

# Rust速習会(1)

## 入門ハンズオン

2018-08-23 @Wantedlyオフィス (白金台)

原 将己 (@qnighy)

実況用ハッシュタグ: #rust\_jp

# 目次

- Rustとは？
- Rustの資料
- Rustのセットアップ
- ハンズオン

Rustとは？

# Rustとは？

- 安全性、並行性、スピードを兼ね備えたプログラミング言語

# Rustとは？

- 安全性、並行性、**スピード**を兼ね備えたプログラミング言語

Benchmarks Gameでは**C/C++と互角**

## The Computer Language Benchmarks Game

Rust versus C gcc fastest programs

vs C vs C++ vs Go vs Java

by faster benchmark performance

<u>reverse-complement</u>	<u>mandelbrot</u>	<u>fasta</u>	<u>spectral-norm</u>	<u>binary-trees</u>
source secs n	source secs	source secs	source secs	source secs
<u>Rust</u> 1.62 995,	<u>Rust</u> 1.75	<u>Rust</u> 1.47	<u>Rust</u> 2.31	<u>Rust</u> 4.31
<u>C gcc</u> 1.75 994,	<u>C gcc</u> 1.64	<u>C gcc</u> 1.32	<u>C gcc</u> 1.99	<u>C gcc</u> 2.54

<u>pidigits</u>	<u>fannkuch-redux</u>	<u>k-nucleotide</u>	<u>n-body</u>	<u>regex-redux</u>
source secs	source secs	source secs	source secs	source secs
<u>Rust</u> 1.74	<u>Rust</u> 9.44	<u>Rust</u> 5.76	<u>Rust</u> 13.13	<u>Rust</u> 2.54
<u>C gcc</u> 1.75	<u>C gcc</u> 8.72	<u>C gcc</u> 5.07	<u>C gcc</u> 9.17	<u>C gcc</u> 1.45

# Rustとは？

- **安全性**、並行性、スピードを兼ね備えたプログラミング言語

メモリ破壊や未定義動作を網羅的に防止する機構

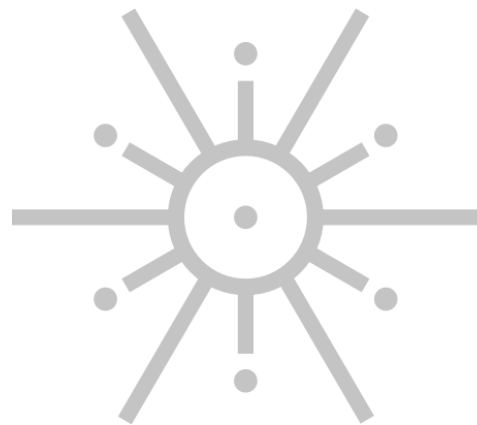
```
error[E0106]: missing lifetime specifier
--> src\main.rs:4:22
4 | fn return_stack() -> &i32 {
  |                    ^ expected lifetime parameter
= help: this function's return type contains a borrowed value, but there is no value for it to be borrowed from
= help: consider giving it a 'static lifetime
```

```
error[E0382]: use of moved value: `s`
--> src\main.rs:11:16
11 |         return (s, s);
    |                - ^ value used here after move
    |                |
    |                value moved here
= note: move occurs because `s` has type `std::string::String`, which does not implement the `Copy` trait
```

# Rustとは？

- 安全性、**並行性**、スピードを兼ね備えたプログラミング言語

正しさが証明された並列プリミティブ  
オーバーヘッドの少ない非同期I/O



Tokio

# Rustとは？

- 使いやすさ重視のコンパイラ

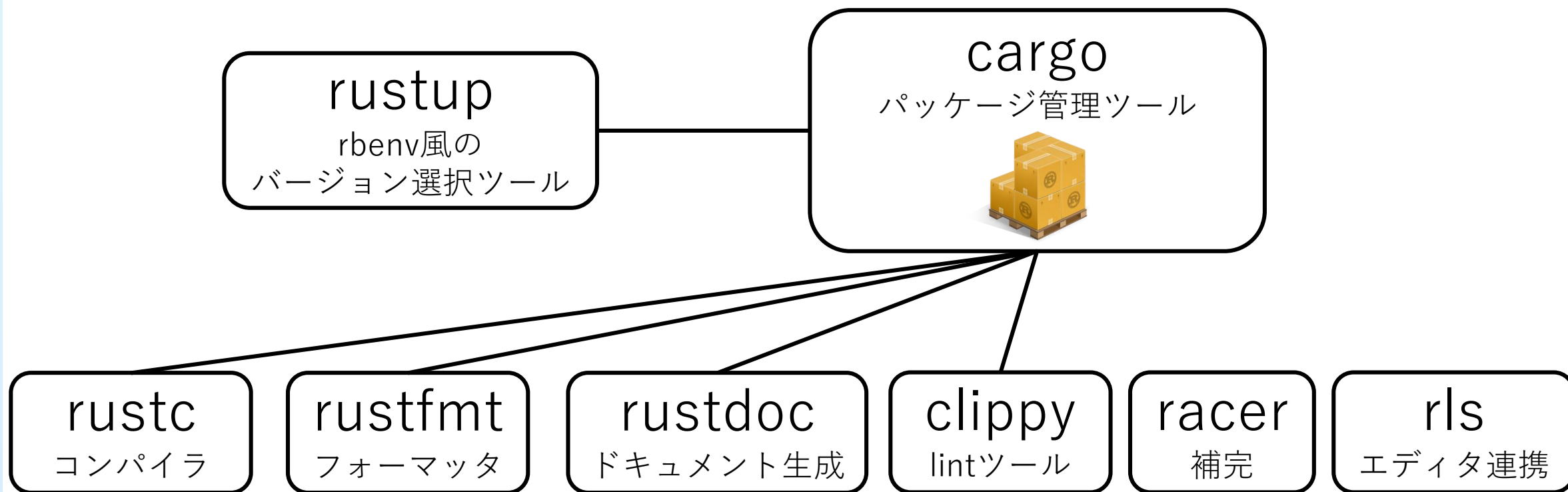
```
error[E0308]: mismatched types
--> src/main.rs:2:7
2 |     f(42);
  |     ^^^
  |     |
  |     expected enum `std::option::Option`, found integral variable
  |     help: try using a variant of the expected type: `Some(42)`
= note: expected type `std::option::Option<i32>`
       found type `{integer}`
```

```
error[E0308]: mismatched types
--> src/main.rs:2:7
2 |     f(42);
  |     ^^^
  |     |
  |     expected &i32, found integral variable
  |     help: consider borrowing here: `&42`
= note: expected type `&i32`
       found type `{integer}`
```



# Rustとは？

- 全部入りツールチェーン



# Rustとは？

- 互換性を保つためのさまざまな工夫

semver

ライブラリがsemverを  
尊重する文化

stable compiler

互換性を保証するコンパイラ

nightly compiler

非互換性と引き換えに  
最新機能を試せる

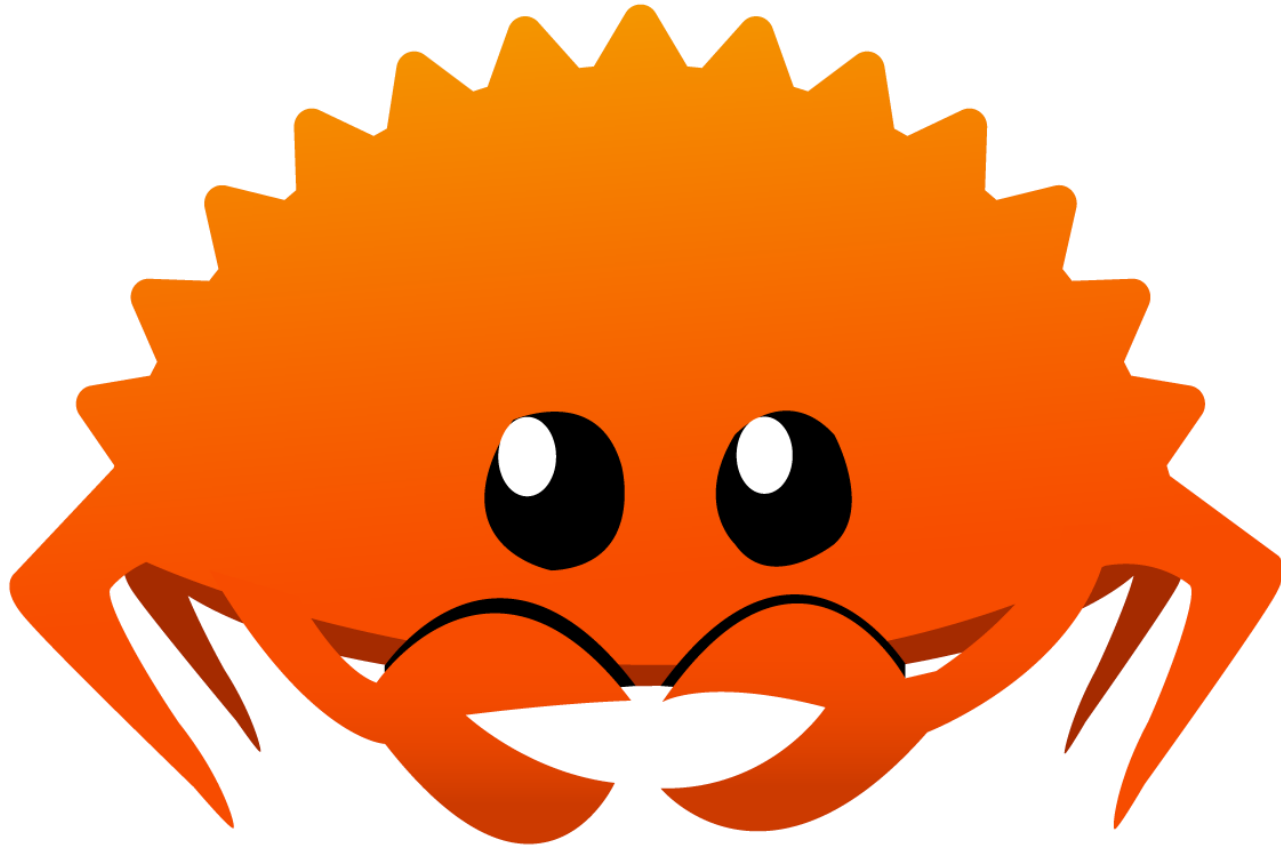
crater

全てのライブラリをコンパイル  
して、互換性を見落しを防ぐ

warning cycle

どうしても必要な破壊的変更は  
猶予期間を持つ

# Rustのマスコット (Ferris)



Crustacean (甲殻類) になぞらえて  
RustユーザーをRustaceanと呼ぶ

そのためRustのマスコットは蟹である

# Rustの資料

# Rustの資料: [doc.rust-lang.org](http://doc.rust-lang.org)

## Rust Documentation

---

Welcome to an overview of the documentation provided by the Rust project. All of these projects are managed by the Docs Team; there are other unofficial documentation resources as well!

Many of these resources take the form of "books"; we collectively call these "The Rust Bookshelf." Some are large, some are small.

## 1 Learn Rust

---

If you'd like to learn Rust, this is the spot for you! All of these resources assume that you have programmed before, but not in any specific language:

### 1.1 The Rust Programming Language

---

Affectionately nicknamed "the book," [The Rust Programming Language](#) will give you an overview of the language from first principles. You'll build a few projects along the way, and by the end, you'll have a solid grasp of the language.

### 1.2 Rust By Example

---

If reading multiple hundreds of pages about a language isn't your style, then [Rust By Example](#) has you covered. While the book talks about code with a lot of words, RBE shows off a bunch of code, and keeps the talking to a minimum. It also includes exercises!

## 2 Use Rust

# Rustの資料: `$ rustup doc`

## Rust Documentation

---

Welcome to an overview of the documentation provided by the Rust project. All of these projects are managed by the Docs Team; there are other unofficial documentation resources as well!

Many of these resources take the form of "books"; we collectively call these "The Rust Bookshelf." Some are large, some are small.

### 1 Learn Rust

---

If you'd like to learn Rust, this is the spot for you! All of these resources assume that you have programmed before, but not in any specific language:

#### 1.1 The Rust Programming Language

---

Affectionately nicknamed "the book," [The Rust Programming Language](#) will give you an overview of the language from first principles. You'll build a few projects along the way, and by the end, you'll have a solid grasp of the language.

#### 1.2 Rust By Example

---

If reading multiple hundreds of pages about a language isn't your style, then [Rust By Example](#) has you covered. While the book talks about code with a lot of words, RBE shows off a bunch of code, and keeps the talking to a minimum. It also includes exercises!

## 2 Use Rust

オフラインでもOK!!!!

# Rustの資料: [doc.rust-lang.org/nightly](https://doc.rust-lang.org/nightly)

## Rust Documentation

---

Welcome to an overview of the documentation provided by the Rust project. All of these projects are managed by the Docs Team; there are other unofficial documentation resources as well!

Many of these resources take the form of "books"; we collectively call these "The Rust Bookshelf." Some are large, some are small.

## 1 Learn Rust

---

If you'd like to learn Rust, this is the spot for you! All of these resources assume that you have programmed before, but not in any specific language:

### 1.1 The Rust Programming Language

---

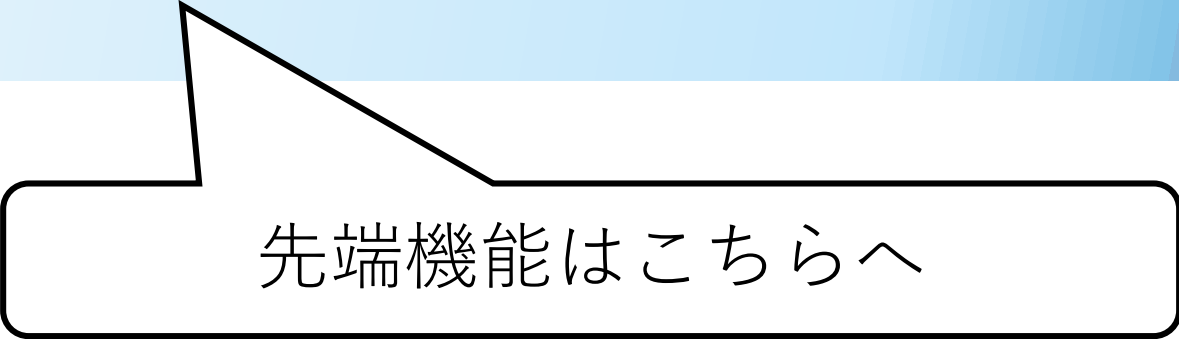
Affectionately nicknamed "the book," [The Rust Programming Language](#) will give you an overview of the language from first principles. You'll build a few projects along the way, and by the end, you'll have a solid grasp of the language.

### 1.2 Rust By Example

---

If reading multiple hundreds of pages about a language isn't your style, then [Rust By Example](#) has you covered. While the book talks about code with a lot of words, RBE shows off a bunch of code, and keeps the talking to a minimum. It also includes exercises!

## 2 Use Rust



先端機能はこちらへ

# Rustの資料: [TRPL](#)

## Foreword

### Introduction

#### 1. Getting Started

##### 1.1. Installation

##### 1.2. Hello, World!

##### 1.3. Hello, Cargo!

#### 2. Programming a Guessing Game

#### 3. Common Programming Concepts

##### 3.1. Variables and Mutability

##### 3.2. Data Types

##### 3.3. How Functions Work

##### 3.4. Comments

##### 3.5. Control Flow

#### 4. Understanding Ownership

##### 4.1. What is Ownership?

##### 4.2. References & Borrowing

##### 4.3. Slices

#### 5. Using Structs to Structure Related Data

##### 5.1. Defining and Instantiating Structs

##### 5.2. An Example Program Using Structs

##### 5.3. Method Syntax

## Foreword

It wasn't always so clear, but the no matter what kind of code you with confidence in a wider varie

Take, for example, "systems-level data representation, and concurrency accessible only to a select few with pitfalls. And even those who practice crashes, or corruption.

Rust breaks down these barriers of tools to help you along the way. can do so with Rust, without taking having to learn the fine points of naturally towards reliable code.

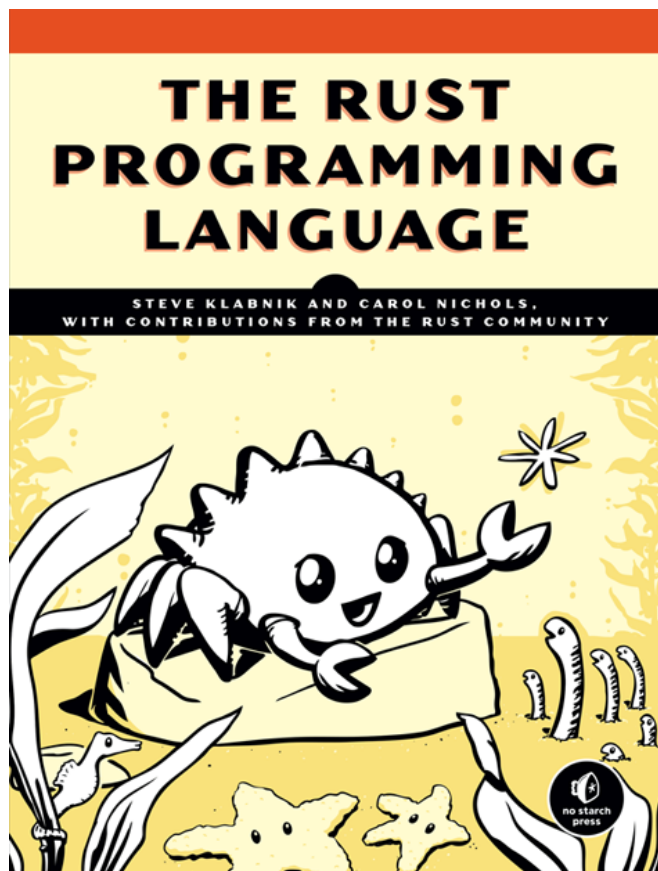
Programmers who are already vers example, introducing parallelism classical mistakes for you. And you confidence that you won't accid

But Rust isn't limited to low-level

# 公式の教科書



# Rustの資料: [TRPL 書籍版](#)



公式の教科書  
の公式の書籍版

# Rustの資料: TRPL日本語版

Last Commit Date of Markdown Sources: Tue Aug 7 00:39:47 2018 +0900

i

## まえがき

すぐにはわかりにくいかもしれませんが、Rust プログラミング言語は、エンパワーメント (empowerment) を根本原理としています: どんな種類のコードを現在書いているにせよ、Rust は幅広い領域で以前よりも遠くへ到達し、自信を持ってプログラムを組む力を与え (empower) ます。

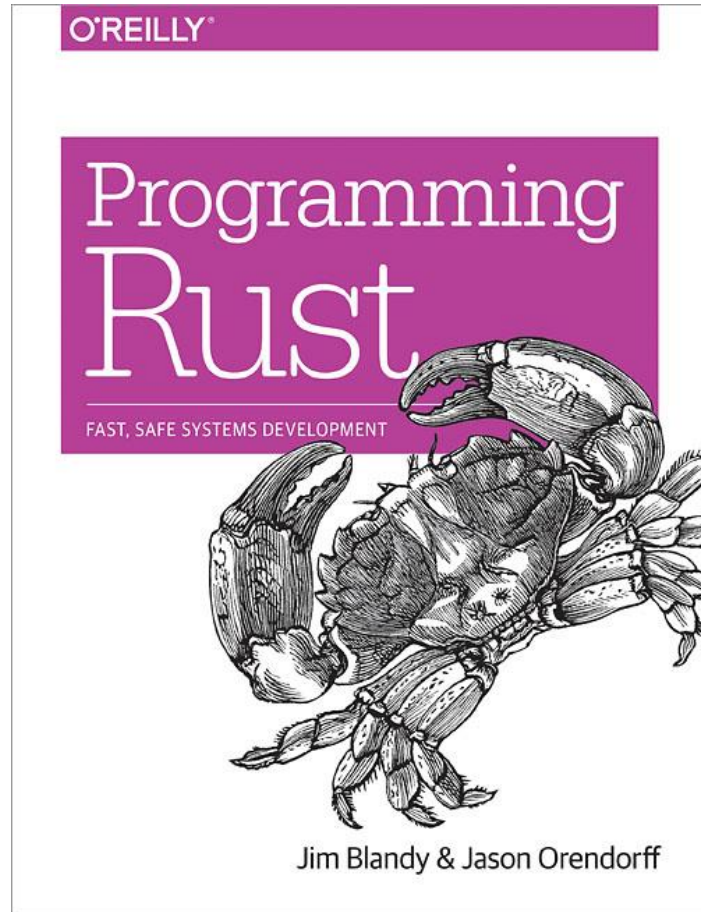
一例を挙げると、メモリ管理やデータ表現、並列性などの低レベルな詳細を扱う「システムレベル」のプログラミングがあります。伝統的にこの分野は難解で、年月をかけてやっかいな落とし穴を回避する術を習得した選ばれし者にだけ可能と見なされています。そのように鍛錬を積んだ者でさえ注意が必要で、さもないと書いたコードがクラッキングの糸口になったりクラッシュやデータ破損を引き起こしかねないのです。

この難しさを取り除くために、Rust は、古い落とし穴を排除し、その過程で使いやすく役に立つ洗練された一連のツールを提供します。低レベルな制御に「下がる」必要があるプログラマは、お決まりのクラッシュやセキュリティホールをリスクを負わず、気まぐれなツールチェーンのデリケートな部分を学ぶ必要なく Rust で同じことができます。さらにいいことに、Rust は、スピードとメモリ使用の観点で効率的な信頼性の高いコードへと自然に導くよう設計されています。

低レベルな制御を取り組むのに、Rust を使用し、プログラマは、スピードとメモリ使用の観点で効率的な信頼性の高いコードへと自然に導くよう設計されています。

## 公式の教科書 の非公式訳

# Rustの資料: 蟹 (オライリー)



# Rustの資料: 日本語の蟹 (オライリー)



# Rustの資料: [keen](#)さんの記事

## Hello World

さっそくRustのコードを書いてみましょう。最初はもちろん「Hello, World」です。

次のコードを、`hello.rs` という名前で保存してください。

```
fn main() {  
    println!("Hello, World");  
}
```

このRustコードをコンパイルするには、次のように `rustc` コマンドを実行します。

```
$ rustc hello.rs
```

これで、同じディレクトリ内に `hello` という実行可能なバイナリファイルができています。

この `hello` をシェルから実行すれば、`Hello, World` と表示されます。

```
$/hello  
Hello, World  
$
```

# Rustの資料: [Rust By Example](#)

## 3.1. Structures

### 3.2. Enums

#### 3.2.1. use

#### 3.2.2. C-like

#### 3.2.3. Testcase: linked-list

### 3.3. constants

## 4. Variable Bindings

### 4.1. Mutability

### 4.2. Scope and Shadowing

### 4.3. Declare first

## 5. Types

### 5.1. Casting

### 5.2. Literals

### 5.3. Inference

### 5.4. Aliasing

## 6. Conversion



Rust By Example

## Structures

There are three types of structures ("structs") that can be created using Rust:

- Tuple structs, which are, basically, named tuples.
- The classic [C structs](#)
- Unit structs, which are field-less, are useful for generics.

```
#[derive(Debug)]
struct Person<'a> {
    name: &'a str,
    age: u8,
}

// A unit struct
struct Nil;

// A tuple struct
struct Pair(i32, f32);

// A struct with two fields
```

# Rustの資料: stdのドキュメント



Crate std

Version 1.28.0 (9634041f0  
2018-07-30)

See all std's items

Primitive Types

Modules

Macros

## Crates

alloc

core

proc\_macro

**std**

std\_unicode

test



Click or press 'S' to search, '?' for more options...

## Crate std

### [\[-\] The Rust Standard Library](#)

The Rust Standard Library is the foundation of portable Rust software, broader Rust ecosystem. It offers core types, like `Vec<T>` and `Option`, standard macros, I/O and multithreading, among many other things.

`std` is available to all Rust crates by default, just as if each one contained it. Therefore the standard library can be accessed in `use` statements through the absolute path `::std`, as in `::std::env::args`.

### [How to read this documentation](#)

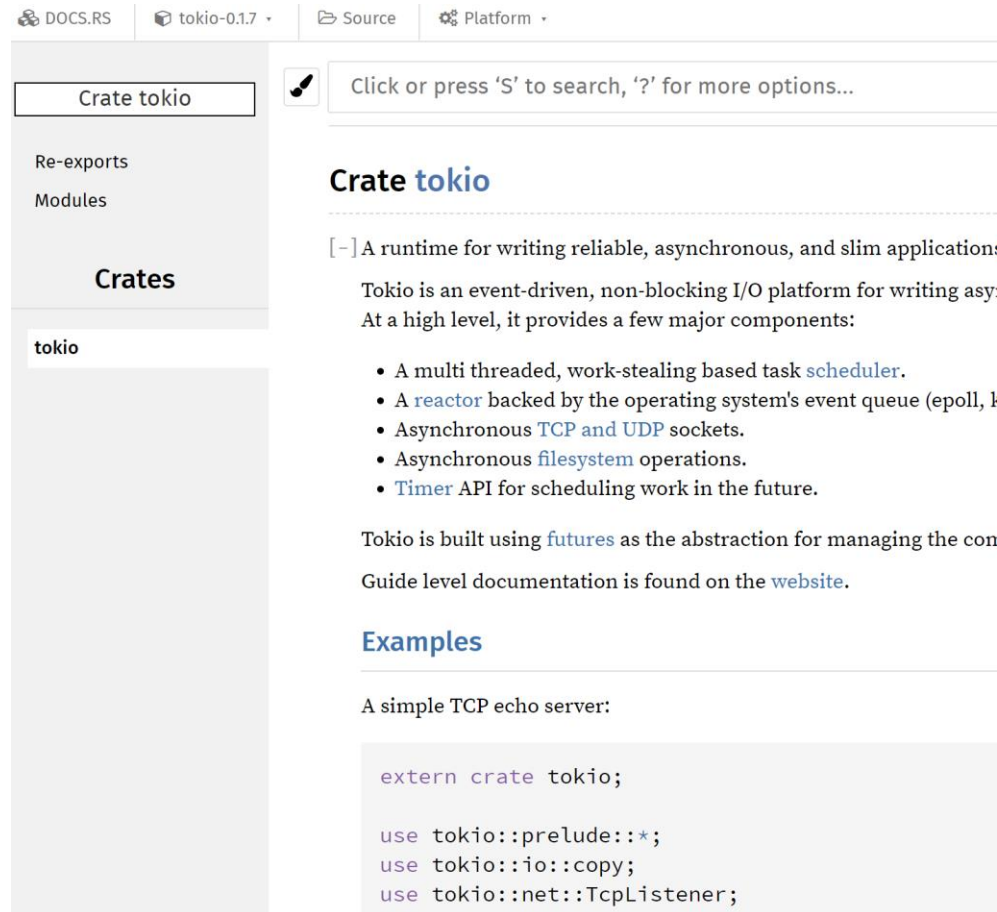
If you already know the name of what you are looking for, the fastest way is to search for it. Otherwise, you may want to jump to one of these useful sections:

- [std::\\* modules](#)
- [Primitive types](#)
- [Standard macros](#)
- [The Rust Prelude](#)

If this is your first time, the documentation for the standard library is very helpful and should generally lead you to interesting places. Still, there are important things to know about the standard library and its documentation!

Once you are familiar with the contents of the standard library you may want to explore the

# Rustの資料: [docs.rs](https://docs.rs)



The screenshot shows the docs.rs website interface. At the top, there are navigation links for 'DOCS.RS', 'tokio-0.1.7', 'Source', and 'Platform'. Below this is a search bar with a magnifying glass icon and the text 'Click or press 'S' to search, '?' for more options...'. The main content area is titled 'Crate tokio' and includes a search icon. On the left sidebar, there are sections for 'Re-exports', 'Modules', 'Crates', and 'tokio'. The main content area contains the following text:

## Crate tokio

[ - ] A runtime for writing reliable, asynchronous, and slim application:

Tokio is an event-driven, non-blocking I/O platform for writing asynchronous applications. At a high level, it provides a few major components:

- A multi threaded, work-stealing based task [scheduler](#).
- A [reactor](#) backed by the operating system's event queue (epoll, kqueue, etc).
- Asynchronous [TCP and UDP](#) sockets.
- Asynchronous [filesystem](#) operations.
- [Timer](#) API for scheduling work in the future.

Tokio is built using [futures](#) as the abstraction for managing the control flow.

Guide level documentation is found on the [website](#).

## Examples

A simple TCP echo server:

```
extern crate tokio;

use tokio::prelude::*;
use tokio::io::copy;
use tokio::net::TcpListener;
```

“docs.rs/パッケージ名”  
で飛べる



# Rustの資料: Cargo Book

## Introduction

### 1. Getting Started

- 1.1. Installation
- 1.2. First Steps with Cargo

### 2. Cargo Guide

- 2.1. Why Cargo Exists
- 2.2. Creating a New Project
- 2.3. Working on an Existing Project
- 2.4. Dependencies
- 2.5. Project Layout
- 2.6. Cargo.toml vs Cargo.lock
- 2.7. Tests
- 2.8. Continuous Integration
- 2.9. Build Cache

### 3. Cargo Reference

- 3.1. Specifying Dependencies
- 3.2. The Manifest Format
- 3.3. Configuration
- 3.4. Environment Variables
- 3.5. Build Scripts
- 3.6. Publishing on crates.io
- 3.7. Package ID Specifications
- 3.8. Source Replacement
- 3.9. External Tools



## The Cargo Book



Cargo is the [Rust package manager](#). Cargo downloads your project, makes packages, and upload them to [crates.io](#). You can contribute to this book on [GitHub](#).

### Sections

#### Getting Started

To get started with Cargo, install Cargo (and Rust) and

The

## Cargo.tomlの書き方とか

# Rustの資料: The Reference

## Introduction

1. Notation
2. Lexical structure
  - 2.1. Input format
  - 2.2. Keywords
  - 2.3. Identifiers
  - 2.4. Comments
  - 2.5. Whitespace
  - 2.6. Tokens
  - 2.7. Paths
3. Macros
  - 3.1. Macros By Example
  - 3.2. Procedural Macros
4. Crates and source files
5. Items and attributes
  - 5.1. Items
    - 5.1.1. Modules
    - 5.1.2. Extern crates
    - 5.1.3. Use declarations
    - 5.1.4. Functions
    - 5.1.5. Type aliases
    - 5.1.6. Structs
    - 5.1.7. Enumerations
    - 5.1.8. Unions
    - 5.1.9. Constant items

For now, this reference is a best-effort document. We strive for docs and lang teams will work together to figure out how best to handle something wrong or missing, file an [issue](#) or send in a pull request.



## Block expressions

### Syntax

```
BlockExpression :  
{  
  InnerAttribute*  
  Statement*  
  Expression?  
}
```

A *block expression* is similar to a module in that it can contain *statements* and end with an *expression* in its namespace scope. Use items can bring new items into the block itself.

A block will execute each statement sequentially. If a block doesn't end in an expression, its value is the value of the last statement.

```
let x: () = { println!("Hello."); };
```

If it ends in an expression, its value and type are the value and type of that expression.

## 辞書

細かい仕様が書いてある  
しかし、未だにコンパイラ  
のほう詳しい

# Rustの資料: The Rustonomicon

Introduction

1. Meet Safe and Unsafe

- 1.1. How Safe and Unsafe Interact
- 1.2. What Unsafe Can Do
- 1.3. Working with Unsafe

2. Data Layout

- 2.1. repr(Rust)
- 2.2. Exotically Sized Types
- 2.3. Other reprs

3. Ownership

- 3.1. References
- 3.2. Aliasing
- 3.3. Lifetimes
- 3.4. Limits of Lifetimes
- 3.5. Lifetime Elision
- 3.6. Unbounded Lifetimes
- 3.7. Higher-Rank Trait Bounds
- 3.8. Subtyping and Variance
- 3.9. Drop Check
- 3.10. PhantomData
- 3.11. Splitting Borrows

4. Type Conversions

- 4.1. Coercions
- 4.2. The Dot Operator



## Drop Check

We have seen how lifetimes provide dangling references. However up to relationship in an inclusive manner *exactly* as long as 'b'. At first glance dropped at the same time as another statements:

```
let x;  
let y;
```

```
{  
  let x;  
  {  
    let y;  
  }  
}
```

Each creates its own scope, clearly do the following?

```
let (x, y) = (vec![], vec![]);
```

Does either value strictly outlive the other. Of course, one of x or y will live. Tuples aren't special in this regard; as of Rust 1.0.

# unsafeを触る前に読む本 (むずかしい)

# Rustの資料: [unstable-book](#)

The Unstable Book

1. Compiler flags
  - 1.1. linker\_flavor
  - 1.2. profile
2. Language features
  - 2.1. aarch64\_target\_feature
  - 2.2. abi\_msp430\_interrupt
  - 2.3. abi\_ptx
  - 2.4. abi\_thiscall
  - 2.5. abi\_unadjusted
  - 2.6. abi\_vectorcall
  - 2.7. abi\_x86\_interrupt
  - 2.8. allocator\_internals
  - 2.9. allow\_fail
  - 2.10. allow\_internal\_unsafe
  - 2.11. allow\_internal\_unstable
  - 2.12. arbitrary\_self\_types
  - 2.13. arm\_target\_feature
  - 2.14. asm
  - 2.15. associated\_type\_defaults
  - 2.16. attr\_literals
  - 2.17. avx512\_target\_feature
  - 2.18. box\_patterns
  - 2.19. box\_syntax
  - 2.20. catch\_expr



## crate\_visibility\_m

The tracking issue for this feature is: [#453](#)

The `crate_visibility_modifier` feature modifier synonymous to `pub(crate)`, ind entire enclosing crate, but not to other cr

```
#![feature(crate_visibility_modifier)
crate struct Foo {
    bar: usize,
}
```



先端機能を知りたい人向け

# Rustの資料: [edition-guide](#)

Introduction

1. What are editions?

1.1. Transitioning your code to a new edition

2. Rust 2015

3. Rust 2018

3.1. Module system

3.1.1. Raw identifiers

3.1.2. Path clarity

3.1.3. More visibility modifiers

3.1.4. Nested imports with use

3.2. Error handling and panics

3.2.1. The ? operator for easier error handling

3.2.2. ? in main and tests

3.2.3. Controlling panics with std::panic

3.2.4. Aborting on panic

3.3. Control flow

3.3.1. Loops can break with a value

3.3.2. async/await for easier concurrency

3.4. Trait system

3.4.1. impl Trait for returning complex types with ease



## Raw identifiers

Minimum Rust Version **nightly**

Rust, like many programming languages, has something to the language, and so you can't name, and other places. Raw identifiers let you allow.

For example, `match` is a keyword. If you try to

```
fn match(needle: &str, haystack: &str)
    haystack.contains(needle)
}
```

You'll get this error:

```
error: expected identifier, found keyword
--> src/main.rs:4:4
   |
4  | fn match(needle: &str, haystack: &str
   |      ^^^^^ expected identifier, found
```

You can write this with a raw identifier:

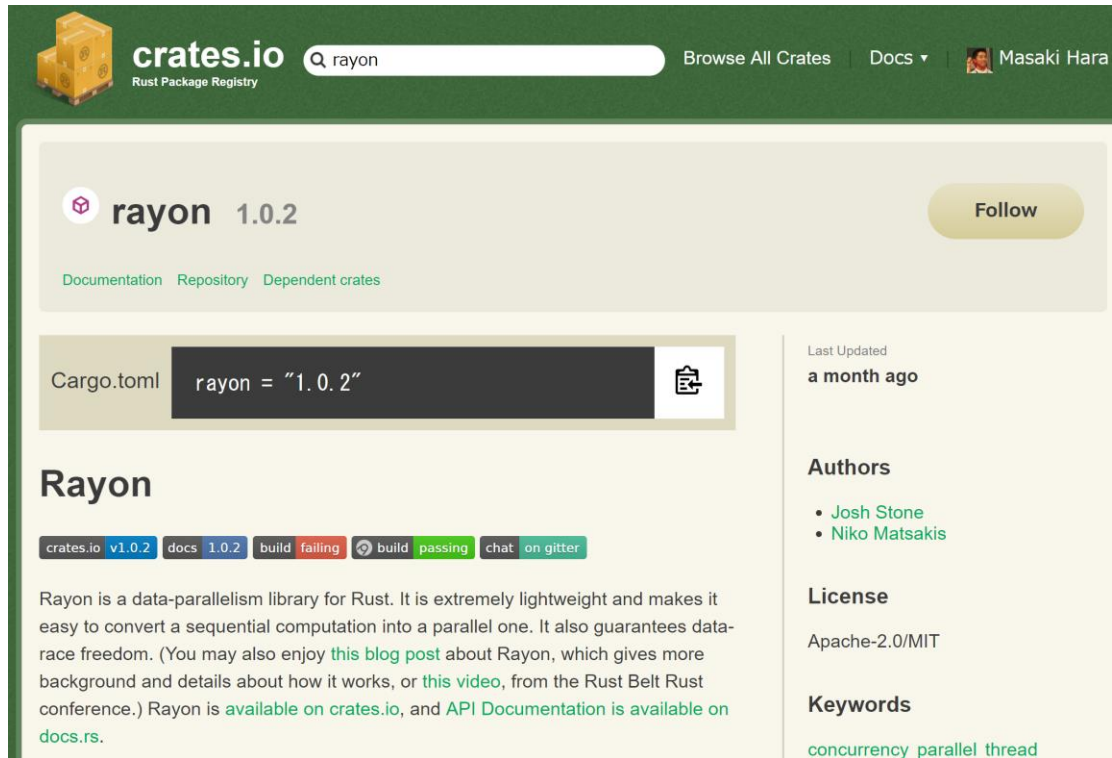
```
#![feature(rust_2018_preview)]
#![feature(raw_identifiers)]

fn r#match(needle: &str, haystack: &str)
    haystack.contains(needle)
}
```

“Rust 2018”の変更点が解説されている

Rustを昔触ってた人にも有益

# Rustの資料: [crates.io](https://crates.io)

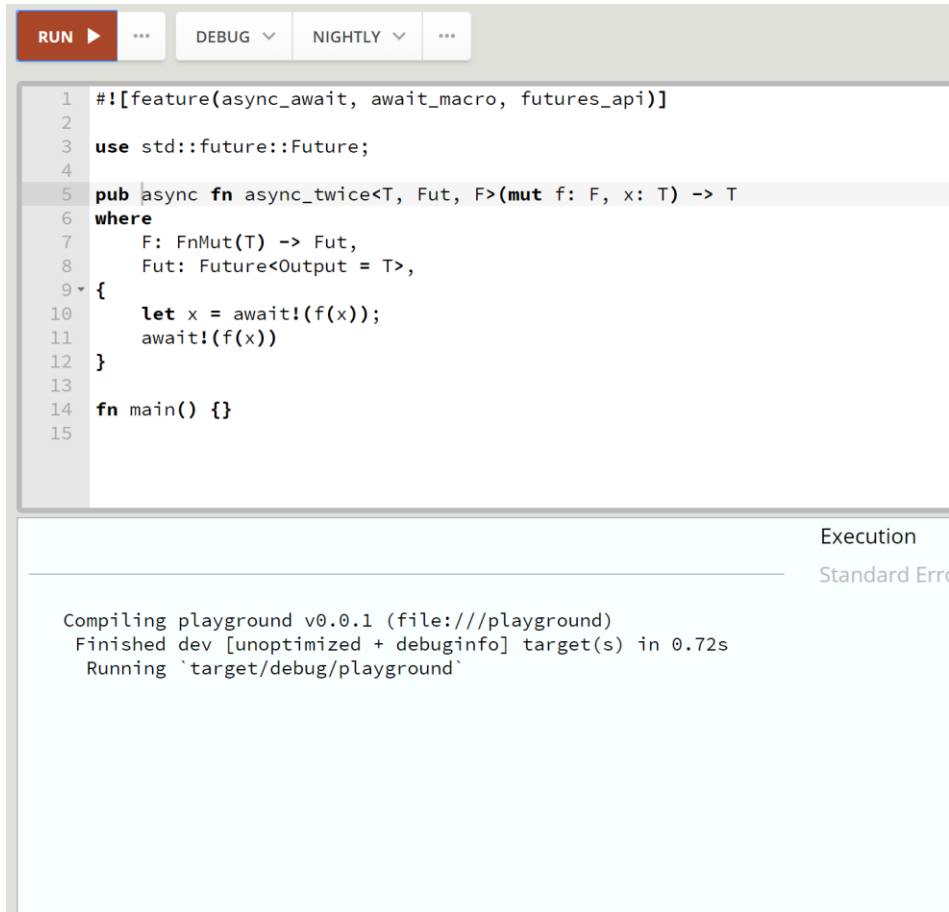


The screenshot shows the crates.io website interface. At the top, there is a search bar with "rayon" entered, and navigation links for "Browse All Crates", "Docs", and a user profile for "Masaki Hara". The main content area features the "rayon 1.0.2" package card, which includes a "Follow" button and links for "Documentation", "Repository", and "Dependent crates". Below the package name, there is a "Cargo.toml" section showing the dependency declaration: `rayon = "1.0.2"`. The package description states: "Rayon is a data-parallelism library for Rust. It is extremely lightweight and makes it easy to convert a sequential computation into a parallel one. It also guarantees data-race freedom. (You may also enjoy [this blog post](#) about Rayon, which gives more background and details about how it works, or [this video](#), from the Rust Belt Rust conference.) Rayon is [available on crates.io](#), and [API Documentation](#) is available on [docs.rs](#)." The right sidebar provides additional details: "Last Updated a month ago", "Authors" (Josh Stone, Niko Matsakis), "License" (Apache-2.0/MIT), and "Keywords" (concurrency, parallel, thread).

Rustのパッケージは  
中央集権的

公式サイトでは検索のほか  
自作パッケージの  
アップロードができる

# Rustの資料: [Rust Playground \(playpen\)](#)



The screenshot shows the Rust Playground interface. At the top, there are buttons for 'RUN', 'DEBUG', and 'NIGHTLY'. Below this is a code editor with the following Rust code:

```
1 #![feature(async_await, await_macro, futures_api)]
2
3 use std::future::Future;
4
5 pub async fn async_twice<T, Fut, F>(mut f: F, x: T) -> T
6 where
7     F: FnMut(T) -> Fut,
8     Fut: Future<Output = T>,
9 {
10     let x = await!(f(x));
11     await!(f(x))
12 }
13
14 fn main() {}
15
```

Below the code editor, there is an 'Execution' section with the following output:

```
Standard Error
Compiling playground v0.0.1 (file:///playground)
Finished dev [unoptimized + debuginfo] target(s) in 0.72s
Running `target/debug/playground`
```

簡単なプログラムは  
Webで試せる

制約

- 1ファイル
- stdと人気上位100パッケージのみ使える

# Rustの資料: [Rust Playground \(playpen\)](#)

```
RUN ▶ ... DEBUG ▾ NIGHTLY ▾ ...
1  #![feature(async_await, await_macro, futures_api)]
2
3  use std::future::Future;
4
5  pub async fn async_twice<T, Fut, F>(mut f: F, x: T) -> T
6  where
7      F: FnMut(T) -> Fut,
8      Fut: Future<Output = T>,
9  {
10     let x = await!(f(x));
11     await!(f(x))
12 }
13
14 fn main() {}
15
```

Execution  
Standard Error

```
Compiling playground v0.0.1 (file:///playground)
Finished dev [unoptimized + debuginfo] target(s) in 0.72s
Running `target/debug/playground`
```

同じサービスがInteger32, LLC.  
によってもホストされている



# Rustの資料: The Rust Community

[Documentation](#)[Install](#)[Community](#)[Contribute](#)

コミュニティーの集まる  
ところ色々へのリンク

---

## The Rust Community

The Rust programming language has many qualities, but Rust's greatest strength is the community of people who come together to make working in Rust a rewarding experience.

We are committed to providing a friendly, safe and welcoming environment for all, regardless of gender, sexual orientation, disability, ethnicity, religion, or similar personal characteristic. Our [code of conduct](#) sets the standards for behavior in all official Rust forums.

If you feel you have been or are being harassed or made uncomfortable by a community member, please [contact](#) any of the [Rust Moderation Team](#) immediately. Whether you are a regular contributor or a newcomer, we care about making the community a safe space for you.

---

## Getting Started

The most important community resources for those new to Rust are:

- [#rust-beginners](#), an IRC channel that loves answering questions at any depth.
- The [Users Forum](#), for discussion of all things Rust.

You may also find help on the question and answer site, [Stack Overflow](#).

---

## News

# Rustの資料: [Rust Blog](#)



The Rust Programming Language Blog

## Posts

Aug 8, 2018

[Launching the 2018 State of Rust Survey](#)

Aug 2, 2018

[Announcing Rust 1.28](#)

Jul 27, 2018

[What is Rust 2018?](#)

Jul 20, 2018

[Announcing Rust 1.27.2](#)

Jul 10, 2018

[Announcing Rust 1.27.1](#)

Jul 6, 2018

[Security Advisory for rustdoc](#)

# Rustの資料: This Week in Rust

## This Week in Rust

---

21 AUG 2018

[This Week in Rust 248](#)

Hello and welcome to another issue of *This Week in Rust!* Rust is a systems language pursuing the trifecta: safety, concurrency, and speed. This is a weekly summary of its progress and community. Want something mentioned? Tweet us at [@ThisWeekInRust](#) or send us a pull request. Want to get involved? [We love contributions.](#)

*This Week in Rust* is openly developed on GitHub. If you find any errors in this week's issue, [please submit a PR.](#)

### Updates from Rust Community

#### News & Blog Posts

- [Announcing Rust 2018 Preview 2.](#) [\[discuss\]](#)
- [Announcing the RLS 1.0 release candidate.](#) [\[discuss\]](#)
- [How Rust's standard library was vulnerable for years and nobody noticed.](#) [\[discuss\]](#)
- [Safe partial initialization in Rust.](#) [\[discuss\]](#)
- [With undefined behavior, anything is possible.](#) [\[discuss\]](#)
- [Rust GraphQL webserver with Warp, Juniper, and MongoDB.](#) [\[discuss\]](#)
- [Programming Servo: Anatomy of a fetch.](#) [\[discuss\]](#)
- [Thanks for asking.](#) An analysis of questions that are asked on r/rust subreddit. [\[discuss\]](#)
- [This week in Rust and WebAssembly 6.](#) [\[discuss\]](#)
- [\[podcast\] New Rustacean news: Rust 1.28.](#) [\[discuss\]](#)

#### Create of the Week

---

## 週刊Rust


コンパイラの更新から新着ブログ記事まで、その週の最新情報が手に入る

# Rustの資料: Fearless Rust Bloggers


Fearless Rust Bloggers	
Dedicated Rust Bloggers	Rust News That Needs Advice:
<a href="#">'static Blog</a>	<a href="#">Jekyll RSS on Github</a>
<a href="#">A rust/piston game developing tutorial</a>	<a href="#">Hugo RSS by default. Read</a>
<a href="#">Aaron Turon</a>	
<a href="#">Adam Perry</a>	<a href="http://brson.github.io">http://brson.github.io</a>
<a href="#">Air Mozilla</a>	<a href="http://featherweightmusings">http://featherweightmusings</a>
<a href="#">Alberto Ruiz - Silicon Island</a>	<a href="http://seanmonstar.com/tag/">http://seanmonstar.com/tag/</a>
<a href="#">Alexis Beingsner</a>	<a href="http://www.willusher.io/archi">http://www.willusher.io/archi</a>
<a href="#">Amethyst Game Engine</a>	<a href="https://blog.guillaume-gome">https://blog.guillaume-gome</a>
<a href="#">Andrew Gallant : BurntSushi</a>	<a href="https://changelog.com/topic">https://changelog.com/topic</a>
<a href="#">aochagavia's blog</a>	<a href="https://chr4.org/blog/catego">https://chr4.org/blog/catego</a>
<a href="#">Are we web yet?</a>	<a href="https://gist.github.com/brson/">https://gist.github.com/brson/</a>
<a href="#">Armin Ronacher</a>	<a href="https://github.com/brson/rus">https://github.com/brson/rus</a>
<a href="#">Baby Steps</a>	<a href="https://github.com/Vagdish/">https://github.com/Vagdish/</a>
<a href="#">Barely Functional - mgattozzi</a>	<a href="https://hashnode.com/n/rus">https://hashnode.com/n/rus</a>
<a href="#">benjaminFRY as blueJEKYLL;</a>	<a href="https://jamey.thesharps.us/s">https://jamey.thesharps.us/s</a>
<a href="#">bheisler.github.io</a>	<a href="https://lab.whitequark.org/n">https://lab.whitequark.org/n</a>
<a href="#">Bibhas Bhattacharya : mobiarch</a>	<a href="https://lukaskalbertodt.githu">https://lukaskalbertodt.githu</a>
<a href="#">blog.rust-lang.org</a>	<a href="https://maniagnosis.crsr.net">https://maniagnosis.crsr.net</a>
<a href="#">Bloggedy blog</a>	<a href="https://polyfractal.com/cater">https://polyfractal.com/cater</a>
<a href="#">bluss' rust bursts</a>	<a href="https://quietmisdreavus.net/">https://quietmisdreavus.net/</a>



Rust関連のRSSを集めたリスト  
OPMLで一括購読できる



# Rustの資料: Rust subreddit



 [注目](#) [新着](#) [上昇度](#) [論争中](#) [トップ](#) [ゴールド](#)



Please read [The Rust Community Code of Conduct](#)


 **【HighLow取引】バイナリーオプション戦略で役立つ五つのヒント**  
thepayout による紹介  
[スポンサードリンク](#) [保存](#) [goldを贈る](#) [問題を報告](#)



 **Hey Rustaceans! Got an easy question? Ask here (34/2018)!** (self.rust)  
 [llogiq](#) [clippy](#) [twir](#) [rust](#) [mutagen](#) [flamer](#) [overflow](#) [bytecount](#) [M](#) が 2日前 \* 投稿 - announcement  
[80個のコメント](#) [シェアする](#) [保存](#) [非表示](#) [goldを贈る](#) [問題を報告](#) [crosspost](#)

 **What's everyone working on this week (34/2018)?** (self.rust)  
 [llogiq](#) [clippy](#) [twir](#) [rust](#) [mutagen](#) [flamer](#) [overflow](#) [bytecount](#) [M](#) が 2日前 投稿 - announcement  
[42個のコメント](#) [シェアする](#) [保存](#) [非表示](#) [goldを贈る](#) [問題を報告](#) [crosspost](#)

 **Idea for a scalable, code review/trust system not only for Rust** (self.rust)  
 dpc\_pw が 3時間前 \* 投稿  
[5個のコメント](#) [シェアする](#) [保存](#) [非表示](#) [goldを贈る](#) [問題を報告](#) [crosspost](#)

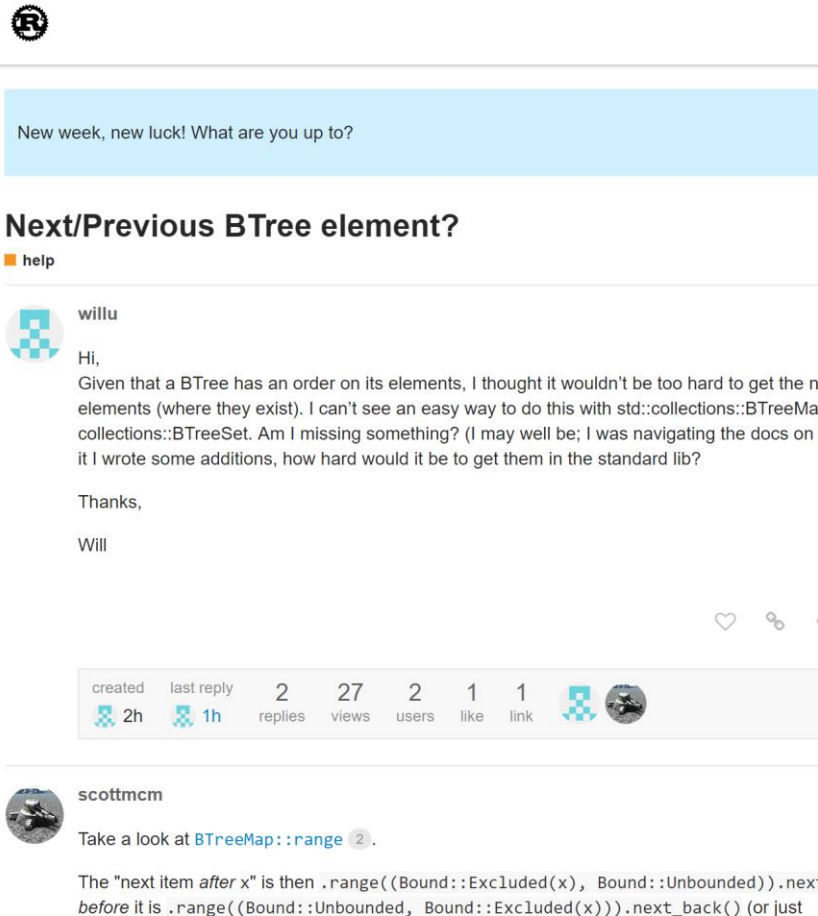
 **Upenn: Rust Programming Course this Fall!** (self.rust)  
 gatoWololo が 8時間前 投稿  
[7個のコメント](#) [シェアする](#) [保存](#) [非表示](#) [goldを贈る](#) [問題を報告](#) [crosspost](#)

 **This Week in Rust 248** (this-week-in-rust.org)  
nasa42 [twir](#) が 6時間前 投稿  
[6個のコメント](#) [シェアする](#) [保存](#) [非表示](#) [goldを贈る](#) [問題を報告](#) [crosspost](#)

 **CVE-2018-1000657: buffer overflow in VecDeque::reserve() in Rust 1. execution** (cve.mitre.org)  
 Shnatsel が 21時間前 投稿  
[63個のコメント](#) [シェアする](#) [保存](#) [非表示](#) [goldを贈る](#) [問題を報告](#) [crosspost](#)

# Rustの資料: [Rust user forum](#)


## Discourse上の公式フォーラム 質問とかアナウンスとか



New week, new luck! What are you up to?

### Next/Previous BTree element?

help


 willu

Hi,  
Given that a BTree has an order on its elements, I thought it wouldn't be too hard to get the n elements (where they exist). I can't see an easy way to do this with `std::collections::BTreeMap` or `std::collections::BTreeSet`. Am I missing something? (I may well be; I was navigating the docs on it I wrote some additions, how hard would it be to get them in the standard lib?)

Thanks,

Will

created 2h last reply 1h 2 replies 27 views 2 users 1 like 1 link

 scottmcm

Take a look at `BTreeMap::range`.

The "next item after x" is then `.range((Bound::Excluded(x), Bound::Unbounded)).next` before it is `.range((Bound::Unbounded, Bound::Excluded(x))).next_back()` (or just

# Rustの資料: Rust 日本語Slack

rust-jp ▾  
● qnighy

Jump to...

All Unreads  
All Threads

Channels +

# code  
# event  
# feed  
# general  
# random

Direct Messages +

slackbot  
● qnighy (you)  
○ dorayakikun

Apps +

#code

☆ | 👤 376 | ✨ 0 | コードを投げて質問しよう

[rustのエラーメッセージの意味がわかりません | teratail](#) — Mond  
前提・実現したいことrustのエラーメッセージの意味がわかりません。シエルから以下のように引数を評価して、+ 1 して表示させるプログラムです。cargo run 123 124 なにが問題なのでしょうか？ 発生している問題・エラーメッセージ

★ teratail[テラテイル]  
[rustのエラーメッセージの意味がわかりません | teratail](#)  
前提・実現したいことrustのエラーメッセージの意味がわかりません。シエルから以下のように引数を評価して、+ 1 して表示させるプログラムです。cargo run 123 124 なにが問題なのでしょうか？ 発生している問題・エラーメッセージ



101 9:28 AM

これだと通りますね。

```
fn main() {  
    let arg: Vec<String> = env::args().collect();  
    println!("{}", add1(&arg[1]));  
}  
  
fn add1(s: &String) -> i32 {  
    let a:i32 = s.parse().unwrap();  
    return a + 1  
}
```

Vec(string)の要素はそのままだと所有権を渡せないとかそんな感じかな



tatsuya6502 9:49 AM

後ほど teratail の方に回答しますね



1 reply 2 days ago

質問・イベント情報・  
翻訳など

# Rustの資料: Stackoverflow

## Questions tagged [rust]

Ask Question

Rust is a language designed for writing highly reliable and fast software in a simple way. It can be used from high-level code down to hardware-specific code, and from big irons to tiny devices.

Watch Tag

Ignore Tag

[Learn more...](#) [Improve tag info](#) [Top users](#) [Synonyms](#)

9,844 questions


[Info](#) [Newest](#) [Frequent](#) [Votes](#) [Active](#) [Unanswered](#)

### 2 votes [Is it possible to group items that need the same attribute?](#)

This is more a style question but still interesting. Is it possible to group things so the attribute is only there once? For instance, in the following code I am using the same attribute three times: ...

1 answer

rust

asked 8 mins ago  
 [purpletentacle](#)  
2,374 ● 1 ● 23 ● 34


7 views

### -3 votes [How to check the size of a vector of char? \[duplicate\]](#)

In Java there is a function called `.length()` that allows you to check the size without a doubt. If I remember correctly, you use `usize` in C++. However I am not able to find such a function in Rust. ...

0 answers

rust

asked 6 hours ago  
 [Adam Palmkvist](#)  
31 ● 2

21 views

### -3 votes [How do I parse a vector into a function?](#)

The idea is to send a set of characters of a vector and let the function display the current correct guesses. Here is ...

Shepmaster氏という守り神がいる  
※Rust開発チームの一人でInteger32, LLC. の共同創業者



# Rustの資料: [Rust RFCs](#)

- Feature Name: (fill me in with a unique ident, my\_awesome\_feature)
- Start Date: 2017-08-02
- RFC PR: [rust-lang/rfcs#2094](#)
- Rust Issue: [rust-lang/rust#44928](#)

## Summary

Extend Rust's borrow system to support **non-lexical lifetimes** -- these are lifetimes that are based on the control-flow graph, rather than lexical scopes. The RFC describes in detail how to infer these new, more flexible regions, and also describes how to adjust our error messages. The RFC also describes a few other extensions to the borrow checker, the total effect of which is to eliminate many common cases where small, function-local code modifications would be required to pass the borrow check. (The appendix describes some of the remaining borrow-checker limitations that are not addressed by this RFC.)

## Motivation

### What is a lifetime?

The basic idea of the borrow checker is that values may not be mutated or moved while they are borrowed, but how do we know whether a value is borrowed? The idea is quite simple: whenever you create a borrow, the compiler assigns the resulting reference a **lifetime**. This lifetime corresponds to the span of the code where the reference may be used. The compiler will infer this lifetime to be the smallest lifetime that it can have that still encompasses all the uses of the reference.

Note that Rust uses the term *lifetime* in a very particular way. In everyday speech, the word *lifetime*

Rustに対する重要な変更の  
提案(RFC)

のうち、承認されたもの

提案は[GitHub上](#)で行われる

# Rustの資料: [Rust Forge](#)



## Rust Forge

This site contains supplementary documentation useful to the members of [The Rust Project](#). To edit it submit PRs against [rust-lang-nursery/rust-forge](#).

### Release Dates

Rust 1.28 stable was released on Fri Aug 03 2018.

**Rust 1.29 stable will be released on Fri Sep 14 2018.**

Rust 1.30 stable will be released on Fri Oct 26 2018.

### Interested in hacking the compiler?

- [The rustc-guide](#) is a book about how rustc works
- [The rustc API docs](#) are hosted here
- [Building rustc with x.py](#).
- [Debugging the compiler](#). Tips for debugging the compiler.
- [Profile queries](#). Tips for tracking what the compiler does.
- [Rustc bug fix procedure](#): Describes the process for bug fixes that may cause existing code to stop compiling.
- [So you want to implement a feature?](#): Describes the procedure for implementing new features in rustc.

## Rustの開発に関する 諸々の資料集

# Rustの資料: [Rust Compiler Guide](#)

## High-level overview of the compiler source

### Crate structure

The main Rust repository consists of a `src` directory, under which there live many crates. These crates contain the sources for the standard library and the compiler. This document, of course, focuses on the latter.

Rustc consists of a number of crates, including `syntax`, `rustc`, `rustc_back`, `rustc_codegen`, `rustc_driver`, and many more. The source for each crate can be found in a directory like `src/libxxx`, where `xxx` is the crate name.

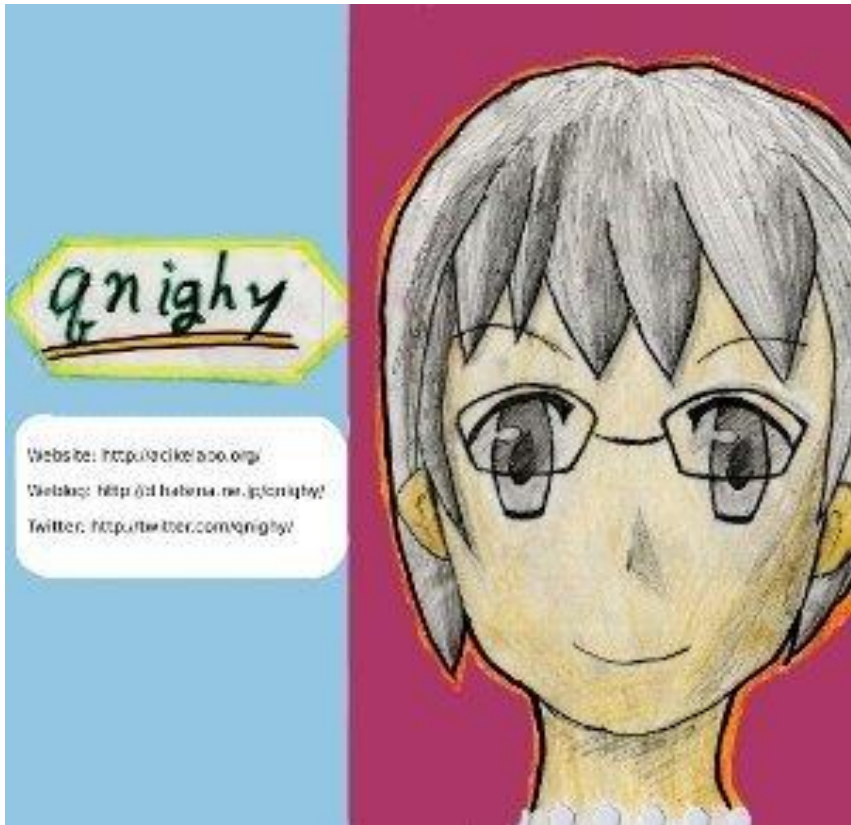
(N.B. The names and divisions of these crates are not set in stone and may change over time. For the time being, we tend towards a finer-grained division to help with compilation time, though as incremental compilation improves, that may change.)

The dependency structure of these crates is roughly a diamond:



## コンパイラの内部構造の解説

# Rustの資料: @qnighy



以下は見てます

- [Twitter](#)
- [Mastodon](#)
- StackOverflow (en/ja)
- teratail

Rustのセットアップ

# rustupを使おう！

## •rustupの利点

1. Rustが公式に推奨する標準的な方法である
2. 急激に進化するRustの最新バージョンを使うことができる
  - 古いコンパイラとの互換性を保証する仕組みは発展途上
3. ユーザー権限で入る (Unix系の場合)
4. 複数バージョンを簡単に切り替えられる
  - 特にstable/nightlyは頻繁に切り替えたい

# Rustのセットアップ



ドキュメント インストール コミュニティ 開発に参加する

Rustは速度、安全性、並行性の3つのゴールにフォーカスしたシステムプログラミング言語です。

[Rustの利用実績](#)

## 特徴

- ゼロコスト抽象化
- ムーブセマンティクス
- 保証されたメモリ安全性
- データ競合のないスレッド
- トレイトによるジェネリクス
- パターンマッチング
- 型推論
- 最小限のランタイム
- 効率的なCバインディング



```
fn main() {
    let greetings = ["Hello", "Hola", "Bonjour",
                    "Ciao", "こんにちは", "안녕하세요",
                    "Cześć", "Olá", "Здравствуйтe",
                    "Chào bạn", "您好", "Hallo",
                    "Hej", "Ahoj", "سلام", "ਸਵਾਗਤ"];

    for (num, greeting) in greetings.iter().enumerate() {
        print!("{}", greeting);
        match num {
            0 => println!("This code is editable and runnable!"),
            1 => println!("¡Este código es editable y ejecutable!"),
            2 => println!("Ce code est modifiable et exécutable !"),
            3 => println!("Questo codice è modificabile ed eseguibile"),
            4 => println!("このコードは編集して実行出来ます！"),
            5 => println!("여기에서 코드를 수정하고 실행할 수 있습니다"),
            6 => println!("Ten kod można edytować oraz uruchomić!"),
            7 => println!("Este código é editável e executável!"),
            8 => println!("Этот код можно отредактировать и запустить"),
            9 => println!("Bạn có thể edit và run code trực tiếp!")
        }
    }
}
```

[rust-lang.org](https://rust-lang.org)

にアクセスして  
インストール

# Rustのセットアップ (Unix系)

```
$ curl https://sh.rustup.rs -sSf | sh
```

を指示されるので実行する

(curl | sh が嫌いな人へ: 毒を食らわば皿まで)



# Rustのセットアップ (Windows)

`rustup-init.exe`

が落ちてくるので実行する

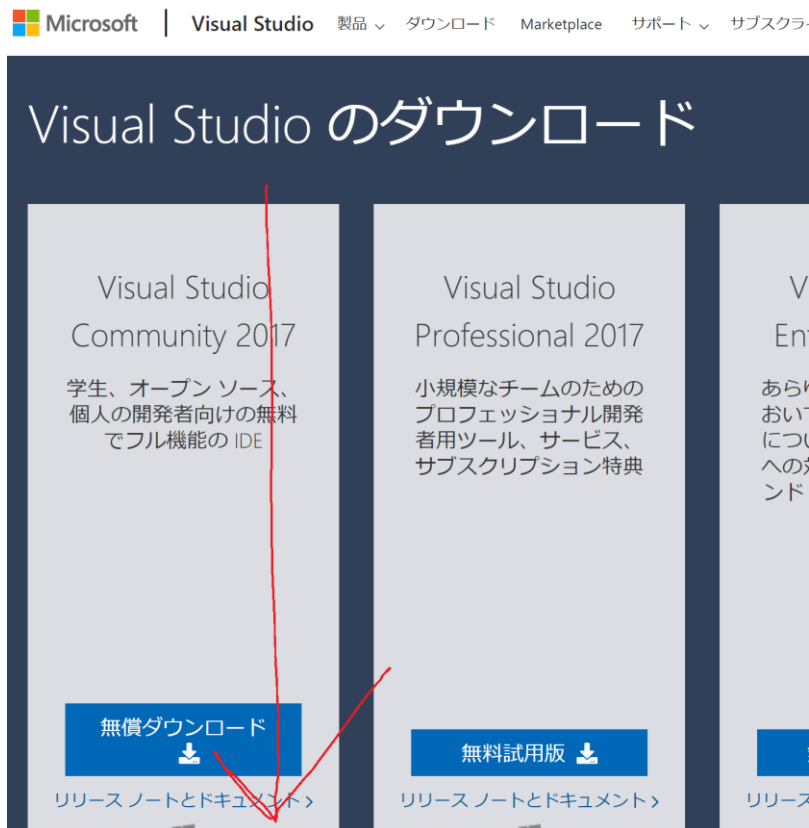
# Rustのセットアップ (Windows)

デフォルトのMSVC系ツールチェーンを使う場合、  
MSVCに含まれるリンカが必要

[aka.ms/buildtools](https://aka.ms/buildtools)

からBuild Tools for Visual Studio 2017 をダウンロード

# Rustのセットアップ (Windows)

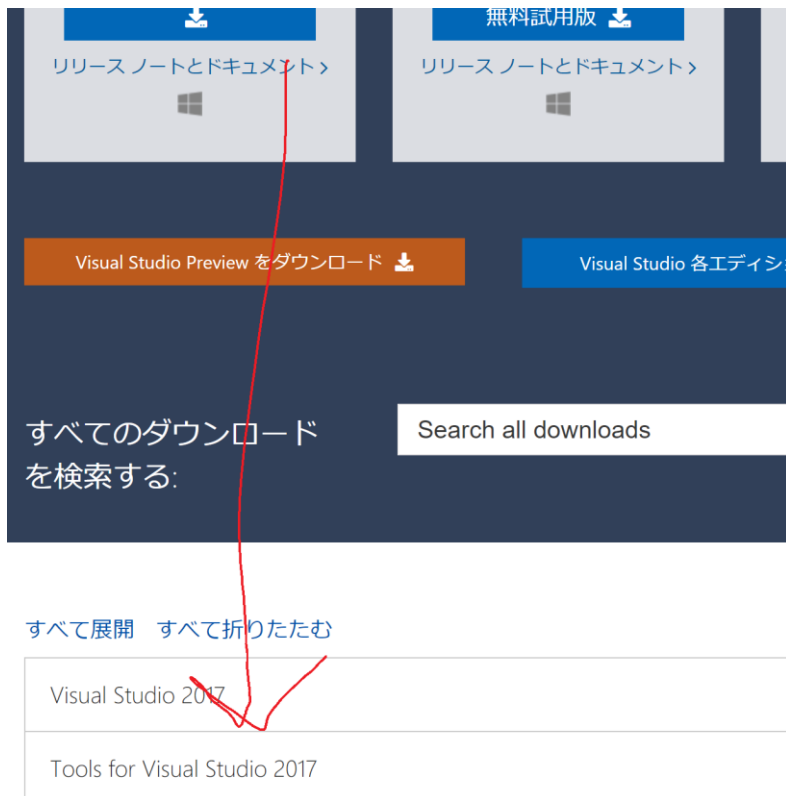


デフォルトのMSVC系ツールチェーンを使う場合、MSVCに含まれるリンクが必要

[aka.ms/buildtools](https://aka.ms/buildtools)

からBuild Tools for Visual Studio 2017 をダウンロード

# Rustのセットアップ (Windows)



デフォルトのMSVC系ツールチェーンを使う場合、MSVCに含まれるリンクが必要

[aka.ms/buildtools](https://aka.ms/buildtools)

からBuild Tools for Visual Studio 2017 をダウンロード

# Rustのセットアップ (Windows)

デフォルトのMSVC系ツールチェーンを使う場合、MSVCに含まれるリンクが必要

Tools for Visual Studio 2017

Build Tools for Visual Studio 2017 これらのビルドツールを使用すると、コマンドライン インターフェイスから Visual Studio をビルドできます。サポートされるプロジェクト:ASP.NET、Azure、C++ デスクトップ、ClickOnce、コンテナー、.NET Core、.NET Desktop、Node.js、Office と SharePoint、Python、TypeScript、単体テスト、UWP、WCF、Xamarin。

Agents for Visual Studio 2017 Agents for Visual Studio 2017 はロードテスト、機能テスト、自動テストに使用できます。詳細については、ALM ブログをご覧ください。

Feedback Client for Visual Studio 2017 Feedback Client for Visual Studio を使用して、ソフトウェアプロジェクトに関するフィードバックを収集します。

Visual Studio 2017 IntelliTrace スタンドアロン コレクター IntelliTrace スタンドアロン コレクターでは、本稼働サーバー上で Visual Studio をインストールせず、またアプリケーションを再展開せずに、アプリの診断データを収集できます。

Performance Tools for Visual Studio 2017 Visual Studio を使用せずにパフォーマンス プロファイルを実行できるようにするスタンドアロンのコマンドライン ツールです。詳細については、「コマンドラインからのプロファイリング ツールの使用」を参照してください。

Remote Tools for Visual Studio 2017 Remote Tools for Visual Studio 2017 を使用すると、Visual Studio がインストールされていないコンピュータでアプリケーションの展開、リモート デバッグ、リモートテスト、パフォーマンスのプロファイル作成、単体テストを実行できます。

[aka.ms/buildtools](https://aka.ms/buildtools)

からBuild Tools for Visual Studio 2017 をダウンロード

# Rustのセットアップ (Windows)

インストールしています - Visual Studio Build Tools 2017 - 15.8.1

×

ワークロード 個別のコンポーネント 言語パック インストールの場所

デフォルトのMSVC系ツールチェーンを使う場合、MSVCに含まれるリンカが必要

Windows (3)

Visual C++ Build Tools  
Microsoft C++ ツールセット、ATL、MFC を使用して Windows のデスクトップ アプリケーションをビルドし...

.NET デスクトップ ビルドツール  
C#、Visual Basic、F# を使用して、WPF、コンソール アプリケーションをビルド...

ユニバーサル Windows プラットフォーム ビルドツール  
ユニバーサル Windows プラットフォーム アプリケーションをビルドするために必要なツールを提供します。

Web & クラウド (4)

Web 開発ビルドツール  
Web アプリケーションのビルドのための MSBuild タスクとターゲット。

Azure 開発ビルドツール  
Azure アプリケーションのビルドのための MSBuild タスクとターゲット。

Office/SharePoint ビルド ツール  
Office アドイン、SharePoint アドイン、VSTO アドインをビルドします。

データストレージとビルドツールの処理  
SQL Server データベース プロジェクトをビルドする

場所

C:\Program Files (x86)\Microsoft Visual Studio\2017\BuildTools 変更...

続行すると、選択した Visual Studio のエディションのライセンスに同意することになります。また、Visual Studio を使用して他のソフトウェアをダウンロードする機能も提供されます。このソフトウェアはサードパーティに関する通知または付属するライセンスに記載のとおり、個別にライセンスされています。続行することで、これらのライセンスにも同意することになります。

- Visual C++ Build Tools のコア機能
- VC++ 2017 バージョン 15.8 v14.15 最新の v141...
- Visual C++ 2017 再頒布可能パッケージの更新...

オプション

- Windows 10 SDK (10.0.17134.0)
- CMake の Visual C++ ツール
- ツールのコア機能のテスト - ビルド ツール
- Windows 8.1 SDK と UCRT SDK
- x86 用と x64 用の Visual C++ ATL
- x86 用と x64 用の Visual C++ MFC
- C++/CLI サポート
- 標準ライブラリのモジュール (試験段階)
- Windows 10 SDK (10.0.16299.0)
- Windows 10 SDK (10.0.15063.0)
- Windows 10 SDK (10.0.14393.0)
- Windows 10 SDK (10.0.10586.0)
- Windows 10 SDK (10.0.10240.0)
- デスクトップ用 VC++ 2015.3 v14.00 (v140) ツー...
- ARM 用 Visual Studio C++ コンパイラとライブ...

必要な領域の合計サイズ: 4.72 GB

インストール

# エディタのセットアップ

- 独自の補完・IDEサポートを持っているエディタ



JetBrainsのIntelliJ IDEAやCLionなど任意のIDEで[Rust Plugin](#)が動作する

# エディタのセットアップ

- RLSによる補完・IDEサポートを持っているエディタ
  - 基本的には[Language Serverクライアント](#)があれば動くはず



Visual Studio Code  
([rls-vscode](#))



Vim/Neovim  
([vim-lsp](#)など)



Emacs  
([lsp-mode](#)など)



Atom  
([atom-ide-rust](#)など)

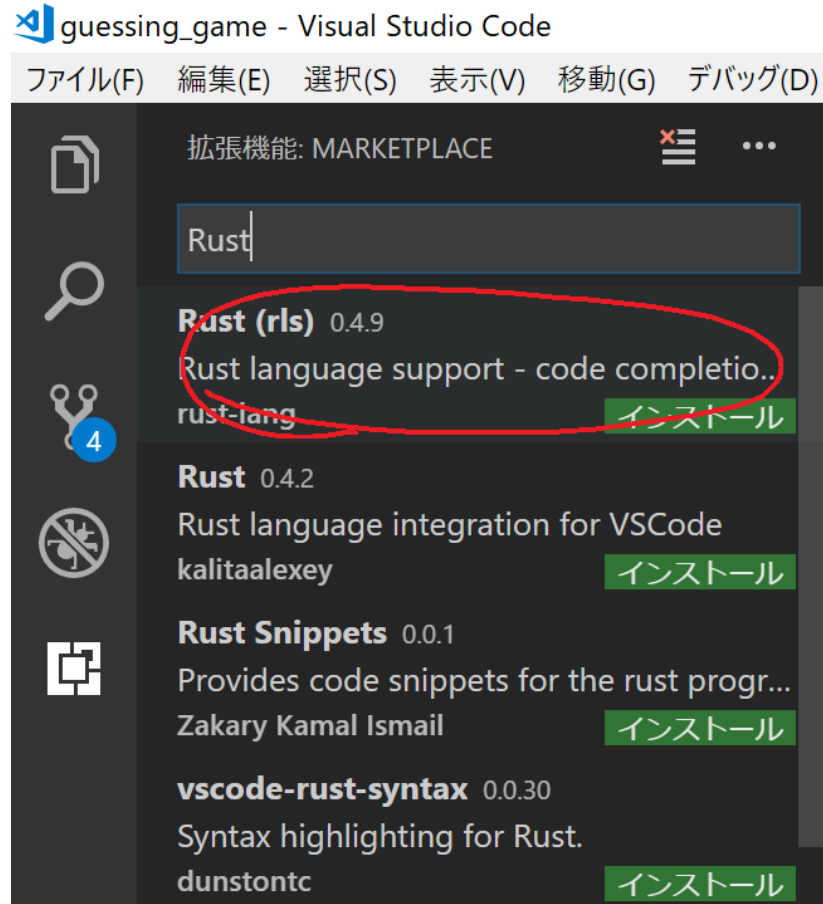


# Visual Studio Codeの場合

- RLSをインストールする (他エディタでも同様)

```
$ rustup component add rls-preview rust-analysis  
rust-src  
info: downloading component 'rls-preview'  
info: installing component 'rls-preview'  
info: downloading component 'rust-analysis'  
info: installing component 'rust-analysis'
```

# Visual Studio Codeの場合



拡張機能一覧を開いて  
“Rust”で検索  
“Rust (rls)”を選択  
してインストールするだけ

# Visual Studio Codeの場合

そのまま補完等が  
利用できるようになる

```
main.rs
1 fn main() {
2   let mut s = String::fr
3 }
4
```

- from fn from(s: Cow<'a, str>) -> String ⓘ
- from
- from
- from\_iter
- from\_iter
- from\_iter
- from\_iter
- from\_iter
- from\_iter
- from\_iter
- from\_iter
- from\_raw\_parts
- from\_str
- from\_utf16
- from\_utf16\_lossy

```
main.rs
1 fn main() {
2   let mut s = String::from("Hello, ");
3
4   s.push
5 }
6
```

- push
- push\_str pub fn push\_str(&mut self, string: &st... ⓘ

# ハンズオン

Cargoを使おう

# Cargoを使おう (§ 1.3)

- コンパイラは `rustc` だが、これを自分で実行することはほぼない。
- `cargo` を経由して実行することがほとんど。

```
$ rustup --version
rustup 1.13.0 (ea9259c1b 2018-07-16)
$ rustc --version
rustc 1.28.0 (9634041f0 2018-07-30)
$ cargo --version
cargo 1.28.0 (96a2c7d16 2018-07-13)
```

# プロジェクトを作ろう

- `cargo new` でプロジェクトを作成できる。

```
$ cargo new hello_cargo --bin
Created binary (application) `hello_cargo`
project
$ cd hello_cargo
```

# プロジェクトを作ろう

- `cargo new` でプロジェクトを作成できる。

```
$ ls
Cargo.toml  src
$ ls src
main.rs
```

# プロジェクトを作ろう

- `cargo new` でプロジェクトを作成できる。

```
$ git status  
On branch master  
  
No commits yet
```

```
...
```



# プロジェクトを作ろう

- プロジェクトの設定は `Cargo.toml` に書かれている

```
[package]
name = "hello_cargo"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]

[dependencies]
```

# プロジェクトを作ろう

- プロジェクトの設定は `Cargo.toml` に書かれている

```
[package]
name = "hello_cargo"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]

[dependencies]
```

# プロジェクトを作ろう

- 一番上のソースコードは `main.rs` (バイナリの場合)

```
fn main() {  
    println!("Hello, world!");  
}
```

# ビルド

- `cargo build` でプロジェクトをビルドできる。
  - `target/debug/hello_cargo` が生成される

```
$ cargo build
  Compiling hello_cargo v0.1.0
  Finished dev [unoptimized + debuginfo]
target(s) in 0.99s
```

# 実行

- `cargo run` で実行
  - 必要なときだけ再ビルドされる

```
$ cargo run
Finished dev [unoptimized + debuginfo]
target(s) in 0.01s
Running `target¥debug¥hello_cargo.exe`
Hello, world!
```

# チェック

- `cargo check` でコンパイルが通るかどうか確認
  - コード生成しないのでbuildより速い

```
$ cargo check
Checking hello_cargo v0.1.0
Finished dev [unoptimized + debuginfo]
target(s) in 0.22s
```

# リリースビルドと実行

- `cargo build --release` でリリースビルドする
  - 高速なバイナリが生成されるが、コンパイルに時間がかかる

```
$ cargo build --release  
$ cargo run --release
```

# ハンズオン

数当てゲーム



# プロジェクト作成

```
$ cargo new guessing_game --bin  
Created binary (application) `guessing_game`  
project  
$ cd guessing_game
```

# プロジェクト作成

```
$ cargo run
  Compiling guessing_game v0.1.0
  Finished dev [unoptimized + debuginfo]
target(s) in 0.01s
  Running `target¥debug¥guessing_game.exe`
Hello, world!
```

# プロジェクトの設定

```
[package]
name = "guessing_game"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]
```

```
[dependencies]
```

# プロジェクトの設定

- gitから自動生成される
- 正しくなければ直す

```
[package]
```

```
name = "guessing_game"
```

```
version = "0.1.0"
```

```
authors = ["Your Name <you@example.com>"]
```

```
[dependencies]
```

# 出力

```
fn main() {  
    println!("Hello, world!");  
}
```

# 出力

```
fn main() {  
    println!("Hello, world!");  
}
```

- `main`という名前の関数から始まる
- 関数の外に処理は書けない

# 出力

```
fn main() {  
    println!("Hello, world!");  
}
```

- Cのprintfみたいな処理
- println! の ! はマクロ呼び出し

# 出力

```
fn main() {  
    println!("Hello, world!");  
}
```

- "str" は文字列 ('c' は文字)



# 出力

```
fn main() {  
    println!("Hello, world!");  
}
```

- 文は ; で終わる
- 改行に構文上の意味はない

# 入力

```
use std::io;

fn main() {
    println!("Guess the number!");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guessed: {}", guess);
}
```

# 入力

```
use std::io;
```

```
fn main() {  
    println!("Guess the number!");  
  
    println!("Please input your guess.");  
  
    let mut guess = String::new();
```

- [prelude](#)以外の関数を使うとき必要
- [std::io](#)モジュール

# 入力

- 変数を宣言するlet文

```
let foo = bar; // immutable
```

```
let mut guess = String::new();
```

```
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");
```

```
println!("You guessed: {}", guess);
```

```
}
```

# 入力

- `mut` をつけると書き込める

```
let mut foo = bar; // mutable
```

```
let mut guess = String::new();  
  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
  
println!("You guessed: {}", guess);  
}
```

# 入力

- 変数の初期化

```
let mut guess = String::new();  
  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
  
println!("You guessed: {}", guess);  
}
```

# 入力

型

関連関数  
(静的メソッドのようなもの)

```
let mut guess = String::new();

io::stdin().read_line(&mut guess)
    .expect("Failed to read line");

println!("You guessed: {}", guess);
}
```

# 入力

- `new`という名前に特別な意味はなく、慣習として使われる

```
let mut guess = String::new();  
  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
  
println!("You guessed: {}", guess);  
}
```



# 入力

- `io`モジュールの[stdin](#)関数
- `std::io::stdin()`とも書ける

```
io::stdin().read_line(&mut guess)
    .expect("Failed to read line");

println!("You guessed: {}", guess);
}
```

# 入力

- Stdin型のハンドルを返す `:new();`

```
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");
```

```
println!("You guessed: {}", guess);
```

```
}
```

# 入力

- メソッド呼び出し

```
let mut guess = String::new();  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
println!("You guessed: {}", guess);  
}
```

# 入力

- & で変数への参照をとる
- &mut なので書き込み可能

```
let mut guess = String::new();  
  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
  
println!("You guessed: {}", guess);  
}
```

# 入力

- 1行でもよい

```
let mut guess = String::new();  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
println!("You guessed: {}", guess);  
}
```

# 入力

- Result型のメソッド

```
let mut guess = String::new();  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
println!("You guessed: {}", guess);  
}
```

# 入力

- `Ok(...)` のときは文字数が入っている  
→ 文字数を返す
- `Err(...)` のときは失敗の理由が入っている  
→ `"Failed to read line"` と出力してプログラムを終了

```
io::stdin().read_line(&mut guess)
    .expect("Failed to read line");

println!("You guessed: {}", guess);
}
```

# 入力

- 文字数は使わずに捨てている

```
let mut guess = String::new(),  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
println!("You guessed: {}", guess);  
}
```



# 入力

- `Result`を無視してもコンパイルは通るが、警告になる

```
Compiling guessing_game v0.1.0 (file:///C:/Users/qnighy/workdir/guessing_game)
warning: unused `std::result::Result` which must be used
--> src/main.rs:10:5
10 |         io::stdin().read_line(&mut guess);
    |         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
= note: #[warn(unused_result)] on by default
= note: this `Result` may be an `Err` variant, which should be handled
Finished dev [unoptimized + debuginfo] target(s) in 0.89s
```

```
let mut guess = String::new();
```

```
io::stdin().read_line(&mut guess);
```

```
println!("You guessed: {}", guess);
```

```
}
```

# 入力

- println! はCの printf のように使える

```
let x = 5;  
let y = 10;  
  
println!("x = {} and y = {}", x, y);
```

```
let mut
```

```
io::std
```

```
.expect(
```

```
println!("You guessed: {}", guess);
```

```
}
```

# 入力

```
use std::io;

fn main() {
    println!("Guess the number!");

    println!("Please input your guess.");

    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("Failed to read line");

    println!("You guessed: {}", guess);
}
```

# 秘密の番号

```
[package]
name = "guessing_game"
version = "0.1.0"
authors = ["Your Name <you@example.com>"]
```

```
[dependencies]
rand = "0.3.14"
```

# 秘密の番号

```
[package]
name = "guessing_game"
version = "0.1.0"
authors = ["Your Name"]
```

```
[dependencies]
rand = "0.3.14"
```

- 乱数ライブラリ [rand](#)を追加
- 3桁の最新バージョンを書いておけば [semver的によしなに](#)してくれる

# 秘密の番号

- 依存関係が変わったときはパッケージインデックスが更新される

```
$ cargo build
  Updating registry `https://github.com/rust-
lang/crates.io-index`
  Downloading rand v0.3.14
  Downloading libc v0.2.43
  Compiling libc v0.2.43
```

# 秘密の番号

- 依存関係も必要なときだけビルドされる  
(./target 以下に入る)

```
Compiling libc v0.2.45  
Compiling rand v0.3.14  
Compiling guessing_game v0.1.0  
(file:///C:/Users/qnighy/workdir/guessing_game)  
Finished dev [unoptimized + debuginfo]  
target(s) in 1m 59s
```

# 秘密の番号

- Cargo.lock を作りなおす

```
$ cargo update
  Updating registry `https://github.com/rust-
lang/crates.io-index`
  Adding bitflags v1.0.4
  Adding fuchsia-zircon v0.3.3
  Adding fuchsia-zircon-sys v0.3.3
```



# 秘密の番号

- "0.3.14" と指定したときは 0.3 以内で更新される

```
Removing rand v0.3.14
```

```
Adding rand v0.3.22
```

```
Adding rand v0.4.3
```

```
Adding winapi v0.3.5
```

```
Adding winapi-i686-pc-windows-gnu v0.4.0
```

```
Adding winapi-x86_64-pc-windows-gnu v0.4.0
```

- dtolnay's semver trick

# 秘密の番号

```
extern crate rand;

use std::io;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
             secret_number);

    println!("Please input your guess.");
```

```
let mut guess = String::new();

io::stdin()
    .read_line(&mut guess)
    .expect(
        "Failed to read line"
    );

println!(
    "You guessed: {}",
    guess);
}
```

# 秘密の番号

- 外部ライブラリの使用宣言  
(Rust2018で廃止予定)

```
extern crate rand;
```

```
use std::io;
```

```
use rand::Rng;
```

```
fn main() {
```

```
    println!("Guess the number!");
```

# 秘密の番号

```
extern crate rand;
```

```
use std::io;
```

```
use rand::Rng;
```

```
fn main() {
```

```
    println!("Guess the number!");
```

- [Rng](#)のトレイトメソッドを呼ぶために必要

# 秘密の番号

```
println!("Guess the nu
```

```
let secret_number =
```

```
    rand::thread_rng().gen_range(1, 101);
```

```
println!("The secret number is: {}", secret_number);
```

- 標準の乱数ソース

# 秘密の番号

- ThreadRng型ではなく Rngトレイトに定義されているメソッド

```
println!("Guess the number");

let secret_number =
    rand::thread_rng().gen_range(1, 101);

println!("The secret number is: {}", secret_number);
```

# 秘密の番号

```
extern crate rand;

use std::io;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
             secret_number);

    println!("Please input your guess.");
```

```
    let mut guess = String::new();

    io::stdin()
        .read_line(&mut guess)
        .expect(
            "Failed to read line"
        );

    println!(
        "You guessed: {}",
        guess);
}
```

# 比較

- まだコンパイルは通らない

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
            secret_number);

    println!("Please input your guess.");
```

```
let mut guess = String::new();

io::stdin().read_line(&mut guess)
    .expect("Failed to read line");

println!("You guessed: {}", guess);

match guess.cmp(&secret_number) {
    Ordering::Less =>
        println!("Too small!"),
    Ordering::Greater =>
        println!("Too big!"),
    Ordering::Equal =>
        println!("You win!"),
}
```



# 比較

- 比較結果をあらわす enum
  - Less
  - Equal
  - Greater

```
extern crate rand;
```

```
use std::io;
```

```
use std::cmp::Ordering;
```

```
use rand::Rng;
```

```
fn main() {
```

```
    println!("Guess the number!");
```

# 比較

```
println!("You guessed: {}", guess)
```

- [Ord::cmp](#)

```
match guess.cmp(&secret_number) {  
  Ordering::Less => println!("Too small!"),  
  Ordering::Greater => println!("Too big!"),  
  Ordering::Equal => println!("You win!"),  
}  
}
```

# 比較

- matchでenumの場合分け
- 網羅性はチェックされる

```
println!("You guessed: {}", guess);
```

```
match guess.cmp(&secret_number) {  
    Ordering::Less => println!("Too small!"),  
    Ordering::Greater => println!("Too big!"),  
    Ordering::Equal => println!("You win!"),  
}
```

```
}
```

# 比較

- 腕 (arm)
- パターン => 式

```
println!("You guessed: {}", guess)

match guess.cmp(&secret_number) {
  Ordering::Less => println!("Too small!"),
  Ordering::Greater => println!("Too big!"),
  Ordering::Equal => println!("You win!"),
}
}
```

# 比較

- ここでコンパイルエラー

```
Compiling guessing_game v0.1.0 (file:///C:/Users/qnighy/workdir/guessing_game)
error[E0308]: mismatched types
  --> src/main.rs:23:21
23 |         match guess.cmp(&secret_number) {
   |                       ^^^^^^^^^^^^^^^^^ expected struct `std::string::String`, found integral variable
   = note: expected type `&std::string::String`
           found type `&{integer}`
error: aborting due to previous error
```

```
println!("You guess {guess}"),
```

```
match guess.cmp(&secret_number) {
    Ordering::Less => println!("Too small!"),
    Ordering::Greater => println!("Too big!"),
    Ordering::Equal => println!("You win!"),
}
}
```

# 比較

• &String 型

• &i32 型

```
guess.guess(&secret_number);  
  
match guess.cmp(&secret_number) {  
    Ordering::Less => println!("Too small!"),  
    Ordering::Greater => println!("Too big!"),  
    Ordering::Equal => println!("You win!"),  
}  
}
```

# 変換

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
            secret_number);

    println!("Please input your guess.");
}
```

```
let mut guess = String::new();

io::stdin().read_line(&mut guess)
    .expect("Failed to read line");

let guess: u32 = guess.trim().parse()
    .expect("Please type a number!");

println!("You guessed: {}", guess);

match guess.cmp(&secret_number) {
    Ordering::Less =>
        println!("Too small!"),
    Ordering::Greater =>
        println!("Too big!"),
    Ordering::Equal =>
        println!("You win!"),
}
```

# 変換

```
let mut guess = String::new();
```

```
io::stdin().read_line()  
    .expect("Failed to read line");
```

• 変数guessを宣言…?

```
let guess: u32 = guess.trim().parse()  
    .expect("Please type a number!");
```



# 変換 - シャドーイング

- この変数 `guess` と

```
let mut guess = String::new();  
io::stdin().read_line().expect("Failed to read line");
```

- この変数 `guess` は同姓同名の別人  
(型も違う)

```
let guess: u32 = guess.trim().parse()  
    .expect("Please type a number!");
```

# 変換 - シャドーイング

- この範囲では上の `guess` を指す

```
let mut guess = String::new();  
  
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");  
  
let guess: u32 = guess.trim().parse()  
    .expect("Not a number");
```

- ここ以降は下の `guess` を指す

# 変換 - シャドーイング

- シャドーイングは  
OCamlなどのプログラミング言語で  
長年採用されてきた  
**由緒正しい**文化です。

# 変換

```
let mut guess =
```

```
io::stdin().read
```

```
.expect("Failed to read line");
```

```
let guess: u32 = guess.trim().parse()
```

```
.expect("Please type a number!");
```

- 末尾改行を削除するために、空白除去するメソッドを使う

# 変換

- 文字列をパースして他の型にする
- パースできないときもあるので **Result**を返す

```
let mut guess =
```

```
io::stdin().read_line(&mut guess)  
    .expect("Failed to read line");
```

```
let guess: u32 = guess.trim().parse()  
    .expect("Please type a number!");
```

# 変換

```
let mut guess = String::new();
```

- 今回は、パースできなかつたら終了する

```
.expect("Failed to parse");
```

```
let guess: u32 = guess.trim().parse()  
.expect("Please type a number!");
```

# 変換

```
let mut guess = String::new();
```

- 今回は、パースできなかつたら終了する

```
.expect("Failed to parse");
```

```
let guess: u32 = guess.trim().parse()  
.expect("Please type a number!");
```

# 変換

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
            secret_number);

    println!("Please input your guess.");
```

```
    let mut guess = String::new();

    io::stdin().read_line(&mut guess)
        .expect("Failed to read line");

    let guess: u32 = guess.trim().parse()
        .expect("Please type a number!");

    println!("You guessed: {}", guess);

    match guess.cmp(&secret_number) {
        Ordering::Less =>
            println!("Too small!"),
        Ordering::Greater =>
            println!("Too big!"),
        Ordering::Equal =>
            println!("You win!"),
    }
}
```



# ループ

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
        secret_number);

    loop {
        println!("Please input your guess.");
```

```
        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = guess.trim().parse()
            .expect("Please type a number!");

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less =>
                println!("Too small!"),
            Ordering::Greater =>
                println!("Too big!"),
            Ordering::Equal =>
                println!("You win!"),
        }
```

```
    }
}
```

# ループ

- break するまで永遠にループ

```
loop {  
    println!("Please input your guess.");  
  
    let mut guess = String::new();  
  
    io::stdin().read_line(&mut guess)  
        .expect("Failed to read line");
```

# ループ

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
             secret_number);

    loop {
        println!("Please input your guess.");
```

```
        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = guess.trim().parse()
            .expect("Please type a number!");

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less =>
                println!("Too small!"),
            Ordering::Greater =>
                println!("Too big!"),
            Ordering::Equal =>
                println!("You win!"),
        }
    }
}
```

# 脱出

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
        secret_number);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();
```

```
        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");

        let guess: u32 = guess.trim().parse()
            .expect("Please type a number!");

        println!("You guessed: {}", guess);

        match guess.cmp(&secret_number) {
            Ordering::Less =>
                println!("Too small!"),
            Ordering::Greater =>
                println!("Too big!"),
            Ordering::Equal => {
                println!("You win!");
                break;
            }
        }
    }
}
```

# 脱出

```
Ordering::Greater =>  
    println!("Too big!"),  
Ordering::Equal => {  
    println!("You win!");  
    break;  
}  
}  
}
```

• 脱出

# エラー処理

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
             secret_number);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");
    }
}
```

```
let guess: u32 =
    match guess.trim().parse() {
        Ok(num) => num,
        Err(_) => continue,
    };

println!("You guessed: {}", guess);

match guess.cmp(&secret_number) {
    Ordering::Less =>
        println!("Too small!"),
    Ordering::Greater =>
        println!("Too big!"),
    Ordering::Equal => {
        println!("You win!");
        break;
    }
}
}
```

# エラー処理

- `Result` に対して `match` することで明示的にエラー処理をしている

```
let guess: u32 = match guess.trim().parse() {  
    Ok(num) => num,  
    Err(_) => continue,  
};
```

# エラー処理

- `Ok(...)` だったときは  
その中身を変数として取り出せる

```
let guess: Result<int> = match guess.trim().parse() {  
  Ok(num) => num,  
  Err(_) => continue,  
};
```



# エラー処理

- `Err(...)` だったときもエラーを取り出せるが、いらないので `_` で捨てる

```
let guess: u... match guess.trim().parse() {  
    Ok(num) => num,  
    Err(_) => continue,  
};
```

# エラー処理

```
let guess: u32 = match guess.trim().parse() {  
    Ok(num) => num,  
    Err(_) => continue,  
};
```

- ループのやり直し

# エラー処理

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);

    println!("The secret number is: {}",
            secret_number);

    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");
    }
}
```

```
let guess: u32 =
    match guess.trim().parse() {
        Ok(num) => num,
        Err(_) => continue,
    };

println!("You guessed: {}", guess);


match guess.cmp(&secret_number) {
    Ordering::Less =>
        println!("Too small!"),
    Ordering::Greater =>
        println!("Too big!"),
    Ordering::Equal => {
        println!("You win!");
        break;
    }
}
}
```

# 最終形 (ネタばらしを削除)

```
extern crate rand;

use std::io;
use std::cmp::Ordering;
use rand::Rng;

fn main() {
    println!("Guess the number!");

    let secret_number =
        rand::thread_rng().gen_range(1, 101);
    
    loop {
        println!("Please input your guess.");

        let mut guess = String::new();

        io::stdin().read_line(&mut guess)
            .expect("Failed to read line");
    }
}
```

```
let guess: u32 =
    match guess.trim().parse() {
        Ok(num) => num,
        Err(_) => continue,
    };

println!("You guessed: {}", guess);

match guess.cmp(&secret_number) {
    Ordering::Less =>
        println!("Too small!"),
    Ordering::Greater =>
        println!("Too big!"),
    Ordering::Equal => {
        println!("You win!");
        break;
    }
}
}
```

# 普通のプログラミング機能の 紹介

# 変数と可変性

```
fn main() {  
    let x = 5;  
    println!("The value of x is: {}", x);  
    x = 6;  
    println!("The value of x is: {}", x);  
}
```

• 変数宣言と初期化

• 再代入 (エラー)

# 変数と可変性

```
fn main() {  
    let mut x = 5;  
    println!("The value of x is: {}", x);  
    x = 6;  
    println!("The value of x is: {}", x);  
}
```

• mutをつけて宣言

• 再代入 (OK)

# 変数と可変性

- 定数

```
fn main() {  
    const MAX_POINTS: u32 = 100_000;  
}
```



# 変数と可変性

- 定数は型が必須

```
fn main() {  
    const MAX_POINTS: u32 = 100_000;  
}
```

# 変数と可変性

- グローバルな定数

```
fn main() {}  
const MAX_POINTS: u32 = 100_000;
```

# 変数と可変性

```
fn main() {  
    let x = 5;  
    let x = x + 1;  
    let x = x * 2;  
    println!("The value of x is: {}", x);  
}
```

- シャドーイング  
(同姓同名の別人)

# 変数と可変性

```
fn main() {  
    let x = 5;  
    let x = x + 1;  
    let x = x * 2;  
    println!("The value of x is: {}", x);  
}
```

- シャドーイング  
(同姓同名の別人)

# 変数と可変性

```
let spaces = "  ";  
let spaces = spaces.length;
```

- 作業途中の値には積極的に同じ名前をつける

# 変数と可変性

```
let mut spaces = " ";  
spaces = spaces.len();
```

- コンパイルエラー

# データ型

- Rustは単相Hindley-Milnerをベースとする型推論がある
- グローバルの型は明示し、ローカルの型は省略できる

# データ型 - 整数

- numクレートに多倍長整数がある
- AtomicとNonZeroという亜種がある

長さ	符号つき	符号なし
8bit	i8	u8
16bit	i16	u16
32bit	i32	u32
64bit	i64	u64
128bit	i128	u128
ポインタと同じ	isize	usize



# データ型 - 整数

- リテラルの書き方

リテラル	例
10進数	98_222
16進数	0xff
8進数	0o77
2進数	0b1111_0000
バイト (u8のみ)	b'A'

# データ型 - 整数以外のプリミティブ

	型	リテラルの例
32bit float	f32	2.0
64bit float	f64	3.0
真偽値	bool	false
Unicode文字	char	'🍕'

# データ型 - 演算

- C言語と同様の四則演算
- 自動キャストはしない

# 複合型

```
let (x, mut y): (i32, i64) = (3, 88);
```

```
let arr: [u8; 3] = [3, 2, 3];
```

```
let slice: &[u8] = &arr;
```

# 建設中

- ここまで来た場合ホワイトボードとかでやります

# 所有権と借用

# 建設中

- ここまで来た場合ホワイトボードでやります

# Rustの今後



# Rustは発展途上

安定版コンパイラ

6週ごとに新しいリリース  
毎回新機能が届けられる

nightlyコンパイラ

ほぼ毎日更新される最先端  
不安定機能の利用が許可されている  
多くの機能が安定化を待ち望まれている

Edition 2018

破壊的変更を非破壊的に導入する新たな試み  
ライブラリごとに導入でき、相互に行き来できる  
2018年末までに正式公開される

Rust RFC

Rustの大きな変更はRFCという門を通る  
コミュニティの議論を経て、大きな合  
意が取れたら開発が開始される

# Rustは発展途上

安定版コンパイラ

```
$ rustup update
```

[Rust Blog](#)でリリースをチェック

nightlyコンパイラ

```
$ rustup toolchain install nightly
```

必要な場所だけでnightlyを使うことができる  
[This Week in Rust](#)で最新情報をチェック

Edition 2018

[Edition Guide](#)を読んでプレビュー版を試そう

Rust RFC

[GitHubリポジトリ](#)で議論が見られる

# Rustの2018年の目標

- 2018年初にユーザーからのサーベイをもとに設定した[目標](#)

## ネットワークサービス

高速非同期サーバーのための機能が急ピッチで開発中  
async/awaitの安定化を待て

## WebAssembly

Emscriptenに依存しなくなった  
Yewなどのクライアントサイドフレームワークが台頭してきている

## コマンドライン

シングルバイナリにクロスコンパイルできる強みを活かす

## 組み込み

小さいランタイムとメタルに近い言語仕様、そしてクロスコンパイルの利便性が強み

# まとめ

- Rustはいい言語だよ