Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

19.3 高级类型

使用 newtype 模式实现类型安全和抽象

- newtype 模式可以：
  - 用来静态的保证各种值之间不会混淆并表明值的单位
  - 为类型的某些细节提供抽象能力
  - 通过轻量级的封装来隐藏内部实现细节

使用类型别名创建类型同义词

- Rust 提供了类型别名的功能：
  - 为现有类型生产另外的名称（同义词）
  - 并不是一个独立的类型
  - 使用 type 关键字
- 主要用途：减少代码字符重复
- （例子）

Never 类型

- 有一个名为 ! 的特殊类型：
  - 它没有任何值，行话称为空类型（empty type）
  - 我们倾向于叫它 never 类型，因为它在不返回的函数中充当返回类型
- 不返回值的函数也被称作发散函数（diverging function）
- （例子）

动态大小的 Sized Trait

- Rust 需要在编译时确定为一个特定类型的值分配多少空间。
- 动态大小的类型（Dynamically Sized Types，DST）的概念：
  - 编写代码时使用只有在运行时才能确定大小的值
- str 是动态大小的类型（注意不是 &str）：只有运行时才能确定字符串的长度
  - 下列代码无法正常工作：
    - let s1: str = "Hello there!";
    - let s2: str = "How's it going?";
  - 使用 &str 来解决：
    - str 的地址
    - str 的长度

Rust 使用动态大小类型的通用方式

- 附带一些额外的元数据来存储动态信息的大小
  - 使用动态大小类型时总会把它的值放在某种指针后边

另外一种动态大小的类型：trait

- 每个 trait 都是一个动态大小的类型，可以通过名称对其进行引用
- 为了将 trait 用作 trait 对象，必须将它放置在某种指针之后
  - 例如 &dyn Trait 或 Box<dyn Trait>（Rc<dyn Trait>）之后

Sized trait

- 为了处理动态大小的类型，Rust 提供了一个 Sized trait 来确定一个类型的大小在编译时是否已知
  - 编译时可计算出大小的类型会自动实现这一 trait
  - Rust 还会为每一个泛型函数隐式的添加 Sized 约束
- （例子）
- 默认情况下，泛型函数只能被用于编译时已经知道大小的类型，可以通过特殊语法解除这一限制

?Sized trait 约束

- （例子）
- T 可能是也可能不是 Sized
- 这个语法只能用在 Sized 上面，不能被用于其它 trait

再见