Rust 编程语言入门

杨旭，微软MVP
Rust、Go、C#开发者

10.7 生命周期（3/4）

深入理解生命周期

- 指定生命周期参数的方式依赖于函数所做的事情
- （例子）

- 从函数返回引用时，返回类型的生命周期参数需要与其中一个参数的生命周期匹配：
- 如果返回的引用没有指向任何参数，那么它只能引用函数内创建的值：
  - 这就是悬垂引用：该值在函数结束时就走出了作用域
- （例子）

Struct 定义中的生命周期标注

- Struct 里可包括：
  - 自持有的类型
  - 引用：需要在每个引用上添加生命周期标注
- （例子）

生命周期的省略

- 我们知道：
  - 每个引用都有生命周期
  - 需要为使用生命周期的函数或 struct 指定生命周期参数
- （例子）

生命周期省略规则

- 在 Rust 引用分析中所编入的模式称为**生命周期省略规则**。
  - 这些规则无需开发者来遵守
  - 它们是一些特殊情况，由编译器来考虑
  - 如果你的代码符合这些情况，那么就无需显式标注生命周期
- 生命周期省略规则不会提供完整的推断：
  - 如果应用规则后，引用的生命周期仍然模糊不清 → 编译错误
  - 解决办法：添加生命周期标注，表明引用间的相互关系

输入、输出生命周期

- 生命周期在：
  - 函数/方法的参数：输入生命周期
  - 函数/方法的返回值：输出生命周期

生命周期省略的三个规则

- 编译器使用 3 个规则在没有显式标注生命周期的情况下，来确定引用的生命周期
  - 规则 1 应用于输入生命周期
  - 规则 2、3 应用于输出生命周期
  - 如果编译器应用完 3 个规则之后，仍然有无法确定生命周期的引用 → 报错
  - 这些规则适用于 fn 定义和 impl 块
- 规则 1：每个引用类型的参数都有自己的生命周期
- 规则 2：如果只有 1 个输入生命周期参数，那么该生命周期被赋给所有的输出生命周期参数
- 规则 3：如果有多个输入生命周期参数，但其中一个是 &self 或 &mut self（是方法），那么 self 的生命周期会被赋给所有的输出生命周期参数

生命周期省略的三个规则 – 例子

- 假设我们是编译器：
- fn first_word(s: &str) -> &str {
- fn first_word<'a>(s: &'a str) -> &str {
- fn first_word<'a>(s: &'a str) -> &'a str {

- fn longest(x: &str, y: &str) -> &str {
- fn longest<'a, 'b>(x: &'a str, y: &'b str) -> &str {

再见