# 关键字

## 目前正在使用的关键字

如下关键字目前有对应其描述的功能。

* as - 强制类型转换，消除特定包含项的 trait 的歧义，或者对 use 和 extern crate 语句中的项重命名
* break - 立刻退出循环
* const - 定义常量或不变裸指针（constant raw pointer）
* continue - 继续进入下一次循环迭代
* crate - 链接（link）一个外部 crate 或一个代表宏定义的 crate 的宏变量
* dyn - 动态分发 trait 对象
* else - 作为 if 和 if let 控制流结构的 fallback
* enum - 定义一个枚举
* extern - 链接一个外部 crate 、函数或变量
* false - 布尔字面值 false
* fn - 定义一个函数或 函数指针类型 (function pointer type)
* for - 遍历一个迭代器或实现一个 trait 或者指定一个更高级的生命周期
* if - 基于条件表达式的结果分支
* impl - 实现自有或 trait 功能
* in - for 循环语法的一部分
* let - 绑定一个变量
* loop - 无条件循环
* match - 模式匹配
* mod - 定义一个模块
* move - 使闭包获取其所捕获项的所有权
* mut - 表示引用、裸指针或模式绑定的可变性性
* pub - 表示结构体字段、impl 块或模块的公有可见性
* ref - 通过引用绑定
* return - 从函数中返回
* Self - 实现 trait 的类型的类型别名
* self - 表示方法本身或当前模块
* static - 表示全局变量或在整个程序执行期间保持其生命周期
* struct - 定义一个结构体
* super - 表示当前模块的父模块
* trait - 定义一个 trait
* true - 布尔字面值 true
* type - 定义一个类型别名或关联类型
* unsafe - 表示不安全的代码、函数、trait 或实现
* use - 引入外部空间的符号
* where - 表示一个约束类型的从句
* while - 基于一个表达式的结果判断是否进行循环

## 保留做将来使用的关键字

如下关键字没有任何功能，不过由 Rust 保留以备将来的应用。

* abstract
* async
* become
* box
* do
* final
* macro
* override
* priv
* try
* typeof
* unsized
* virtual
* yield

## 原始标识符

原始标识符（Raw identifiers）允许你使用通常不能使用的关键字，其带有 r# 前缀。r# 前缀需同时用于函数名和调用

fn r#match(needle: &str, haystack: &str)

# 基本语法

## 引用

默认情况下，Rust 将 prelude 模块中少量的类型引入到每个程序的作用域中。如果需要的类型不在 prelude 中，你必须使用 use 语句显式地将其引入作用域。

use std::io;

## 入口

main 函数是程序的入口点，fn 语法声明了一个新函数，() 表明没有参数，{ 作为函数体的开始。

fn main() {

## 变量

### 赋值

使用 mut 来使一个变量可变

let foo = 5; // 不可变  
let mut bar = 5; // 可变

不能对不可变变量二次赋值

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

### 隐藏

可以定义一个与之前变量同名的新变量，而新变量会 隐藏 之前的变量。可以用相同变量名称来隐藏一个变量，以及重复使用 let 关键字来多次隐藏，

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

隐藏与将变量标记为 mut 是有区别的。当不小心尝试对变量重新赋值时，如果没有使用 let 关键字，就会导致编译时错误。通过使用 let，我们可以用这个值进行一些计算，不过计算完之后变量仍然是不变的。

mut 与隐藏的另一个区别是，当再次使用 let 时，实际上创建了一个新变量，我们可以改变值的类型，但复用这个名字。

## 常量

声明常量使用 const 关键字，并且必须注明值的类型。常量可以在任何作用域中声明，包括全局作用域

Rust 常量的命名规范是使用下划线分隔的大写字母单词，并且可以在数字字面值中插入下划线来提升可读性

const MAX\_POINTS: u32 = 100\_000;

## 数据类型

每一个值都属于某一个 数据类型（data type），这告诉 Rust 它被指定为何种数据，以便明确数据处理方式

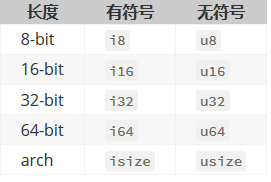
Rust是静态类型（statically typed）语言，在编译时就必须知道所有变量的类型。根据值及其使用方式，编译器通常可以推断出要用的类型。当多种类型均有可能时，必须增加类型注解

### 标量类型

标量（scalar）类型代表一个单独的值。Rust 有四种基本的标量类型：整型、浮点型、布尔类型和字符类型。

#### 整型

整数是一个没有小数部分的数字。



isize 和 usize 类型依赖运行程序的计算机架构：64 位架构上它们是 64 位的， 32 位架构上它们是 32 位的。