Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

17.2 使用 trait 对象来存储不同类型的值

有这样一个需求

- 创建一个 GUI 工具：
  - 它会遍历某个元素的列表，依次调用元素的 draw 方法进行绘制
  - 例如：Button、TextField 等元素

- 在面向对象语言里：
  - 定义一个 Component 父类，里面定义了 draw 方法
  - 定义 Button、TextField 等类，继承与 Component 类

为共有行为定义一个 trait

- Rust 避免将 struct 或 enum 称为对象，因为它们与 impl 块是分开的
- trait 对象有些类似于其它语言中的对象：
  - 它们某种程度上组合了数据与行为
- trait 对象与传统对象不同的地方：
  - 无法为 trait 对象添加数据
- trait 对象被专门用于抽象某些共有行为，它没其它语言中的对象那么通用
- （例子）

Trait 对象执行的是动态派发

- 将 trait 约束作用于泛型时，Rust 编译器会执行单态化：
  - 编译器会为我们用来替换泛型类型参数的每一个具体类型生成对应函数和方法的非泛型实现。
- 通过单态化生成的代码会执行静态派发（static dispatch），在编译过程中确定调用的具体方法
- 动态派发（dynamic dispatch）：
  - 无法在编译过程中确定你调用的究竟是哪一种方法
  - 编译器会产生额外的代码以便在运行时找出希望调用的方法
- 使用 trait 对象，会执行动态派发：
  - 产生运行时开销
  - 阻止编译器内联方法代码，使得部分优化操作无法进行

Trait 对象必须保证对象安全

- 只能把满足对象安全（object-safe）的 trait 转化为 trait 对象
- Rust 采用一系列规则来判定某个对象是否安全，只需记住两条：
  - 方法的返回类型不是 Self
  - 方法中不包含任何泛型类型参数
- （例子）

再见