Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

19.5 宏

宏 macro

- 宏在 Rust 里指的是一组相关特性的集合称谓：
  - 使用 macro_rules! 构建的声明宏（declarative macro）
  - 3 种过程宏
    - 自定义 #[derive] 宏，用于 struct 或 enum，可以为其指定随 derive 属性添加的代码
    - 类似属性的宏，在任何条目上添加自定义属性
    - 类似函数的宏，看起来像函数调用，对其指定为参数的 token 进行操作

函数与宏的差别

- 本质上，宏是用来编写可以生成其它代码的代码（元编程，metaprogramming）
- 函数在定义签名时，必须声明参数的个数和类型，宏可处理可变的参数
- 编译器会在解释代码前展开宏
- 宏的定义比函数复杂得多，难以阅读、理解、维护
- 在某个文件调用宏时，必须提前定义宏或将宏引入当前作用域；
- 函数可以在任何位置定义并在任何位置使用

macro_rules! 声明宏（弃用）

- Rust 中最常见的宏形式：声明宏
  - 类似 match 的模式匹配
  - 需要使用 marco_rules!
- （例子）

基于属性来生成代码的过程宏

- 这种形式更像函数（某种形式的过程）一些
  - 接收并操作输入的 Rust 代码
  - 生成另外一些 Rust 代码作为结果
- 三种过程宏：
  - 自定义派生
  - 属性宏
  - 函数宏
- 创建过程宏时：
  - 宏定义必须单独放在它们自己的包中，并使用特殊的包类型
- （例子）

自定义 derive 宏

- 需求：
  - 创建一个 hello_macro 包，定义一个拥有关联函数 hello_macro 的 HelloMacro trait
  - 我们提供一个能自动实现 trait 的过程宏
  - 在它们的类型上标注 #[derive(HelloMacro)]，进而得到 hello_macro 的默认实现
- （例子）

类似属性的宏

- 属性宏与自定义 derive 宏类似
  - 允许创建新的属性
  - 但不是为 derive 属性生成代码
- 属性宏更加灵活：
  - derive 只能用于 struct 和 enum
  - 属性宏可以用于任意条目，例如函数
- （例子）

类似函数的宏

- 函数宏定义类似于函数调用的宏，但比普通函数更加灵活
- 函数宏可以接收 TokenStream 作为参数
- 与另外两种过程宏一样，在定义中使用 Rust 代码来操作 TokenStream
- （例子）

再见