Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

19 高级特性

主要内容

- 不安全 Rust
- 高级 Trait
- 高级 类型
- 高级函数和闭包
- 宏

19.1 不安全 Rust

匹配命名变量

- 隐藏着第二个语言，它没有强制内存安全保证：Unsafe Rust（不安全的 Rust）
  - 和普通的 Rust 一样，但提供了额外的“超能力”
- Unsafe Rust 存在的原因：
  - 静态分析是保守的。
    - 使用 Unsafe Rust：我知道自己在做什么，并承担相应风险
  - 计算机硬件本身就是不安全的，Rust 需要能够进行底层系统编程

Unsafe 超能力

- 使用 unsafe 关键字来切换到 unsafe Rust，开启一个块，里面放着 unsafe 代码
- Unsafe Rust 里可执行的四个动作（unsafe 超能力）：
  - 解引用原始指针
  - 调用 unsafe 函数或方法
  - 访问或修改可变的静态变量
  - 实现 unsafe trait
- 注意：
  - unsafe 并没有关闭借用检查或停用其它安全检查
  - 任何内存安全相关的错误必须留在 unsafe 块里
  - 尽可能隔离 unsafe 代码，最好将其封装在安全的抽象里，提供安全的API

解引用原始指针

- 原始指针
  - 可变的：*mut T
  - 不可变的：*const T。意味着指针在解引用后不能直接对其进行赋值
  - 注意：这里的 * 不是解引用符号，它是类型名的一部分。
- 与引用不同，原始指针：
  - 允许通过同时具有不可变和可变指针或多个指向同一位置的可变指针来忽略借用规则
  - 无法保证能指向合理的内存
  - 允许为 null
  - 不实现任何自动清理
- 放弃保证的安全，换取更好的性能/与其它语言或硬件接口的能力

解引用原始指针

- （例子）
- 为什么要用原始指针？
  - 与 C 语言进行接口
  - 构建借用检查器无法理解的安全抽象

调用 unsafe 函数或方法

- unsafe 函数或方法：在定义前加上了 unsafe 关键字
  - 调用前需手动满足一些条件（主要靠看文档），因为 Rust 无法对这些条件进行验证
  - 需要在 unsafe 块里进行调用
- （例子）

创建 unsafe 代码的安全抽象

- 函数包含 unsafe 代码并不意味着需要将整个函数标记为 unsafe
- 将 unsafe 代码包裹在安全函数中是一个常见的抽象
- （例子）

使用 extern 函数调用外部代码

- extern 关键字：简化创建和使用外部函数接口（FFI）的过程。
- 外部函数接口（ FFI ，Foreign Function Interface）：它允许一种编程语言定义函数，并让其它编程语言能调用这些函数
- （例子）
- 应用二进制接口（ABI，Application Binary Interface）：定义函数在汇编层的调用方式
- “C”ABI是最常见的 ABI，它遵循 C 语言的 ABI

从其它语言调用 Rust 函数

- 可以使用 extern 创建接口，其它语言通过它们可以调用 Rust 的函数
- 在 fn 前添加 extern 关键字，并指定 ABI
- 还需添加 #[no_mangle] 注解：避免 Rust 在编译时改变它的名称
- （例子）

访问或修改一个可变静态变量

- Rust 支持全局变量，但因为所有权机制可能产生某些问题，例如数据竞争
- 在 Rust 里，全局变量叫做静态（static）变量
- （例子）

静态变量

- 静态变量与常量类似
- 命名：SCREAMING_SNAKE_CASE
- 必须标注类型
- 静态变量只能存储 ‘static 生命周期的引用，无需显式标注
- 访问不可变静态变量是安全的

常量和不可变静态变量的区别

- 静态变量：有固定的内存地址，使用它的值总会访问同样的数据
- 常量：允许使用它们的时候对数据进行复制
- 静态变量：可以是可变的，访问和修改静态可变变量是不安全（unsafe）的
- （例子）

实现不安全（unsafe）trait

- 当某个 trait 中存在至少一个方法拥有编译器无法校验的不安全因素时，就称这个 trait 是不安全的
- 声明 unsafe trait：在定义前加 unsafe 关键字
  - 该 trait 只能在 unsafe 代码块中实现
- （例子）

何时使用 unsafe 代码

- 编译器无法保证内存安全，保证 unsafe 代码正确并不简单
- 有充足理由使用 unsafe 代码时，就可以这样做
- 通过显式标记 unsafe，可以在出现问题时轻松的定位

再见