<https://www.rust-lang.org/zh-CN/>

高性能

Rust 速度惊人且内存利用率极高。由于没有运行时和垃圾回收，它能够胜任对性能要求特别高的服务，可以在嵌入式设备上运行，还能轻松和其他语言集成。

可靠性

Rust 丰富的类型系统和所有权模型保证了内存安全和线程安全，让程序员在编译期就能够消除各种各样的错误。

生产力

Rust 拥有出色的文档、友好的编译器和清晰的错误提示信息， 还集成了一流的工具——包管理器和构建工具， 智能地自动补全和类型检验的多编辑器支持， 以及自动格式化代码等等。

<https://baike.baidu.com/item/Rust%E8%AF%AD%E8%A8%80/9502634>

Rust致力于成为优雅解决高并发和高安全性系统问题的编程语言 [18] ，适用于大型场景，即创造维护能够保持大型系统完整的边界。这就导致了它强调安全，内存布局控制和并发的特点。标准Rust性能与标准C++性能不相上下。

内存安全：

Rust语言系统设计于保证内存安全，它在安全代码里不允许空指针，悬垂指针和数据竞争。数值只能用一系列固定形式来初始化，要求所有输入已经被初始化。在其它语言中复制函数指针或者有效或者为空，比如在链表和二叉树等数据结构中，Rust核心库提供Option类型，用来测试指针是否有值。Rust同时引入添加语法来管理生命周期，而且编译器通过租借检查器来说明相关理由。

内存管理：

Rust不像Go,Java以及.NET Framework那样使用自动垃圾回收系统。不同的是Rust通过RAII来管理内存和资源，还可选引用计数。Rust以低开销提供资源确定性管理。Rust也支持值的栈分配并不表现暗箱。

Rust里也有引用概念（用&符号），不包含运行时引用计数。使得此类指针的安全性已获得租用检查器的编译时验证，阻止悬垂指针和其它形式的未定义行为。

所有权：

Rust有一个所有权系统，所有的值都有一个唯一的属主，值的有效范围跟属主的有效范围一样。值可以通过不可变引用&T传递，可变引用&mut T传递，T值递。在任何时候，或者有多个不可变引用，或者只有一个可变引用。Rust编译器在编译时执行这些规则同时检查所有引用的有效性。

类型多态：

Rust的类型系统支持一种类似类型类的机制，叫“traits”,是被Haskell激发灵感的。这是一种用于特定同质法的设施，通过给类型变量声明添加约束来实现。其它来自Haskell的特性，如更高类型多态还没有支持。

Rust为以let关键字声明的变量而产生类型推导特性。这样的变量不需要赋初值来判断类型。如果某个分支代码没能给变量赋初值将会产生一个编译时错误。对变量进行多次赋值要使用mut关键字标记。

函数可以接收泛型参数，但通常要求泛型实现某种特性或者几种特性。在这种函数里面，这种泛型值只能通过这些特性来使用。这就是说一个泛型函数在定义的时候就能完成类型检查。这是对标C++模板，那种本质鸭式，只能在以具体类型实例化后完成检查。C++概念解决同样的问题，并且有希望能成为C++20标准的一部分。

然而，Rust泛型的实现与C++模板的典型实现类似：每次实例化都会生成一份单独的代码这被称作单态，和经常在Java和Haskell中使用的类型擦除方案类比。单态的好处是为每一个具体用例提供优化代码，缺点是增长了运行时和很多结果文件。

Rust里面的对象系统是基于实现，特性和结构化类型的。实现扮演的角色跟其它语言中的类相似，以关键字impl来定义。继承和多态是由特性提供；它们允许方法被定义或者混合在实现里面。结构类型用于定义字段。实现和特性自己无法定义字段，而且只有特性能够提供继承。至于其它好处，这个阻止了多继承中的菱形继承问题，像C++一样。换句话说，Rust支持接口继承，但是通过复合替代实现继承；参见继承复合。

版本记录编辑

<https://www.zhihu.com/topic/19674381/intro>

Rust是一种系统级编程语言，注重高性能、可靠性和生产力[1]，支持结构化编程、函数式编程、面向对象编程等多种编程范式。

Rust是编译型语言，没有运行时(Runtime)和垃圾回收(Garbage Collector)。Rust使用所有权机制来实现自动内存管理，并以此来保证内存安全。Rust是开源项目，目前托管在Github上，Rust和所有其他官方项目都采用Apache许可证2.0和MIT许可证双重许可[2]。Rust官方软件包管理器为Cargo。

https://zh.wikipedia.org/zh-cn/Rust

Rust是由Mozilla[11]主导开发的通用、编译型编程语言。设计准则为“安全、并发、实用”，[12][13]支持函数式、并发式、过程式以及面向对象的程序设计风格。

Rust语言原本是Mozilla员工Graydon Hoare的私人计划，而Mozilla于2009年开始赞助这个计划 [14]，并且在2010年首次公开[15]。也在同一年，其编译器源代码开始由原本的OCaml语言转移到用Rust语言，进行bootstrapping工作，称做“rustc”[16]，并于2011年实际完成[17]。这个可自我编译的编译器在架构上采用了LLVM做为它的后端。

第一个有版本号的Rust编译器于2012年1月发布。[18]Rust 1.0是第一个稳定版本，于2015年5月15日发布。[19]

Rust在完全开放的情况下开发，并且相当欢迎社区的反馈。在1.0稳定版之前，语言设计也因为透过撰写Servo网页浏览器排版引擎和rustc编译器本身，而有进一步的改善。它虽然由Mozilla资助，但其实是一个共有项目，有很大部分的代码是来自于社区的贡献者。[20]

Rust的设计目标之一，是要使设计大型的互联网客户端和服务器的任务变得更容易[21]。因此更加强调安全性、存储器配置、以及并发处理等方面的特性。

Rust的性能

在性能上，具有额外安全保证的代码会比C++慢一些，例如对Rust的数组进行操作时默认会检查索引是否越界，尽管可以通过一些方式[22]绕过[23]，而C++则不会，但是如果以C++也手工提供保证的情况下，则两者性能上是相似的[24]。

语法

Rust的语法设计，与C语言和C++相当相似，区块（block）使用大括号隔开，流程控制的关键字如if、else、while等等。在保持相似性的同时，Rust也加进了新的关键字，如用于模式匹配（pattern matching）的match（与switch相似）则是使用C／C++系统编程语言的人会相对陌生的概念。尽管在语法上相似，Rust的语义（semantic）和C／C++非常不同。

内存安全

为了提供存储器安全，它的设计不允许空指针和悬空指针[25] [26]。 指针只能透过固定的初始化形态来建构，而所有这些形态都要求它们的输入已经分析过了[27]。Rust有一个检查指针生命期间和指针冻结的系统，可以用来预防在C++中许多的类型错误，甚至是用了智能指针功能之后会发生的类型错误。

所有权

Rust有设计了一个所有权系统，其中所有值都有一个唯一的所有者，并且值的作用域与所有者的作用域相同。值可以通过不可变引用（&T）、可变引用（&mut T）或者通过值本身（T）传递。任何时候，一个变量都可以有多个不可变引用或一个可变引用，这实际上是一个显式的读写锁。Rust编译器在编译时强制执行这些规则，并检查所有引用是否有效。[28][29]

内存管理

早期的Rust虽然有垃圾回收系统，但非如Java或.NET平台的全自动垃圾回收。Rust 1.0已不再使用垃圾回收器，而是全面改用基于引用计数的智能指针来管理内存。

类型与多态

它的类型系统直接地模仿了Haskell语言的类型类概念，并把它称作“traits”，可以把它看成是一种特设多态。Rust的作法是透过在宣告类型变量（type variable）的时候，在上面加上限制条件。至于Haskell的高端类型变量（Higher-kinded polymorphism）则还未支持。

类型推导也是Rust提供的特性之一，使用let语法宣告的变量可以不用宣告类型，亦不需要初始值来推断类型。但如果在稍后的程序中从未指派任何值到该变量，编译器会发出编译时（compile time）错误[30]。 函数可以使用泛型化参数（generics），但是必须绑定Trait。没有任何方法可以使用方法或运算符，又不宣告它们的类型，每一项都必确明确定义。

Rust的对象系统是基于三样东西之上的，即实现（implementation）、Trait以及结构化资料（如struct）。实现的角色类似提供Class关键字的编程语言所代表的意义，并使用impl关键字。继承和多态则透过Trait实现，它们使得方法（method）可以在实现中被定义。结构化资料用来定义字段。实现和（trait）都无法定义字段，并且只有（trait）可以提供继承，藉以躲避C++的“钻石继承问题”（菱型缺陷）。

除了编译器和标准库，Rust生态系统还包括用于软件开发的额外组件。官方推荐使用Rustup，一个Rust工具链安装程序来管理这些组件。

Cargo

Cargo是Rust的软件包管理器，用来下载和构建依赖关系。Cargo还充当了Clippy和其他Rust组件的封装器。它要求项目遵循一定的目录结构。[35]

Cargo.toml文件指定了项目所需的依赖和版本要求，告诉Cargo哪些版本的依赖关系与该包兼容。Cargo默认从crates.io中获取依赖，但Git仓库和本地文件系统中的包也可以作为依赖。[36]