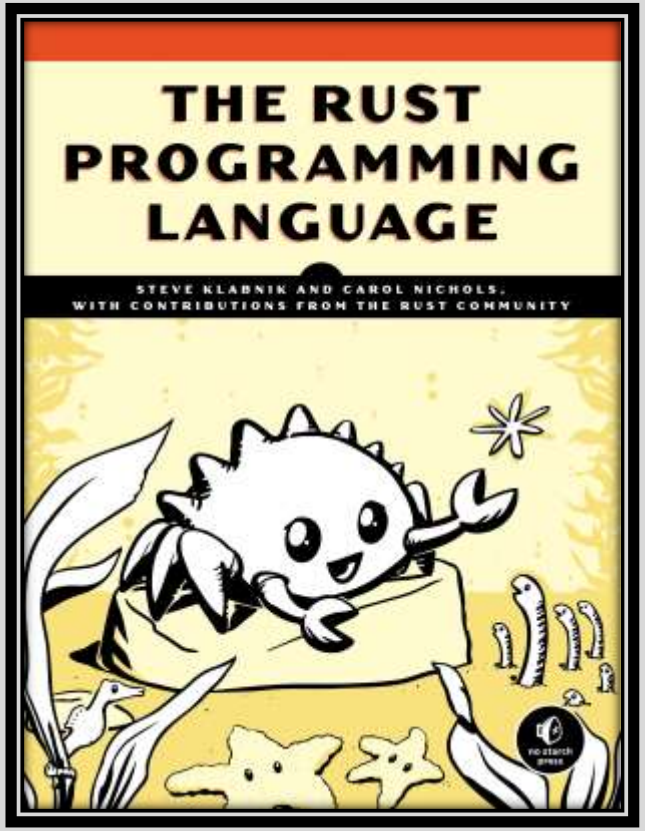
杨德睿 · 启元实验室

2022/05

大纲 · TOC

- Why Another?
- Rust **跨领域能力**
  - 分层抽象：对比RISC-V特权结构设计
  - L1 Bare Metal Rust 裸机开发
  - L2 Cross Rust 交叉编译
  - L3 Std Rust 标准库
  - L0 Meta Rust 元编程
- Rust **生态**
  - 构建工具：Cargo
  - 包管理平台：Crates.io
  - RFC、团队与社区



Why Another?

编程语言需要针对自己要解决的问题设置**关注点**

本节观点来自陈天的Rust培训视频
(陈天@知乎/喜欢历史的程序君@Bilibili)

简洁

透明

性能

安全

学习难度

可移植性

表达能力

故障修复

部署

C

对结构化编程的有效实践

Böhm-Jacopini 理论

任何可计算函数可用顺序、分支和循环的组合表示

简洁

透明

性能

安全

学习难度

可移植性

表达能力

故障修复

部署

Erlang
Elixir

自动重启、热更新

和基于虚拟机的实时性

不要担心异常

`Let it crash!`

处理业务的复杂性，错误处理和重启交给运行环境

简洁

透明

性能

安全

学习难度

可移植性

表达能力

故障修复

部署

Python

以最简单的形式解释执行

自动管理类型和内存

拿我的游标卡尺来!

简洁

透明

性能

安全

学习难度

可移植性

表达能力

故障修复

部署

Java

(in early days)

自动垃圾收集

一次编译，任意运行

~~结构体~~

~~指针~~

~~模板/非擦除泛型~~

~~多继承~~

~~宏~~

守序善良

Python

规律而强大的库和高尚的语法，可以让大多数码农生活得更好。

中立善良

Go

平衡是十分重要的事，单方面地强调高级或性能，是无法达到至善的。

混乱善良

.NET

有自己的一套道德标准，虽然不至于为恶，但也不和大众的道德标准完全相同。

守序中立

Java

只要是规定，不管结果是好是坏，都必须遵行无误。

绝对中立

C

相信底层的性能，因此拒绝任何高级特性。

混乱中立

Scala

躲避权威、憎恨限制、挑战传统，完全的个人主义。

守序邪恶

Haskell

在理论上会依循 Ph.D 的标准，而不管是否简单易懂。

中立邪恶

PHP

为了爽可以做出任何事，一切都是为了爽，就这么简单。

混乱邪恶

JavaScript

全栈工程师会因为贪婪、憎恨或欲望而做出任何事。所幸代码大多杂乱无章，其团体大多组织散乱。

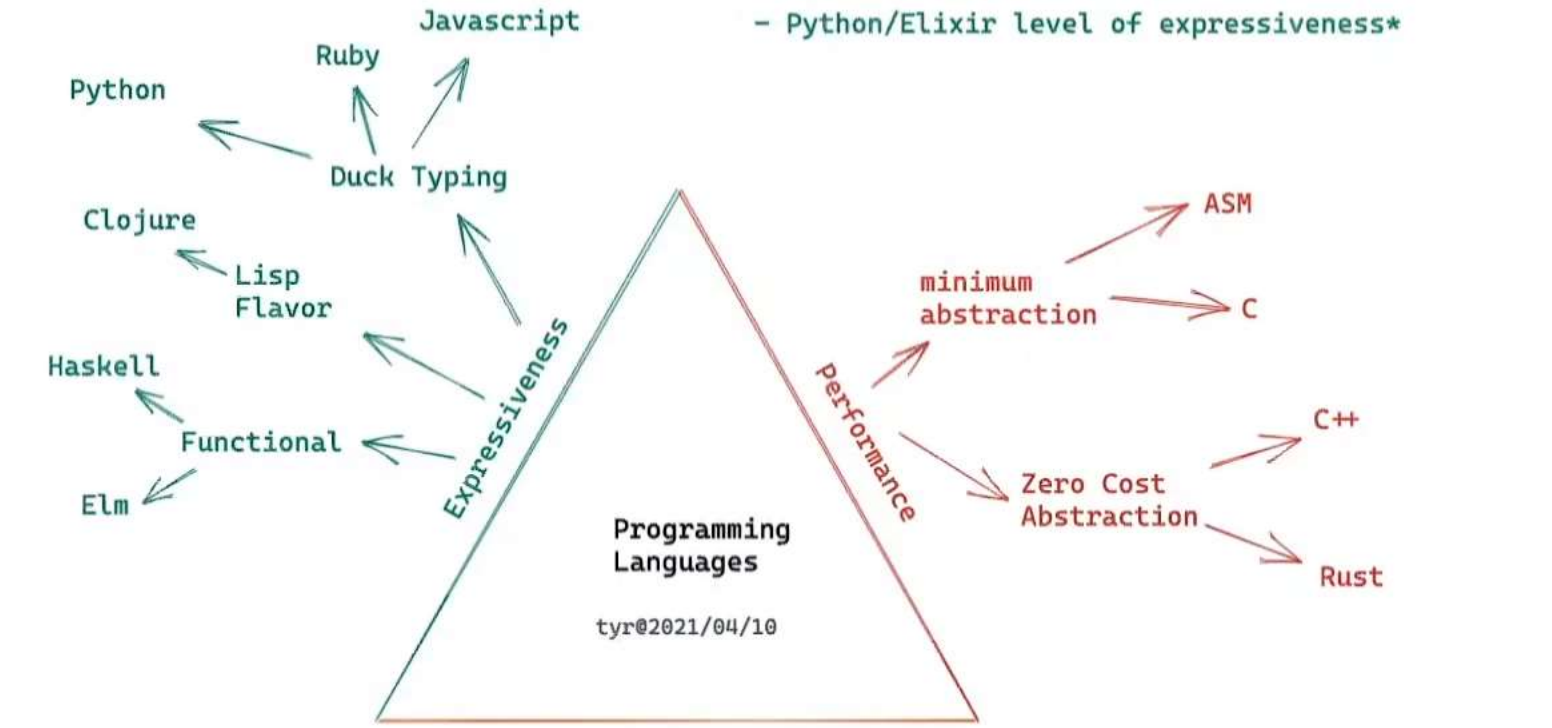
Rust

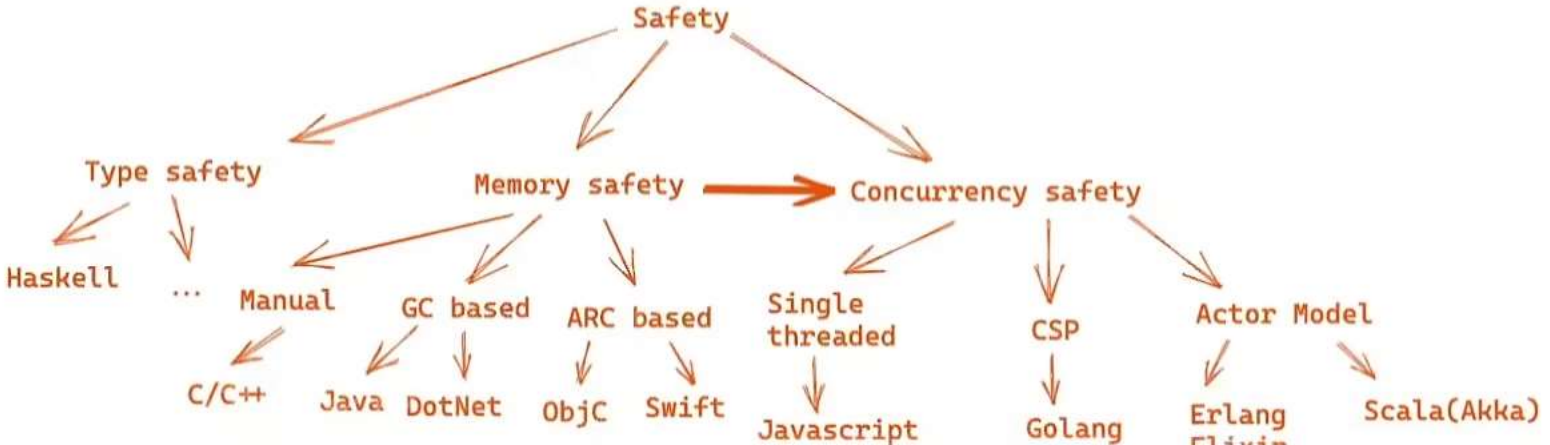
零成本抽象、

内存和并发安全

既要对机器友好，也要考虑工程师的感受

Rust:
- Type, Memory and Concurrency safety
- C/C++ level performance
- Python/Elixir level of expressiveness*



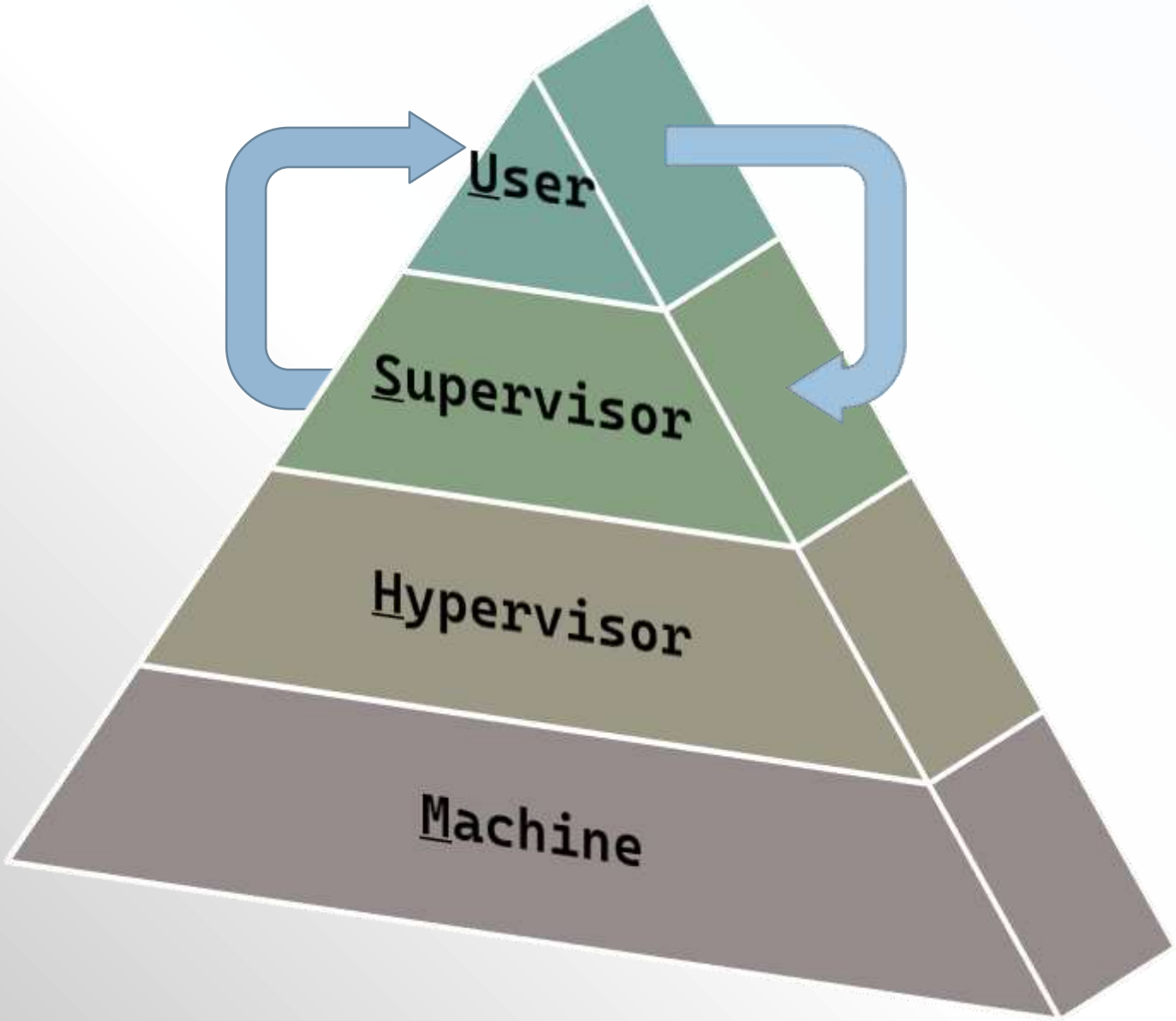


观点

- 所有权、生命周期、基于标记的并发安全：安全性降低了**零成本抽象**原则下的开发难度
- 众多特性共同支撑零成本抽象：默认不可变、移动语义、自定义析构、值或引用传递、卫生宏……
- 以上特性 C++(11+) 全都有，但 Rust 的安全性特性和良好设计的语法降低了开发难度
- 更好的治理原则提高了委员会效率：谨慎引入模板元编程，不保证绝对向前兼容、官方编译器团队……
- 零成本抽象的结果：可裁剪 ⇒ 最小核心 + 可选扩展 + 分层服务 ⇒ Rust 的跨领域能力

Rust 跨领域能力

分层抽象		RISC-V vs Rust
裸机开发	L1	Bare Metal Rust
交叉编译	L2	Cross Rust
标准库	L3	Std Rust
元编程	L0	Meta Rust



RISC-V 特权结构

下层：

通过专用硬件控制上层运行

向上层提供运行环境和运行时服务

<jonanin> any history behind the name?
<graydon> jonanin: "rust"?
<jonanin> yeah
<graydon> people keep asking and I keep making up different explanations.
<graydon> from an email exchange with an early private reviewer of rustboot:
<graydon> >> I love the name. I take it that it refers to your scavenging the
<graydon> >> skeletal hulks of dead languages, now covered in vines ...?
<graydon> >>
<graydon> >> A little. Also big metallic things. And rusts and smuts, fungi. And it's a
<graydon> >> nice substring of "robust".
<jonanin> hah
<jonanin> interesting
<graydon> IOW I don't have a really good explanation. it seemed like a good name. (also a substring of "trust", "frustrating", "rustic" and ... "thrust"?)
<graydon> I think I named it after fungi. rusts are amazing creatures.
<graydon> Five-lifecycle-phase heteroecious parasites. I mean, that's just _crazy_.
<graydon> talk about over-engineered for survival
<jonanin> what does that mean? :]
<graydon> fungi are amazingly robust
<graydon> to start, they are distributed organisms. not single cellular, but also no single point of failure.
<graydon> then depending on the fungi, they have more than just the usual 2 lifecycle phases of critters like us (somatic and gamete)
<jonanin> ohhh
<jonanin> those kind of phases
<graydon> they might have 3, 4, or 5 lifecycle stages. several of which might cross back on one another (meet and reproduce, restart the lineage) and/or self-reproduce or reinfect
<jonanin> but i mean
<jonanin> you have haploid gametes and diploid somatic cells right? what else could there be?
<graydon> and in rusts, some of them actually alternate between multiple different hosts. so a crop failure or host death of one sort doesn't kill off the line.
<graydon> they can double up!
<graydon> http://en.wikipedia.org/wiki/Dikaryon
<graydon> it's madness. basically like someone was looking at sexual reproduction and said "nah, way too failure-prone, let's see how many other variations we can do in parallel"
<jonanin> I can't really understand that lol. I'm only 3/4 the way through my *highschool* bio class
<jonanin> which is not much
<graydon> !
<jonanin> I understood maybe half the words on that page
<evanmcc> that's totally insane
<jonanin> so a gamete becomes two different organisms in parallel?
<graydon> highschool? gosh. I ... definitely was not landing patches on other people's compilers in highscool. precocious! you have a bright future in programming
<rumbleca> rust never sleeps ...
<graydon> jonanin: something like this, yeah. I think basically they have lifecycle phases that are part of two separate reproduction cycles at the same time or something. it's very confusing. I took a mycology course trying to understand all this and it got far too complex for me to follow
<graydon> anyway, I remember being kinda into them back when I was picking the name.
<graydon> but then everyone thinks it's a pun on "chrome" so maybe we should stick with that
<jonanin> hahahha

Rust 历史

- 2006 Graydon Hoare 开始开发
- 2009 Mozilla 聘用 Graydon Hoare
- 2010 成为 Mozilla 官方项目
- 2011 迁移到 LLVM，自举
- 2012 首个版本号
- 2013 Mozilla 基金会宣布将与三星集团合作开发浏览器排版引擎 Servo，此引擎将由 Rust 来实现
- 2015/5/16 Rust 1.0.0 发布

“Rust”：

- 一种真菌：多形态、分布式、鲁棒性
- 读音近似：robust, trust, fast, …
- ~~Chrome：铬 | Rust：锈~~

L0
Bare Metal Rust

Rust 有一套内生的、最小的语法定义，不依赖解释器、虚拟机甚至堆分配、基本I/O，只需要一套图灵式的指令集、一个主存空间和一些通用寄存器，因此可以支持裸机开发。

no-std 模式

- 只能使用核心库
  - Rust 核心库是 Rust 标准库的无依赖基础。它是语言与其库之间的可移植粘合剂，定义了所有 Rust 代码的内在和原始构建块。它不链接到上游库、系统库和 libc。
  - 核心库是最小的：它甚至不知道堆分配，也不提供并发或 I/O。这些东西需要平台集成，而这个库与平台无关。
- 自定义 alloc、panic_handler，
- 使用裸函数（#[naked]）和内联汇编（core::arch::asm!）直接操作机器

```
//! # The Rust Core Library
//!
//! The Rust Core Library is the dependency-free[^free] foundation of [The
//! Rust Standard Library](../std/index.html). It is the portable glue
//! between the language and its libraries, defining the intrinsic and
//! primitive building blocks of all Rust code. It links to no
//! upstream libraries, no system libraries, and no libc.
//!
//! [^free]: Strictly speaking, there are some symbols which are needed but
//!          they aren't always necessary.
//!
//! The core library is *minimal*: it isn't even aware of heap allocation,
//! nor does it provide concurrency or I/O. These things require
//! platform integration, and this library is platform-agnostic.
```

L1
Cross Rust

得益于最小化的核心和编译器的 LLVM 中间态，Rust 可以实现一次编写，任意编译到各种各样的硬件平台和虚拟环境。

甚至一些“奇怪”的环境，比如 Web Assembly

Rust

一门赋予每个人
构建可靠且高效软件能力的语言。

L2
Std Rust

Rust 标准库是可移植 Rust 软件的基础，这是一组针对更广泛的 Rust 生态系统的最小且经过实战测试的共享抽象。 它提供了核心类型，例如 Vec<T> 和 Option<T>，库定义的对语言原语的操作，标准库宏，I/O 和多线程，以及许多其他东西。

```
538  // From implies Into
539  #[stable(feature = "rust1", since = "1.0.
540  #[rustc_const_unstable(feature = "const_c
541  impl<T, U> const Into<U> for T
542  where
543      U: ~const From<T>,
544  {
545      /// Calls `U::from(self)`.
546      ///
547      /// That is, this conversion is whate
548      /// <code>[From]&lt;T&gt; for U</code
549      fn into(self) -> U {
550          U::from(self)
551      }
552  }
```

L0
Meta Rust

利用泛型静态分发、宏、属性元数据和自定义构建过程，Rust 在工具链内部实现了强大的代码生成能力。

Rust 语法特性

变量默认不可变	移动语义	读写互斥	所有权
生命周期	自定义析构与 RAII	模式匹配	闭包
类型系统	泛型	基于标记的并发安全	协程异步

Rust 生态

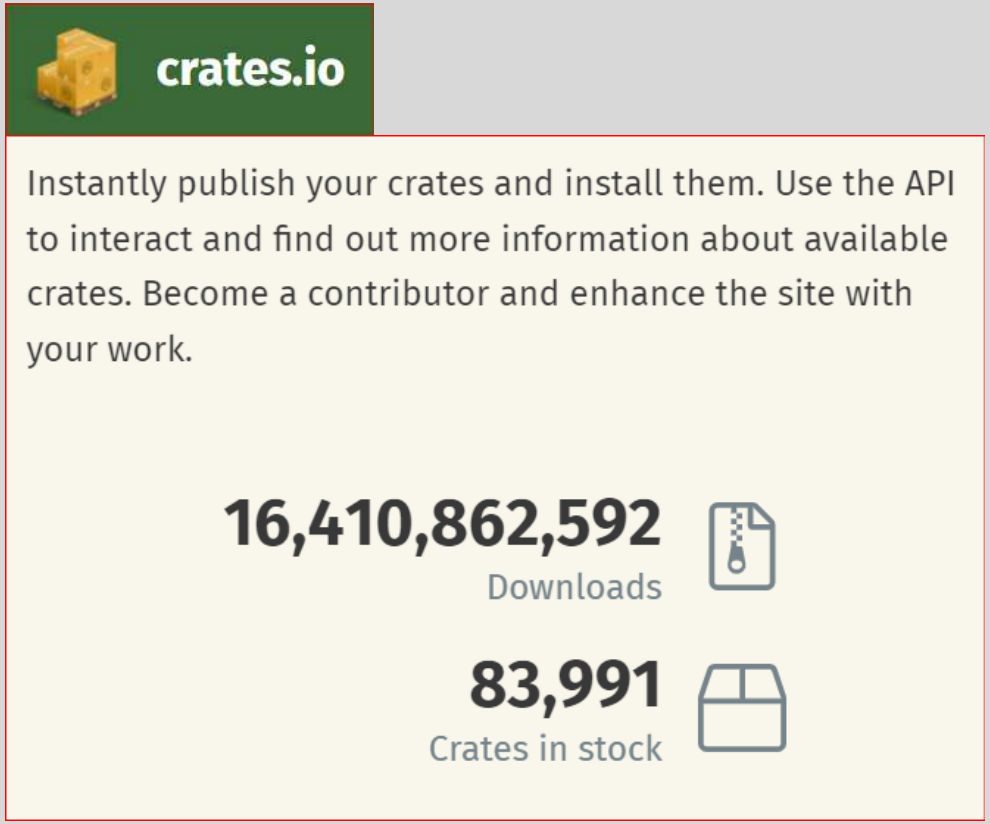
- 构建工具：Cargo
- 包管理平台：Crates.io
- RFC、团队与社区

Cargo

- 使用 `TOML` 描述性文档，结构简洁，逻辑分离到 `build.rs`
- 内置语法检查、获取依赖项、构建、测试、文档生成等一系列功能
- `xtask` 模式扩展
- 应用程序二进制分发
- 开源，更新迅速，文档齐全
- 关键是统一!
- ~~比 `CMake` 高到不知哪里去了~~

Crates.io

- 现代语言的必要条件：包管理器
  - Jvm : Maven central
  - .Net : Nuget Gallery
  - JavaScript: npm
  - Python : PyPI(Python Package Index)
  - Rust : Crates.io
  - ~~C++ : ?~~



RFC、团队与社区

- 语言本身的演进依赖 Github 工作流：Issue → PR → feature
- 每个重要决定都是从征求意见稿（RFC）开始的。任何人都可以参与提案的讨论，权衡利弊以便达成共识
- 定期发布版本
  - edition：3 年
  - stable：6 周
  - nightly：每天
- 组织结构：核心团队–专项团队–工作组–社区


感谢观看！