独立

同时

无畏并发

定义

使用线程同时运行代码

用户级线程以及系统级线程

用户级线程通常也称为轻量级线程、协程或绿色线程 户程序自己实现的线程模型 操作系统无关 操作系统级线程通常也称为重量级线程或内核线程 由操作系统内核来实现的线程模型 操作系统的支持

1:1线程模型 N:M线程模型 1:1线 模型中，每个用户级线程都直接映射到一个操作系统级线程上 而在N:M线程模型中，多个用户级线 被映射到少量的操作系统级线程上

1:1线程模型中 创建、销毁、同步和通信 操作系统 可靠和稳定 开销和限制 创建、销毁、同步和通信等操作都由用户空间的线程库来管理 加高效 线程调度和管理方式

用户级线程是由用户程序自己实现的线程模型，它们与操作系统无关 操作系统级线程 由操作系统内核来实现的线程模型，需要操作系统的支持

实现线程的方式

运行时

Move闭包和线程

消息

共享状态的并发

通过Send和SyncTrait来扩展并发

Send 和 Sync trait

- Rust 语言的并发特性较少，目前讲的并发特新都来自标准库（而不是语言本身）
- 无需局限于标准库的并发，可以自己实现并发
- 但在 Rust 语言中有两个并发概念：
  - std::marker::Sync 和 std::marker::Send 这两个 trait

Send：允许线程间转移所有权

- 实现 Send trait 的类型可在线程间转移所有权
- Rust 中几乎所有的类型都实现了 Send
  - 但 Rc<T> 没有实现 Send，它只用于单线程情景
- 任何完全由 Send 类型组成的类型也被标记为 Send
- 除了原始指针之外，几乎所有的基础类型都是 Send

Sync：允许从多线程访问

- 实现 Sync 的类型可以安全的被多个线程引用
- 也就是说：如果 T 是 Sync，那么 &T 就是 Send
  - 引用可以被安全的送往另一个线程
- 基础类型都是 Sync
- 完全由 Sync 类型组成的类型也是 Sync
  - 但，Rc<T> 不是 Sync 的
  - RefCell<T> 和 Cell<T> 家族也不是 Sync 的

手动来实现 Send 和 Sync 是不安全的

- 记住上面这句话即可