Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

15.5 RefCell<T>和内部可变性

内部可变性（interior mutability）

- 内部可变性是 Rust 的设计模式之一
- 它允许你在只持有不可变引用的前提下对数据进行修改
  - 数据结构中使用了 unsafe 代码来绕过 Rust 正常的可变性和借用规则

RefCell<T>

- 与 Rc<T> 不同，RefCell<T> 类型代表了其持有数据的唯一所有权。

回忆一下：
借用规则

在任何给定的时间里，你要么只能拥有一个可变引用，要么只能拥有任意数量的不可变引用

引用总是有效的

RefCell<T> 与 Box<T> 的区别

Box<T>

- **编译阶段**强制代码遵守借用规则
- 否则出现错误

RefCell<T>

- 只会在**运行时**检查借用规则
- 否则触发 panic

借用规则
在不同阶段进行检查的比较

编译阶段

- 尽早暴露问题
- 没有任何运行时开销
- 对大多数场景是最佳选择
- 是 Rust 的默认行为

运行时

- 问题暴露延后，甚至到生产环境
- 因借用计数产生些许性能损失
- 实现某些特定的内存安全场景（不可变环境中修改自身数据）

RefCell<T>

- 与 Rc<T> 相似，只能用于单线程场景

选择 Box<T>、Rc<T>、RefCell<T> 的依据

	Box<T>	Rc<T>	RefCell<T>
同一数据的所有者	一个	多个	一个
可变性、借用检查	可变、不可变借用 （编译时检查）	不可变借用 （编译时检查）	可变、不可变借用 （运行时检查）

- 其中：即便 RefCell<T> 本身不可变，但仍能修改其中存储的值

内部可变性：可变的借用一个不可变的值

- （例子）

使用 RefCell<T> 在运行时记录借用信息

- 两个方法（安全接口）：
  - borrow 方法
    - 返回智能指针 Ref<T>，它实现了 Deref
  - borrow_mut 方法
    - 返回智能指针 RefMut<T>，它实现了 Deref

使用 RefCell<T> 在运行时记录借用信息

- RefCell<T> 会记录当前存在多少个活跃的 Ref<T> 和 RefMut<T> 智能指针：
  - 每次调用 borrow：不可变借用计数加 1
  - 任何一个 Ref<T> 的值离开作用域被释放时：不可变借用计数减 1
  - 每次调用 borrow_mut：可变借用计数加 1
  - 任何一个 RefMut<T> 的值离开作用域被释放时：可变借用计数减 1

- 以此技术来维护借用检查规则：
  - 任何一个给定时间里，只允许拥有多个不可变借用或一个可变借用。

将 Rc<T> 和 RefCell<T> 结合使用来实现一个拥有多重所有权的可变数据

- （例子）

其它可实现内部可变性的类型

- Cell<T>：通过复制来访问数据
- Mutex<T>：用于实现跨线程情形下的内部可变性模式

再见