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

reverse-complement

source	secs	n
Rust	**1.62**	995,
C gcc	1.75	994,

mandelbrot

source	secs
Rust	1.75
C gcc	1.64

fasta

source	secs
Rust	1.47
C gcc	1.32

spectral-norm

source	secs
Rust	2.31
C gcc	1.99

binary-trees

source	secs
Rust	4.31
C gcc	2.54

pidigits

source	secs
Rust	**1.74**
C gcc	1.75

fannkuch-redux

source	secs
Rust	9.44
C gcc	8.72

k-nucleotide

source	secs
Rust	5.76
C gcc	5.07

n-body

source	secs
Rust	13.13
C gcc	9.17

regex-redux

source	secs
Rust	2.54
C gcc	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

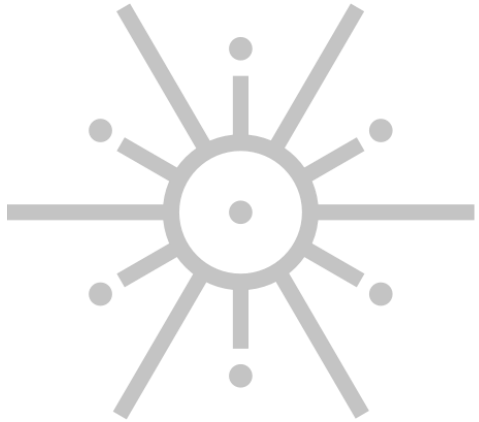
```
error[E0106]: missing lifetime specifier
 --> src\main.rs:4:22
  |
4 | fn return_stack() -> &i32 {
  |                      ^ expected lifetime parameter
  |
  = help: this function's return type contains a borrowed value, but there is no
 value for it to be borrowed from
  = help: consider giving 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 no
t 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？

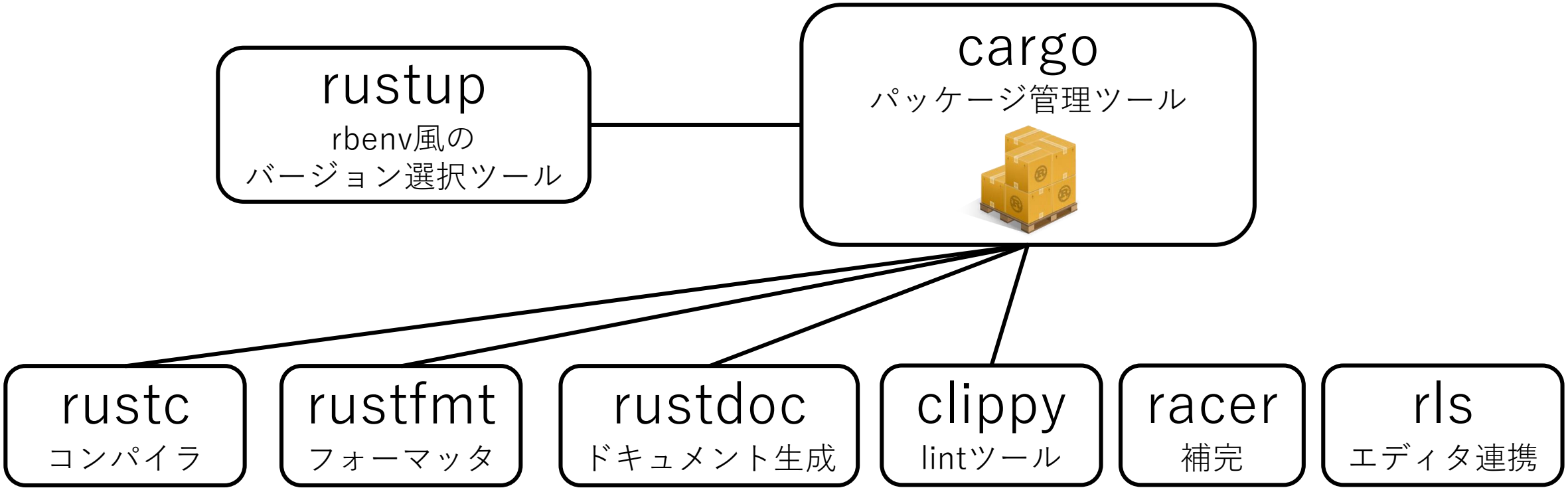
- 使いやすさ重視のコンパイラ

```
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イン



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

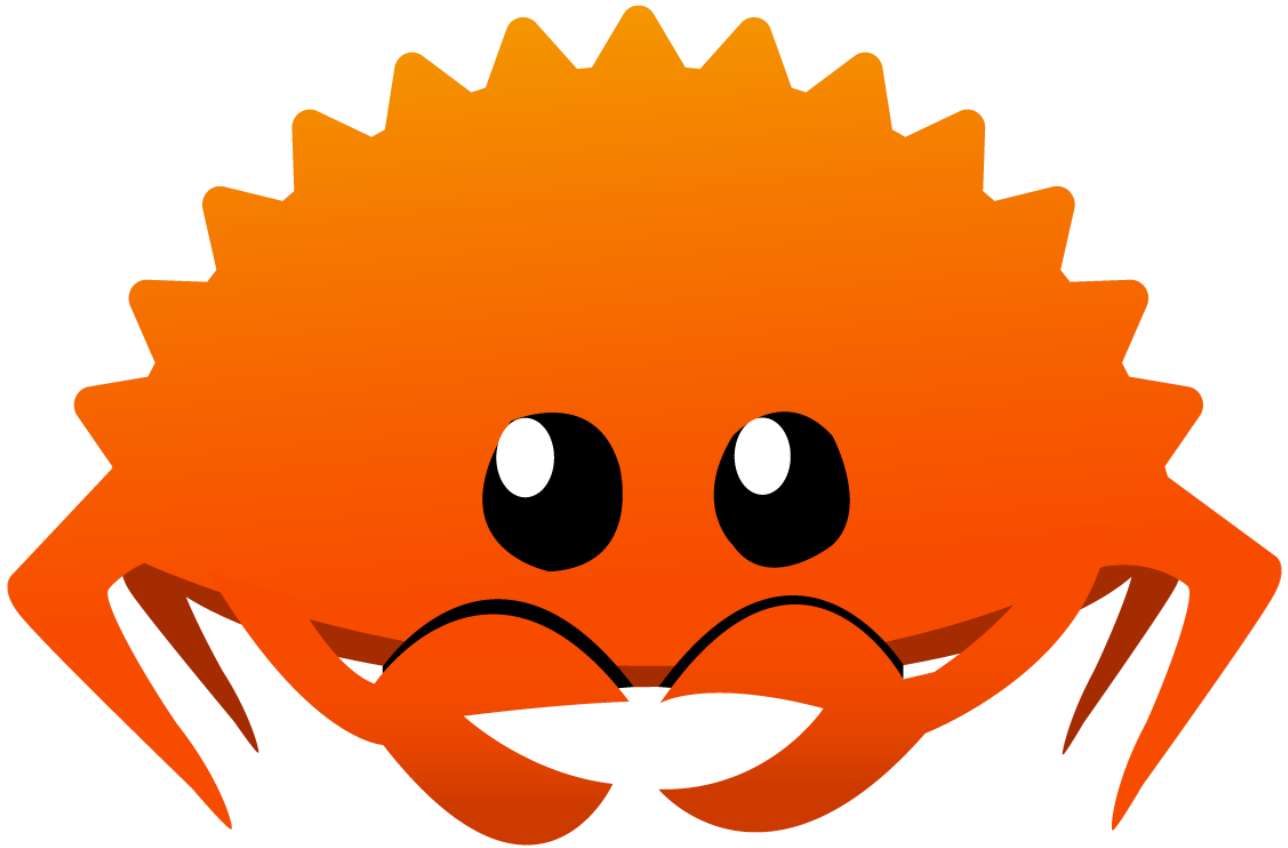
stable compiler
互換性を保証するコンパイラ

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、互換性の見落しを防ぐ

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

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

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

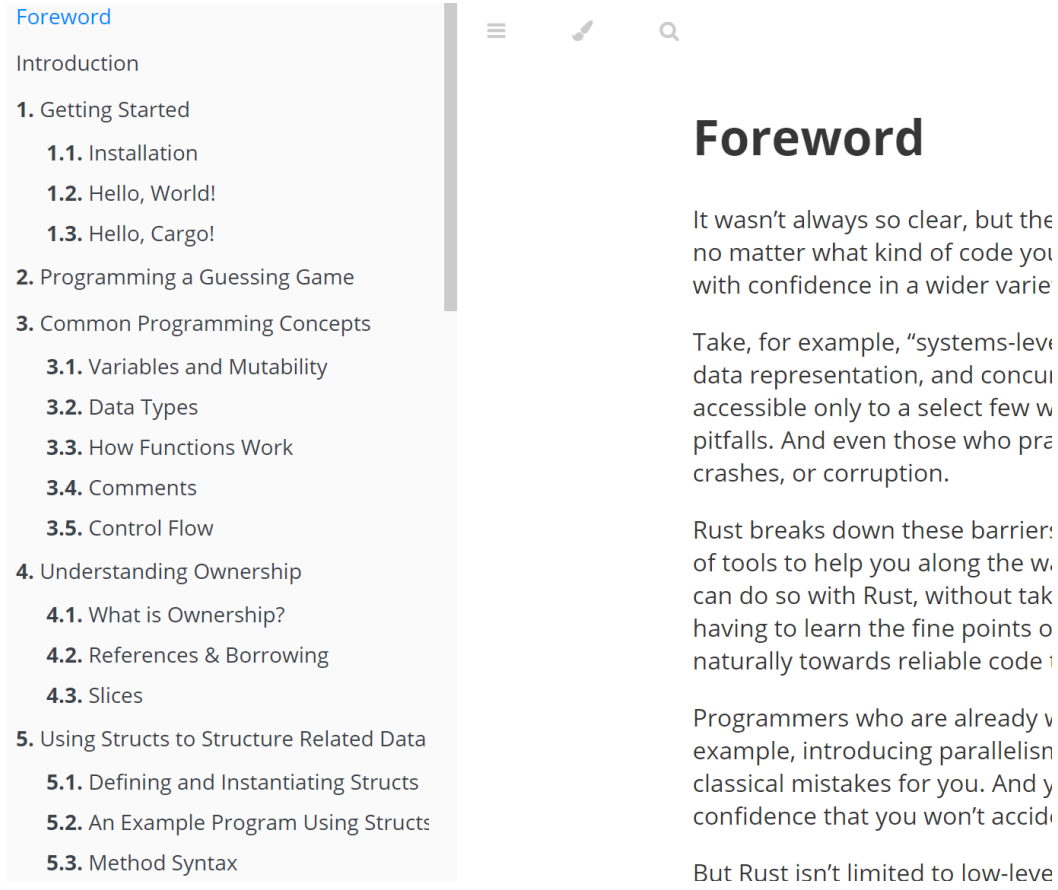
If reading multiple hundreds of pages about a language isn't your style, then Rust By Example has you covered. While the book talks about code with a lot of words, RBE shows off a bunch of code, and keeps the talking to a minimum. It also includes exercises!

2 Use Rust

先端機能はこちらへ

Rustの資料: TRPL

公式の教科書



Foreword

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

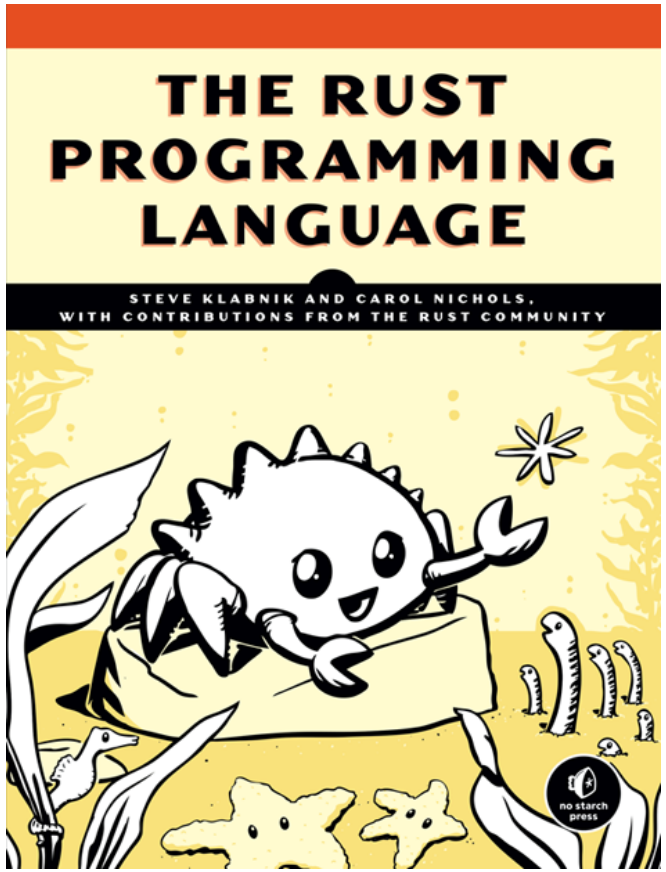
Take, for example, "systems-leve
data representation, and concur
accessible only to a select few w
pitfalls. And even those who pra
crashes, or corruption.

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

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

But Rust isn't limited to low-leve

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

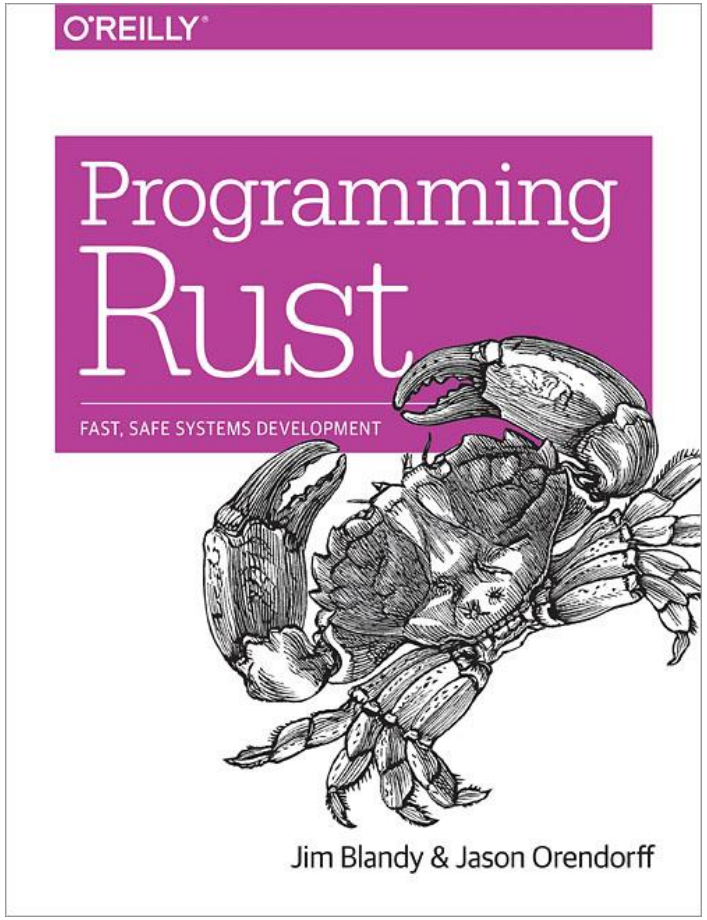
すぐにはわかりにくいかもしれませんが、Rust プログラミング言語は、エンパワーメント (empowerment) を根本原理としています: どんな種類のコードを現在書いているにせよ、Rust は幅広い領域で以前よりも遠くへ到達し、自信を持ってプログラムを組む力を与え (empower) ます。

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。

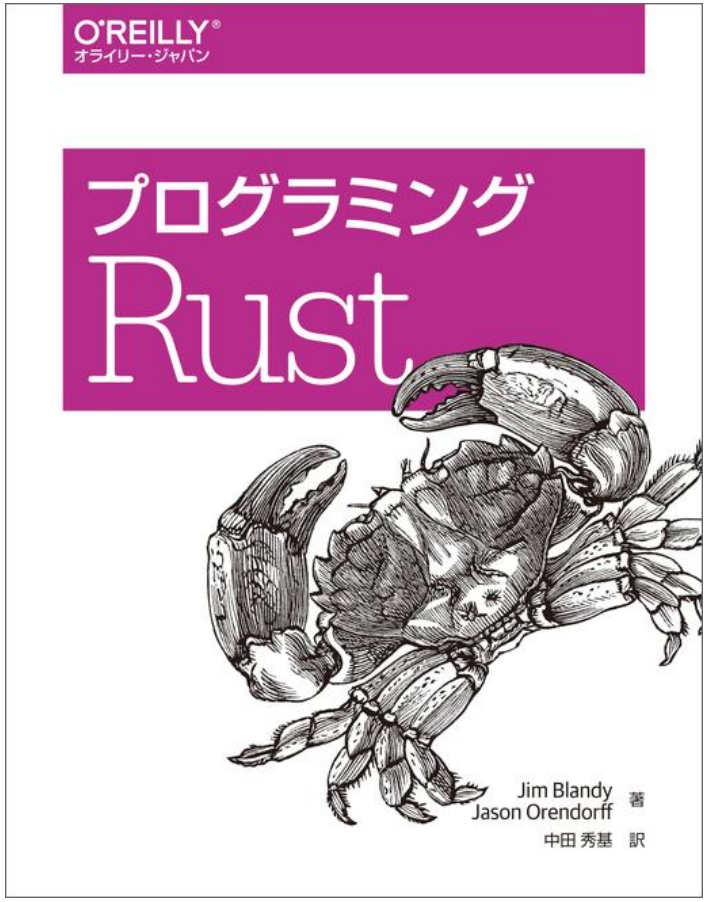
この難しさを取り除くために、Rust は、古い落とし穴を排除し、その過程で使いやすく役に立つ洗練された一連のツールを提供します。低レベルな制御に「下がる」必要があるプログラマは、お決まりのクラッシュやセキュリティホールのリスクを負わず、気まぐれなツールチェーンのデリケートな部分を学ぶ必要なく Rust で同じことができます。さらにいいことに、Rust は、スピードとメモリ使用の観点で効率的な信頼性の高いコードへと自然に導くよう設計されています。

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るはずです。

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

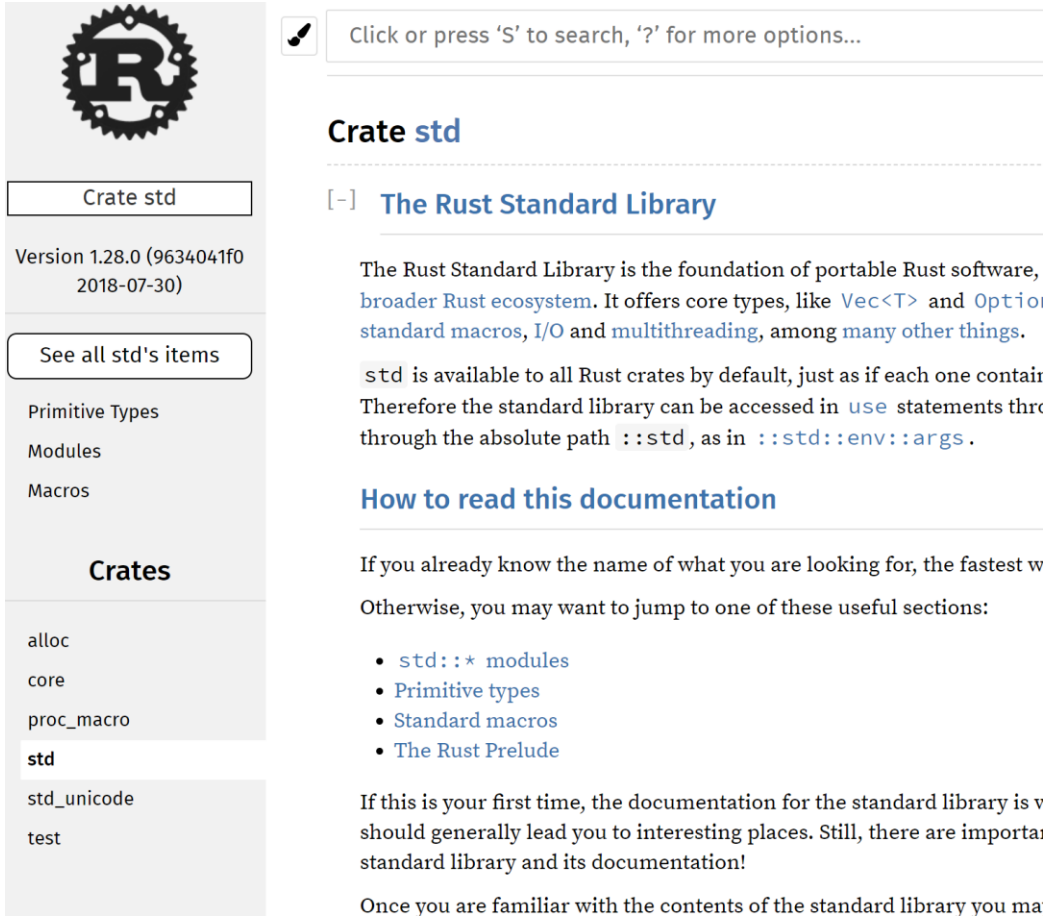
Structures

There are three types of structures ("structs") that can be created us

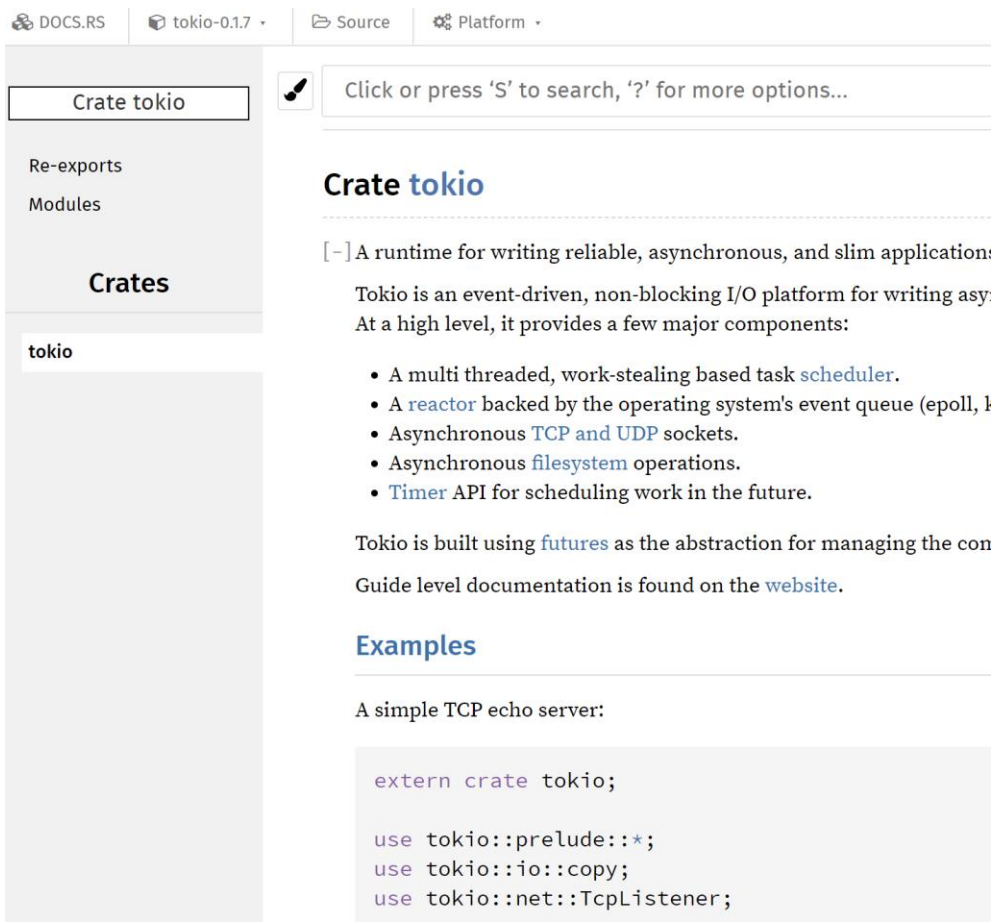
- Tuple structs, which are, basically, named tuples.
- The classic C structs
- Unit structs, which are field-less, are useful for generics.

```
#[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



Rustの資料: docs.rs



"docs.rs/パッケージ名"で飛べる

Rustの資料: Cargo Book

The

The Cargo Book

Cargo is the Rust *package manager*. Cargo downloads
your project, makes packages, and upload them to cr
You can contribute to this book on GitHub.

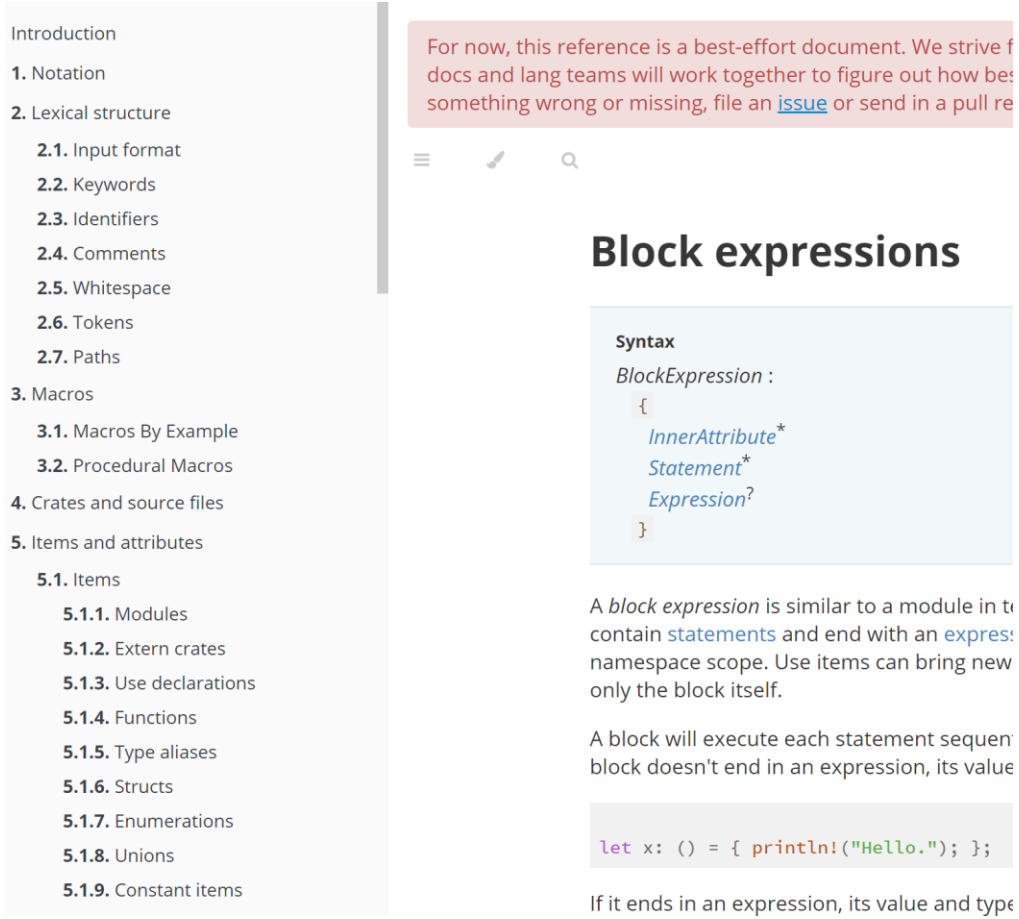
Sections

Getting Started

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

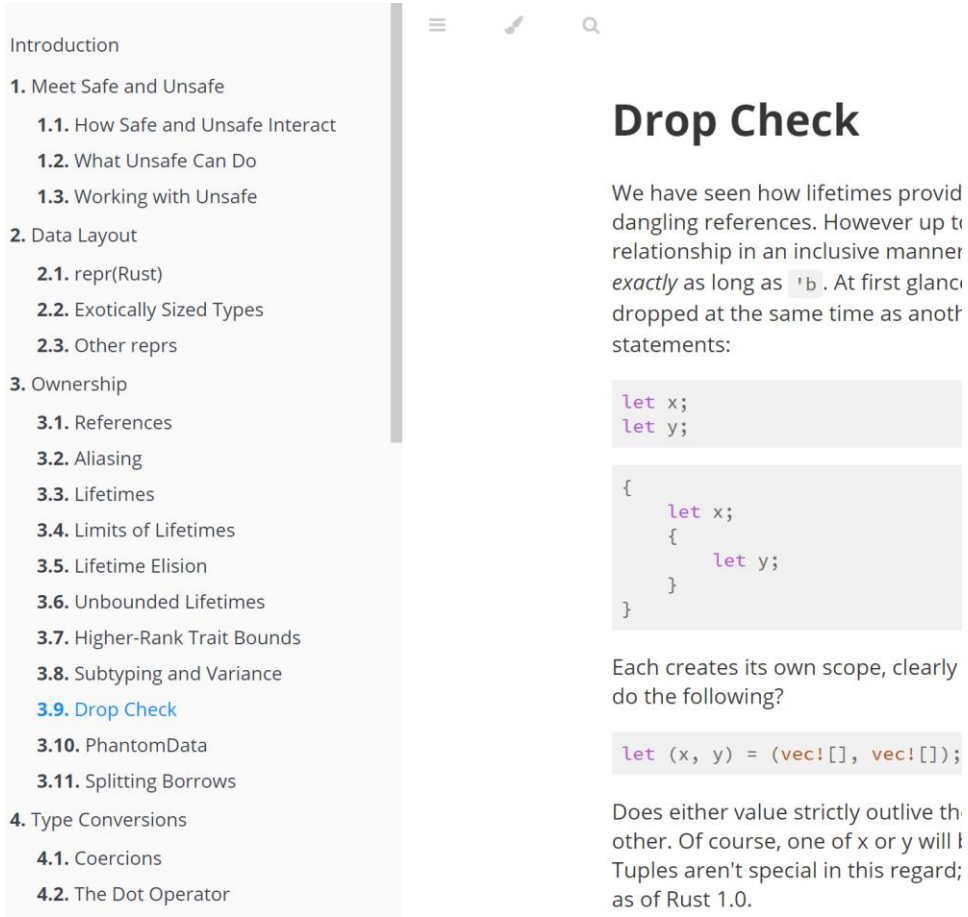
Cargo.tomlの書き方とか

Rustの資料: The Reference



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が詳しい

Rustの資料: The Rustonomicon



Drop Check

We have seen how lifetimes provid
dangling references. However up t
relationship in an inclusive manner
exactly as long as `'b`. At first glanc
dropped at the same time as anoth
statements:

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

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

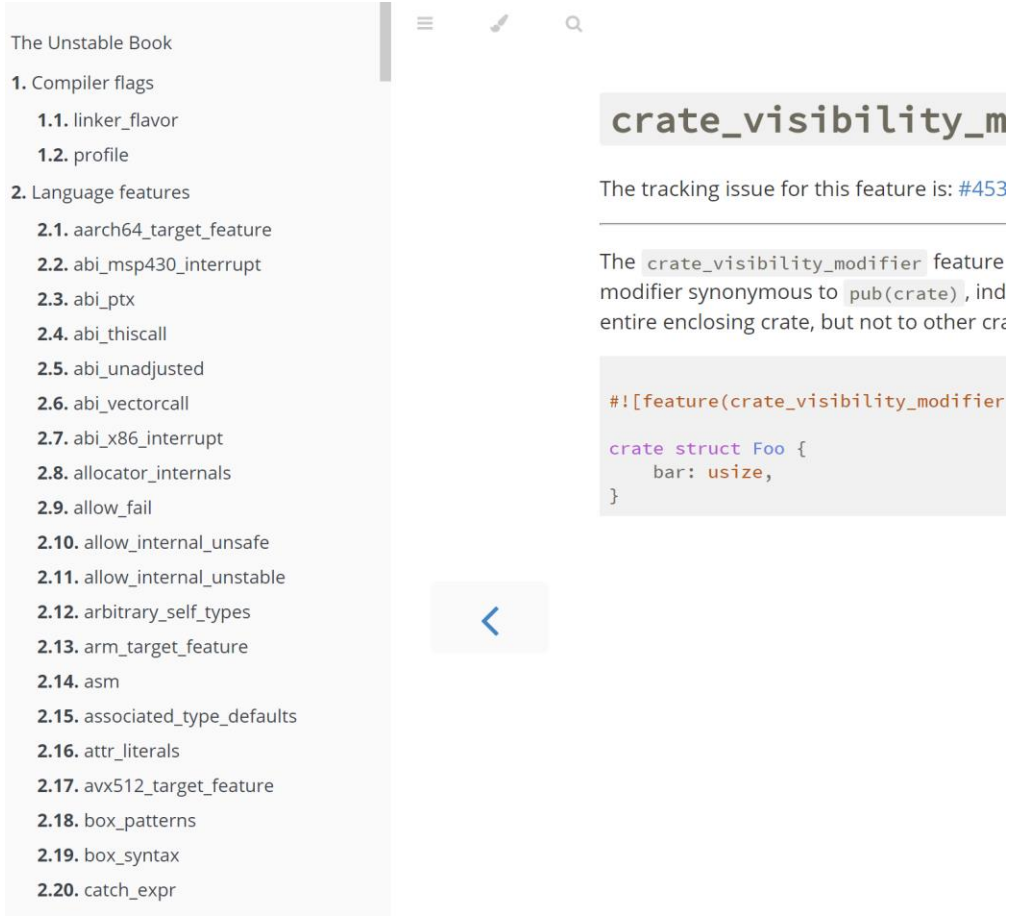
Each creates its own scope, clearly
do the following?

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

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

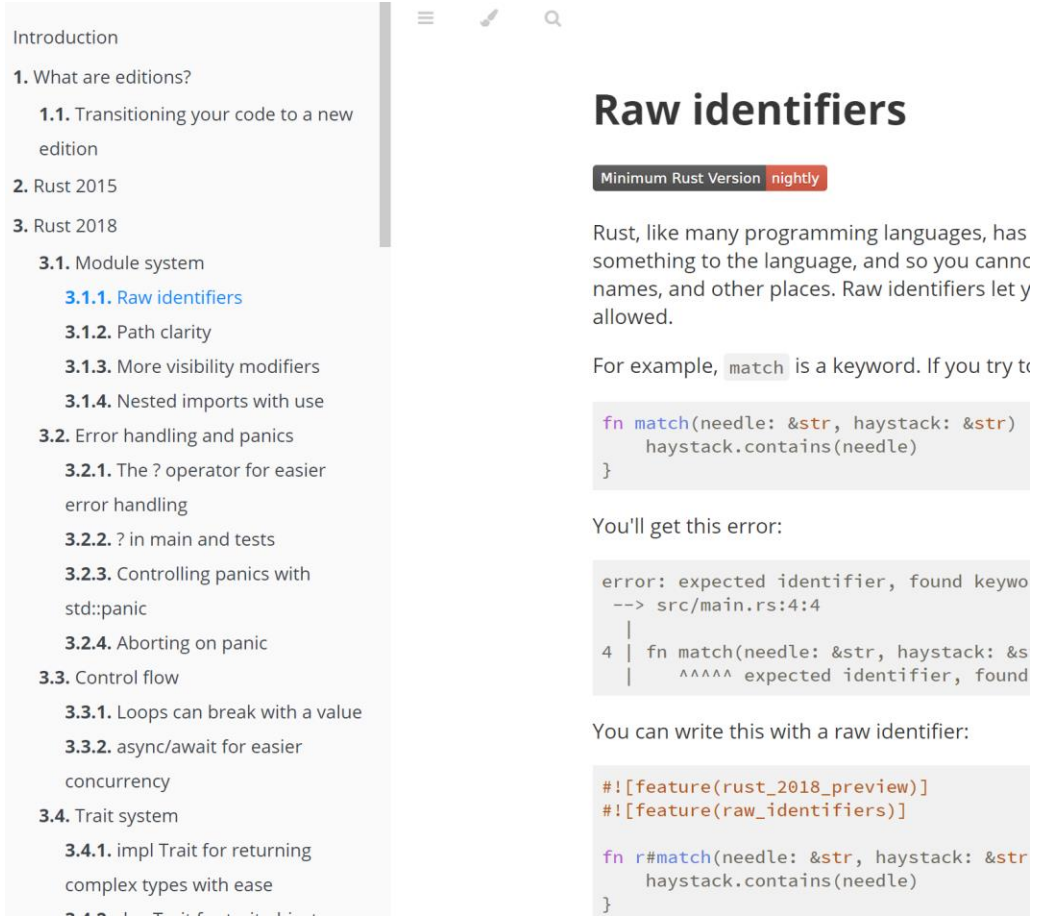
unsafeを触る前に読む本
(むずかしい)

Rustの資料: unstable-book



先端機能を知りたい人向け

Rustの資料: edition-guide



“Rust 2018” の変更点が解説されている

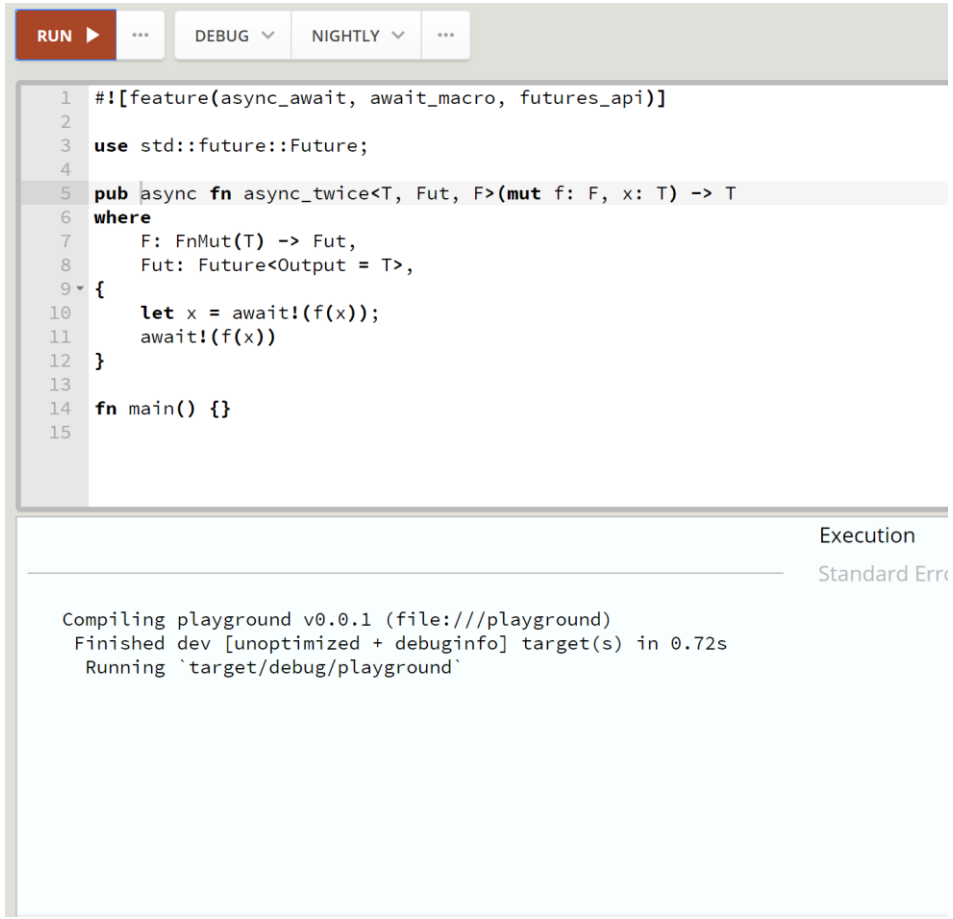
Rustを昔触ってた人にも有益

Rustの資料: crates.io

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)

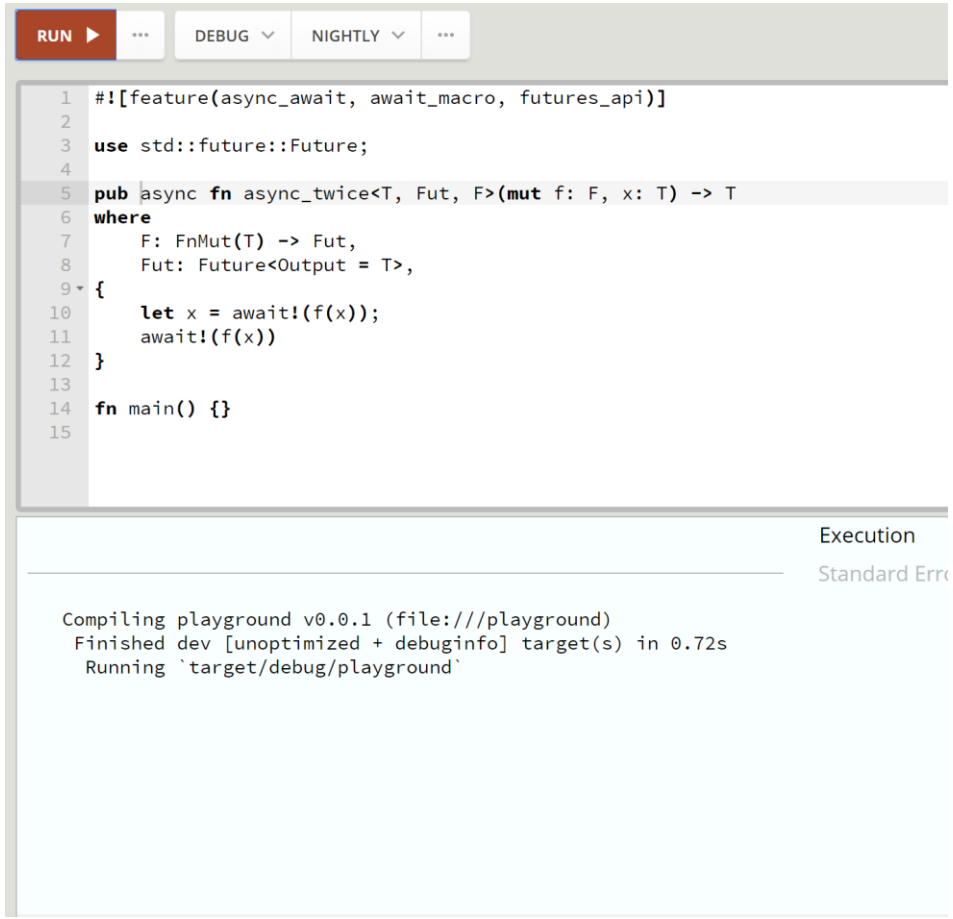


簡単なプログラムは
Webで試せる

制約

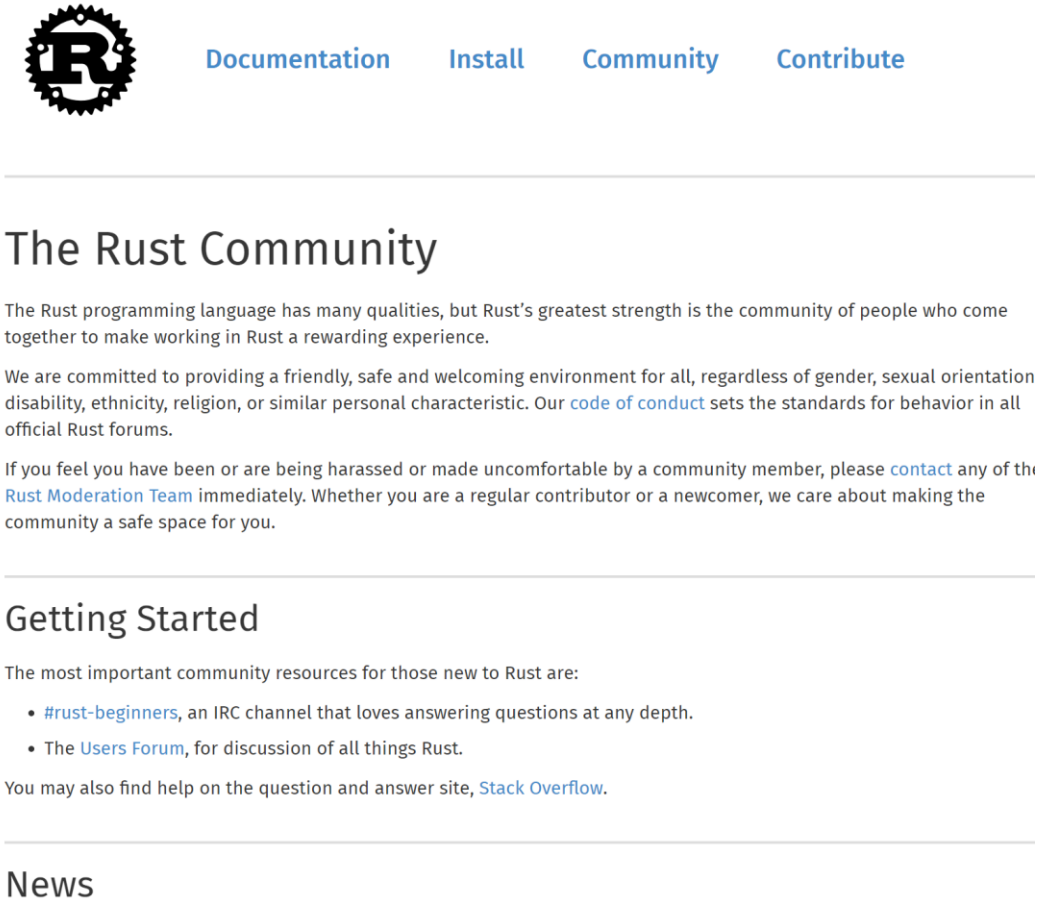
- 1ファイル
- stdと人気上位100パッケージのみ使える

Rustの資料: Rust Playground (playpen)



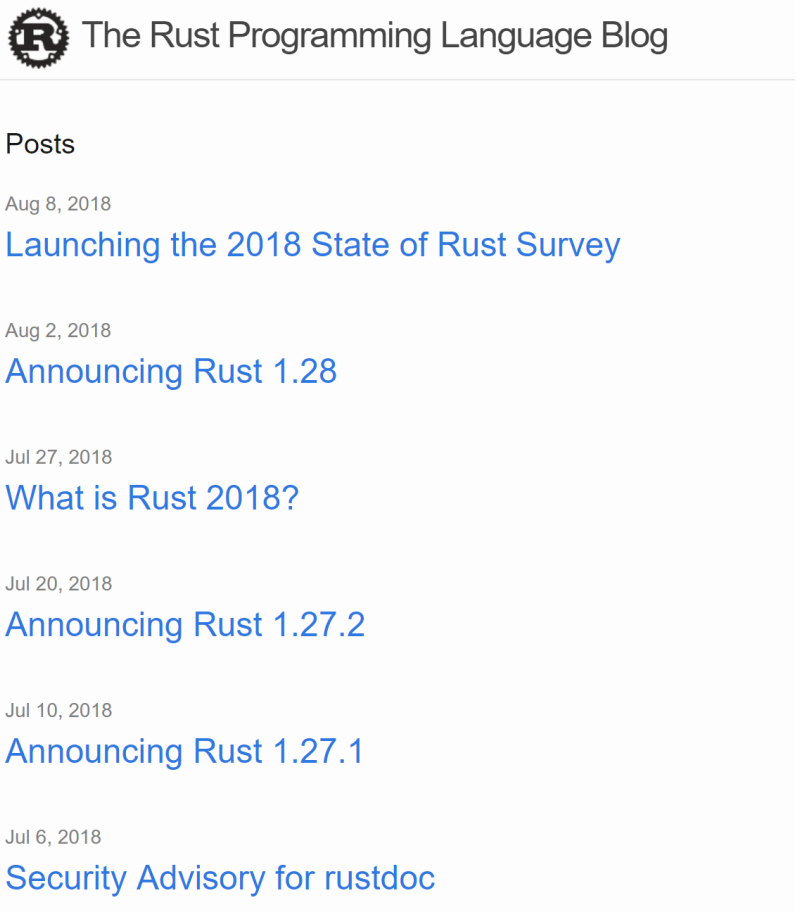
同じサービスがInteger32, LLC.によってもホストされている

Rustの資料: The Rust Community

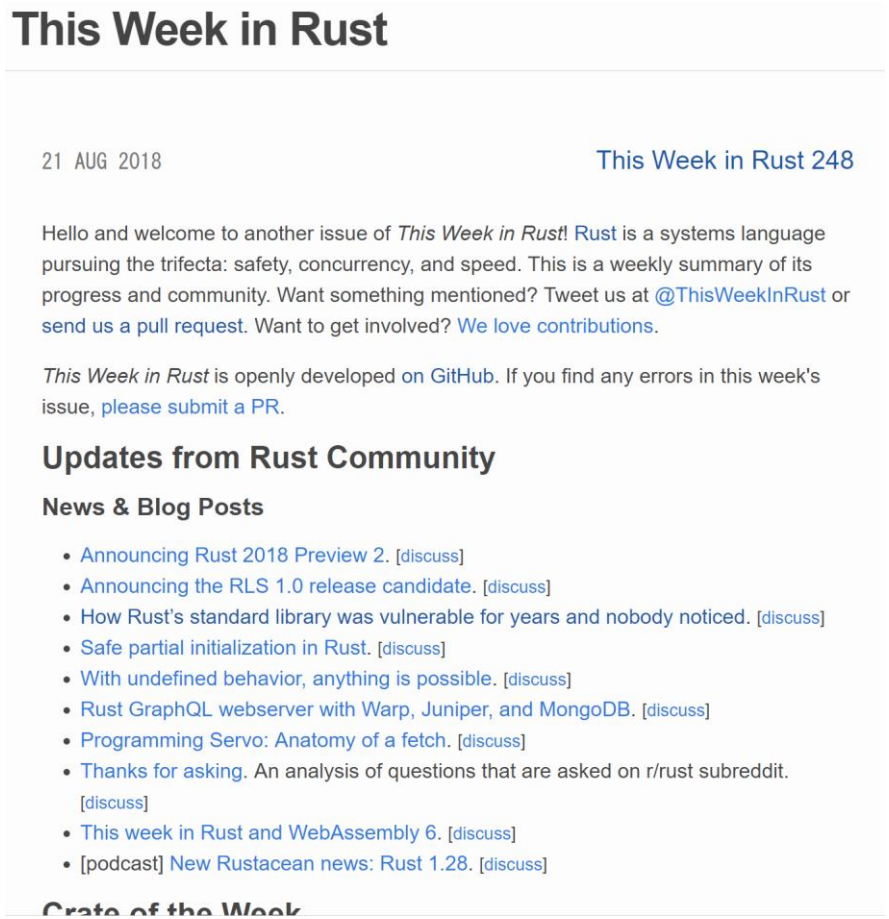


コミュニティーの集まるところ色々へのリンク

Rustの資料: Rust Blog



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]

Crate of the Week

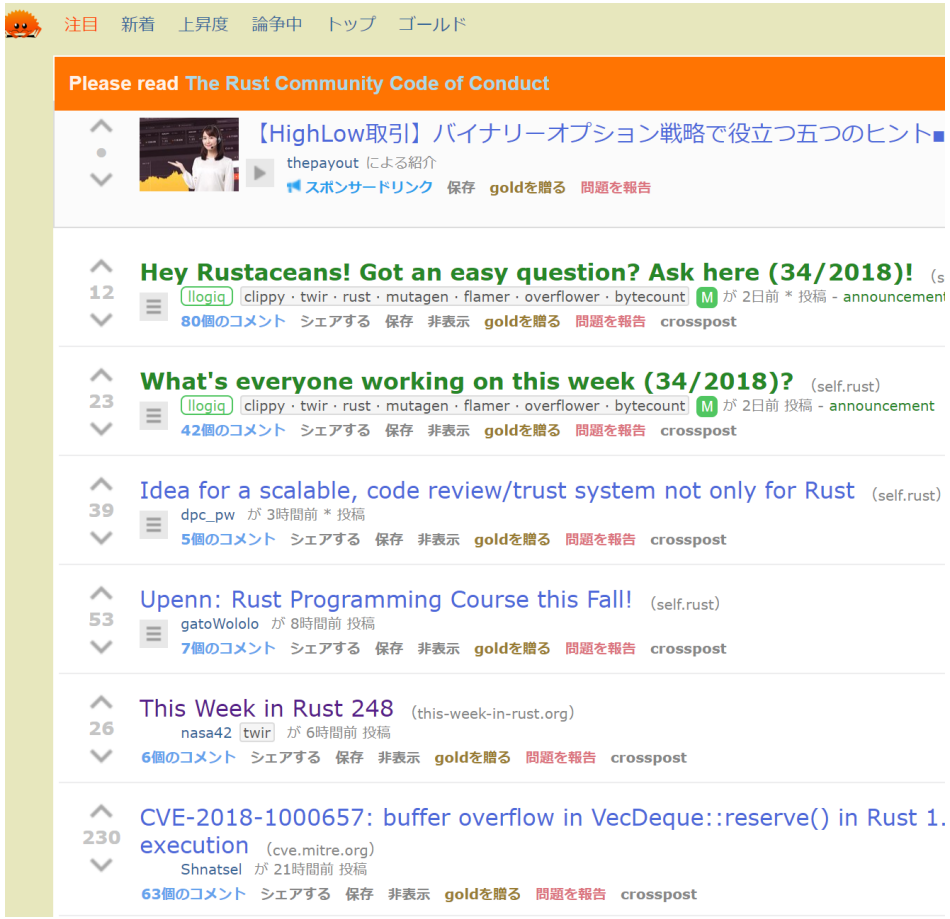
週刊Rust
コンパイラの更新から新着ブログ記事まで、その週の最新情報が手に入る

Rustの資料: Fearless Rust Bloggers

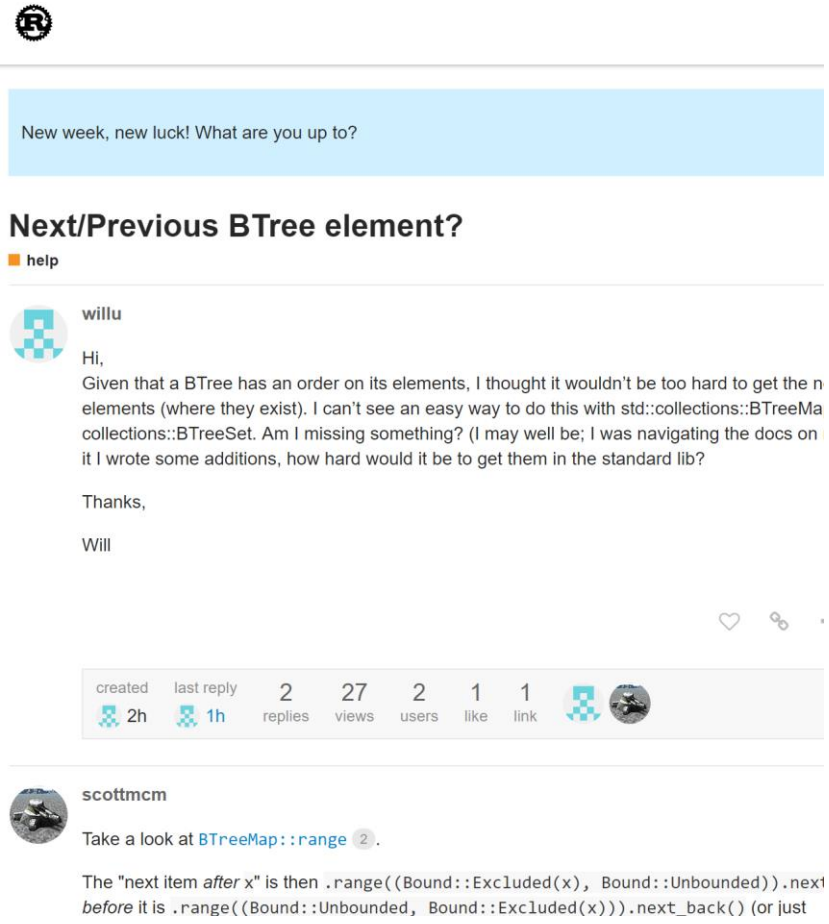
Fearless Rust Bloggers	
Dedicated Rust Bloggers	Rust News That Needs
	Advice:
'static Blog	Jekyll RSS on Github
A rust/piston game developing tutorial	Hugo RSS by default. Read
Aaron Turon	
Adam Perry	http://brson.github.io
Air Mozilla	http://featherweightmusings
Alberto Ruiz - Silicon Island	http://seanmonstar.com/tag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

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



Rustの資料: Rust user forum



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::BTreeMa
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` 2.

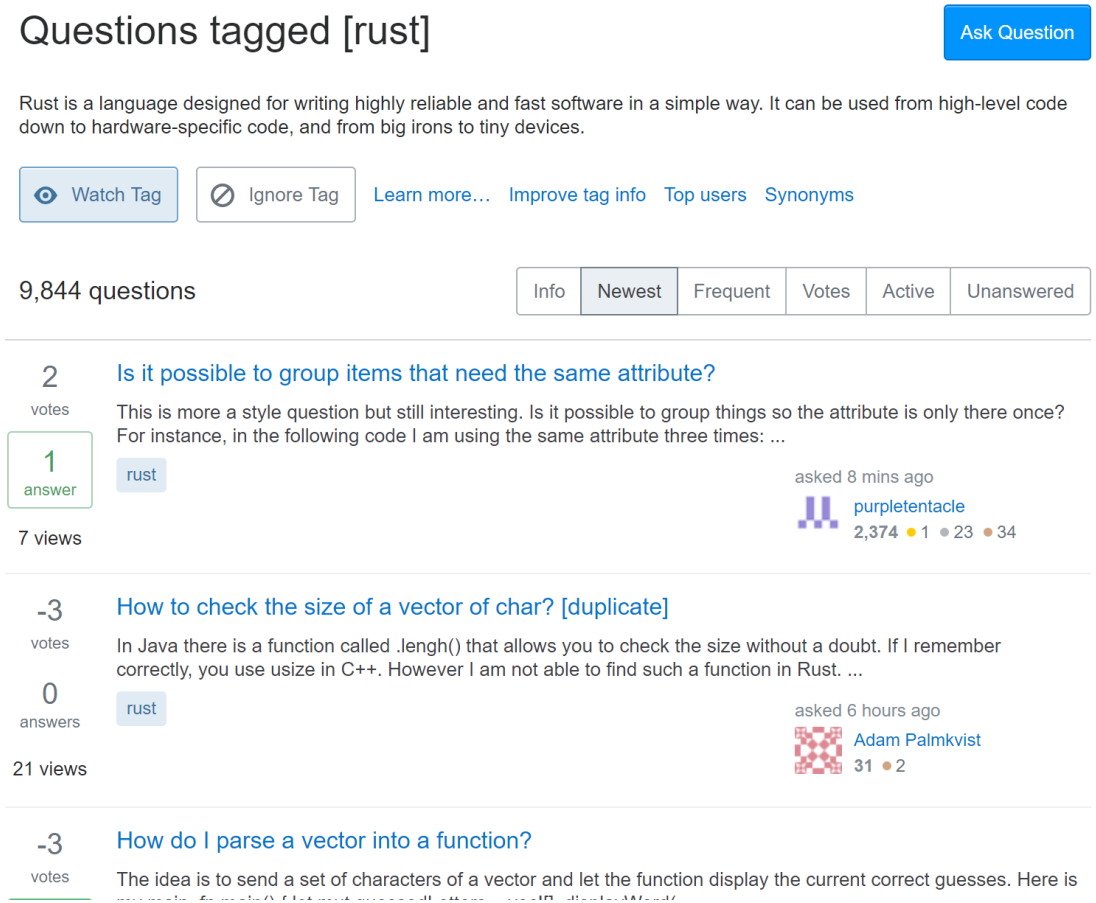
The "next item *after* x" is then `.range((Bound::Excluded(x), Bound::Unbounded)).next`
before it is `.range((Bound::Unbounded, Bound::Excluded(x))).next_back()` (or just

Discourse上の公式フォーラム
質問とかアナウンスとか

Rustの資料: Rust 日本語Slack

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
1 answer
7 views

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: ...

rust

asked 8 mins ago
purpletentacle
2,374 ● 1 ● 23 ● 34

-3 votes
0 answers
21 views

How to check the size of a vector of char? [duplicate]

In Java there is a function called .lengh() 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. ...

rust

asked 6 hours ago
Adam Palmkvist
31 ● 2

-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?

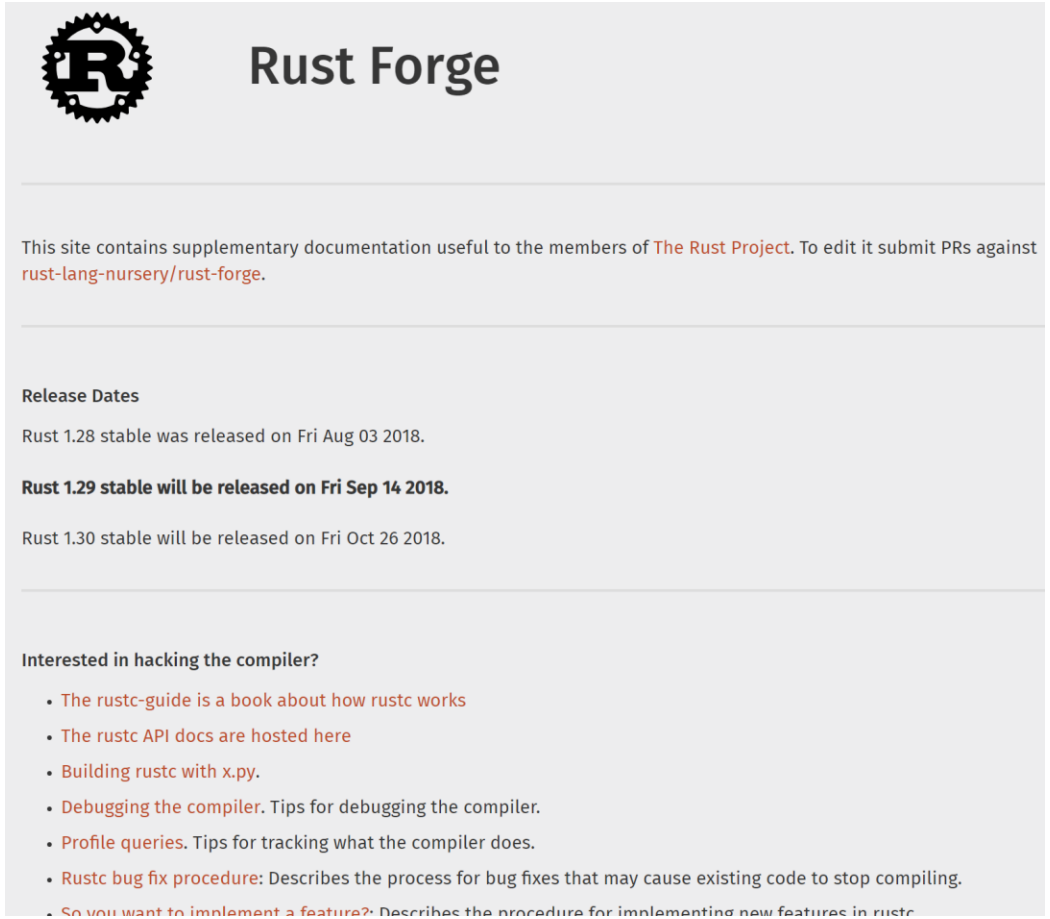
The basic idea of the borrow checker is that values may not be mutated or moved while they are borrowed, but how do we know whether a value is borrowed? The idea is quite simple: whenever you create a borrow, the compiler assigns the resulting reference a **lifetime**. This lifetime corresponds to the 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の開発に関係する
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.

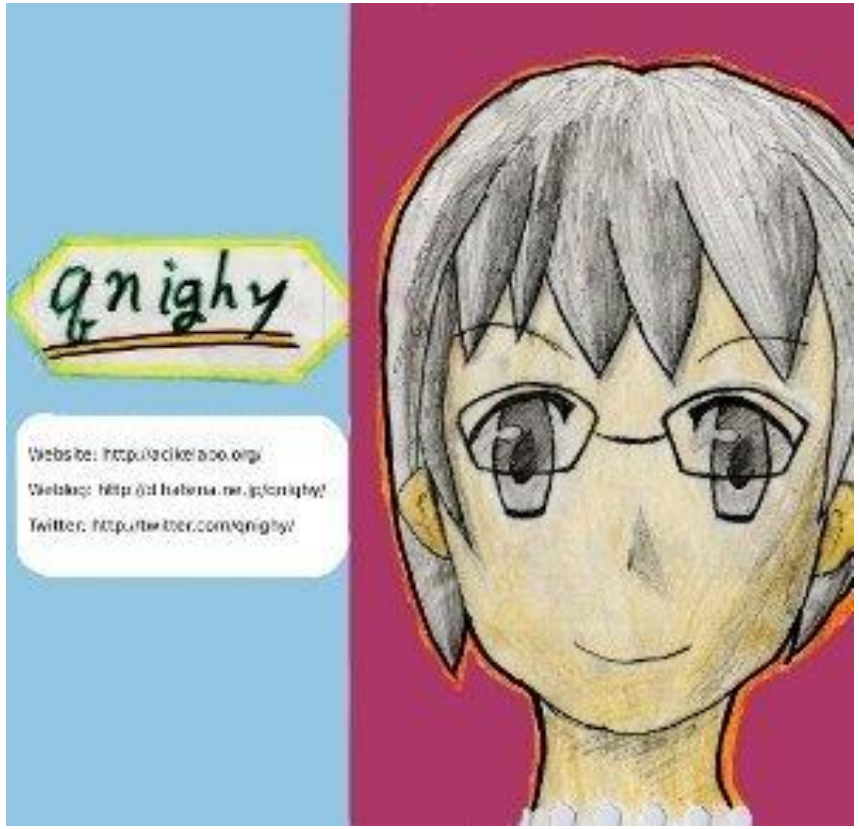
Rustc consists of a number of crates, including `syntax`, `rustc`, `rustc_back`, `rustc_codegen`, `rustc_driver`, and many more. The source for each crate can be found in a directory like `src/libXXX`, where `XXX` is the crate name.

(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

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

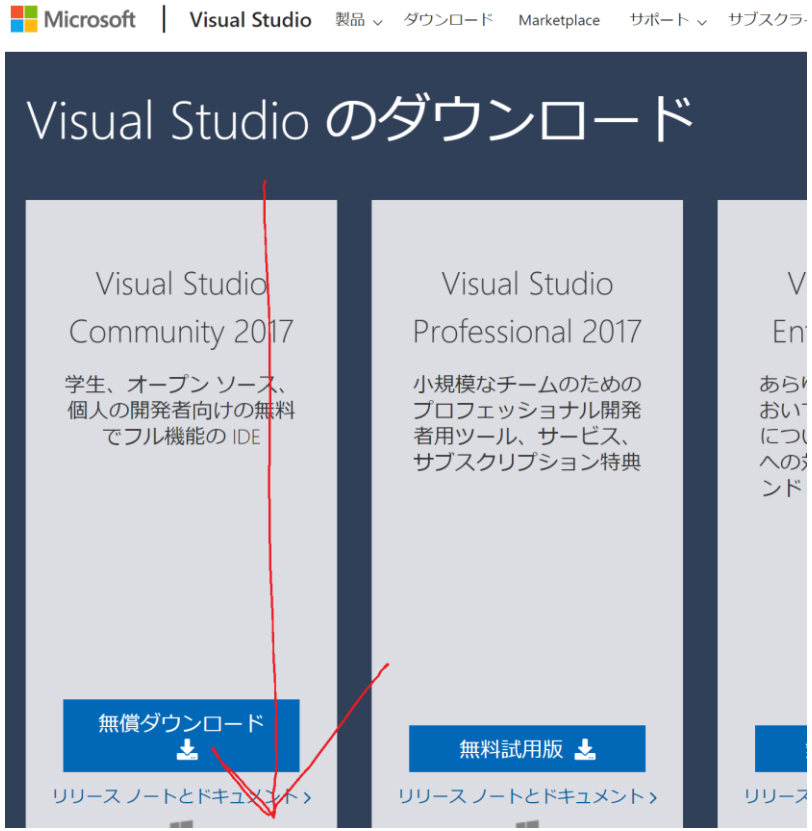
Rustのセットアップ (Windows)

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

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)

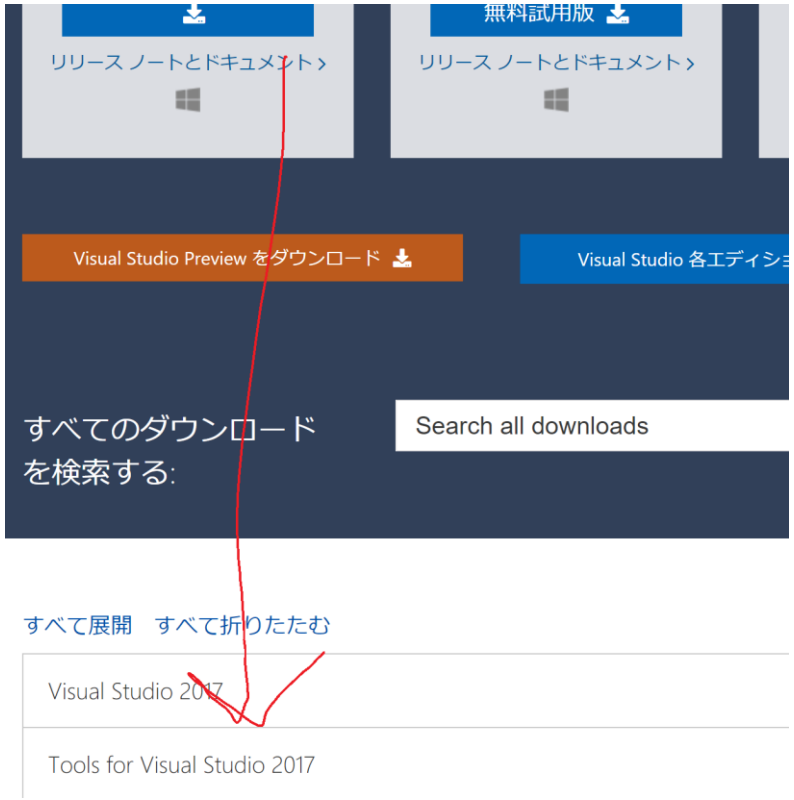


デフォルトのMSVC系ツールチェインを使う場合、
MSVCに含まれるリンカが必要

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)

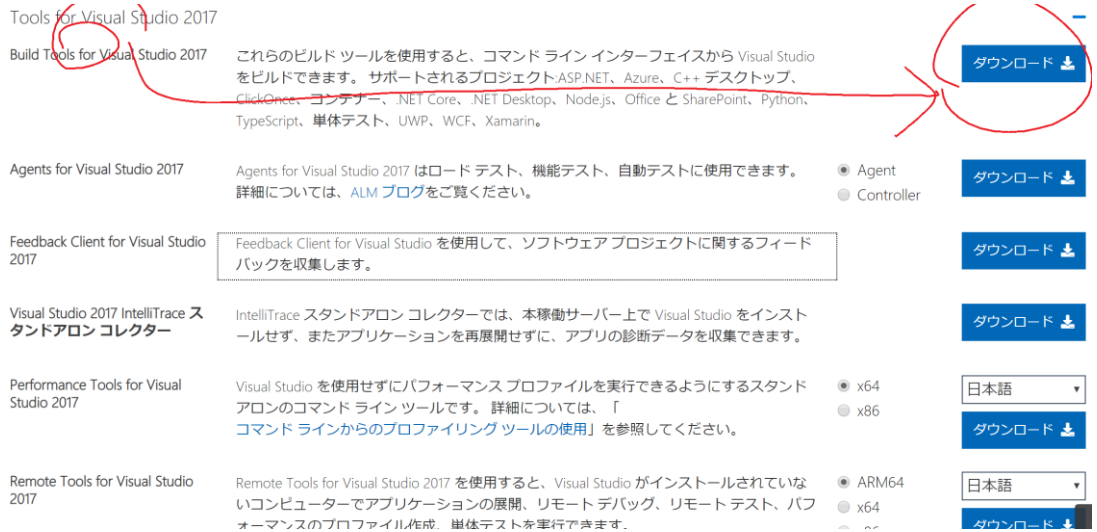


デフォルトのMSVC系ツールチェインを使う場合、
MSVCに含まれるリンカが必要

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系ツールチェインを使う場合、
MSVCに含まれるリンカが必要

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系ツールチェインを使う場合、
MSVCに含まれるリンカが必要

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)

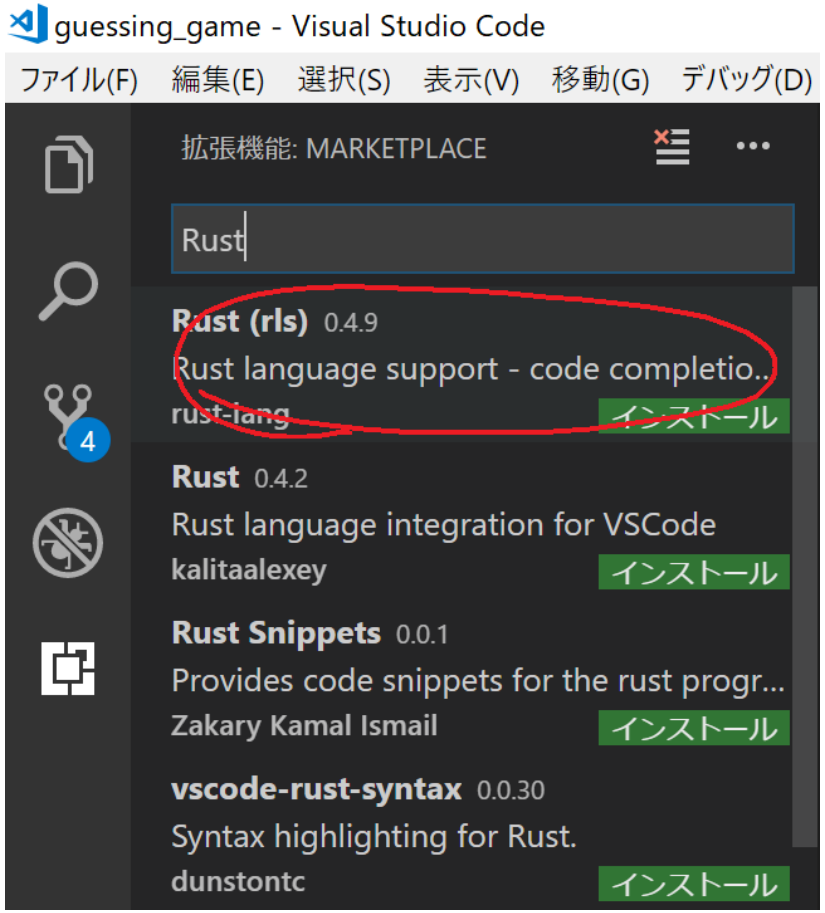
Atom
(atom-ide-rustなど)

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）

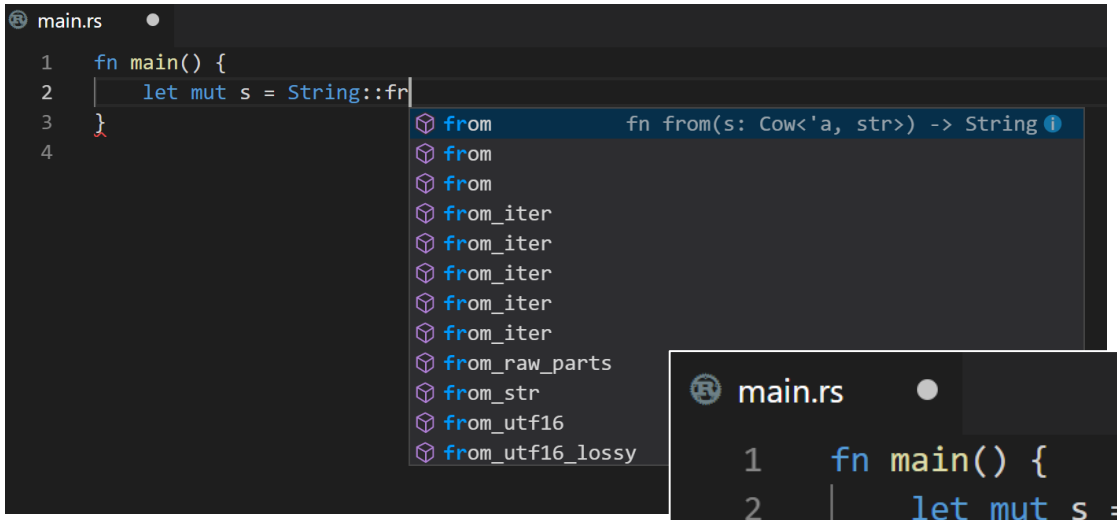
```
$ rustup component add rls-preview rust-analysis
rust-src
info: downloading component 'rls-preview'
info: installing component 'rls-preview'
info: downloading component 'rust-analysis'
info: installing component 'rust-analysis'
```

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

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 mu                                  w();

    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

```
    let mut guess = Strin                ();

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

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

- Result型のメソッド

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_must_use)] 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 =          .: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 Na

[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: {}", gues

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

- Ord::cmp

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`, f
ound 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

```
    gu                                    );

    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_l
    .expect("Failed t    ead line");

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

- 変数guessを宣言…?

変換 – シャドーイング

- この変数guessと

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

io::stdin().rea
    .expect("Failed
                    ad line");

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

- この変数guessは**同姓同名の別人**
  (型も違う)

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()
                                  ");
```

- 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      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();

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

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

- 今回は、パースできなかったら終了する

変換

```
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!");
```

- 今回は、パースできなかったら終了する

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: u32 = 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.len();
```

- 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
コミュニティーの議論を経て、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
```

必要な場所でだけ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	WebAssembly
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	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	組み込み
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	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