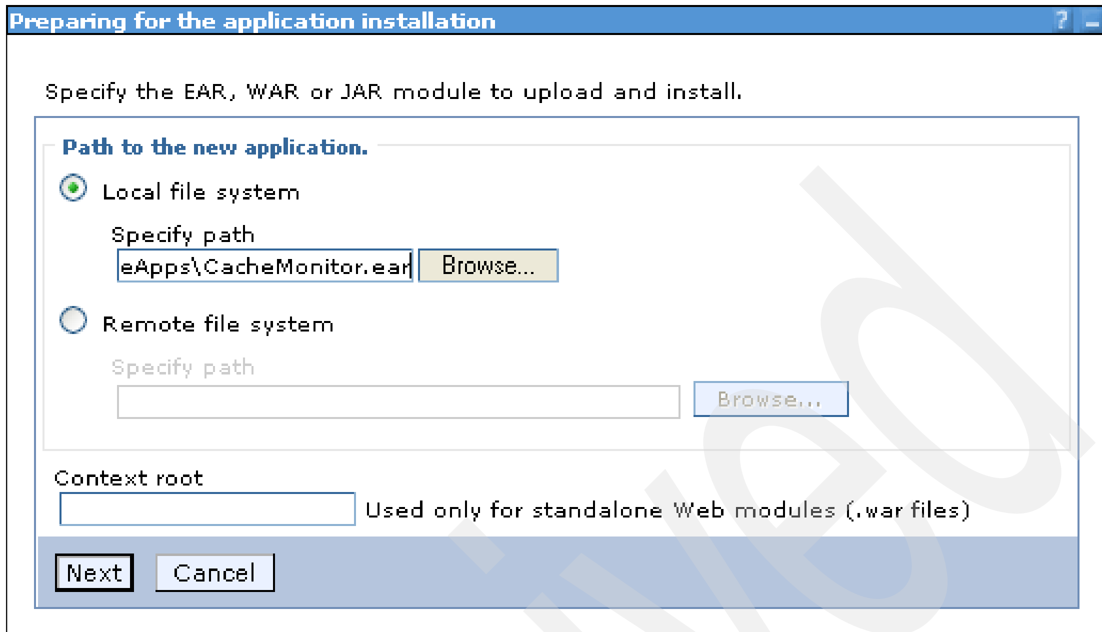



*Figure 8-4 CacheMonitor.ear*

e. Click **Next** on the Preparing for the application installation screen. You will receive a warning; click **Continue**.

f. On the Select installation options screen, accept the default values and click **Next**.

g. On the Map modules to servers screen, select the application server to monitor. Select both server names for the servers to monitor from the list of available servers you can map to. Then, check the box beside the DynaCache Monitor module, and click **Apply**. As part of the entry in the server column of the table, you should now see both server names as shown below in Figure 8-5.

Archived

**Map modules to servers**

Specify targets such as application servers or clusters of application servers where you want to install the modules contained in your application. Modules can be installed on the same application server or dispersed among several application servers. Also, specify the Web servers as targets that will serve as routers for requests to this application. The plug-in configuration file (plugin-cfg.xml) for each Web server is generated based on the applications which are routed through it.

Clusters and Servers:

WebSphere:cell=WC_demo_cell,node=WC_demo_node,server=server1
WebSphere:cell=WC_demo_cell,node=webserver1_node,server=webserver1

Apply

| Select | Module | URI | Server |
|---|---|---|---|
| ☐ | Dynamic Cache Monitor | CacheMonitor.war,WEB-INF/web.xml | WebSphere:cell=WC_demo_cell,node=WC_demo_node,server=server1<br>WebSphere:cell=WC_demo_cell,node=webserver1_node,server=webserver1 |

*Figure 8-5 Map modules to servers*

h. On the Map virtual hosts for Web modules screen, select **VH_demo** as the virtual host (Figure 8-6).

**Map virtual hosts for Web modules**

Specify the virtual host where you want to install the Web modules contained in your application. You can install Web modules on the same virtual host or disperse them among several hosts.

⊞ Apply Multiple Mappings

| Select | Web module | Virtual host |
|---|---|---|
| ☐ | Dynamic Cache Monitor | admin_host |

default_host
admin_host
VH_demo
VH_demo_Tools
VH_demo_Admin
VH_demo_OrgAdmin
VH_demo_Preview

*Figure 8-6 Map virtual hosts for Web modules*

i. On the Map security roles to users/groups screen, accept the default values and click **Next**.

j. On the Summary screen, accept the default values and click **Finish**.

k. Save the changes.

2. Regenerate the Web server plug-ins.

   a. Using the WebSphere Administrative Console, navigate to **Servers** → **Web Servers**.

   b. Select **webserver1** → **Generate Plug-in** (Figure 8-7).

Archived

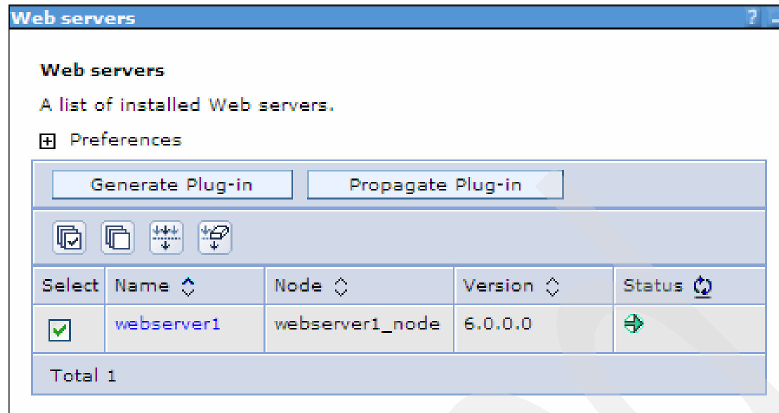



*Figure 8-7 Web servers*

3. Restart the Web server and application server.

   a. Stop WC_demo by selecting **Control Panel** → **Administrative Tools** → **Services** → **IBM WebSphere Application Server V6 - WC_demo** → **Stop**.

   b. Stop IHS by selecting **Control Panel** → **Administrative Tools** → **Services** → **IBM HTTP Server for WebSphere Commerce (demo)** → **Stop**.

   c. Start IHS by selecting **Control Panel** → **Administrative Tools** → **Services** → **IBM HTTP Server for WebSphere Commerce (demo)** → **Start**.

   d. Start WC_demo by selecting **Control Panel** → **Administrative Tools** → **Services** → **IBM WebSphere Application Server V6 - WC_demo** → **Start**.

4. Verify that you can access the Cache Monitor by pointing your Web browser to:

```
http://host_name/cachemonitor
```

   You should now be able to see the Cache Monitor screen in your browser as shown in Figure 8-8.

Archived

*Figure 8-8 Cache Monitor*

5. Reading cache statistics

   Use the cache monitor's main screen to watch cache hits versus misses. By comparing these two values, determine how much DynaCache is helping your application, and whether there are any additional steps you can take to improve performance further and decrease the load on the Application Server.

   Cache Monitor displays the cache statistics described in Table 8-2.

*Table 8-2 Cache statistics*

| Cache statistic | Description |
|---|---|
| Cache Size | The maximum number of entries that the cache can hold. |
| Used Entries | The number of cache entries used. |

| Cache statistic | Description |
|---|---|
| Cache Hits | The number of request responses that are served from the cache. |
| Cache Misses | The number of request responses that are cacheable but cannot be served from the cache. |
| LRU Evictions | The number of cache entries removed to make room for new cache entries. |
| Explicit Removals | The number of cache entries removed or invalidated from the cache based on cache policies or were deleted from the cache through the cache monitor. |
| Default priority | Specifies the default priority for all cache entries. Lower priority entries are moved from the cache before higher priority entries when the cache is full. You can specify the priority for individual cache entries in the cache policy. |
| Servlet Caching Enabled | If servlet caching is enabled, results from servlets and Java Server Pages (JSP) files are cached. |
| Disk Offload Enabled | Specifies if entries that are being removed from the cache are saved to disk. |

## 8.3 Caching ConsumerDirect store

The ConsumerDirect store included with WebSphere Commerce contains the most commonly used shopping functions. Consumer Direct supports commerce transactions involving products, services, or information between businesses and consumers. A user, either registered or not, can shop at ConsumerDirect and enjoy a variety of features provided by the store. It is available on all the editions of WebSphere Commerce.

In this tutorial, you will use the default implementation of ConsumerDirect sample store to cache the catalog-related pages.

### 8.3.1 Catalog subsystem URLs

Catalog subsystem URLs include all logic and data relevant to a catalog, including categories, products and their attributes, items, and groupings of each, and any associations or relationships among them.

Archived

A typical catalog page's flow looks like this:

```
StoreCatalogDisplay → TopCategoriesDisplay → CategoryDisplay →
ProductDisplay
```

Table 8-3 shows what the URLs in this group can do.

*Table 8-3 Catalog subsystem URLs*

| URLs | Description |
|---|---|
| StoreCatalogDisplay | Display all the catalogs for a given store. However, in the default ConsumerDirect store, this URL is not used and is mapped to TopCategoriesDisplay. |
| TopCategoriesDisplay | Display the root categories for a given catalog. |
| CategoryDisplay | Display a category within a catalog. |
| ProductDisplay | Display a catalog entry. |

In order to cache these catalog URLs, we use the recommended strategy of full-page caching. WebSphere Commerce v6 sample stores are based on Struts main servlet name, which is different from WebSphere Commerce v5. The new servlet name is ECActionServlet; the old Stores main servlet name was RequestServlet.

Add a full page cache-entry for ConsumerDirect store into the cachespec.xml file:

```
WAS_install_root\profiles\demo\installedApps\WC_demo_cell\WC_demo.ea
r\Stores.war\WEB-INF\cachespec.xml
```

The cache-entry structure is shown in Example 8-1.

*Example 8-1 Full page cache-entry structure*

```
<cache-entry>
 <class>servlet</class>
 <name>com.ibm.commerce.struts.ECActionServlet.class</name>
 <property name="consume-subfragments">true</property>
 <property name="save-attributes">false
   <exclude>jspStoreDir</exclude>
 </property>

 <!-- TopCategoriesDisplay?storeId=s&catalogId=s -->
   ........................
   ........................
 <!-- CategoryDisplay?storeId=s&catalogId=s&categoryId=s -->
   ........................
```

```
   .........................
 <!-- ProductDisplay?storeId=s&productId=s -->
   .........................
   .........................
</cache-entry>
```

You are now ready to create cache-ids for each of the URLs created in Example 8-1:

- TopCategoriesDisplay
- CategoryDisplay
- ProductDisplay

## 8.3.2 TopCategoriesDisplay

TopCategoriesDisplay displays the root categories for a catalog. ConsumerDirect uses this URL as the home page for the store Web site. The URL structure and parameter values are shown in Figure 8-9.

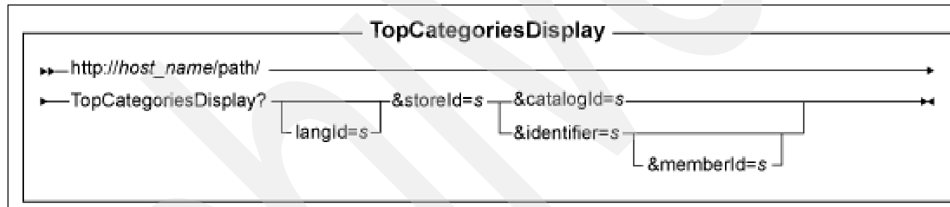



*Figure 8-9 TopCategoriesDisplay*

Table 8-4 explains the parameter values that are used by the TopCategoriesDisplay URL.

*Table 8-4 TopCategoriesDisplay parameter values*

| Parameter name | Description |
|---|---|
| langId | Sets or resets the preferred language for the duration of the session. The supported languages for a store are found in the STORELANG table. |
| storeId | (Required) The reference number of the store associated with the categories. |
| catalogId | The reference number for the catalog that is associated with the given store. The catalog contains root categories. |
| identifier | The external identifier for the category. |

| Parameter name | Description |
|---|---|
| memberId | The reference number of the member who owns the category. The memberId along with the identifier uniquely identify the category. If a memberId is omitted, then the owner of the current store along with the identifier are used to uniquely identify the category. |

By default ConsumerDirect uses the following parameters:

- StoreId
- CatalogId
- CategoryId (ConsumerDirect store specific parameter)

Based on this information, add the cache-id for TopCategoriesDisplay shown in Example 8-2.

*Example 8-2 TopCategoriesDisplay cache-id*

```
<cache-id>

 <component id="" type="pathinfo">
  <required>true</required>
  <value>/TopCategoriesDisplay</value>
 </component>

 <component id="storeId" type="parameter">
  <required>true</required>
 </component>

 <component id="catalogId" type="parameter">
  <required>true</required>
 </component>

 <component id="categoryId" type="parameter">
  <required>false</required>
 </component>

</cache-id>
```

Now check that cachespec.xml is loaded correctly by verifying the TopCategoriesDisplay cache policy in Cache Monitor.

1. Check that cachespec.xml is reloaded.

   a. Point your Web browser to the Cache Monitor:

      `http://host_name/cachemonitor`

Archived

b. Navigate to **Cache Policies** → /webapp/wcs/stores/com.ibm.commerce.struts.ECActionServlet.class.

c. You will see the TopCategoriesDisplay cache policy that you defined. It should look like Figure 8-10.

Cache Policy Detail for /webapp/wcs/stores/com.ibm.commerce.struts.ECActionServlet.class

Cache ID Rule 0 | timeout: 0 | priority: 0 | inactivity: 0 | idGenerator: null | metaDataGenerator: null | properties:

| Component | ID | Type | Ignore Value | Method | Field | Required | Values | Not-Values | Index | Multiple IDs |
|---|---|---|---|---|---|---|---|---|---|---|
| Component 0 | | pathinfo | false | null | null | true | /TopCategoriesDisplay | | | false |
| Component 1 | storeId | parameter | false | null | null | true | | | | false |
| Component 2 | catalogId | parameter | false | null | null | true | | | | false |
| Component 3 | categoryId | parameter | false | null | null | false | | | | false |

*Figure 8-10 TopCategoriesDisplay cache policy*

2. Verify that TopCategoriesDisplay URL is being cached.

a. Clear any existing cache and reset statistics using Cache Monitor by selecting **Cache Monitor** → **Cache Statistics** → **Clear Cache** → **Reset Statistics**.

b. The cache statistics are reset as shown in Figure 8-11.

| Statistic | Value |
|---|---|
| Cache Size | 2000 |
| Used Entries | 0 |
| Cache Hits | 0 |
| Cache Misses | 0 |
| LRU Evictions | 0 |
| Explicit Removals | 0 |
| Default Priority | 3 |
| Servlet Caching Enabled | Yes |
| Disk Offload Enabled | Yes |

*Figure 8-11 Cache Statistics*

c. Open a new Web browser and point to the ConsumerDirect home page:

```
http://host_name/webapp/wcs/stores/servlet/TopCategoriesDisplay?langId=-1&storeId=10001&catalogId=10001
```

The home page is shown in Figure 8-12.

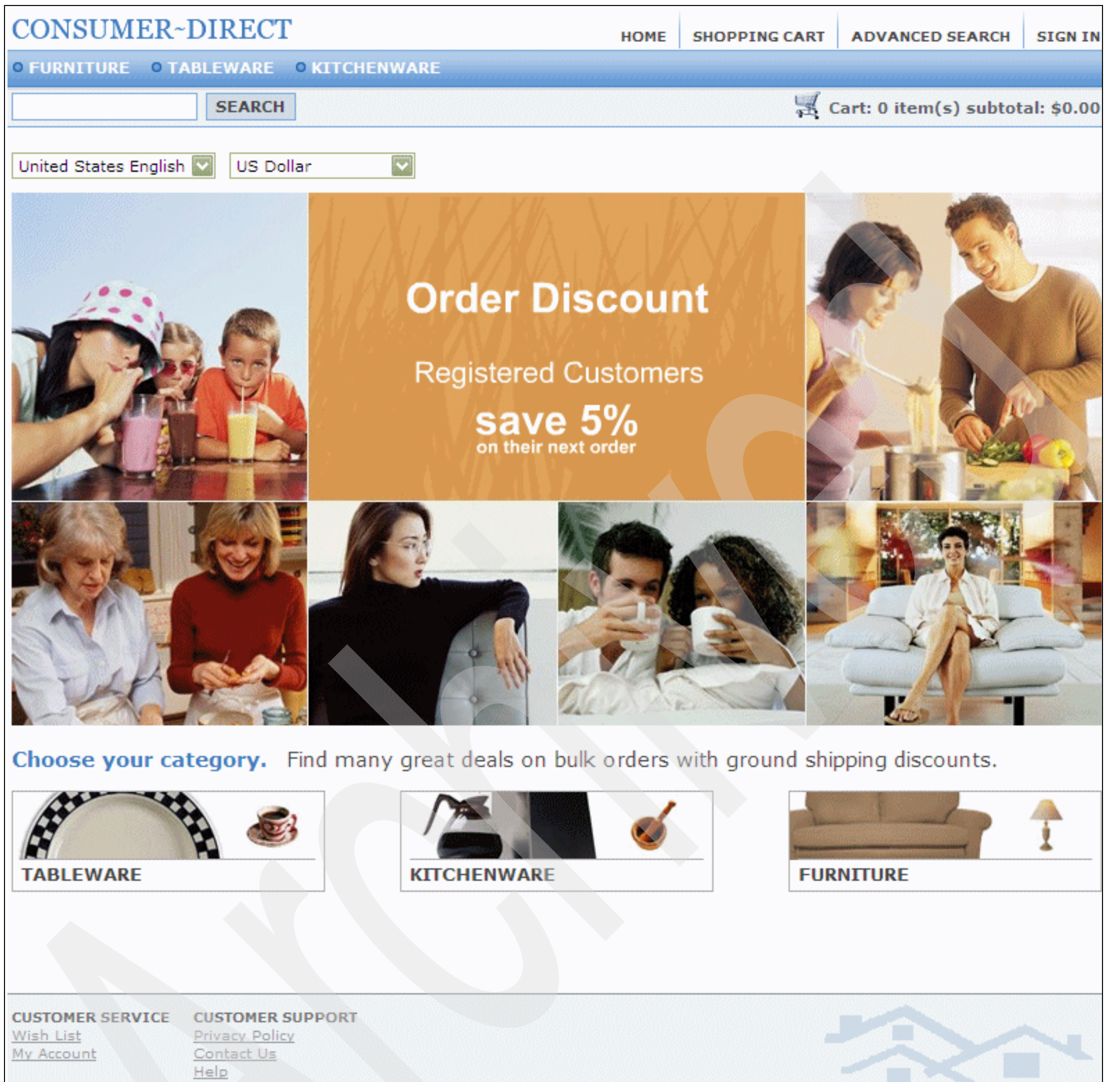



*Figure 8-12 ConsumerDirect Home page*

d. Refresh the Cache Monitor to view updated statistics. You will see some Cache Misses because the cache was cleared. This is shown in Figure 8-13.

| Statistic | Value |
|---|---|
| Cache Size | 2000 |
| Used Entries | 4 |
| Cache Hits | 12 |
| Cache Misses | 4 |
| LRU Evictions | 0 |
| Explicit Removals | 0 |
| Default Priority | 3 |
| Servlet Caching Enabled | Yes |
| Disk Offload Enabled | Yes |

*Figure 8-13 Cache Statistics after TopCategoriesDisplay call*

3. Find the fragments that we do not want to consume with the full page.

   a. Navigate to this cached page using Cache Monitor and selecting **Cache Contents** → .../ECActionServlet.class Template → ...:pathinfo=/TopCategoriesDisplay Cache ID.

   b. To find the included JSPs that are consumed with this full page cache, look for the `"CONSUMED include"` term.

      i. You will find the first entry for TopCategoriesDisplay.jsp. This is the parent JSP and we want it to be consumed - this is the correct behavior.

      ii. CachedHeaderDisplay.jsp is the second JSP that is consumed.

      CachedHeaderDisplay.jsp is a dynamic fragment that causes the home page content to change when a user logs in. However, it is cached with the TopCategoriesDisplay page. The result is that the home page content does not change when the TopCategoriesDisplay page is displayed again. When another user logs in and displays the TopCategoriesDisplay page, the CachedHeaderDisplay is loaded from the cache and displays the previous user's details.

      To correct this you need to stop CachedHeaderDisplay.jsp from being consumed by TopCategoriesDisplay. Create a new cache-entry in your cachespec.xml file as shown in Example 8-3.

Archived

*Example 8-3 CachedHeaderDisplay.jsp cache-entry*

```
<cache-entry>

 <class>servlet</class>

<name>/ConsumerDirect/include/styles/style1/CachedHeaderDisplay.jsp</name>

<name>/ConsumerDirect/include/styles/style2/CachedHeaderDisplay.jsp</name>
 <property name="do-not-consume">true</property>
 <property name="save-attributes">false</property>

 <cache-id>

  <component id="storeId" type="parameter">
   <required>true</required>
  </component>

  <component id="catalogId" type="parameter">
   <required>true</required>
  </component>

  <component id="DC_userType" type="attribute">
   <required>false</required>
   <not-value>-1002</not-value>
  </component>

  <component id="DC_lang" type="attribute">
   <required>true</required>
  </component>

 </cache-id>

</cache-entry>
```

iii. Continue to search for more consumed fragments in Cache Monitor.

The next entry is MiniShopCartDisplay.jsp.

MiniShopCartDisplay.jsp displays the number of items in the user cart and the subtotal. We handle the caching of this JSP fragment differently from the rest of the page. Use command-based invalidation to re-cache the MiniShopCartDisplay.jsp every time the user makes a change to their shopping cart.

Add the snippet shown in Example 8-4 to the cachespec.xml file.

Archived

*Example 8-4 /MiniShopCartDisplay.jsp*

```
<cache-entry>
 <class>servlet</class>
 <name>/ConsumerDirect/include/MiniShopCartDisplay.jsp</name>
 <property name="do-not-consume">true</property>
 <property name="save-attributes">false</property>
 <cache-id>
  <component id="DC_storeId" type="attribute">
   <required>true</required>
  </component>
  <component id="DC_userId" type="attribute">
   <required>false</required>
   <not-value>-1002</not-value>
  </component>
  <component id="DC_lang" type="attribute">
   <required>true</required>
  </component>
  <component id="DC_curr" type="attribute">
   <required>true</required>
  </component>
  <priority>1</priority>
  <timeout>3600</timeout>
  <inactivity>600</inactivity>
 </cache-id>
 <dependency-id>DC_storeId
  <component id="DC_storeId" type="attribute">
   <required>true</required>
  </component>
 </dependency-id>
 <dependency-id>DC_userId
  <component id="DC_userId" type="attribute">
    <required>true</required>
  </component>
 </dependency-id>
 <dependency-id>MiniCart</dependency-id>
 <dependency-id>MiniCart:DC_storeId
  <component id="DC_storeId" type="attribute">
    <required>true</required>
  </component>
 </dependency-id>
 <dependency-id>MiniCart:DC_userId
  <component id="DC_userId" type="attribute">
    <required>true</required>
  </component>
 </dependency-id>
 <dependency-id>MiniCart:DC_storeId:DC_userId
  <component id="DC_storeId" type="attribute">
    <required>true</required>
```

```
  </component>
  <component id="DC_userId" type="attribute">
    <required>true</required>
  </component>
 </dependency-id>
</cache-entry>

<cache-entry>
 <class>command</class>
 <sharing-policy>not-shared</sharing-policy>
 <name>com.ibm.commerce.order.commands.OrderCalculateCmdImpl</name>

<name>com.ibm.commerce.order.commands.PromotionEngineOrderCalculateCmdImpl</nam
e>
 <name>com.ibm.commerce.orderitems.commands.OrderItemMoveCmdImpl</name>

<name>com.ibm.commerce.usermanagement.commands.UserRegistrationAddCmdImpl</name
>

<name>com.ibm.commerce.usermanagement.commands.UserRegistrationUpdateCmdImpl</n
ame>
 <!-- Used by the advanced order  -->
 <name>com.ibm.commerce.order.commands.OrderProcessCOCmdImpl</name>
 <name>com.ibm.commerce.orderitems.commands.OrderItemAddCOCmdImpl</name>
 <name>com.ibm.commerce.orderitems.commands.OrderItemDeleteCOCmdImpl</name>
 <name>com.ibm.commerce.orderitems.commands.OrderItemUpdateCOCmdImpl</name>
 <name>com.ibm.commerce.order.commands.OrderCancelCOCmdImpl</name>
 <!-- Used by the classic order  -->
 <name>com.ibm.commerce.order.commands.OrderProcessCmdImpl</name>
 <name>com.ibm.commerce.orderitems.commands.OrderItemAddCmdImpl</name>
 <name>com.ibm.commerce.orderitems.commands.OrderItemDeleteCmdImpl</name>
 <name>com.ibm.commerce.orderitems.commands.OrderItemUpdateCmdImpl</name>
 <name>com.ibm.commerce.order.commands.OrderCancelCmdImpl</name>

 <invalidation>MiniCart:DC_storeId:DC_userId
  <component type="method" id="getCommandContext">
   <method>getStoreId</method>
   <required>true</required>
  </component>
  <component type="method" id="getCommandContext">
   <method>getUserId</method>
   <required>true</required>
  </component>
 </invalidation>
</cache-entry>
```

iv. Continue to search for more consumed fragments in Cache Monitor.

The next fragment is CachedSidebarDisplay.jsp.

This stays constant and is best left being consumed with the page for better performance.

v. The next fragments in the Cache Monitor are ContentContainerTop.jsp and ContentSpotDisplay.jsp.

These can be added to cachespec.xml by navigating to WebSphere Commerce Accelerator by selecting **ConsumerDirect** → **Store** → **Content** → **Home spot - Top and/or Bottom**.

If the home spots are updated frequently, it is better to cache this fragment separately by adding an entry as shown in Example 8-5.

*Example 8-5 ContentContainerTop.jsp*

```
<cache-entry>
 <class>servlet</class>
 <name>/ConsumerDirect/Snippets/Marketing/Content/ContentSpotDisplay.jsp</name>
 <property name="do-not-consume">true</property>
 <property name="save-attributes">false</property>
 <cache-id>
  <component id="emsName" type="parameter">
   <required>true</required>
  </component>
  <component id="DC_storeId" type="attribute">
   <required>true</required>
  </component>
  <component  id="DC_lang" type="attribute">
   <required>true</required>
  </component>
  <inactivity>600</inactivity>
 </cache-id>
</cache-entry>
```

vi. Continue to search for more consumed fragments in Cache Monitor.

The next entry is for StoreCatalogProductESpot.jsp. This alternates the eSpots on the main page. In order to avoid consuming these with TopCategoriesDisplay, so that the eSpots change, add the cache-entry shown in Example 8-6 to cachespec.xml file.

*Example 8-6 StoreCatalogProductESpot.jsp*

```
<cache-entry>
 <class>servlet</class>
 <name>/ConsumerDirect/include/StoreCatalogProductESpot.jsp</name>
 <property name="do-not-cache">true</property>
 <cache-id>
 </cache-id>
</cache-entry>
```

Archived

vii. The last fragment is CachedFooterDisplay.jsp. If the footer is session dependant then follow Example 8-7 and cache CachedFooterDisplay.jsp separately.

*Example 8-7 CachedFooterDisplay.jsp*

```
<cache-entry>
 <class>servlet</class>
 <name>/ConsumerDirect/include/styles/style1/CachedFooterDisplay.jsp</name>
 <name>/ConsumerDirect/include/styles/style2/CachedFooterDisplay.jsp</name>
 <property name="do-not-consume">true</property>
 <property name="save-attributes">false</property>
 <cache-id>
  <component id="storeId" type="parameter">
   <required>true</required>
  </component>
  <component id="DC_userType" type="attribute">
   <required>false</required>
   <not-value>-1002</not-value>
  </component>
  <component id="DC_lang" type="attribute">
   <required>true</required>
  </component>
 </cache-id>
</cache-entry>
```

4. Verify that the fragments are now cached separately.
   a. Select the **Cache Policies** page in Cache Monitor and verify all the modified fragments are listed in the cache policies.
   b. Clear the cache using Cache Monitor’s main page.
   c. Resubmit the request for TopCategoriesDisplay home page.
   d. Navigate to the **Cache Contents** page in Cache Monitor. Verify the fragments you modified are now cached separately and *not* consumed any longer.

      You will see the following templates cached with their Cache IDs:

```
/webapp/wcs/stores/ConsumerDirect/include/MiniShopCartDisplay.jsp
/webapp/wcs/stores/ConsumerDirect/include/styles/style1/CachedFoo
terDisplay.jsp
/webapp/wcs/stores/ConsumerDirect/include/styles/style1/CachedHea
derDisplay.jsp
/webapp/wcs/stores/com.ibm.commerce.struts.ECActionServlet.class
```

## 8.3.3 CategoryDisplay

The CategoryDisplay URL displays a category within a catalog. Figure 8-14 shows the URL structure and parameter values.

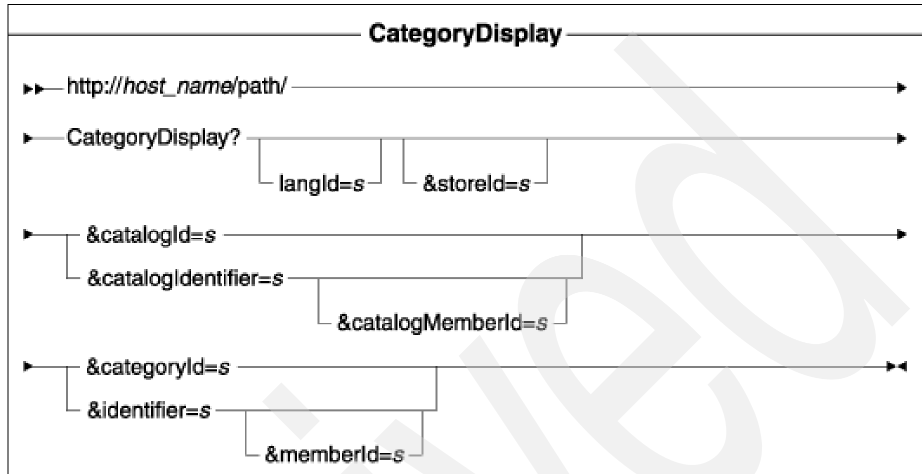



*Figure 8-14 CategoryDisplay structure*

Table 8-5 describes the parameter values that are used by CategoryDisplay.

*Table 8-5 CategoryDisplay parameter values*

| Parameter names | Description |
|---|---|
| langId | Sets or resets the preferred language for the duration of the session. The supported languages for a store are found in the STORELANG table. |
| storeId | The store's reference number associated with the category being displayed. |
| catalogId | (Required) The reference number of the catalog in which the category exists. You must specify either catalogId or catalogIdentifier. |
| catalogIdentifier | (Required) The external identifier for the catalog. You must specify either catalogIdentifier or catalogId. |
| catalogMemberId | The reference number of the member who owns the catalog. |
| categoryId | The unique reference number of the category to be displayed. |
| identifier | The external identifier for the category. |

Archived

| Parameter names | Description |
|---|---|
| memberId | The reference number of the member who owns the category. The memberId along with the identifier uniquely identify the category. If a memberId is omitted, then the owner of the current store along with the identifier are used to uniquely identify the category. |

By default ConsumerDirect uses the following parameters:

- StoreId
- CatalogId
- CategoryId
- PageView (ConsumerDirect store-specific parameter)
- CurrentPage (ConsumerDirect store-specific parameter)

Based on information from Table 8-5 on page 201, Example 8-8 shows how to add the cache-id for CategoryDisplay:

*Example 8-8 CategoryDisplay cache-id*

```
<cache-id>

 <component id="" type="pathinfo">
  <required>true</required>
  <value>/CategoryDisplay</value>
 </component>

 <component id="storeId" type="parameter">
  <required>true</required>
 </component>

 <component id="catalogId" type="parameter">
  <required>true</required>
 </component>

 <component id="categoryId" type="parameter">
  <required>true</required>
 </component>

 <component id="pageView" type="parameter">
  <required>false</required>
 </component>

 <component id="currentPage" type="parameter">
  <required>false</required>
 </component>

</cache-id>
```

Perform the following steps to verify that the cachespec.xml file is loaded correctly by verifying the cache policy in Cache Monitor.

1. Verify cachespec.xml is reloaded.

   a. Point your Web browser to the Cache Monitor:

      `http://host_name/cachemonitor`

   b. Navigate to **Cache Policies** → /webapp/wcs/stores/com.ibm.commerce.struts.ECActionServlet.class

   c. Figure 8-15 shows the CategoryDisplay cache policy that you defined in Example 8-8.

Cache ID Rule 1 timeout: 0 priority: 0 inactivity: 0 idGenerator: null metaDataGenerator: null properties:

| Component | ID | Type | Ignore Value | Method | Field | Required | Values | Not-Values | Index | Multiple IDs |
|---|---|---|---|---|---|---|---|---|---|---|
| Component 0 | | pathinfo | false | null | null | true | /CategoryDisplay | | | false |
| Component 1 | storeId | parameter | false | null | null | true | | | | false |
| Component 2 | catalogId | parameter | false | null | null | true | | | | false |
| Component 3 | categoryId | parameter | false | null | null | true | | | | false |
| Component 4 | pageView | parameter | false | null | null | false | | | | false |
| Component 5 | currentPage | parameter | false | null | null | false | | | | false |

*Figure 8-15 CategoryDisplay Cache Policy*

2. Verify that the CategoryDisplay URL is being cached.

   a. Navigate in the ConsumerDirect store to any of the CategoryDisplay pages as shown in Figure 8-16. For example, select **Home Page** → **FURNITURE**.

Archived

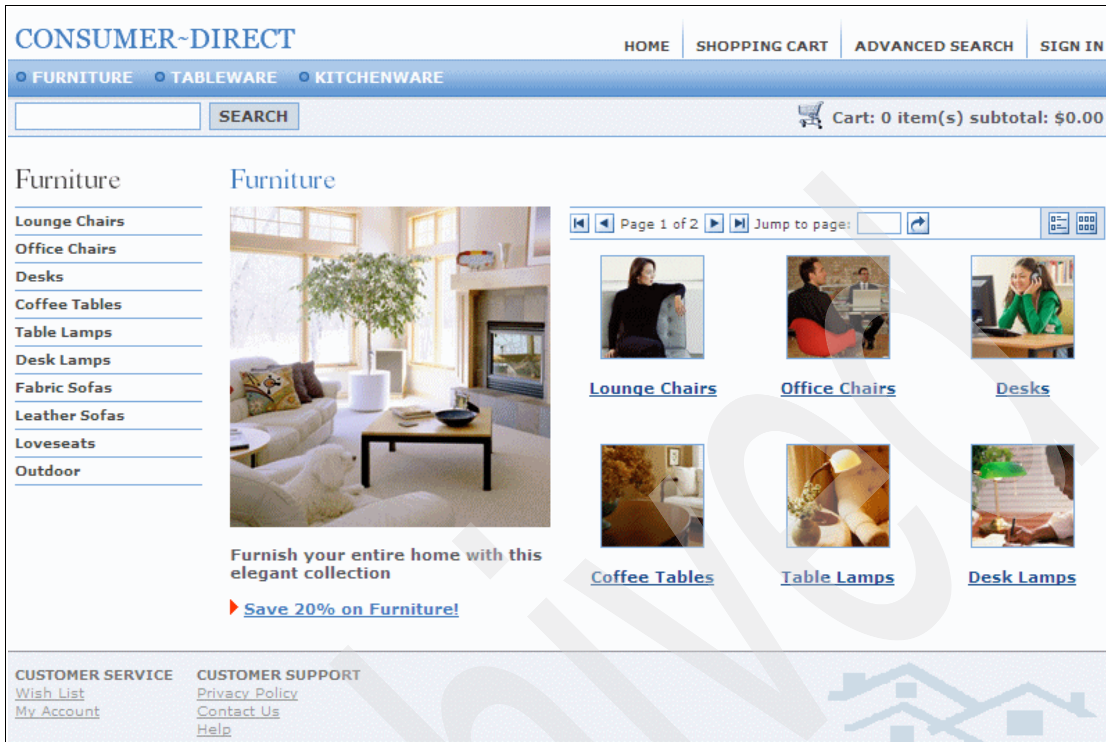



*Figure 8-16 CategoryDisplay*

3. Using Cache Monitor, navigate to **Cache Contents** → .../ECActionServlet.class Template → ...:pathinfo=/CategoryDisplay Cache ID

   b. Notice that all the fragments that must be separately cached are the same as TopCategoriesDisplay, so they are already cached separately. The rest of the contents of the CategoryDisplay page is consumed with the full page caching of CategoriesDisplay.

## 8.3.4 ProductDisplay

The ProductDisplay URL displays a catalog entry, which consists either of a single item or of all the items contained within a product, package, or bundle. Figure 8-17 shows the URL structure and parameter values.

Archived

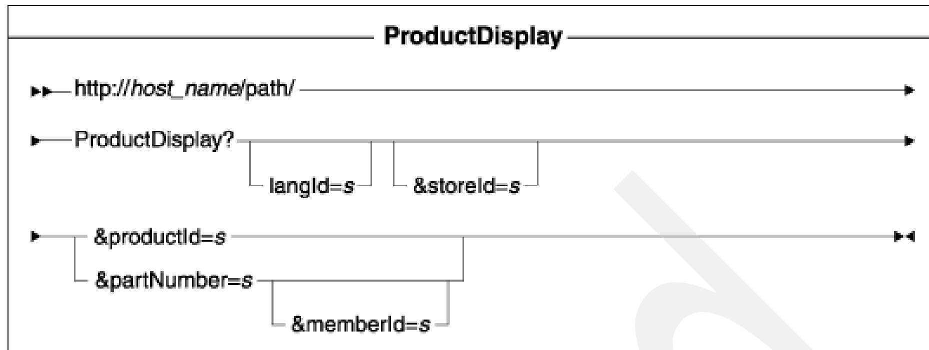


*Figure 8-17 ProductDisplay URL structure*

Table 8-6 describes the parameter values that are used by the ProductDisplay URL.

*Table 8-6 ProductDisplay parameter values*

| Parameter names | Description |
|---|---|
| langId | Sets or resets the preferred language for the duration of the session. The supported languages for a store are found in the STORELANG table. |
| storeId | The store's reference number associated with the catalog entry to be displayed. |
| productId | (Required) The reference number for the catalog entry (item, product, package, or bundle) that is to be displayed. You must specify either productId or partNumber. |
| partNumber | (Required) The store's unique identifier (or code) for the catalog entry that is to be displayed. You must specify either partNumber or productId. |
| memberId | The reference number of the member who owns the catalog entry. The memberId, along with the partNumber, uniquely identifies the catalog entry. If the memberId is omitted, then the owner of the current store and the partNumber are used to uniquely identify the catalog entry. |

By default ConsumerDirect uses the following parameters:

- StoreId
- ProductId
- SummaryOnly (ConsumerDirect store-specific parameter)

Based on Table 8-6, Example 8-9 shows how to add the cache-id for CategoryDisplay.

*Example 8-9 ProductDisplay cache-id*

```
<cache-id>

 <component id="" type="pathinfo">
  <required>true</required>
  <value>/ProductDisplay</value>
 </component>

 <component id="storeId" type="parameter">
  <required>true</required>
 </component>

 <component id="productId" type="parameter">
  <required>true</required>
 </component>

 <component id="summaryOnly" type="parameter">
  <required>false</required>
 </component>

</cache-id>
```

Perform the following steps to verify that the cachespec.xml file is loaded correctly by verifying the cache policy in Cache Monitor.

1. Verify cachespec.xml is reloaded.

   a. Point your Web browser to the Cache Monitor:

      `http://host_name/cachemonitor`

   b. Navigate to **Cache Policies** → /webapp/wcs/stores/com.ibm.commerce.struts.ECActionServlet.class

   Figure 8-18 shows the ProductDisplay cache policy that you defined in Example 8-9.

Archived

| Cache ID Rule 2 | timeout: 0 | priority: 0 | inactivity: 0 | idGenerator: null | metaDataGenerator: null | properties: | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Component** | **ID** | **Type** | **Ignore Value** | **Method** | **Field** | **Required** | **Values** | **Not-Values** | **Index** | **Multiple IDs** |
| Component 0 | | pathinfo | false | null | null | true | /ProductDisplay | | | false |
| Component 1 | storeId | parameter | false | null | null | true | | | | false |
| Component 2 | productId | parameter | false | null | null | true | | | | false |
| Component 3 | summaryOnly | parameter | false | null | null | false | | | | false |

*Figure 8-18 ProductDisplay Cache Policy*

2. Verify the CategoryDisplay URL is being cached.

   a. Navigate in the ConsumerDirect store to any of the ProductDisplay pages. For example, Figure 8-19 shows the results of selecting **Home Page** → **FURNITURE** → **Lounge Chairs**.

*Figure 8-19 ProductDisplay*

b. Using Cache Monitor, navigate to **Cache Contents** → .../ECActionServlet.class Template → ...:pathinfo=/ProductDisplay Cache ID

Notice that all the fragments that must be separately cached are indeed already not consumed. The rest of the contents of the ProductDisplay page is consumed with the full page cache.

9

# Benchmarking DynaCache

In this chapter, we discuss benchmarking DynaCache. Any Web application has its target uses and users. Web application performance (how fast a page is returned) and capability (how many concurrent users the system can support) are key issues in ensuring the users are left satisfied with the service and the Web-based business can attain the desired profits. Unlike Web application functionality, however, performance and capability are concepts that are inherently vague and relative. The purpose of a benchmark, then, is to make their discussion more concrete and to provide a point of reference for their measurement.

This chapter contains the following:

1. A brief discussion of the general benchmarking methodology and key considerations for benchmarking Web applications and WebSphere Commerce with DynaCache specifically
2. A discussion of the benchmark-creation testing process
3. A discussion of the design, execution, and results of a sample benchmark-creation test for a WebSphere Commerce system with DynaCache

© Copyright IBM Corp. 2006. All rights reserved.

## 9.1 Overview

A benchmark is a set of performance test results used for comparison purposes. A good and reliable benchmark serves as a reference point for improvement or degradation measurement when a Web application undergoes any changes.

### 9.1.1 Benchmarking benefits

Benchmarking plays an important role in evaluating a Web application. It provides valuable information, explicitly or implicitly, for a Web-based business both before and throughout its operation.

- A benchmark provides a quantitative expression of business requirements.

  Before building a Web application, any business organization will have certain types of requirements, such as the cost and profit associated with building and maintaining the application, acceptable time for a page to be returned to the user, the number of users the system can handle concurrently, and so on. These requirements are both relative and interdependent. A benchmark can provide a clear cross-product view of the requirements by means of numbers, tables, and figures.

- A benchmark reliably predicts the performance of a production system under stress and after long-term operation.

  A typical Web application can be accessed by end-users without limits on location, time, or duration. As a result, it is bound to experience periods of peak demand and operate under stress. A typical Web application may also go several years without major upgrades. How will the system behave under stress, and what would be the best possible time for a system upgrade? Measuring the difference in resource usage between the current state of the system and the benchmark can help answer these questions.

- A benchmark may help identify performance issues.

  Any Web application will undergo some changes, such as patches, layout modifications, or functional enhancements in its lifetime. All such changes are possible candidates for performance degradation. Before finalizing a change to the production site, it is highly beneficial to run a suite of tests to identify potential performance issues. A benchmark can serve as an excellent reference point for those tests.

### 9.1.2 Benchmarking considerations

Although a benchmark, as a reference point, is merely a set of test results, it can reflect many aspects of the Web application, such as cost, profit, user

Archived

satisfaction, and so forth. Furthermore, it can be used for both current evaluation and for prediction.

To define and to obtain a benchmark, we need to carefully investigate what our target criteria are. Then, by investigating the workload characteristics of the Web site, and comparing the test configuration with the production environment, we can understand how the test results will reflect the behavior of a real production site.

### Target criteria

The target criteria are the business scenarios that are given to the benchmarking team for planning benchmarking tests. Here are three typical types of target criteria.

- Mapping business requirements to the benchmark

  The benchmark should reflect the business requirements and provide enough information to ascertain that the Web application can be used to fulfill the business requirements. The benchmark should be selected based on the type of Web application under consideration: an online store may worry most about the number of orders the application can handle, while an online download site may worry most about the download speed.

- Predicting workload for special events

  Every year, e-Commerce sites see seasonal peaks of shopping activity, Christmas gift sales or back-to-school supply sales being familiar examples. What the possible workload will be during such special events and how the system can handle it need to be considered early.

- Estimating long-term system status

  A Web application can be expected to operate for several years. We would like have a rough idea about how many new users will be registered, how many new products will be added, how many orders will be placed, and so on, over the course of its operation. This kind of information should be considered during benchmarking setup.

### Workload characteristics

It goes without saying that a benchmark test is only as good as its input. It is very difficult to predict Web site traffic and weight the importance of different kinds of traffic. It is a good idea to break down the workload into different categories so that the results can be analyzed.

- Choose a good mix of test cases

  Good test cases reflect the typical workload of a site. Selecting the most frequent and the most important user activities is key. Because the benchmark will be used as a reference point and is unlikely to be recreated,

Archived

we need to make sure that it is comparable with other performance-type tests and contains enough information for comparison purposes.

- Account for error conditions

  Benchmarking setup should also include considerations for error conditions, such as a wrong password or wrong credit card information supplied by the user since a typical system behaves differently upon encountering these. Your own experience should convince you that handling invalid passwords and credit card information is nearly as common as entering the correct information first time.

- Test for system failures

  We must always have a plan for partial or total system failure. How long does the system need to resume normal operation? What kind of information will be lost when the system fails? If, as a preventative measure, we were to perform periodic system backups, how long will they take and will they fit into the desired maintenance window?

### Test environment versus production environment

After a Web site goes live, using it as a test environment is very dangerous. Therefore, common practice is to keep a separate test environment. The best possible scenario is when the test environment and production environment are identical. However, due to cost and maintenance considerations, that is rarely the case; the test environment is usually both smaller and simpler.

In most cases, we use the test environment both to create benchmarks and to run any further tests. We then map the results onto the production environment. The following information about both test and production environments is useful for establishing this mapping:

- Resources versus capacity

  CPU, memory, and hard drive characteristics are key for the capacity of a system. Doubling CPU or memory, however, does not yield twice the capacity. To get an accurate mapping, we must carefully monitor the test environment resource usage during test runs. Generally speaking, if you decide to make the test environment, say, a fifth the size of the production environment, then the components should be scaled to a fifth the size too. Memory may need to be the same as in the production environment. For example, a test database server would need to have the same memory as production because you will be testing with a full size database. The same applies to application servers and the JVM size.

- Capacity, workload, and response time

  The number of concurrent users and the response time are two inseparable measurements. Higher capacity can handle more concurrent users. More



Archived

concurrent users mean longer response times. Different Web applications will have different relationships between capacity, workload, and response time. We should consider this relationship when mapping the benchmarking results from the test environment to the production environment.

## 9.1.3 Benchmarking DynaCache

Since this book focuses on the use of DynaCache with WebSphere Commerce v.6.0, we next mention several points specific to benchmarking DynaCache.

### Obtaining the baseline results and measuring improvements

The test of system performance without using DynaCache yields the baseline results. Getting the results with DynaCache may require several parameter-tuning runs. Together, these two sets of results are used for direct comparison between system performance without and with DynaCache. We typically will also want to run additional tests on the system with DynaCache to investigate system scalability, saturation point, and so forth. All of these results belong in the benchmark.

### Identifying possible impacts of DynaCache

In general, DynaCache will improve system performance, but it might also have side effects:

- JVM heap and fragmentation

  DynaCache uses JVM heap to store cached objects. A large number of cached objects occupy a large portion of the JVM heap and contribute to heap fragmentation. During benchmark-creation tests, we need to monitor the JVM usage in relation to the number of DynaCache entries, adjusting the number of entries as necessary.

- DynaCache overhead

  Cached object search and invalidation use resources. If the cache hit ratio is too low or the cache refresh rate is too high, the resulting overhead might be significant. It is important to keep the hit ratio high by adjusting the number of DynaCache entries as well as the number of cacheable objects.

- Time required to stop the WebSphere Commerce server with the “Flush to disk” option enabled

  DynaCache contents can be flushed to a file on a disk when the WebSphere Commerce server is shut down. If the cache size is big, the shutdown may take a long time. On the other hand, if we disable the “Flush to disk” option, the cached objects will be lost and will need to be regenerated and cached after the WebSphere Commerce server is restarted. Several experiments may be necessary to establish the optimal strategy for your site.

### Tuning DynaCache parameters

Several DynaCache basic parameters are outlined in this section. Despite being relatively simple, they have a significant effect on DynaCache performance and, therefore, need to be considered as part of the benchmark-creation testing activities.

- Number of cache entries

  Increasing the number of cache entries can optimize performance by increasing the hit ratio and reducing the movement of cached objects. There is no universal rule for defining this number, but the number of cacheable pages in the site, the JVM heap size, typical user activities, and the average cacheable object size are all among the aspects to consider.

- Invalidation policy

  Because, after a while, it becomes impossible to keep all the cached objects in memory, an invalidation policy must be defined. The choice of invalidation policy has significant effect on DynaCache performance and has to be considered carefully. Refer to Chapter 3, “DynaCache invalidation” on page 91 for details.

- Disk offload and the size of offload files

  With the “Enable disk offload” option turned on, if there is no space for a new object, an existing cached object is pushed to disk according to the offload policy in effect. Depending on the policy and the number of cached objects, the size of the offload files may become very large, negatively impacting the overall system performance. Consequently, we should investigate the utility of enabling disk offload as part of our benchmark-creation testing activities.

## 9.2 Benchmark creation process

How do we a create a benchmark? Test environment, test results, and test result analysis are the key components of the benchmark-creation process, and we consider them next.

### 9.2.1 Setting up benchmark-creation tests

To generate a benchmark we need to set up a test environment, select a tool to be used to simulate Web application users, and develop the test scenarios.

### Environment

The test environment for benchmarking should be as similar to the production environment as possible. At a minimum, it should contain all the major production

Archived

environment components with similar layout. Usually, the test environment is a scaled-down version of the production site.

## Tools

A number of load-testing tools can be used for Web application testing. Several considerations go into selecting a load-testing tool; for instance:

- Can the tool provide all the functionality necessary for testing?
- What are the tool's hardware requirements, particularly for generating large volumes of workload or simulating a large number of concurrent users?
- Does the test report produced by the tool contain enough detail, and is the format convenient?
- How easily can the testing scripts be created or modified?

## Test scenarios for benchmark-creation testing

The scenarios for benchmark-creation testing should cover the most common and typical user behavior. The workload characteristics should be similar to those in the actual site usage. For e-Commerce applications, some of the characteristics to be considered are as follows:

- Buy versus browse ratio
- Average size of shopping cart
- Guest versus registered user ratio

If the production site already exists you should use the web-logs to define the workload characteristics.

Benchmark-creation tests are not the same as system tests or performance tests.

- Benchmark-creation tests build up a set of reference points and are not intended for finding defects or solving performance issues. However, they *can* be combined with other types of tests.
- Functional coverage of benchmark-creation tests is typically much smaller than that of system or performance tests, including only the most prominent elements of a site's daily workload.
- Because of their limited coverage, benchmark-creation tests rarely include special-purpose features, such as discounts, promotions, or marketing experimentation: some features may not be always active in the site, others do not have a significant impact on the performance. If necessary, we can create separate benchmarks for such features.

Archived

## 9.2.2 Executing tests and recording results

We must have a clear idea of the metrics and conditions for the benchmark-creation tests and know what kind of information to record for future reference. Rerunning a test might be very expensive – and sometimes impossible (for instance, due to lack of unique data generated at runtime).

### Performance metrics

Performance measurements come in two flavors:

- Activity metrics

  Activity metrics reflect the system's reaction to workload and can be used to measure business requirement fulfillment and customer satisfaction. Examples of such metrics include:

  - Number of Web site hits
  - Page response times
  - Number of orders placed
  - Volume of data transferred

- Resource metrics

  Resource metrics reflect system capacity and are important for predicting system behavior under stress or after long-term operation. Examples of such metrics include:

  - CPU usage
  - RAM usage
  - JVM heap usage
  - Hard disk usage

### Execution conditions

By execution conditions we mean understanding and controlling the execution of benchmarking tests so that we know the initial conditions of the test and can repeat the tests. Here are two sets of best practices to help you manage test execution.

#### *Test repeatability*

- To ensure that a test can be repeated, we should back up the environment prior to the test run. Database and configuration are the two most important parts of the environment.
- We should run comparison-type tests (for example, WebSphere Commerce with and without DynaCache) in the same environment.

### *Constancy of workload*

- We should keep the number of concurrent users constant across comparison-type tests to ensure the workload remains the same.
- Think time between two hits has a significant impact on workload: longer think time means lower workload and vice versa. To ensure the workload remains the same, we should keep think time constant across comparison-type tests as well.
- Whether DynaCache has been preloaded, JSP pages precompiled, and the database warmed up also impact the actual workload on the system. In benchmark-creation tests with DynaCache, we should make sure DynaCache is fully loaded and stabilized, JSP pages precompiled, and the database warmed up. Typically, allow half an hour warm up and a one hour run.
- Make two or three runs to confirm the consistency of the results.

### Information to be recorded

The baseline result forms the basis of a benchmark. Usually, we also run additional tests, characterized by changes to workload, configuration, features, or hardware. These additional tests are all part of the benchmark. For example:

- The size of workload can be used to find the saturation point and study how the system behaves under stress. The type of workload can reflect the business model, such as B2B versus B2C.
- We should test any newly-introduced as well as any special-purpose features if we expect them to impact performance or if their impacts are unknown.
- The optimal values of DynaCache parameters depend on a number of aspects, including workload size and type, hardware, and software. Changes to configuration usually require a lot of experimentation. We should keep thorough records of all the changes introduced in each test run in order to get the best tuning.
- Running tests with different hardware configurations (for example, a different number of CPUs or amount of memory) provides important insight into system scalability.
- System performance is bound to change after long-term operation, due, in no small part, to changes to the database and logs. Running a test with a larger database and logs will give us an idea of system performance after extended periods of operation.

## 9.2.3 Interpreting and analyzing the test results

The test results that we obtain may not give us the desired information directly. Therefore, interpreting and analyzing them is a necessary and important part of

Archived

the benchmark-creation work. This phase is also crucial for making corrections to the initial assumptions, setup, and so forth, should the results prove contrary to our expectations. It is very important to review the application logs to make sure there are no unexpected errors. If the test involves a database, you should also review the database for deadlocks and any other database issues.

# 9.3 Benchmarking example

There are no unique rules for creating a benchmark. In the preceding sections, we have discussed some general benchmarking process considerations. In this section, we use a WebSphere Commerce version 6.0 system with DynaCache to illustrate the process. In addition, our results demonstrate clear performance benefits that DynaCache brings.

## 9.3.1 Test environment

Our choice of test environment, like yours, was constrained by the hardware that was available to us. We would have liked to run the tests on P5 servers using a SAN, and stored the database on disks with non-volatile cache.

The hardware we used is, however, a typical component of many small to medium sized Web sites today. The results show for the non-cached benchmark we were limited by the speed of the processors and not the disks; however, when caching is used processing power is no longer the gating factor, and more performance analysis is required to see how we can keep response times constant for an increasing numbers of users.

### Topology

Most WebSphere Commerce customers use a multi-tier cluster environment. The environment is easy to maintain and easy to scale by adding more WebSphere Commerce server machines as the business expands. In our benchmarking example, we used a 3-tier environment with two clustered servers, as Figure 9-1 on page 219 illustrates.

Archived

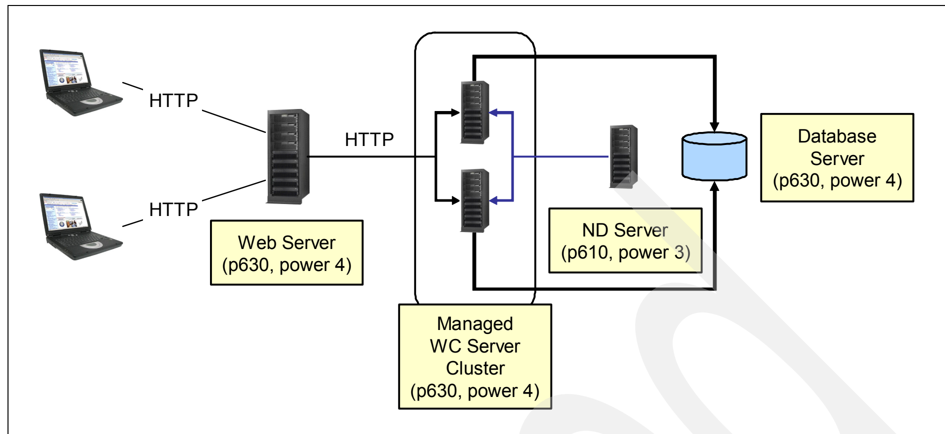



*Figure 9-1 Topology of the benchmarking environment*

## Stack configuration

As Table 9-1 shows, we are using AIX® machines at each node of our test environment. The software stack follows the requirements outlined in the WebSphere Commerce version 6.0 installation guide available at:

http://www.elink.ibmlink.ibm.com/public/applications/publications/cgibin/pbi.cgi?CTY=US&FNC=SRX&PBL=GC104259

*Table 9-1 Stack configuration*

| Node | Hardware | Software |
|---|---|---|
| Database | p630<br>CPU: 4x1.45GHz Power 4<br>Mem: 16 GB | AIX5.3 ML01<br>DB2 8.2.3 |
| Commerce(1) | p630<br>CPU: 4x1.45GHz Power 4<br>Mem: 16 GB | AIX5.3 ML01<br>DB2 8.2.3 Client<br>WAS 6.0.2.5 |
| Commerce(2) | p630<br>CPU: 4x1.45GHz Power 4<br>Mem: 16 GB | AIX5.3 ML01<br>DB2 8.2.3 Client<br>WAS 6.0.2.5 |
| Web Server | p630<br>CPU: 4x1.45GHz Power 4<br>Mem: 16 GB | AIX5.3 ML01<br>IHS 6.0.2<br>WAS Plugin 6.0.2.5 |
| WAS ND Server | p610<br>CPU: 1x450MHz Power 3<br>Mem: 8GB | AIX5.3 ML01<br>WAS ND 6.0.2.5 |



Archived

## 9.3.2 Test data set and scenario

The test data set and scenario represent a medium sized Web site with 50,000 registered customers. This is typical of a specialized retailer.

### Data set

The size of the data set is an important consideration in WebSphere Commerce benchmarking. How big should the catalog be? How many registered users should there be in the system? Sometimes, we also need to consider other elements, such as the number of orders, campaigns, discounts, and so on. For our tests, the WebSphere Commerce database initially contained the following data[1]:

- Users
  - 50,000 registered users
  - 150,000 guest users
- Catalog: 500,000 entries
  - 10 top-level categories
  - 20 second-level categories per top-level category
  - 25 third-level categories per second-level category
  - 10 products per third-level category
    - 8 available (in-stock) items
    - 1 backorder (out-of-stock) item
    - 3 attributes (1 integer and 2 string attributes) per product
  - 1 top-level dynamic kit category
  - 10 second-level dynamic kit categories
  - 10 third-level dynamic kit categories per second-level dynamic kit category
  - 20 dynamic kits per third-level dynamic kit category
    - 20 available (in-stock) items per dynamic kit
- Orders
  - 170,000 completed orders
  - 20,000 pending orders
  - 10,000 canceled orders

### Test scenario

WebSphere Commerce implements several business models, such as B2C, B2B, and hosting, with user behavior differing widely by model.

In our tests, we use a B2C (consumer direct) store. To reduce the amount of preparatory work, we use a custom version of the WebSphere Commerce consumer direct starter store with the data described here.

[1] For detailed explanations of WebSphere Commerce terminology, please refer to the WebSphere Commerce v6.0 Information Center at
http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp

For a B2C e-commerce site, the typical workload is dominated by customer activities. To reflect that, we assumed the following workload distribution in our tests:

- Users:
  - Guest users: 60%
  - Existing registered users: 20%
  - New registered users: 20%
- Buy ratio:
  - Browse only: 85%
  - Browse and add to cart: 5%
  - Browse, add to cart, add billing/shipping information, view order summary: 5%
  - Browse, add to cart, add billing/shipping information, view order summary, add payment information, and complete order: 5%
- Errors on logon
  - Wrong logon ID or password for an existing registered user: 10%

Figure 9-2 on page 222 depicts the overall scenario flow.

Archived

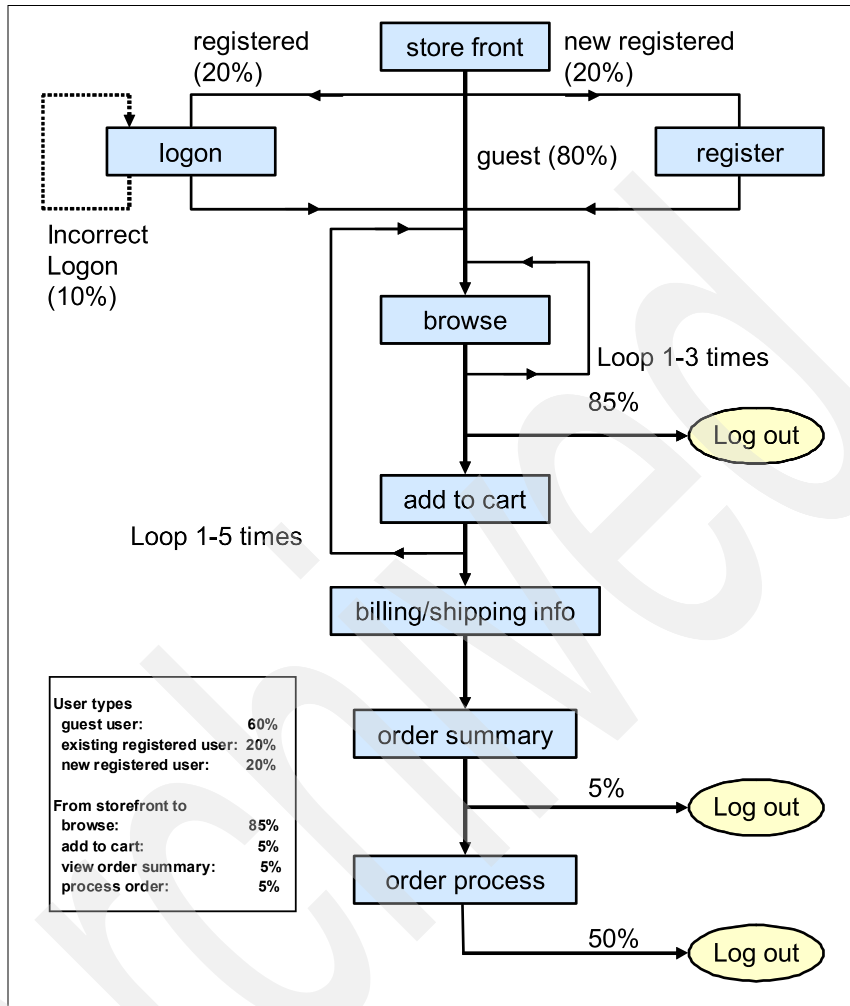



*Figure 9-2 Scenario flow diagram*

As mentioned previously, think time is a particularly important parameter for the workload. The best way to assess the characteristics of think time is to analyze an existing Web activity log. As a general rule, however, end users of a Web application tend to spend large amounts of time between hits. The longer the think time, the more concurrent users it takes to create the same workload. In the context of benchmarking, this means that, to get the same amount of workload for the benchmark, we require more machines that simulate end users, because the maximum number of concurrent users that a single machine can simulate is limited. Conversely, for our test purposes, we can reduce the think time in order

to reduce the number of concurrent users and the number of machines required to simulate them.

Table 9-2 lists the think times that we used in our tests.

*Table 9-2 think times in the tests*

| Web action | Think time (second) | Web action | Think time (second) |
|---|---|---|---|
| Select shipping address | 5 | View order summary | 10 |
| Add shipping address | 10 | Add payment info | 10 |
| Display category | 3 | Display product | 3 |
| Log off | 2 | View shipping method | 5 |
| Log on | 5 | Display top category | 5 |
| Go to log on page | 0 | Visit store front | 0 |
| Add item to cart | 5 | Register user | 10 |
| View billing address | 5 | Go to register page | 0 |
| Process order | 5 | | |

## 9.3.3 Execution and results

As stated at the opening of the Benchmarking example section, the purpose of our tests is to create a benchmark for a WebSphere Commerce system with DynaCache, as well as to show performance improvements that DynaCache brings. As discussed in “Obtaining the baseline results and measuring improvements” on page 213, we therefore need a baseline test without DynaCache and a test with the finalized parameter set, reflected by the cachespec.xml descriptor, to serve as the new DynaCache baseline for any possible changes at a future time. Sometimes, we also require additional tests, for example, to confirm that we get performance benefit from such changes or to investigate system behavior under a different set of circumstances.

We ran four tests in preparation of this book:

- 100 concurrent users without DynaCache
- 100 concurrent users with caching the product display page only
- 100 concurrent users with the cachespec.xml file provided as part of the WebSphere Commerce consumer direct starter store

- 200 concurrent users with the cachespec.xml file provided as part of the WebSphere Commerce consumer direct starter store

We ran each test for 3 hours, using the JVM heap size of 1GB and identical scenarios, ratios, and think times (see “Test data set and scenario” on page 220).

## 100 concurrent users without DynaCache

Table 9-3 shows the resource usage for the baseline test without DynaCache.

*Table 9-3 Resource usage in the test with 100 users without DynaCache*

| | CPU usage | Minimum JVM heap usage | Hard disk usage increase |
|---|---|---|---|
| Commerce server 1 | 46.27% | 300MB | |
| Commerce server 2 | 47.39% | 300MB | |
| Database server | **100%** | | 23MB |
| Web server | 2.72% | | |

As we can see, the database server has reached its capacity with 100% CPU usage. Since, according to the test scenario, *browse only* accounts for 85% of the site activity, the database update volume should be very low, and the bulk of database activity should come from retrieval. To narrow down the most common or expensive types of retrieval, we next look at the response times per activity.

*Table 9-4 Response times in the test with 100 users without DynaCache*

| Web action | Average response time (second) | Count | Total time (second) |
|---|---|---|---|
| Overall | 6.524 | 35,459 | 231334.516 |
| Select shipping address | 0.465 | 447 | 207.855 |
| Add shipping address | 0.301 | 321 | 96.621 |
| Display category | 2.277 | 16,137 | 36743.949 |
| Log off | 0.29 | 795 | 230.55 |
| Log on | 0.508 | 473 | 240.284 |
| Go to log on page | 0.376 | 473 | 177.848 |
| Add item to cart | 1.235 | 1,078 | 1331.33 |

| Web action | Average response time (second) | Count | Total time (second) |
|---|---|---|---|
| View billing address | 1.14 | 865 | 986.1 |
| Process order | 1.368 | 92 | 125.856 |
| View order summary | 2.219 | 192 | 426.048 |
| Add payment info | 1.314 | 92 | 120.888 |
| Display product | **35.338** | 5,305 | 187468.09 |
| View shipping method | 0.618 | 863 | 533.334 |
| Display top category | 0.335 | 5,388 | 1804.98 |
| Visit store front | 0.2 | 2,128 | 425.6 |
| Register user | 0.61 | 405 | 247.05 |
| Go to register page | 0.373 | 405 | 151.065 |

From Table 9-4 and Figure 9-3 on page 226, we can see that the response time for the product display page is huge (35.3 seconds) *and* the page is used 81% (=187468/231334x100%) of the total time. Without caching, the server has to retrieve product information from the database whenever a user browses the catalog. But product pages are user-independent, can be “shared” by users, and are likely to remain unchanged for days or even months. This makes the product display page an excellent first candidate for dynamic caching.

Archived

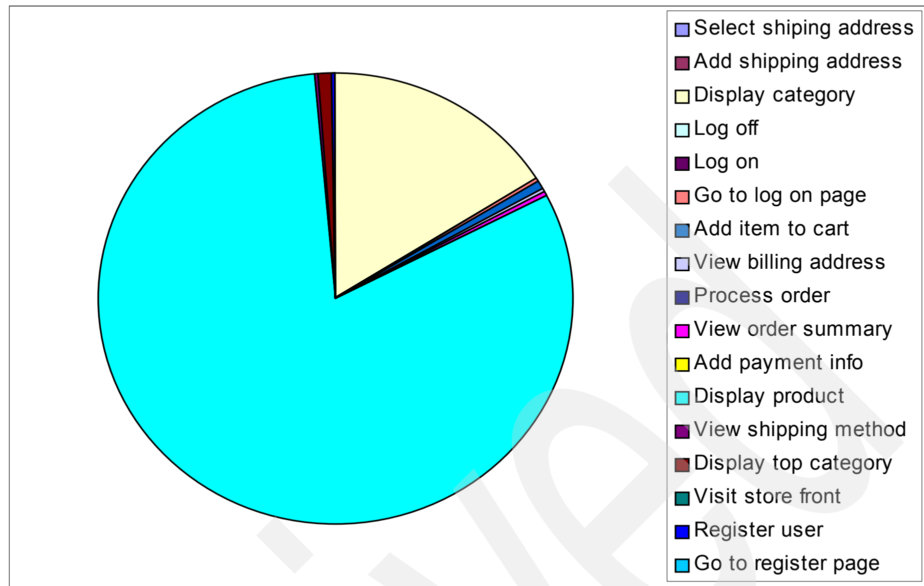



*Figure 9-3 Total time for URL actions in the100 users test without DynaCache*

### 100 concurrent users caching the product display page only

In this test, we enabled the DynaCache full page caching of the product display page. Since the page contains a header bar, footer bar, and a mini shopping cart, all realized as JSP fragments, we also had to enable fragment caching in cachespec.xml.

To get the benefit from product display page caching, we fully populated the cache prior to the formal test by running a browsing-only scenario and using the cache monitor to get information about the state of the cache.

Table 9-5 and Table 9-6, and Figure 9-4 on page 228 summarize the results of the test.

*Table 9-5 Resource usage in the test with 100 users and caching of product display page*

| | CPU usage | Minimum JVM heap usage | Hard disk usage increase |
|---|---|---|---|
| Commerce server 1 | 85.98% | 350MB | |
| Commerce server 2 | 86.13% | 350MB | |
| Database server | 36.12% | | 40MB |

| | CPU usage | Minimum JVM heap usage | Hard disk usage increase |
|---|---|---|---|
| Web server | 4.72% | | |

Comparing Table 9-5 on page 226 with Table 9-3 on page 224, we can observe that the CPU usage on the database server has dropped to 36.12% from 100%, while the CPU usage on both WebSphere Commerce servers has increased. The database server is thus no longer the bottleneck.

*Table 9-6 Response times in the test with 100 users and caching of product display page*

| Web action | Average response time (second) | Count | Total time (second) |
|---|---|---|---|
| Overall | 1.625 | 210,572 | 342179.5 |
| Select shipping address | 0.595 | 2,623 | 1560.685 |
| Add shipping address | 0.336 | 1,890 | 635.04 |
| Display category | **2.946** | 95,236 | 280565.256 |
| Log off | 0.34 | 4,924 | 1674.16 |
| Log on | 0.589 | 2,654 | 1563.206 |
| Go to log on page | 0.465 | 2,654 | 1234.11 |
| Add item to cart | 1.694 | 6,514 | 11034.716 |
| View billing address | 1.558 | 5,380 | 8382.04 |
| Process order | 2.022 | 595 | 1203.09 |
| View order summary | 3.404 | 1,179 | 4013.316 |
| Add payment info | 1.795 | 595 | 1068.025 |
| Display product | **0.135** | 31,714 | 4281.39 |
| View shipping method | 0.807 | 5,376 | 4338.432 |
| Display top category | 0.484 | 31,769 | 15376.196 |
| Visit store front | 0.182 | 12,377 | 2252.614 |
| Register user | 0.758 | 2,546 | 1929.868 |
| Go to register page | 0.43 | 2,546 | 1094.78 |

Archived

As we can see from Table 9-6, the response time for the product display page is now only 0.135 seconds, down considerably from 35.338. Because of this dramatic drop, the total number of pages visited in 3 hours has increased from 35,459 to 210,572, marking a six-fold increase in throughput. This is typical of many implementations of DynaCache.

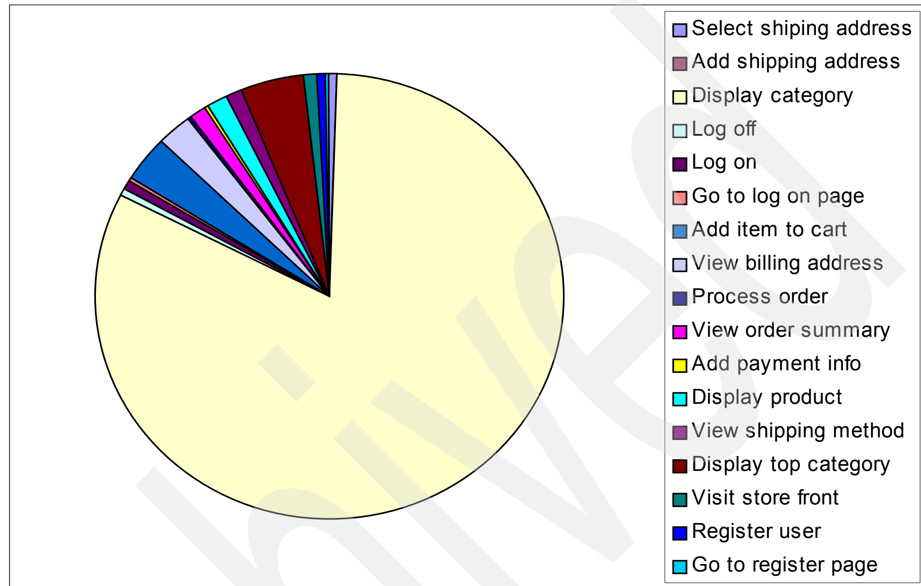



*Figure 9-4 Total time for URL actions in the test with 100 users and caching of product display page*

## 100 concurrent users using cachespec.xml

In the previous test, we saw a considerable performance improvement brought on by DynaCache. We also saw that the category display page now consumes most of the total time, and the CPU usage on both WebSphere Commerce servers is up. Dynamically caching other pages should help us improve response time and reduce resource usage. In this next test, we use the sample cachespec.xml file provided with the WebSphere Commerce consumer direct starter store, which enables caching of a number of other pages (including the category display page) as well as WebSphere Commerce commands.

Table 9-7 and Table 9-8 on page 229, and Figure 9-5 on page 230 summarize the results of the test.

*Table 9-7 Resource usage in the test with 100 users and sample cachespec.xml*

| | CPU usage | Minimum JVM heap usage | Hard disk usage increase |
|---|---|---|---|
| Commerce server 1 | 18.74% | 450MB | |
| Commerce server 2 | 18.01% | 420MB | |
| Database server | 8.17% | | 58MB |
| Web server | 6.36% | | |

*Table 9-8 Response times in the test with 100 users test and sample cachespec.xml*

| Web action | Average response time (second) | Count | Total time (second) |
|---|---|---|---|
| Overall | 0.238 | 289,004 | 68782.952 |
| Select shipping address | 0.385 | 3,596 | 1384.46 |
| Add shipping address | 0.276 | 2,609 | 720.084 |
| Display category | **0.197** | 130,859 | 25779.223 |
| Log off | 0.292 | 6,644 | 1940.048 |
| Log on | 0.355 | 3,683 | 1307.465 |
| Go to log on page | 0.305 | 3,683 | 1123.315 |
| Add item to cart | 0.82 | 9,097 | 7459.54 |
| View billing address | 0.787 | 7,381 | 5808.847 |
| Process order | 0.86 | 810 | 696.6 |
| View order summary | 1.21 | 1,646 | 1991.66 |
| Add payment info | 0.784 | 811 | 635.824 |
| Display product | 0.129 | 43,591 | 5623.239 |
| View shipping method | 0.467 | 7,372 | 3442.724 |
| Display top category | 0.127 | 43,648 | 5543.296 |
| Visit store front | 0.157 | 16,864 | 2647.648 |
| Register user | 0.433 | 3,355 | 1452.715 |

Archived

| Web action | Average response time (second) | Count | Total time (second) |
|---|---|---|---|
| Go to register page | 0.332 | 3,355 | 1113.86 |

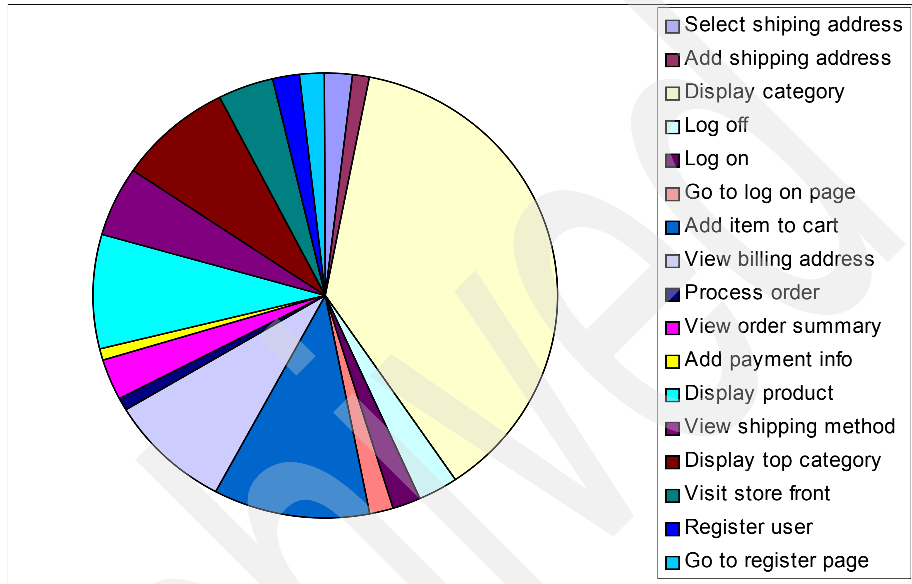



*Figure 9-5 Total time for URL actions fin test with 100 users and sample cachespec.xml*

We see a significant drop in CPU usage for both the WebSphere Commerce and database servers, as well as an increase in the total number of pages visited in 3 hours, from 210,572 to 289,004. We still see the category display page dominating the total time column of Table 9-8 on page 229. However, this results largely from the sheer number of hits it gets (~50% of total hits); the response time has gone down from 2.496 seconds to 0.197 seconds.

## 200 concurrent users using cachespec.xml

]The test with sample cachespec.xml shows that our test environment is very capable of handling 100 users: the page response times are low, as is the CPU usage on all the servers. We can therefore use the test results as the final benchmark of WebSphere Commerce with DynaCache.

What happens if we increase the number of concurrent users, however? Will the throughput (number of pages) and CPU usage increase linearly? What about the response time?

The final benchmark-creation test, which we tackle in this section, is an illustration of the kinds of additional tests that we would typically include in a benchmark suite, intended to investigate issues of performance under stress, scalability, and so on. Here, we double the number of concurrent users to see how the system reacts.

Table 9-9 and Table 9-10 and Figure 9-6 on page 232 summarize the results of the test.

*Table 9-9 Resource usage in the test with 200 users test and sample cachespec.xml*

| | **CPU usage** | **Minimum JVM heap usage** | **Hard disk usage increase** |
|---|---|---|---|
| Commerce server 1 | 27.99% | 450MB | |
| Commerce server 2 | 28.02% | 430MB | |
| Database server | 13.74% | | 101MB |
| Web server | 10.2% | | |

*Table 9-10 response times in the test with 200 users and sample cachespec.xml*

| **Web action** | **Average response time (second)** | **Count** | **Total time (second)** |
|---|---|---|---|
| Overall | 0.938 | 468,303 | 439268.214 |
| Select shipping address | 1.145 | 5,837 | 6683.365 |
| Add shipping address | 0.988 | 4,129 | 4079.452 |
| Display category | 1.004 | 211,692 | 212538.768 |
| Log off | 0.801 | 10,872 | 8708.472 |
| Log on | 0.858 | 6,044 | 5185.752 |
| Go to log on page | 0.797 | 6,044 | 4817.068 |
| Add item to cart | 1.666 | 14,876 | 24783.416 |
| View billing address | 1.749 | 12,118 | 21194.382 |
| Process order | 1.85 | 1,306 | 2416.1 |
| View order summary | 2.354 | 2,689 | 6329.906 |
| Add payment info | 1.839 | 1,306 | 2401.734 |

Archived

| Web action | Average response time (second) | Count | Total time (second) |
|---|---|---|---|
| Display product | 0.757 | 70,515 | 53379.855 |
| View shipping method | 1.328 | 12,107 | 16078.096 |
| Display top category | 0.655 | 70,617 | 46254.135 |
| Visit store front | 0.478 | 27,261 | 13030.758 |
| Register user | 0.845 | 5,445 | 4601.025 |
| Go to register page | 1.201 | 5,445 | 6539.445 |

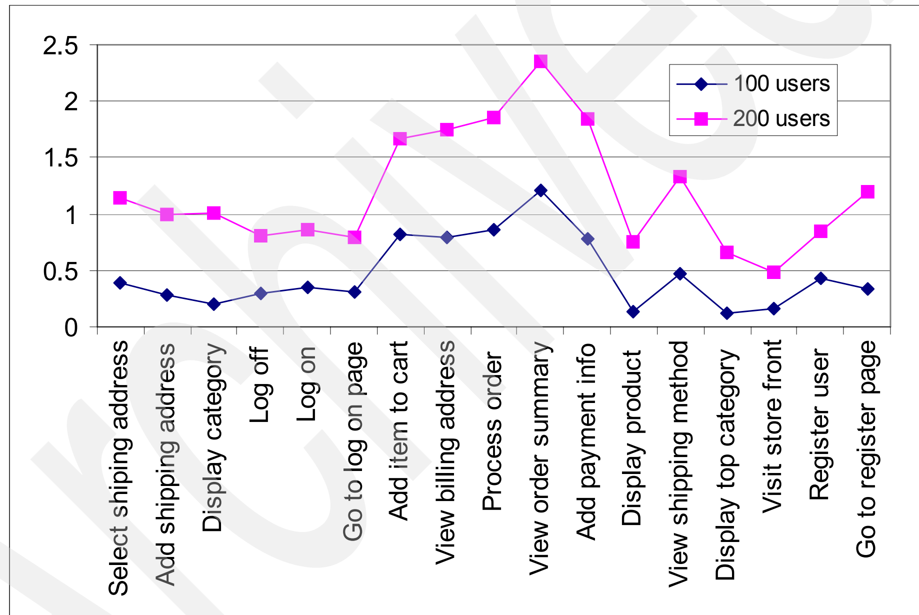



*Figure 9-6 Response times in the tests with 100 and 200 users and sample cachespec.xml*

The test results show that the throughput, response time, and CPU usage do not increase linearly. Doubling the number of concurrent users does not double the throughput, even though a considerable amount of unused resources remains.

As a next step, we would run more tests to determine the saturation point.



Archived

The results also confirm the common-sense observation that the response time is unlikely to stay unchanged when the number of concurrent users increases – but we need to take more measurements to determine why the response time is increasing.

## 9.4 Conclusion

In this chapter, we have discussed the general benchmarking methodology and key considerations for benchmarking Web applications and WebSphere Commerce with DynaCache specifically. Although there is no hard and fast rule for planning, designing, or obtaining a benchmark, we have attempted to provide enough guidance to make the task practicable. A benchmark is grounded in business requirements and can be used for both current and future, ongoing, evaluation of system performance. It is our sincere conviction that every site should have a benchmark.

In the final section of the chapter, we have also given an example of benchmarking WebSphere Commerce with DynaCache. As our tests have shown, the use of DynaCache in WebSphere Commerce systems is not merely an option, but an imperative. Well-defined and customized dynamic caching can improve system performance ten- or even a hundred-fold, helping us shorten response times, increase workload, get the most out of the hardware we have, and inform us where best to spend our hardware budget to improve performance or support more customers as the popularity of our Web site grows.

Archived

Archived

10

# Case study: A DynaCache anti-pattern

It is a truism to say we learn from our mistakes. This case study is based on a real Web site. We can learn from the experience of the IBM service team who helped to get the Web site on track again. Of course this would never happen to us. Or would it?

One of the interesting lessons to draw from this case study, whether or not you are involved in troubleshooting a problematic Web site, is how to use the many options to configure DynaCache. You have read the theory in the previous chapters. Now you can read about putting the theory into practice in a real-world scenario.

© Copyright IBM Corp. 2006. All rights reserved.

## 10.1 Online shop project brief

This case study follows the re-release of an online shop using WebSphere Commerce. The subject company hired another company to create a customized WebSphere Commerce application, and consulted with IBM on machine sizing and configuration. The online shop application was developed, integrated with the back-end systems, and tested. Only then was performance considered.

Once stress and load testing commenced the company become aware that performance was well short of expectations. On minimal user load, the quickest page was over 20 seconds at the 95 percentile (that is, 5% of the pages took over 20 seconds). The slowest pages were well over 90 seconds.

It was also determined that the WebSphere Commerce application was using 100% of the CPU, even though very little content was being delivered.

By the time user testing was completed, the team customizing the WebSphere Commerce application had all but disbanded. This severely limited the options available to adjust the application.

## 10.2 Issues encountered

During the performance tuning process on the online shop, a number of issues were encountered. The only options available for performance tuning were environmental setup. WebSphere Commerce application modification was not possible because most of the application developers were unavailable, but without significant gains in performance the online shop project was at risk of being cancelled.

In this section we discuss the issues encountered in the order they were discovered and resolved. The following issues were handled:

1. DynaCache not enabled
2. Inability to cache page fragments
3. Cache invalidations causing severe performance impacts
4. Cached page size greater than 200Kb
5. Large numbers of duplicated similar cache areas

### 10.2.1 DynaCache not enabled

The first issue we identified was that DynaCache had not been enabled.

Archived

Although this seems obvious in hindsight, it was not considered by the application development team because they were focused on function, not performance. Since this was the first WebSphere application to be deployed for the company that had response time issues, the integration team had no prior knowledge of DynaCache.

Just enabling DynaCache in it's “out of the box” configuration did not yield any significant gains in page performance. The cachespec.xml needed customization.

### 10.2.2 Inability to cache page fragments

Initial attempts at configuring cachespec.xml were unsuccessful. Caching fragments caused the application to fail in unexpected ways. For fragment caching to be successful, the fragments must be independent. The design of the application was that the page fragments were dependant on the parent page for information. For example, the page header displayed a user name which was derived from page headers and the navigation menus were customized based on the identified user profile.

Once full page caching (com.ibm.commerce.server.RequestServlet.class) was enabled, performance improvements were realized.

Having enabled full page caching, some fragments (such as the mini-shopping cart) had to be un-cached. The version of WebSphere Application Server being used did not support the “do not cache” option on consumed fragments. Consequently, these fragments needed to be invalidated using a short timeout (1 second). This workaround is scheduled to be eliminated in the next version of the online shop using a later release of WebSphere Commerce.

To set up caching to work with WebSphere Commerce we needed to determine the parameters being passed to WebSphere Commerce. We set all parameters passed to WebSphere Commerce in NonEncryptedParameters and analyzed the logs of all Web traffic generated during testing, determining the parameters from the URL request lines. We also enabled DoubleClickMonitoredCommands and excluded the full page JSPs being cached. Both of these segments are modified within the <instance>.xml file.

Full page caching provided significant enough gains in performance to allow the online shop to be implemented in production. Statistics gained from the test runs now indicated the slowest pages were 10 seconds at the 95 percentile (that is, 5% of the pages taking over 10 seconds). Previously, the same pages were over 90 seconds at the 95 percentile. Although this was over a 900% gain in performance, it was only just inside the acceptable range and further improvements would need to be made in the near future.

Archived

## 10.2.3 Cache invalidations causing severe performance impacts

Cache invalidation was not carefully considered before the shop was opened to the public. Initially the online shop cache pages were set to never expire. This had great performance, but content updates became a problem. We found there was no way to identify the cache pages invalidated by database updates. Consequently, a method was developed to invalidate the entire cache when data propagation from the staging instance to the production database was completed.

Emptying the cache caused an immediate and catastrophic performance impact. As mentioned before, the online shop was open to the public when this was found.

Because there was no identifiable correlation between cache pages and database updates, and we could not invalidate the entire cache in one operation, the only remaining solution was to invalidate pages after an elapsed time period.

The cache invalidation policy was changed to invalidate pages 8 hours after page generation. This was a less than ideal policy, but it was the best compromise available at the time. The business unit responsible for owning and running the online shop was made aware that it could be as much as 8 hours from product update to cached page update. This meant old data could still be on display to real customers.

A proposal to build an application to warm up the cache was rejected for three reasons:

1. No accurate traffic pattern had emerged to show the pages to pre-warm the cache with.
2. The inefficiency of the WebSphere Commerce application code ensured a warm-up operation would cause an operational outage to the online shop.
3. During a programmatic warm-up, the online shop would perform so poorly it appeared to users to be unavailable.

## 10.2.4 Cached page sizes greater than 200Kb

It was discovered during testing that pages constructed by WebSphere Commerce were large and filled with white space. To avoid poor performance, pages are compressed by the Web server prior to transmission to the user's browser.

Unfortunately, the uncompressed pages are stored by DynaCache in the JVM. With up to 3000 cached pages stored, this could take 600Mb of heap memory.

Archived

Care had to be taken to ensure the maximum heap memory could accommodate this amount of storage.

The initial heap size of the online shop was too small and the JVM would regularly run out of memory during peak loads or abnormal circumstances.

To avoid memory problems, the number of in-memory entries in the cache was reduced to 2000 to provide an appropriate memory buffer. Any entries over 2000 are stored in the disk cache. Only the least recently used cache entries were stored to disk. It was also found to be quicker to restore a cached page from disk than to re-render its contents.

Since making this change, the online shop has run continuously for a month without any restarts or failures.

### 10.2.5 Large numbers of duplicated similar cache areas

On inspection of the cache contents, we found a large number of what appeared to be duplicated pages. The duplicated pages were found to be the same page called with slightly different parameters.

Because we are caching full pages, we are unable to reduce the number of calling parameters. The solution to this problem is to redesign the pages into cacheable fragments and call and cache the fragments with a minimum number of parameters.

## 10.3 Lessons learned from the exercise

The following are key findings from retro-fitting DynaCache onto an existing WebSphere Commerce application. The points will be discussed in detail in the paragraphs that follow.

1. Include DynaCache in the initial design of WebSphere Commerce applications.
2. The best constructed cachespec.xml will not compensate for poor application design.
3. Simulate Web site traffic using a workload as close to production traffic as possible.
4. Invalidate as little as possible.
5. Warm up your cache offline.

Archived

### 10.3.1 Include DynaCache in the design of applications

When designing a WebSphere Commerce application, to ensure performance, you must incorporate DynaCache into the initial design. Specific techniques to apply to your design include the following:

- Design the pages in independent cacheable fragments.
- Ensure the fragments are of reasonable size.
- Consider cached fragment invalidation.
- Ensure you invalidate as little of the cache as possible with each data change.
- Avoid cache entries requiring timeouts.

### 10.3.2 Retrofitting DynaCache will only be a limited success

Retrofitting a WebSphere Commerce application with a DynaCache will not solve all your performance issues. Where possible try fragment caching prior to using full page caching. Also do not define dependency IDs that are not going to be used.

### 10.3.3 Use accurate workload traffic for simulation

When testing your application for performance, simulate actual user traffic as closely as possible. Analyze traffic from a running system to profile how users will visit your site. Only with accurate load simulation can you expect to learn how your WebSphere Commerce application will perform under pressure.

### 10.3.4 Invalidate as little as possible

Remember the golden invalidation rule: “Less is more.” The less you invalidate, the less time you spend rebuilding the cached content. The more you can get from the cache, the better your WebSphere Commerce application will perform.

### 10.3.5 Warm up the cache

Where possible, warm up your cache offline before bringing WebSphere Commerce online. Do this by running scripts of pre-prepared operations against the offline instance. This way your site will perform better for the first user through the door.

Archived

## 10.4 Changes in the next version of the online shop

A project to enhance the online shop for the company is in progress. This is giving us an opportunity to correct some past mistakes.

Once these solutions were implemented in the test environment, the online shop reached its performance targets of every page responding within five seconds at the 95th percentile (that is, all pages return a response to the browser within five seconds, 95% of the time) under anticipated peak load. This is over a four-fold improvement in performance on the online shop that was initially released, and a twenty to thirty times improvement on the original Web site.

The application changes that were implemented were:

1. Breaking all pages into cacheable fragments
2. Reducing the number of dependency IDs
3. Removing cache page expiries
4. Incorporating DB triggers to update the CACHEIVL table
5. Writing a scheduled task to clean the CACHEIVL table
6. Fixing inefficiencies in the search fragments

### 10.4.1 Break all pages into cacheable fragments

The online shop pages are broken into cacheable and non-cacheable fragments.

For example, the custom header, which varies on user ID, is its own fragment. The side menus are a separate fragment. The footer is a separate fragment. The body of the page is another separate fragment.

Any fragments with user-specific data are kept to a minimum size. That means, if a fragment is used to display a product and a rental plan together, and the rental plan is based on a business or residential user type, the fragment would be broken into two fragments.

Only the parameters required to uniquely create a fragment are passed to that fragment, and the cachespec.xml is being modified to work with these new fragments instead of caching the full pages.

### 10.4.2 Reduce the number of dependency IDs

The dependency IDs for the cached pages of the shop are being reduced to the minimum values required to uniquely identify the page for invalidation.

For example, for a cached page that refers to a single product, we use that product as a dependency ID. For a cached page that refers only to products of a

Archived

specific group, we use the group ID as a dependency ID. By using this type of method, we could construct an invalidation scheme based on database updates. This was achieved by using database triggers, but more on that later.

Also, reducing the number of dependency IDs for the cache elements further reduces the memory footprint in the JVM of the cached pages.

### 10.4.3 Remove cache page expiries

The update to the online shop ensured cache entries will no longer expire. Cache entries are only removed using invalidation rules.

Now only page fragments are being cached (not full pages). The sub-fragments that required short expiries because they were consumed fragments are no longer require as cache entries and there are no more 1 second timeouts.

### 10.4.4 Incorporate DB triggers to update the CACHEIVL table

As mentioned earlier, the online shop was altered to enable invalidation of pages based on database updates. This was achieved by using database triggers to populate the CACHEIVL table whenever updates were performed. This enabled only the relevant cache fragments to be invalidated.

For example, if the details for an “acme” widget are updated in the database, the database trigger will put entries in the CACHEIVL table to remove:

- All cache page entries with specific information on the updated “acme” widget.
- All cache page entries that deal only with “acme” widgets and have multiple “acme” widgets on the one page.
- All cache page entries that deal with any brand of widget and may have a number of widgets from different manufacturers.

### 10.4.5 Write a scheduled task to clean the CACHEIVL table

As the CACHEIVL table does not clear automatically, run the DBClean as a scheduled task to remove old entries from the CACHEIVL table.

### 10.4.6 Fix inefficiencies in the search fragments

The search fragment was the most inefficient operation in the online shop. This function was re-written to reduce the number of DB operations from over a thousand SQL queries per call to fewer than twenty.



Archived

Although this is not a DynaCache operation, it was included in this list because it highlights the need to optimize the application as well as the DynaCache. Frequently called base operations that take too long to complete will adversely impact the overall performance of your site.

## 10.5 Conclusion

Significant gains can be made by retro-fitting DynaCache to an existing WebSphere Commerce instance, but the best results can only be achieved if performance is designed into the application.

Archived

Archived

# Seven steps to get started caching your WebSphere Commerce Web site

*I've been working through your tutorial on caching and have seen incredible results. We went from 8.5 hits/second and 0.18 transactions/second to 172.2 hits/second and 2.86 transactions/second. Thanks!*

*WebSphere Commerce Customer*

How do you get started with caching your WebSphere Commerce site? You read this book, and now are wondering, “What do I do next?” Or perhaps you saw this chapter in the contents and thought, “That’s a good place to start.”

To help you get started we have made the conclusion to this book a brief list of steps you can take to tune up your WebSphere Commerce Web site with DynaCache.

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

# 11.1 Servlet caching

*What:* Servlet caching is equivalent to full-page caching and provides the same benefits. When you configure a page to be full-page (or servlet) cached, its entry in the cache contains the servlet along with the content from its JSP fragments. Think of this as taking the complete request to the servlet and making one cached object.

*How:* Take the sample cachespec.xml from the samples directory in WebSphere Commerce that is under the ConsumerDirect directory. This sample will show you how to cache the browsing pages in the sample B2C WebSphere Commerce site. Refer to Figure 2-12 on page 51 to see how to change the PATHINFO to match your store if you have used custom names. See 2.7.4, "DynaCache full page caching" on page 68 for more detailed information on servlet caching and the consume-subfragments property.

*Why:* We use servlet caching whenever possible because it gives the greatest performance benefit. Caching at the servlet level allows you to maximize what is contained in a given cache entry, thereby minimizing the time required to return the cached information on its subsequent requests. Servlet caching entirely removes the execution of Java code and database lookups on a cache-hit. It also removes execution of most of the runtime code in WebSphere Commerce.

# 11.2 Caching personalized fragments

*What:* Some of the pages you cached at the servlet level will have content that needs to vary page to page depending on many different factors. With these pages, you need to exclude the fragments containing varying content from the servlet's cache using the **do-not-consume** property, and cache the variable fragments using JSP caching (fragment caching) separately.

*How:* A cachespec.xml entry must be made for each fragment you wish to cache separately. The fragment being excluded must be self-executing. Not all fragments in WebSphere Commerce are self-executing. To test if the fragment is self-executing, pass the fragment's URL to a Web browser, including the necessary parameters. See 2.7.5, "DynaCache fragment caching" on page 69 for more information on fragment caching.

*Why:* The page, and its personalized page fragments, are stored in separate cache entries. When the page is transmitted again, the pieces are assembled from the fragments in the cache. Keep in mind that the more pieces there are to assemble into a page, the longer it will take to retrieve entries from the cache. In

general, combining servlet caching with fragment caching results in near full page caching performance for pages with personalized content.

## 11.3 Excluding self-executing fragments from the cache

*What:* A fragment is sometimes truly uncacheable, but that does not make the complete request uncacheable. Use the do-not-cache property to exclude truly uncacheable fragments from servlet caching the full page.

*How:* To use do-not-cache successfully, the fragment must still be self-executing. With some reworking, nearly every fragment can be made self-executing. Tell the WebSphere cache to ignore a fragment in a servlet by having the fragment's JSP defined in the cachespec.xml with the do-not-cache property. See 2.7.5, “DynaCache fragment caching” on page 69 for more information on do-not-cache.

*Why:* The technique of not caching a fragment should be used very rarely and only when it cannot be avoided. Very few things are not cacheable and you need to weigh the value of not caching versus the possible changes needed to make it cacheable.

## 11.4 Fragment caching

*What:* You will find cases, like the flow of a shopping order request, that are not amenable to servlet-level caching but still have cacheable fragments. In these cases use JSP fragment caching, but by itself.

*How:* See 2.7.5, “DynaCache fragment caching” on page 69 for more detailed information on fragment caching.

*Why:* These areas are still important to cache and good performance gains can be achieved so it is important not to give up looking for performance tweaks too soon.

## 11.5 Command caching

*What:* The final option open to you at this point is Command caching. Implementing business logic as a Java command allows the logic to be cached, yielding performance benefits when the same result set is repeatedly returned. Command caching can be viewed as result set caching. WebSphere Commerce uses command caching in its runtime.

*How:* See 2.7.3, “Command interface” on page 63 for more information on Command caching.

*Why:* When you have concluded that you have exhausted the possibilities of servlet and JSP caching, look at the possibility of using command caching.

## 11.6 Invalidation

*What:* Invalidation is the term applied to a number of different techniques used to flag cache entries as out-of-date and force their eviction from the cache. Next time the page or page fragment is requested, or the command re-executed, the content is re-generated and the cache entry stored away for use next time.

*How:* The simplest (and discouraged) method of invalidation is to used a time-based method. Usually, time-based methods do not make best use of the cache and do not guarantee accurate (timely) page contents.

The recommended cache invalidation technique is to use fine-grained invalidation methods like command-based invalidation using dependency and invalidation IDs. A cache entry has dependency IDs defined for each component that it relies on. When an object is cached for this entry, the dependency IDs are generated and associated to it. Invalidation IDs are constructed based on methods and fields generated by command-based invalidation commands. When an invalidation ID is generated, the entries in the cache that have a dependency ID matching the generated invalidation ID are invalidated.

A final way to configure automatic cache invalidation is using the CACHEIVL table in combination with database triggers.

See Chapter 3, “DynaCache invalidation” on page 91.

*Why:* Invalidation is essential to remove out of date cache entries so they can be regenerated with different contents. We recommend using invalidation rules to automate the invalidation of cache entries. This allows you to maximize the time content stays in the cache as well it make sure that any content change is presented to customers immediately.

If you do not have invalidation rules configured, then you will need to invalidate all of the cache when content changes. This means that everything will be removed from the cache, not just the changed pages. The invalidation of the complete cache is not recommended and should be avoided whenever possible. See Chapter 3 for more detailed information on fragment caching.

Archived

# 11.7 Replication

*What:* Identify cache entries to be shared. These are likely to be content cached at the servlet level, since that does not contain personalized cache content. Do not replicate any cached JSP fragments that do not have a long life, or are not reusable across users, unless their creation costs are very high.

*How:* Add an element to their cache entries

```
<sharing-policy>Value<sharing-policy>
```

See Chapter 4, “Clustering DynaCache” on page 107 for more details.

*Why:* In a clustered environment, certain cache entries are highly reusable across users and can be shared among servers in a cluster.

At a minimum you need to use replication to make sure invalidation messages are sent to all members of a cluster and Web pages are showing current information.

Some cache entries are very expensive to recreate, and even with low usage, it can be worth bearing the costs of replication to achieve even a small cache hit rate.

In a dynamic clustered environment (see Appendix B, “Caching in WebSphere Extended Deployment” on page 257), there is the opportunity to partition a very large cache between different members of a cluster, and so increase the proportion of the cache that is held in memory rather than on disk.

Archived

Archived

A

# Web services caching

The Web services runtime in WebSphere Application Server and the Web services framework in WebSphere Commerce both provide caching to help improve the performance of Web Service requests.

© Copyright IBM Corp. 2006. All rights reserved. 

# WebSphere Web service caching support

WebSphere Application Server caches Web service requests. Instead of processing the request it returns the cached response if there is one. The same fundamental caching principle, which we described in chapter 2, is employed again. A cache key is computed from a hash of the SOAP envelope. When a Web service request is made, the SOAP envelope is hashed and the key used to retrieve a previously cached response (or to store the response to the new request as a cache entry).

Details of Web service caching support can be found from the WebSphere information center at:

```
http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.web
sphere.express.doc/info/exp/ae/rdyn_webservicescaching.html
```

# WebSphere Commerce Web service caching

Caching Web services using WebSphere Application Server is similar to caching static Web pages. As long as the request and response do not change, it is the simplest and most effective type of Web services caching. However, for those Web service requests that have dynamic data, such as a timestamp, simple caching does not work. The hash algorithm always generates a different hash key value even if the request is unchanged and appears cacheable. As an example, an OAGIS XML message contains CreationDateTime which is a mandatory element of ApplicationArea. Since this element changes on a per request basis, the hash algorithm produces a different value even though the request could be cacheable.

WebSphere Commerce caches Web service requests that contain dynamic elements. The WebSphere Commerce Web services framework leverages DynaCache to cache Web service requests instead of using WebSphere Web services caching.

## Overview of the WebSphere Commerce Web services framework

The WebSphere Commerce Web service framework uses the Model-View-Controller (MVC) approach to process Web service requests. As shown in Figure 1, when the Web service request enters the system, the processing steps are broken down as follows:

1. Controller: Determine the business logic
2. Model: Execute the business logic
3. View: Render the response



Archived

Controller commands are used to perform the business logic of the operation and JSPs are used to create the Web service response. By using both types of assets, DynaCache is leveraged to cache Web service requests that contain dynamic elements which do not change the result.

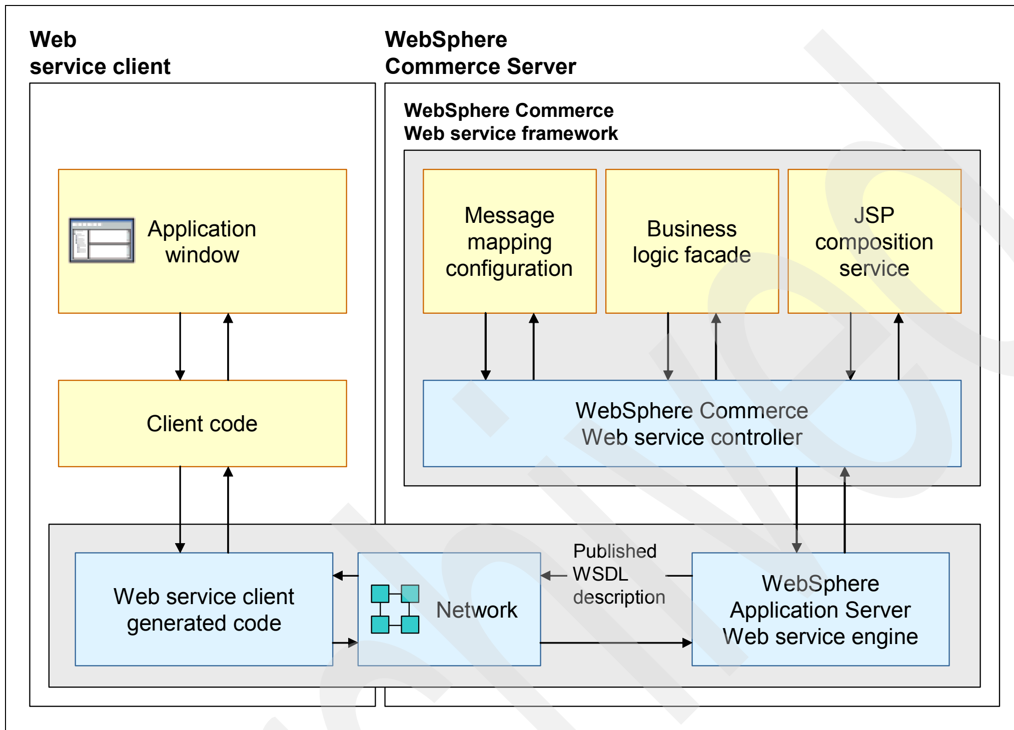



*Figure A-1 Web services framework*

DynaCaching of Web services requests and Web requests only differ in how the business logic is determined. Web requests use the Struts configuration file to identify the controller command interface and JSP response. Web service requests use the messaging mapping component to identify the controller command interface and JSP response.

The message mapping component uses the XML document type to identify the controller command interface and JSP response. It also breaks the XML document down into a set of name-value-pairs that are used as input parameters to the controller command. Based on the XPath configuration, the XML elements of the Web service request are represented in a similar name-value-pair format to a Web request. For simple XML documents, the structure is mapped exactly like a Web request. The mapping facility uses an additional list structure to represent more complex XML documents. The resulting data structure is similar

to the one created for Web requests and DynaCaching of Web services is able to reuse the same controller and task commands as a Web request.

The WebSphere Commerce Web services framework uses the SOAP element binding to route the request to the Web Service controller. It passes the XML body of the SOAP request to the controller. The controller uses the message mapping feature to work out which controller command to use to service the request. It then parses the XML body into name-value pairs and passes them to the controller command.

Message mapping is also used to construct the success or error response. The name-value-pairs returned by the business logic are merged with the initial name-value-pairs and passed as request parameters and attributes to the JSP responsible for rendering the response.

If you specify that a request only returns data in the message mapping configuration then the controller command calls the JSP directly to build the response. In this case, the name-value-pairs returned from the message mapping will be the request parameters and attributes to the JSP.

The Web service response is built with JSPs and leverages existing JSP technology and assets. The JSP uses the Web service definition to generate the document structure and XML element tags of the SOAP response. The user task is to write the JSP code to populate the XML elements. Although the purpose of the JSP is to return XML instead of HTML, the same JSP concepts apply, including JSTL, JSP fragments, and caching.

### Related Information

WebSphere Commerce Web services:
http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.webservices.doc/concepts/cwvwc55webservicesguide08.htm

Message mapping:
http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.integration.doc/concepts/ccvmapper.htm

## Caching the business logic

Business logic is implemented as a controller command that extends the cacheable command, so you create a command caching policy to cache business logic. Of course, the business logic must be cacheable and the command implementation must implement the appropriate caching hooks.



Archived

In many cases business logic is used to update data. In these cases try to bypass the business logic and call the JSP directly to build the response.

# Caching the response

The concepts that apply to caching JSPs for Web requests also apply to the caching of JSPs for generating Web service responses. The only difference is that the JSP produces XML instead of HTML.

To cache a full Web service create the JSP cache entry that produces the Web service response, as shown in Example A-1. The parameters passed to the JSP appear as the request parameters and the command context and request properties appear in the request attribute. These parameters and attributes are used as cache keys to specify the cache entry for the JSP.

*Example: A-1 Cache entry for a Web service*

```
<cache-entry>
           <class>servlet</class>
           <name>/webservices/MyCompany/ProductInformation.jsp</name>
           <property name="save-attributes">false</property>

           <cache-id>
               <component id="productId" type="parameter">
                   <required>true</required>
               </component>
               <component id="CommandContext" type="attribute">
        <method>getStoreId</method>
        <required>true</required>
      </component>
           </cache-id>
      </cache-entry>
```

A JSP fragment can be reused in other JSPs so if a JSP caching policy is created for a fragment, other JSP will reuse the cached fragment. Suppose, for example, a response returns a list of products. Suppose a product node is a separate JSP fragment; then each product fragment is cached separately. When a different list of products is displayed in another page, the cached products are used rather than re-executing the JSP fragment for each individual product.

For an example of caching a Web service response, refer to step 8 of the Defining an inbound Web service tutorial.

http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.webservices.doc/tutorial/twvinboundws.htm

Archived

B

# Caching in WebSphere Extended Deployment

In October 2004, WebSphere released a new offering called WebSphere Extended Deployment (XD). This product is the first on demand offering from WebSphere. This appendix explains how to configure a WebSphere Commerce application to use WebSphere XD to eliminate offloading DynaCache to disk storage.

© Copyright IBM Corp. 2006. All rights reserved.

# Introduction to WebSphere XD

WebSphere XD extends WebSphere Network Deployment (ND) by providing on demand capabilities in three separate areas:

- Dynamic operations
- High performance computing
- Extended manageability

The most notable change in the deployment of a WebSphere XD system is the introduction of an On Demand Router (ODR) to manage the dynamic allocation of resources.

For a list of references on WebSphere XD, refer to the resource section at the end of this appendix.

## Dynamic operations

WebSphere Extended Deployment monitors the Application Server environment and makes optimizations or recommendations based the servers' behavior. This is called *WebSphere Dynamic Operations*. WebSphere XD tries to meet the demands of work coming into the system by balancing resources in the system according to policies and goals.

To manage resources WebSphere XD creates a virtualized environment. Resources in the virtual environment are unified into a pool, called a *node group*. For our purposes a node group is simply a collection of managed computer systems into which WebSphere deploys applications[1]. This group of machines defines a boundary for cluster formation. Within a node group one or more dynamic clusters are created, and the node group's computing power will be distributed between clusters according to the policies and goals that have been established. This is done by assigning priorities to the requests the system receives based on these policies and goals.

Resources are expanded and contracted within the node group to ensure the business goals are achieved. In this way, clusters are no longer a static entity, as in a WebSphere ND environment, and become dynamic. In XD these clusters are known as dynamic clusters.

## High performance computing

High performance computing has three main aspects: data partitioning for high volume transaction applications, the high availability manager for failover

[1] For information on having more than one node per machine consult WebSphere documentation.

Archived

support, and the WebSphere Partitioning Facility (WPF) for relieving the bottlenecks that sometimes occur in large transactional environments.

## Extended manageability

The extended manageability function within WebSphere Extended Deployment provides the ability to configure the dynamic operation environment and visualization capability to enable the administrator to understand the operational state of the environment.

## On-demand router

The topology of a WebSphere XD environment is different than a typical cluster setup. Most notable is the introduction of an On Demand Router (ODR) node. Figure B-1 shows an example of WebSphere Commerce running in an XD environment.

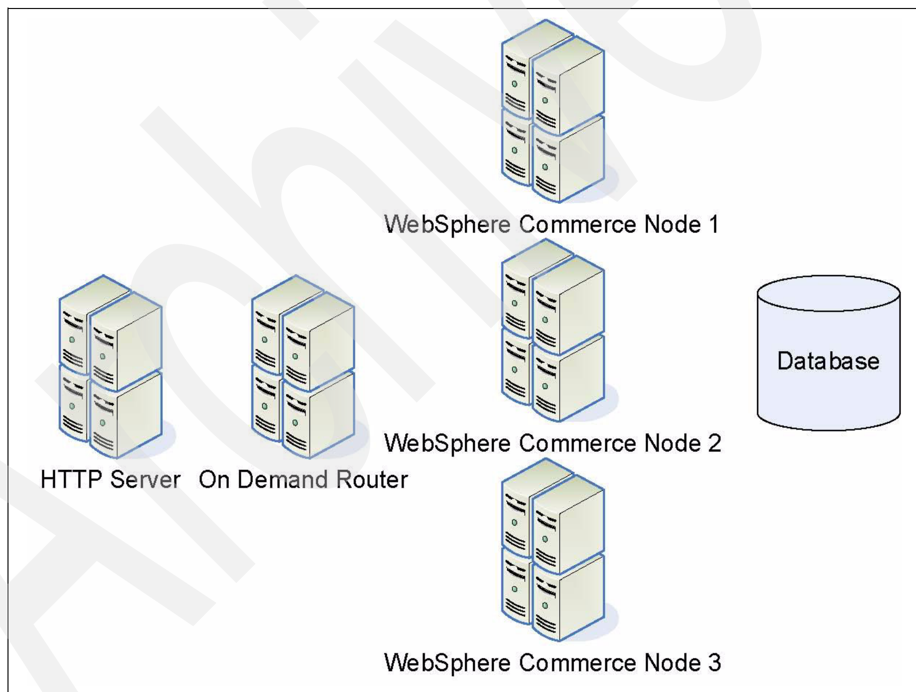



*Figure B-1 WebSphere Commerce in a WebSphere XD environment*

The ODR routes traffic, controls logic, and handles caching and partition decision-making in the WebSphere XD environment. It is an application that runs on WebSphere Application Server.

All requests are routed through the ODR. It is possible to have a cluster of ODRs so that the routing does not become a bottleneck in the system. In a partitioned environment, the ODR extracts the partition name from the HTTP request and routes it to the application server hosting the application instance that is currently serving this partition.

The partition name is constructed from information contained in the HTTP request. HTTP Partitioning assumes incoming HTTP requests contain sufficient information to identify the partition associated with the request in its URL. There is a special file, called partitions.xml, which specifies two configuration lists: expressions and partitions. The expression list consists of all regular expressions that will be used to extract valid partition names from incoming HTTP requests. The partitions list contains all valid partition names identifying all partitions that should be managed by XD and activated in back-end target servers.

For existing applications the deployment specialist must find some constant and recurring information in each HTTP request to construct a partition name. For new applications the partition name can be designed into the HTTP request.

The scenario described in the next section uses HTTP partitioning, where partitions are defined using the URL of an HTTP request.

**Note:** In order for an application to be recognized as partitioned, it must contain a Partitioned Stateless Session Bean (PSSB). A PSSB is a stateless session bean that implements the PartitionHandlerLocal interface, and can utilize the PartitionManager to create partitions from the WPF framework. A PSSB must be introduced into the WebSphere Commerce EAR as a prerequisite to partitioning it. You can find more information about the PSSB and the PartitionHandlerLocal interface in the XD references listed at the end of this appendix.

## WebSphere Commerce and WebSphere XD

We have already discussed the problem of managing large caches in WebSphere Commerce. We recommended extensive use of disk caching to reduce the size of the in-memory cache and reduce its impact on the Java heap. Clustering in a WebSphere ND environment gave us no solution to this problem because in an WebSphere ND clustered environment each server has the same elements in its cache as the other servers, and the same elements in its memory cache.

Would it be possible to partition the cache differently for each server and then route requests to the server which is most likely to have a requested element in

memory? This is just what WebSphere XD can do for your WebSphere Commerce Web site.

Figure B-2 illustrates a catalog that contains 50,000 products. Each node has enough memory to hold at most 20,000 pages at any given time. Suppose the next request is for the product with the id 20001. This will result in a cache miss, and since the cache of each node is full, an entry will be expelled or offloaded to disk to make room for the requested page.

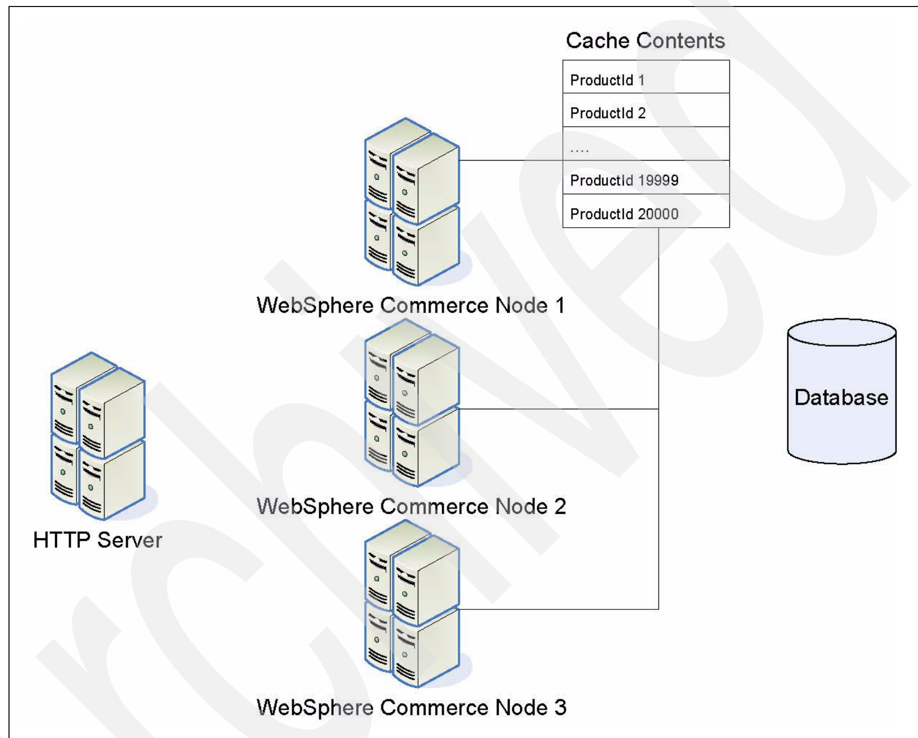



*Figure B-2 Traditional Commerce topology with a large catalog*

In a WebSphere XD environment it is possible for all of the catalog pages to be in-memory at one time. Create partitions in your cluster using WebSphere Partitioning Facility (WPF) and divide the catalog among the nodes, or groups of nodes. The partition could be defined using the product identifier (for example, products 1-15000 are assigned to partition 1, products 15000-30000 are assigned to partition 2, and so forth). A node (or node group) is then responsible for a subset of the catalog and it could be contained entirely in memory. Figure B-3 on page 262 illustrates the partitioned XD environment.

Archived

> **Note:** To achieve maximum performance benefits, each partition must be carefully defined to be no larger than the memory you have decided to allocate from the JVM heap on each node. If the number of catalog pages each partition is responsible for is at most the size of the in-memory cache, then disk caching is avoided altogether. Avoiding disk cache is not always achievable and is not necessary to get reasonable performance.

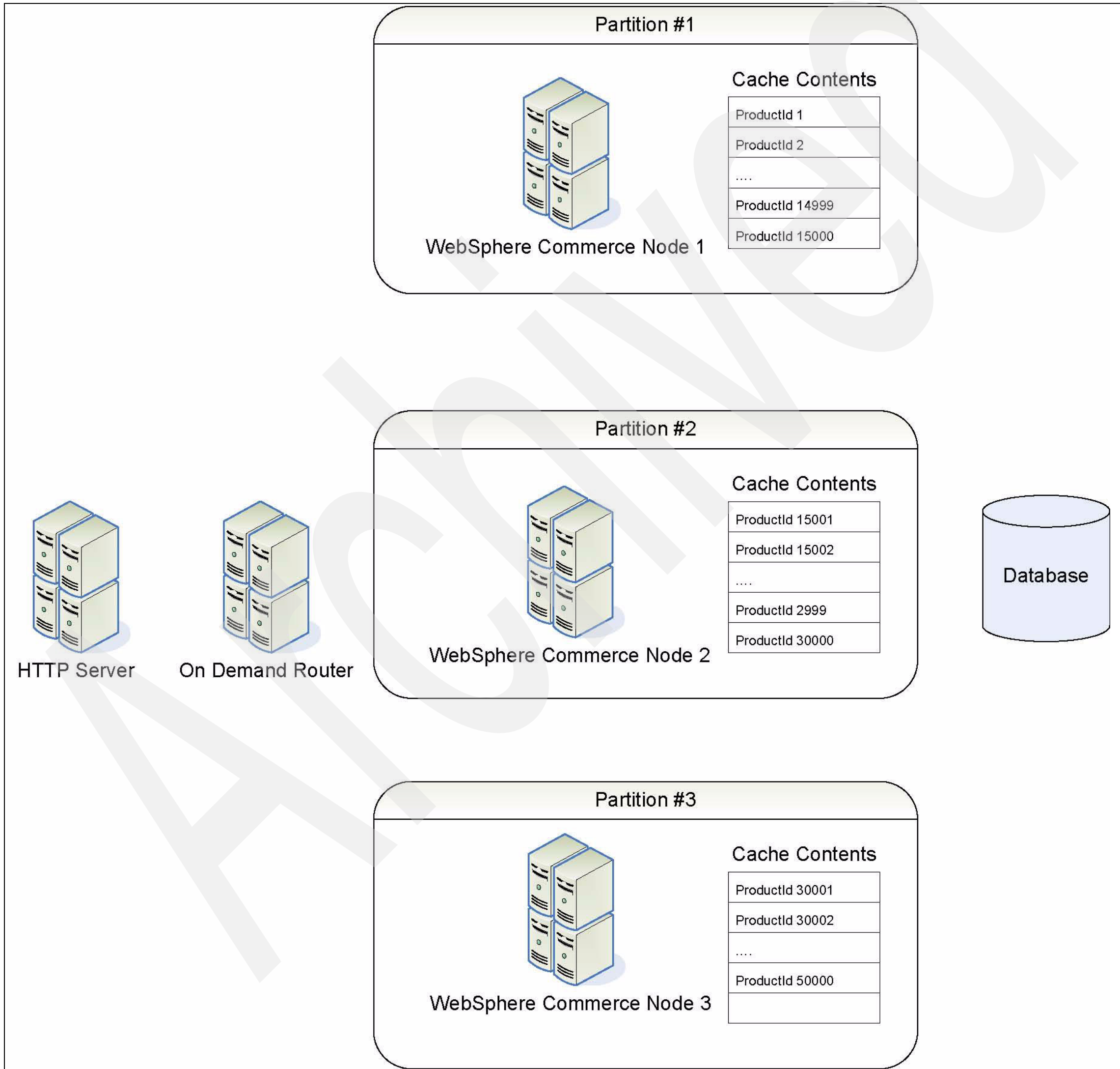



*Figure B-3 Partitioned WebSphere Commerce topology*

What happens when a request is received from a Web client? The request is received by the ODR. The ODR uses the expression defined in partitions.xml to identify the product ID in the incoming HTTP request and map the request to a partition. The request is then sent to a node in the partition. If the page has been accessed before, and not invalidated, the node will have the catalog page in memory. When looking at what partitions to create you should start with separating browsing traffic from buying traffic. Once you have achieved this level of partitioning you can then determine if more partitioning is required.

# References

XD Information Center:

http://publib.boulder.ibm.com/infocenter/wxddoc51/index.jsp?topic=/com.ibm.wasxd.doc/cwpfoverview.html

WPF User Guide:

http://publib.boulder.ibm.com/infocenter/wxddoc51/topic/com.ibm.wasxd.doc/WPFUserGuide.pdf

Archived

Archived

C

# Sales Center caching

IBM Sales Center provides call center representatives with the functionality they need to service and up-sell cross-channel customers.

DynaCache plays an important role to improve the performance of IBM Sales Center. This appendix shows how DynaCache command caching support improves the performance on constructing the response Business Object Document (BOD) from a WebSphere Commerce server.

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

# IBM Sales Center

The IBM Sales Center architecture is comprised of the IBM Sales Center client, WebSphere Commerce server and a messaging architecture.

A data model is used to cache business objects on the client. The default IBM Sales Center data model contains model objects that represent the operator, customers, orders, products, and other commonly used objects. If a valid data model instance (data object) is not available, a service request is sent to the server to retrieve the information to create or update the data model in the client.

Figure C-1 illustrates the messaging architecture.

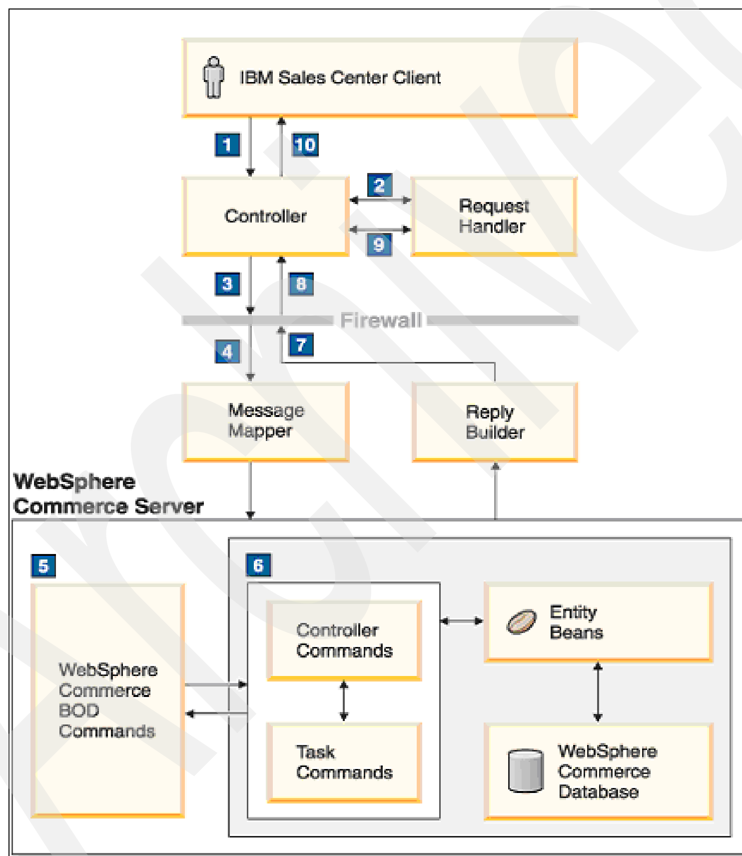



*Figure C-1 IBM Sales center architecture*

The interactions are as follows:

1. The IBM Sales Center client performs a service request.



Archived

2. The service request handler prepares a Business Object Document message.
3. A message is sent from the client to the host machine.
4. The message mapper receives the message and maps the Business Object Document to a WebSphere Commerce BOD command.
5. The WebSphere Commerce BOD command is invoked.
6. The WebSphere Commerce BOD command calls a WebSphere Commerce Controller command, which may call one or more task commands.
7. The reply or response builder constructs the response BOD.
8. A response is returned to the client machine.
9. The request handler receives and handles the response BOD.
10. The client user interface is updated on screen.

## Caching the response in Sales Center

As well as caching the data model in the client, further performance gains can be made by caching the BOD that is returned to the Sales Center client by the Sales Center server in response to a service request.

Constructing the BOD may be a very resource consuming process. With the use of command caching the response BOD can be cached, and returned on the next request rather than recomputing it again. For example, the response builder command for GET store request (see Example C-1) is cacheable based on the retrieved properties such as search criteria, store ID, language ID, user parent member ID, user member group IDs and user roles.

*Example: C-1* Cache Entry for the response BOD of Get Store request

```
<cache-entry>
        <class>command</class>
        <sharing-policy>not-shared</sharing-policy>
        <name>
            com.ibm.commerce.telesales.messaging.bodreply.ShowStoreCach
eCmdImpl
        </name>
        <cache-id>
            <component type="method" id="getSearchCriteria">
                <required>true</required>
            </component>
            <component type="method" id="getStoreId">
                <required>true</required>
```

```
            </component>
            <component type="method" id="getLanguageId">
                <required>true</required>
            </component>
            <component type="method" id="getUserParentMemberId">
                <required>false</required>
            </component>
            <component type="method" id="getUserMemberGroupIds">
                <required>false</required>
            </component>
            <component type="method" id="getUserRoles">
                <required>false</required>
            </component>
            <component type="method"
id="getEligibleTradingAgreementIds">
                <required>false</required>
            </component>
            <priority>1</priority>
            <inactivity>3600</inactivity>
            <timeout>86400</timeout>
        </cache-id>
        <dependency-id>Stores</dependency-id>
    </cache-entry>
```

The result of the response builder command is cached and is used by subsequent calls to avoid constructing the same response BOD again. The cached response BOD for GET store request is invalidated only if the store-related information is updated; for example, when the state of the store is changed. For example, to list stores with names beginning with “Special,” the response BOD is constructed and cached. It takes time to search for the eligible list of stores using a database query. Subsequent calls to get the list of stores with names beginning with “Special” will be very fast. The cached response BOD is returned rather than executing the search query and there is no need to create the response BOD again.

Archived

# Abbreviations and acronyms

| | |
|---|---|
| **API** | Application Programming Interface |
| **ASP** | Active Server Page |
| **CGI** | Common Gateway interface |
| **CPU** | Central Processing Unit |
| **CSF** | Critical Success Factor |
| **DB** | Database |
| **DCPE** | DynaCache Policy Editor |
| **DRS** | Data Replication Service |
| **EJB** | Enterprise Java Bean |
| **FAQ** | Frequently Asked Question |
| **GMT** | Greenwich Mean Time |
| **HTML** | HyperText Markup Language |
| **HTTP** | HyperText Transfer Protocol |
| **I/O** | Input/output |
| **IBM** | International Business Machines Corporation |
| **IHS** | IBM HTTP Server |
| **IT** | Information Technology |
| **ITSO** | International Technical Support Organization |
| **JDBC** | Java Database Connection |
| **JDK** | Java Development Kit |
| **JMS** | Java Message Service |
| **JNDI** | Java Naming and Directory interface |
| **JSP** | Java Server Page |
| **JSR** | Java Specification Request |
| **JSTL** | Java Standard Tag Library |
| **JVM** | Java Virtual Machine |
| **LRU** | Least Recently Used |
| **MVC** | Model-View-Controller |
| **ND** | Network Deployment |
| **OAGIS** | Open Applications Group Integration Specification |
| **ODR** | On-Demand Router |
| **PHP** | Hypertest Pre-Processor, Previously, Personal Home Page |
| **PMI** | Performance Management Instrumentation |
| **POJO** | Plain Old Java Object |
| **RAD** | Rapid Application Development |
| **RAM** | Random Access Memory |
| **RMI/IIOP** | Remote Method Invocation/Internet Interoperability Protocol |
| **SDK** | Software Development Toolkit |
| **SI** | System Integration |
| **SOAP** | SOAP |
| **SPOF** | Single Point Of Failure |
| **SQL** | Structured Query Language |
| **SSL** | Secure Sockets Layer |
| **TCP/IP** | Transmission Control Protocol / Internet Protocol |
| **TTL** | Time To Live |
| **UK** | United Kingdom |
| **URI** | Universal Resource Identifier |
| **URL** | Universal Resource Locator |
| **WAN** | Wide Area Network |
| **WSDL** | Web Services Definition Language |
| **XD** | Extended Deployment |
| **XML** | eXtensible Markup Language |

Archived

© Copyright IBM Corp. 2006. All rights reserved. 

Archived

# Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this Redbook.

## IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 273. Note that some of the documents referenced here may be available in softcopy only.

- *Best Practices and Tools for Creating WebSphere Commerce Sites,* SG24-6699-00

## Online resources

These Web sites and URLs are also relevant as further information sources:

- Enhancing the performance of WebSphere Commerce applications through dynamic caching

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0603_crick/0603_crick.html

- Caching WebSphere Commerce pages with the WebSphere Application Server dynamic cache service

  http://www-128.ibm.com/developerworks/websphere/library/techarticles/0405_caching/0405_caching.html

- Tutorial: Improve WebSphere Commerce performance with dynamic caching

  http://www-128.ibm.com/developerworks/websphere/library/tutorials/0507_crick/0507_crick_reg.html

- Disk cache enhancements

  http://www-1.ibm.com/support/docview.wss?uid=swg27007969

- Dynamic cache and data replication service tuning guide

  http://www-1.ibm.com/support/docview.wss?rs=180&context=SSEQTP&q1=dynamic+cache+tuning+guide&uid=swg27006431&loc=en_US&cs=utf-8&lang=en

© Copyright IBM Corp. 2006. All rights reserved. 

- Clearing up the dynamic cache using WebSphere Application Server scripting/mbean API

  http://www-1.ibm.com/support/docview.wss?uid=swg21243042
- WebSphere Commerce InfoCenter

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp
- WebSphere XD InfoCentre:

  http://publib.boulder.ibm.com/infocenter/wxddoc51/index.jsp?topic=/com.ibm.wasxd.doc/cwpfoverview.html
- WebSphere Partition Facility User Guide:

  http://publib.boulder.ibm.com/infocenter/wxddoc51/topic/com.ibm.wasxd.doc/WPFUserGuide.pdf
- WebSphere Commerce Web Services:

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.webservices.doc/concepts/cwvwc55webservicesguide08.htm
- Web Service Caching

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.express.doc/info/exp/ae/rdyn_webservicescaching.html
- Message Mapping:

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/topic/com.ibm.commerce.integration.doc/concepts/ccvmapper.htm
- Performance tuning

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/topic/com.ibm.websphere.nd.doc/info/ae/ae/tprf_tuneprf.html
- Struts

  http://publib.boulder.ibm.com/infocenter/wchelp/v6r0m0/index.jsp?topic=/com.ibm.commerce.developer.doc/concepts/csdstrutskeycompons.htm
- Tiles

  http://struts.apache.org/1.x/struts-tiles/
- CacheAdvisor

  http://alphaworks.ibm.com/tech/cacheadvisor
- Dynacache Policy Editor

  http://www.alphaworks.ibm.com/tech/cachepolicyeditor
- Java Techology, Diagnosis Documentation

  http://www-128.ibm.com/developerworks/java/jdk/diagnosis/142.html



Archived

## How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this web site:

**ibm.com**/redbooks

## Help from IBM

IBM Support and downloads

**ibm.com**/support

IBM Global Services

**ibm.com**/services

Archived

Archived

# Index


## A

activity metrics
  see benchmarking, activity metrics
Akamai 29
analytics 125
  tools 125
anti-pattern 137, 235
  build more slack 21
APAR
  IY88656 101
  PK13460 85, 149
application tier 4
associated components 28
AttributeValueUpdateCmdImpl 96
auto detect
  see network, auto detect

## B

B2B 5, 122, 217, 220
B2C 5, 122–123, 180, 217, 220–221, 246
bandwidth
  see network, bandwidth
baseline
  see benchmarking, baseline
benchmarking 12, 14, 209–211, 213–218, 222–223, 230–231, 233
  activity metrics 216
  baseline 12, 213, 217, 223–224
  benefits 210
  considerations 210
  creation process 214
  DynaCache 209, 213
best candidates for caching 11, 45
bottlenecks 7, 12, 14, 20–22, 259
  build more slack, see anti-pattern, build more slack
  choke point 20
broken system 12
browse-buy ratio 122
budget 12, 233
bursts
  see network, bursts
Business to Business (B2B) site, see B2B
Business to Consumer (B2C) site, see B2C

## C

cache
  administration 39
  analysis 5
  applications generally scale better 10
  capacity 14, 123, 212
  clear button 105
  client 5, 7–8, 27, 29–31, 42, 51, 62, 64, 67, 114, 117, 124, 134, 149, 153–154, 166, 168, 263
  ConsumerDirect store 189
  contents 88, 195, 200, 204, 208
  design 126
  entry 47–49, 52, 54–55, 57–59, 62–63, 67–69, 71–77, 93–94, 96–98, 116, 130, 132, 136, 140–141, 144, 160–162, 167, 190, 195–196, 198–200, 248, 255
    default priority 189
  entry element overview 48
  entry ids 157
  filter 42–43
  fragment 6, 33–34, 41, 50, 69, 71–72, 77, 93, 116, 126, 128, 134–138, 140, 154, 195–196, 198–200, 226, 237, 240–242, 246–248, 255
  fragmentation 80–83, 173, 213
  full-page 126
  generic concepts 25
  hit 26, 51, 60–61, 71, 74, 99, 164, 188–189, 213, 249
  id 26–27, 40, 43, 47–49, 51–53, 58–59, 62–64, 67, 70–74, 77–79, 88–89, 93–94, 98–100, 116–117, 130, 132, 136, 140–141, 146–147, 158, 160–166, 168–169, 192, 196–197, 199–200, 202, 206, 255
    overview 49
  identifiers 26, 40, 49
  instance 39, 84, 88–89, 105, 127, 134, 137–138, 143–145, 174
  item dependencies 28
  java objects 60
  Misses 189, 194
  monitor 25, 35, 41, 53, 69–70, 87–90, 92, 103,


© Copyright IBM Corp. 2006. All rights reserved.

127, 133, 179, 184, 187–188, 192–196, 198–200, 203–204, 206, 208
explicit removals 189
policy 52, 58–59, 63–64, 76, 87–88, 121–122, 124–126, 130, 134–135, 137, 140, 157, 161–162, 168, 184, 189, 192–193, 200, 203, 206
prepared statement 13
programming support 52
proxy server 29
service 26, 37, 59
servlet filtering 168
servlets and JSPs 60
size 188
sizing formula 146
statistics 87–88, 188, 193
strategies 33
unnecessary recreation of page data 5
warming 143
cacheability, see cacheable
cacheable 6, 25, 33–34, 40–41, 69, 121, 123, 125–126, 136–137, 153–155, 161–162, 189, 213–214, 239–241, 247, 252, 254, 267
fragments 5, 241, 247
CacheableCommandImpl 64–65, 67, 126, 128
CachedFooterDisplay.jsp 49–50, 200
CachedHeaderDisplay.jsp 94, 195–196, 200
CachedSidebarDisplay.jsp 198
cache-entry 267–268
cache-id 267–268
CACHEIVL 75, 102, 241–242, 248
table 101–102
cacheKey 100
CacheMonitor.ear, see cache monitor
cachespec.xml 35, 38–41, 45–48, 50, 53, 59, 61–62, 67, 70, 73–78, 84, 92–93, 95–96, 116, 126–127, 138, 141, 144, 147, 154–155, 159, 161, 172, 179, 184, 190, 192, 195–196, 199, 203, 206, 223–224, 226, 228–232, 237, 239, 241, 246–247
catalog 6, 44–45, 57, 84, 92–93, 103, 123, 126, 142, 174, 189, 191, 201, 204–205, 220, 225, 261–263
jsp 6
subsystem 189
CategoryDisplay 54, 56–57, 97, 190–191, 201–204, 206–207
Central Processing Unit
cycles 34
usage 10, 216, 224, 226–232
child JSP fragment 6
choke point
see bottlenecks, choke point 20
Christmas 15, 23, 211
client 266–267
code
changes 11, 63
review 126
com.ibm.commerce.struts.ECActionServlet.class 75–77
command
based invalidation 55–56, 58, 75, 96
interface 52, 57, 63, 68
objects 39, 52, 61, 63
concurrent users 122
connection sizing
see network, connection sizing
consolidating website pages 20
constancy of workload 217
consultant
see also experience 13–14
ConsumerDirect
jspStoreDir issue 170
consume-subfragments 68–69, 71–72, 75, 77, 135, 140
ContentContainerTop.jsp 199
ContentSpotDisplay.jsp 199
cookies 42, 47, 52, 70, 73, 116, 135, 139, 143, 173
CPU, cycles
see Central Processing Unit, cycles
CPU, usage
see Central Processing Unit, usage
CSF
see consume-subfragments
custom content 123
custom JVM Properties 85
custom monitoring level 150

## D

data or persistence tier 4
Data Replication Service 107–108, 110–113
database 23
connections 8
server 6–7, 34, 124, 212, 224, 227
triggers 75, 102, 128, 241–242, 248
tuning 13
datasource 9
connection pool 9

DCPE
see DynaCache, Policy Editor
deadlock 9
default priority
see cache, entry, default priority
defects 11, 215
delay-invalidations 57–59
demand increases 10
dependency 28, 40, 47, 49, 53–55, 57, 59, 73–75, 78, 86, 89, 93–98, 100, 102, 104–105, 146–149, 160–162, 166–167, 169, 174, 197, 240–241, 248, 268
based 27
id 28–29, 47–49, 53–55, 59, 74–75, 86, 89, 93–95, 98, 161–162, 167, 197
items 28, 49, 175
dependency-id 268
design
documents 12
pattern 4
developerWorks 152
dial-up connections 15
disk cache 26, 133–134, 147
cleanup 147–148
enhancements 134–135
eviction algorithm 135
offload 35, 38, 84, 86, 88, 106, 144, 182, 214
enabled 189
scan times 135
displaying cache information 87
dissatisfied customers 15
distributed architecture 4
Distributed Replication Service 39, 113
distributedMap 39, 100
distributedObjectCache 39
do-not-cache 71–72, 126, 135–137, 172, 199, 247
do-not-consume 69–70, 72, 94, 116, 126, 140, 196–197, 199–200
DRS
see Disk Replication Service
DynaCache
class element 153
field element 162–163
fragment caching 69
full page caching 68
history 34
Ignore-value 158, 160, 163
invalidation API 100, 102
overhead 213
Policy Editor 125
DynaCache.jar 38
DynaCacheEsi 142
dynamic clusters 258
DynamicCacheAccessor object 100

## E

Easter 23
ECActionServlet 75–77, 190, 193, 195, 200, 203–204, 206, 208
Eclipse plug-in 125
e-commerce web site 92, 122–123, 221
types 5
edge
statistics 88
tier 4
Edge Side Include 29, 116, 136, 142, 154
EJB pools 13
eMarketing spot 70
enable disk offload 84, 182, 214
ESI
see Edge Side Include
eSpots 179, 199
experience
see also consultants xii–xiii, 11, 13, 31, 34, 74, 105, 145, 210, 212, 235
explicit removals
see cache, monitor, explicit removals
extended manageability
see WebSphere, Extended Deployment, extended manageability
extreme load pressures 142

## F

failover 10, 109–110, 112, 175, 258
detection 10
filter objects
see servlet, filter objects
flush 43–44, 128–129, 131, 175
flush to disk 84, 182, 213
FlushToDiskOnStop 38
Funnel 8

## G

garbage collection 13, 79, 81–82, 146, 173
GC
see garbage collection

getDistributedMap() 100
group objects together 28, 135

## H

hard disk usage 216, 224, 226, 229, 231
hash table on disk 173
hashing algorithm 26
heap fragmentation 80, 83, 213
hit rate 26, 249
horizontal Scaling 23
HTML
  see Hyper-Text Markup Protocol
HTOD
  see hash table on disk
htodCleanupFrequency 85, 148
htodDelayOffload 85–86, 148
htodDelayOffloadEntriesLimit 85–86, 149
HTTP 304
  see Hyper-Text Transfer Protocol, 304
HTTP, GET request
  see Hyper-Text Transfer Protocol, GET request
HTTP, servers
  see Hyper-Text Transfer Protocol, servers
HTTP, session object
  see Hyper-Text Transfer Protocol, session object
HTTP, web server
  see Hyper-Text Transfer Protocol, web server
httpd.conf 172
HttpServletResponse object 173
Hyper-Text Markup Protocol 4–5, 15–16, 31, 33, 41–44, 63, 68–69, 254–255
  page 5
Hyper-Text Transfer Protocol
  304 30
  GET request 30
  servers 23
  session object 42–43
  web server 29

## I

IBM Tech-line 14
  capacity planning guides 14
Identifying cache objects 121, 123
Idgenerator 48, 158, 166
  and metadatagenerator sub-elements 166
If-Modified-Since 30
images 4, 8, 30–31, 33
inactivity 48, 99, 158, 164
infoCenter 133
invalidation 27–28, 34–35, 38–39, 41, 47–49, 53–59, 70, 73–74, 80, 88, 91–98, 101–102, 104–106, 114, 121, 126–127, 135, 137, 146, 160, 162, 168, 174–175, 179, 196, 198, 213–214, 238, 240–242, 248
  best practices and techniques 104
  element 167–168
  generator sub-element 168
  mechanisms 27, 92–93
  policies 92
  programmatic 92
  rules 49, 55–57, 74–75, 93, 95–96, 98, 101, 104, 248
  the whole cache 105
inventory
  list items 6
  numbers 6
IP sprayers 7
isCacheEnabled() 100
IT Specialists 14

## J

J2EE
  see Java 2 Extended Edition
Java
  Dictionary class 38
  heap 81, 173, 260
  Messaging Service 111, 176
    message tuning 13
  Naming and Directory Interface 39, 174
  Native Interface 81
  script 4
  SDK Release 1.3.1, Service Refresh 7 81
  Server Page 4–6, 27, 31, 37, 39–41, 43–44, 47–48, 50–52, 60–61, 63, 68–69, 71, 77, 115, 121, 123–125, 128, 131, 134, 136, 143, 153–154, 170, 172–173, 189, 195–196, 217, 226, 237, 246–249, 253–255
    forward 44
    Tag Language 44, 128, 254
  Server Pages
    self-executing 69, 124, 137, 246–247
  Server Tag Library 128
  Specification Request 168 41, 134
  Virtual Machine
    heap 21, 38, 80, 146, 213–214, 216, 224,

226, 229, 231, 262
tuning 13
Java 2 Extended Edition xiii, 4, 8, 11, 26, 39, 41, 129
Java Database Connection-2
connection pools 13
drivers 81
java.util.HashMap 26
JDBC-2, connection pools
see Java Database Connection-2, connection pools
JDBC-2, drivers
see Java Database Connection-2, drivers
JMS
see Java, Messaging Service
JMS, message tuning
see Java, Messaging Service, message tuning
JNDI
see Java, Naming and Directory Interface
JNI
see Java, Native Interface
JSP
see Java, Server Page
JSP, forward
see Java, Server Page, forward
JSR 168
see Java, Specification Request 168
JSTL
see Java, Server Page, Tag Language
JVM, heap
see Java, Virtual Machine, heap
JVM, tuning
see Java, Virtual Machine, tuning

## K

kCluster 81–83
keep-alive 7–8, 13
kilobytes per second 122

## L

language as part of the cache key 34
Large Object Area, configuring 83
latency 31, 124
least recently used 38, 73, 84, 89, 99, 146, 164, 173, 189, 239
license costs 23
load 14, 16
locale 49, 123
locale-sensitive 124
long running
test 12
loop n invariation 20
LRU
see least recently used

## M

maintenance 106, 212
major increases in traffic 10
marketing
campaign 92, 101
page 6
marketing.jsp 6
maximum obtainable system output 16
maximum processing rate 16
memory
caching 26, 146
usage/dump analysis 13
mentoring 12
metadatagenerator 48, 158, 166–167
milestones 12
mini shopping cart 70, 72, 74, 226
mini-cart
see mini shopping cart
MiniShopCartDisplay.jsp 70, 116, 196, 200
missing indexes 21
modelling 12, 14
Model-View-Controller 4, 252
monitoring 39, 90, 149–151
multi-node 12

## N

name element 153
network 7, 21, 23, 84, 258
auto detect 7
bandwidth 30–31, 122
bursts 8, 18
cards 7
connection sizing 13
design 7
gateways 30
settings 13
speed 7
wide area 7
Node Group 258, 261
non-functional requirements 121
Not-value 135, 138–139, 158, 163

number of connections 8
n-way 12

## O
objectives 11–12
ODR
  see on Demand router
offload location 84, 182
on Demand router 258–259
Operating system tuning 13
OutOfMemory 80
overflow requests 10

## P
page
  fragments 5, 137, 179, 236–237, 242, 246
  modification time 30
  rarely change 5
  views per second 122
parameter-tuning 213
participants 12
password 212, 221
pathlength 14, 20
pCluster 81–83
peak loading 15
peak versus average usage 122
peer to peer 111
percentile 14–15, 236–237, 241
performance
  best practice 11
  foundation 11
  mantra 21
  metrics 216
  objectives 11
  problems 3, 7
  requirements 10–11, 122–123
  terminology 14
  tests 12, 31, 215
performExecute() method 40, 58, 68
persistence models 27
persist-to-disk property 147
personalization 72, 123
personalized
  fragments 172
  information 33, 68, 72, 126
personalized fragments 246
personnel 12
pinned objects 81–82
planning
  activity 11
  phase 10, 12
portlet caching 134–135, 140–141
predicting workload 211
primary object store 134
processing
  overhead 6
  time 15, 105, 124
ProductAttributeUpdateCmdImpl 96
ProductDisplay 54, 190–191, 204–208
production system loads 11
project
  heading for trouble 11
  inception 10
  life cycle 13
  planning 10
property sub-element 165
proxy
  caches 30
  servers 30
push 80, 116
putting items into the DynaCache 60

## Q
queue 8–10, 13, 15, 18, 21, 89, 113
  settings 13

## R
race conditions 58
RAM usage
  see random access memory, usage
random access memory
  usage 216
Rational Application Developer v6 12
redbooks Web site 273
  Contact us xiv
refactoring 20
regression testing 11
remote method invocation/internet interoperability protocol
  buffer pools 13
reorganizing disk layout 135
request
  activity 16
  attributes 42–43, 77
required element 154, 163
resource metrics 216

Archived

response time 14–15, 18–19, 122
retro-fitting 239, 243
reusability 34
reverse proxy 31
RMI/IIOP buffer pools
  see remote method invocation/internet interoperability protocol, buffer pools
roadmap 11
routers 7
rule
  based 27
  of thumb 21
rule-based invalidation 97

## S

SAN
  see storage area network
saturation point 17, 213, 217, 232
scalability 8, 14, 23, 35, 213, 217, 231
scheduled invalidation 92, 101
scheduler threads 9
secure sockets layer 8, 143
security
  implications 174
  sensitive 124
self-executing, see Java, Server Pages, self executing
serialization and de-serialization 147
server restart 38, 59, 106, 149–150
server Side Caching 30
servlet 4, 13, 27, 40–43, 52, 60–61, 68, 76–77, 134, 153–154, 165, 172–173, 189, 246
  caching 37, 140–141, 189
  filter objects 43
  filters 42–43
  responses 31
  technology 42
  thread pool 9
servlet caching 246–247
session
  dependent (SD) caching 168
  independent (SI) caching 168
shared resource 20
sharing-policy 154–155
shopping cart
  see also mini shopping cart 28, 70, 72, 74, 93, 122, 196, 215, 226, 237
shut down 38, 106, 173, 213
simple object access protocol 39, 154, 252, 254
skills 12–14
  gaps analysis 12
skip-cache 135, 137–138
Skip-cache-attribute 135
SOAP
  see simple object access protocol
software stack 180
solution
  architect 28, 134
  design 14
special events 122, 211
spill-over cache 134
SSL
  see secure sockets layer
stability 12, 106
stale
  cache items 38
  entries 92
  items 27
  objects 73
stateful session bean 109, 112
static
  content 29, 59, 172
  data 4
statistics 39, 88, 90, 188, 193, 195, 237
storage area network 174, 218
StoreCatalogDisplay 47, 51–52, 54, 56, 72–73, 78, 190
StoreCatalogProductESpot.jsp 199
Store-cookies property 135, 139
Struts 39, 44, 75–76, 129–131, 134, 190, 253
subject matter expert 12
subnet 7
switches 7, 10, 82
synchronization of resources in cache 135
system
  administrator 38
  design 11
  network 7
  Performance Tuning 12
  resources 6

## T

TCP/IP
  see transmission control protocol/internet protocol
technical sales support 14

template 86, 88–89, 102–104, 131, 147–148
terminology 3, 12, 14, 220
test xiii, 11–12, 105, 146, 209, 212, 214–215, 236, 238, 240
  environment versus production environment 212
  repeatability 216
  server 12
  tool scripting 13
Thanksgiving 15, 23
think time 122, 217, 222
thread
  dump analysis 13
  synchronization 20
throughput 14, 16–17, 122
  Plateau 16
  Saturation 17
tiers 4, 7, 10
Tiles 39, 131, 134
time based 27, 35, 248
  invalidation 74
time-out 98, 105
  settings 7
timestamp 30, 252
Tivoli Performance Viewer 151
tooling 12, 93, 152
tools and methodology 124
TopCategoriesDisplay 54, 56, 78, 190–193, 195, 199–200, 204
TopCategoriesDisplay.jsp 195
TPV
  see Tivoli Performance Viewer
traffic 7, 10, 30, 77, 106, 115, 123, 125, 211, 237–240, 259
training 13
transactional 58, 259
transfer time 15
transmission control protocol/internet protocol 7–8
triggered invalidation 75
tuning
  parameters 8–9
  points 8, 13

**U**

uncacheable 126–127, 137, 247
upper bound 135
Used Entries 188
Users
  activity 16
  arriving 16
  logging in 16
  sending requests 16

**V**

value element 163
value/not-value ranges 135, 138
Vertical Scaling 23
virtualized environment 258
volatility 6

**W**

wait time 15, 19
WAN
  see network, wide area
warm shut down 106
warm up 106, 217, 238, 240
web
  browsers 30
  server logs 125
  services xi, 62, 153–154, 166, 251–254
  site
    performance issues 5
    under stress 5
  tier 4
WebSphere
  Command Framework 56, 126, 128, 170
  Commerce
    DC_ variables 168
  Commerce Accelerator 93
  Commerce Administration Console 101
  Commerce Business Edition 35
  Commerce scheduler. 101
  Commerce Web service caching 252
  Commerce web sites xi, 288
  Dynamic Operations 258
  Extended Deployment xii, 257–261
    extended manageability 258–259
  Funnel model 8
  ND
    see WebSphere, Network Deployment
  Network Deployment 258
  Partitioning Facility 259, 261
  Portal Server 39
  Studio Application Developer 125
  Web service caching support 252
  XD



Archived

see Websphere, Extended Deployment
weekends 12
worker threads 10
workload 4, 12, 145, 211–213, 215–217, 221–222, 233, 239–240
characteristics 211, 215
WPF
see Websphere, Partitioning Facility
wsadmin 87, 118

## X

XPath 253

Archived

Archived

IBM

Redbooks

Mastering DynaCache in WebSphere Commerce

(0.5” spine)
0.475”<->0.875”
250 <-> 459 pages

Archived

Archived

Archived

# Mastering DynaCache in WebSphere Commerce

**Dramatically improve Web site performance**

**Learn from practical examples and tutorials**

**Benchmark Web site performance with and without Dynacache**

This IBM Redbook describes how to use WebSphere DynaCache to improve the performance of WebSphere Commerce Web sites.

Today's Web sites are a demanding mixture of static images surrounded by mini-shopping carts, e-marketing spots, and other eye-catching fragments, all of which change from view to view and user to user. Sites must be richly featured and personalized to attract customers – and they must deliver this content at a high level of performance as well. But the richness and personalization customers want is often the enemy of good Web site performance.

DynaCache technology gives Web site developers a robust tool for achieving excellent Web site performance. It can be applied retrospectively to existing Web sites whose performance is not meeting the owning company's requirements. It is even better applied from the beginning of a J2EE Web project, and will yield performance gains well beyond those achieved at a comparable cost by adding more hardware or rewriting the solution.

This book leads you through an explanation of what caching is, and what is special about caching Web sites. It then describes the capabilities offered by WebSphere DynaCache and how to most effectively make use of those capabilities. The discussion is enhanced by practical examples and tutorials to help you configure DynaCache and implement a sample WebSphere Commerce store. Finally, the book describes how to approach benchmarking for an online store, and how to quantify the effectiveness of a dynamic caching policy on site performance. It also presents a case study of a real-world Web site problem that was turned around by an IBM team applying DynaCache technology.

**INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION**

**BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE**

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

**For more information:**
**ibm.com**/redbooks

SG24-7393-00 ISBN 0738489522