



Introduction to Rust

今回の発表の目的

- Rust とそのエコシステムの魅力をわかってもらう
 - Rust がよいものであることはなんとなく知ってると思うので、Rust を使ってみたいって思ってもらえるようなことを話したい
- Rust の始め方を覚えてもらって暇になったときにいつでも Rust を始めてもらえるようにする

Rust ってなに

(想定知識)

- プログラミング言語
 - と、その処理系
- 人気っぽい
 - なんか調べれば出てきそう
- 安全性・並行性を売りにしていることで有名っぽい
 - なんか調べれば出てきそう
- 学習が大変なことで有名っぽい
 - なんか調べれば出てきそう

なぜ Rust か？

Rust プログラミング言語

https://www.rust-lang.org/ja/

インストール 学ぶ Playground ツール ガバナンス コミュニティ ブログ 日本語 (ja)

Rust

効率的で信頼できるソフトウェアを
誰もがつくれる言語

はじめる バージョン 1.46.0

なぜRustか？

パフォーマンス

Rustは非常に高速でメモリ効率が高く、ランタイムやガベージコレクタがないため、パフォーマンス重視のサービスを実装できますし、組込み機器上で実行したり他の言語との調和も簡単にできます。

信頼性

Rustの豊かな型システムと所有権モデルにより、メモリ安全性とスレッド安全性が保証されます。さらに様々な種類のバグをコンパイル時に排除することが可能です。

生産性

Rustには優れたドキュメント、有用なエラーメッセージを備えた使いやすいコンパイラ、および統合されたパッケージマネージャとビルドツール、多数のエディタに対応するスマートな自動補完と型検査機能、自動フォーマッタといった一流のツール群が数多く揃っています。

なぜRustか？

- パフォーマンス
 - はやい
- 信頼性
 - きびしい

なぜRustか？

- パフォーマンス
 - (なにもしなくても) はやい
- 信頼性
 - (なにもしなくても) きびしい
- パフォーマンスと信頼性は有名な話なので割愛！w
- 生産性
 - たのしい
 - これの話をします

生産性

Rustには優れたドキュメント、有用なエラーメッセージを備えた使いやすいコンパイラ、および統合されたパッケージマネージャとビルドツール、多数のエディタに対応するスマートな自動補完と型検査機能、自動フォーマッタといった一流のツール群が数多く揃っています。

- そんなんどんな言語にもあるやん…？と思うかもしれませんぐ！！！（大声）

優れたドキュメント

- がある
 - 公式が入門のための教材（後述）や教科書（後述）を用意するなどドキュメントの拡充に非常に意欲的
 - なんでも安易にドキュメント化する文化がある
 - The * Book
 - チュートリアルからコンパイラの内部構造まで…
- を作れる
 - cargo doc
 - これが簡単なおかげでエコシステム全体でドキュメントのクオリティが高い
 - あと型がついてるからそれだけでドキュメントとして成立しちゃったり

標準ライブラリ

Rust標準ライブラリAPI完全ガイド。

エディションガイド

Rustのエディションに関する手引き。

CARGOブック

Rustのパッケージマネージャとビルドシステムに関する本。

RUSTDOCブック

あなたのcrateに素晴らしいドキュメントを用意する方法を学ぼう。

RUSTCブック

Rustコンパイラで利用可能なオプションに精通しましょう。

コンパイラエラーインデックス

Rustコンパイラが出す可能性のあるエラーの詳細な説明。

アプリケーションドメインのスキルを得る

コマンドラインブック

便利なコマンドラインアプリケーションをRustで作る方法を学ぼう。

WEBASSEMBLYブック

Rustを用いてWebAssemblyを介したブラウザネイティブライブラリを作る。

組込みブック

マイクロコントローラおよびその他の組込みシステムにおけるRustに習熟しよう。

有用なエラーメッセージを備えた使いやすいコンパイラ

- 丁寧もはや直し方を教えてくれる

```
error[E0384]: cannot assign twice to immutable variable `x`
--> src/main.rs:3:5
2 |     let x = 1;
|     |
|     |
|     first assignment to `x`
|     help: make this binding mutable: `mut x`
3 |     x = 2;
|     ^^^^^ cannot assign twice to immutable variable
```

統合されたパッケージマネージャとビルドツール

- cargo というビルドツールが標準で用意されていて、非常に優秀
- 何が優秀かって難しいけどとにかくやるだけなんですよね
- 依存関係の解決が高速だし Dependency Hell も発生しない
 - 参考: How Rust Solved Dependency Hell -
<https://stephencoakley.com/2019/04/24/how-rust-solved-dependency-hell>
- SemVer をみんな使っているので幸せ

多数のエディタに対応するスマートな自動補完と型検査機能、自動フォーマッタ

- まあこれは LSP 実装があるってだけなのでそれはそうって感じですね 🤪

ほかにも… 1

- Clippy
 - コード規約とか文化とかを理解するのには時間がかかるのでしんどいがClippyはそういうのを自動で見つけて指摘・たまに自動修正してくれる
- コード規約を積極的に公式の側で決めてくれるので考えることが少なくて楽
 - これとか <https://rust-lang.github.io/api-guidelines/>

Prefix	Cost
as_	Free
to_	Expensive
into_	Variable

- With a few exceptions, the get_ prefix is not used for getters in Rust code. The get naming is used only when there is a single and obvious thing that could reasonably be gotten by a getter.

ほかにも… 2

- 公式のコンパイラバージョン管理ツール・インストーラ Rustup
 - <https://rustup.rs/>
 - まずインストールでつまずくことはない
 - 別バージョンのコンパイラに一瞬だけ切り替えるとかも簡単
- 完成物の配布が楽
 - LLVM がバックエンドなので対応アーキテクチャが豊富
 - クロスコンパイルがコンパイルオプションだけで完結
 - シングルバイナリもやるだけ
 - C で書かれたライブラリを使ったりしていると面倒だけど
対応策はたくさんあるしそういうのを pure Rust で
置き換えた実装がだいたい存在するので困ることはほとんどない

OK

- Rust のエコシステムは心地がいいので
書いてて非本質な部分で台パンしたくなることが少ないよ
- おれがエコシステムが微妙な文化圏にばかり居ただけかもしけんが
- 一番魅力が詰まっているのは公式サイトなんで読んでくれや
 - <https://www.rust-lang.org/>

Rustでつくろう

2018年、Rustコミュニティは、いくつかの異なる領域におけるプログラミング体験を改善していくと決めました（[2018年のロードマップ](#)をご覧ください）。これらについて、たくさんの高品質なクレートやいくつかの素晴らしい入門ガイドが見つかります。



コマンドライン

Rustの強力なエコシステムならCLIツールを素早く作れます。Rustはアプリのメンテナンスを信頼できるものにし、その配布も簡単です。

ツールを作る



WebAssembly

Rustを使ってJavaScriptをモジュール単位で高速化しましょう。[npm](#)に公開しwebpackでバンドルすればすぐに使えます。

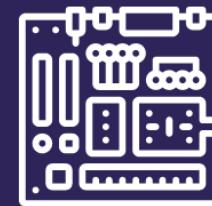
WEBアプリケーションを書く



ネットワーク

予測可能なパフォーマンス。極小のリソースフットプリント。堅固な信頼性。Rustはネットワークサービスにぴったりです。

サーバーを動かす



組込み

低リソースのデバイスがターゲットですか？高レベルの利便性を損なわずに低レベルの制御をしたいですか？Rustにお任せください。

組込から始める

(補足) 逆にRustで書かないかなってやつ

- スクリプティング
 - まず一つファイルで実行したりしない
- おもしろ型ハック
 - Haskell (GHC) みたいな奇想天外な型レベル機能があるわけじゃない
- Web フロントエンド
 - なんかそういうのはあるけどメインストリームとしては高負荷な計算を WebAssembly に押し付けようということであってフロントエンド全体を Rust で書こうというきもちはそんなにないらしいです

Rustを始めよう！



インストール

学ぶ

Playground

ツール

ガバナンス

コミュニティ

ブロケ

日本語(ja)

Rust を学ぶ

Rustを始めよう

「the book」としても親しまれているプログラミング言語Rustは、この言語の概観を基本原理から説明します。読み進める中で複数のプロジェクトを構築し、読み終わるまでにはこの言語の確かな理解が得られます。

あるいはRustlingsなら、コマンドライン上で、Rustのツールチェインのダウンロードとセットアップ方法を確認しながら、Rustの構文の基本的な読み書きを学ぶことができます。Rust by Exampleをあなたの環境で動かすようなものです。

ひとつの言語について何百ページも読むのがあなたの好みに合わなければ、
Rust By Exampleにお任せください。the bookはコードをたくさんの中で説明しますが、 RBE (Rust By Example)はたくさんの中を示し、説明は最小限です。練習問題もあります！

THE BOOKを読もう！

RUSTLINGSコースをやってみよう！

RUST BY EXAMPLEを見てみよう！

Rustlings

- <https://github.com/rust-lang/rustlings/>
 - 実際にエラーを修正していく形で Rust を読み書きできるようになる教材
 - とにかく手を動かし始められるしおすすめ
 - オンラインでも:
<https://gitpod.io/#https://github.com/rust-lang/rustlings>

```
1 // variables4.rs
2 // Make me compile! Execute the command `rustlings hint variables4` if you want a hint :)
3
4 // I AM NOT DONE
5
6 fn main() {
7     let x: i32;
8     println!("Number {}", x);
9 }
10
```

>_ /workspace/rustlings ×

```
✓ Successfully ran exercises/variables/variables3.rs!
! Compiling of exercises/variables/variables4.rs failed! Please try again. Here's the output:
error[E0381]: borrow of possibly-uninitialized variable: `x`
--> exercises/variables/variables4.rs:8:27
 |
8 |     println!("Number {}", x);
|           ^ use of possibly-uninitialized `x`

error: aborting due to previous error
```

For more information about this error, try `rustc --explain E0381`.

```
>_ /workspace/rustlings ×
```

```
8 |     println!("Number {}", x);
|          ^ use of possibly-uninitialized `x`
```

error: aborting due to previous error

For more information about this error, try `rustc --explain E0381`.

hint

Oops! In this exercise, we have a variable binding that we've created on line 7, and we're trying to use it on line 8, but we haven't given it a value. We can't print out something that isn't there; try giving x a value! This is an error that can cause bugs that's very easy to make in any programming language -- thankfully the Rust compiler has caught this for us!



Preview README.md

variables4.rs ×

```
1 // variables4.rs
2 // Make me compile! Execute the command `rustlings hint variables4` if you want a hint :)
3
4 // I AM NOT DONE
5
6 fn main() {
7     let x: i32 = 2;
8     println!("Number {}", x);
9 }
10
```

>_ /workspace/rustlings ×

✓ Successfully ran exercises/variables/variables4.rs!

⚠️⚠️ The code is compiling! ⚠️⚠️

Output:

=====

Number 2

=====

You can keep working on this exercise,
or jump into the next one by removing the `I AM NOT DONE` comment:

```
2 // Make me compile! Execute the command `rustlings hint variables4` if you want a hint :)
3
4 // I AM NOT DONE
5
6 fn main() {
```

そのほかの教材

- The Rust Programming Language - <https://doc.rust-lang.org/book/>
 - 日本語版: <https://doc.rust-jp.rs/book/second-edition/>
 - Rust の教科書; 順当に学ぶならこれ
- Rust by Example - <https://doc.rust-lang.org/stable/rust-by-example/>
 - 日本語版: <https://doc.rust-jp.rs/rust-by-example-ja/>
 - 日本語で一番ハードルが低いのはこれかも？
- keen 『実践 Rust 入門 [言語仕様から開発手法まで]』 技術評論社, 2019
 - 紙の本

書いてて使えそうな資料とか

- The Rust Standard Library - <https://doc.rust-lang.org/std/>
 - これブックマークしておくとよし
- <https://lib.rs/>
 - crates.io よりクレートが探しやすいかも
- Rust Playground - <https://play.rust-lang.org/>
- Rust Language Cheat Sheet - <https://cheats.rs/>
- Rust API Guidelines - <https://rust-lang.github.io/api-guidelines/>
- Rust Cookbook - <https://rust-lang-nursery.github.io/rust-cookbook/>

Q&A

ありがとうございました！

