Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

16.3 共享状态的并发

使用共享来实现并发

- Go 语言的名言：**不要用共享内存来通信**，*要用通信来共享内存*。
- Rust 支持通过共享状态来实现并发。
- Channel 类似单所有权：一旦将值的所有权转移至 Channel，就无法使用它了
- 共享内存并发类似多所有权：多个线程可以同时访问同一块内存

使用 Mutex 来每次只允许一个线程来访问数据

- Mutex 是 mutual exclusion（互斥锁）的简写
- 在同一时刻，Mutex 只允许一个线程来访问某些数据
- 想要访问数据：
  - 线程必须首先获取互斥锁（lock）
    - lock 数据结构是 mutex 的一部分，它能跟踪谁对数据拥有独占访问权
  - mutex 通常被描述为：通过锁定系统来保护它所持有的数据

Mutex 的两条规则

- 在使用数据之前，必须尝试获取锁（lock）。
- 使用完 mutex 所保护的数据，必须对数据进行解锁，以便其它线程可以获取锁。

Mutex<T> 的 API

- 通过 Mutex::new(数据) 来创建 Mutex<T>
  - Mutex<T> 是一个智能指针
- 访问数据前，通过 lock 方法来获取锁
  - 会阻塞当前线程
  - lock 可能会失败
  - 返回的是 MutexGuard（智能指针，实现了 Deref 和 Drop）
- （例子）

多线程共享 Mutex<T>

- （例子）

多线程的多重所有权

- （例子）

使用 Arc<T> 来进行原子引用计数

- Arc<T> 和 Rc<T> 类似，它可以用于并发情景
  - A：atomic，原子的
- 为什么所有的基础类型都不是原子的，为什么标准库类型不默认使用Arc<T>？
  - 需要性能作为代价
- Arc<T> 和 Rc<T> 的 API 是相同的
- （例子）

RefCell<T>/Rc<T> vs Mutex<T>/Arc<T>

- Mutex<T> 提供了内部可变性，和 Cell 家族一样
- 我们使用 RefCell<T> 来改变 Rc<T> 里面的内容
- 我们使用 Mutex<T> 来改变 Arc<T> 里面的内容
- 注意：Mutex<T> 有死锁风险

再见