Rust 编程语言入门

杨旭，微软MVP
Rust、Go、C#开发者

9.4 什么时候应该用 panic!

总体原则

- 在定义一个可能失败的函数时，优先考虑返回 Result
- 否则就 panic!

编写示例、原型代码、测试

- 可以使用 panic!
  - 演示某些概念：unwrap
  - 原型代码：unwrap、expect
  - 测试：unwrap、expect

有时你比编译器掌握更多的信息

- 你可以确定 Result 就是 Ok：unwrap
- （例子）

错误处理的指导性建议

- 当代码最终可能处于损坏状态时，最好使用 panic!
- 损坏状态（Bad state）：某些假设、保证、约定或不可变性被打破
  - 例如非法的值、矛盾的值或空缺的值被传入代码
  - 以及下列中的一条：
    - 这种损坏状态并不是预期能够偶尔发生的事情。
    - 在此之后，您的代码如果处于这种损坏状态就无法运行。
    - 在您使用的类型中没有一个好的方法来将这些信息（处于损坏状态）进行编码。

场景建议

- 调用你的代码，传入无意义的参数值：panic!
- 调用外部不可控代码，返回非法状态，你无法修复：panic!
- 如果失败是可预期的：Result
- 当你的代码对值进行操作，首先应该验证这些值：panic!

为验证创建自定义类型

- （例子）
- 创建新的类型，把验证逻辑放在构造实例的函数里。
- （例子）
- getter：返回字段数据
  - 字段是私有的（上例中）：外部无法直接对字段赋值

再见