Rust 编程语言入门

杨旭，微软 MVP
Rust、Go 开发者

16 无畏并发

并发

- Concurrent：程序的不同部分之间独立的执行
- Parallel：程序的不同部分同时运行

- Rust 无畏并发：允许你编写没有细微 Bug 的代码，并在不引入新 Bug 的情况下易于重构
- 注意：本课程中的“并发”泛指 concurrent 和 parallel。

16.1 使用线程同时运行代码

进程与线程

- 在大部分 OS 里，代码运行在**进程（process）**中，OS 同时管理多个进程。
- 在你的程序里，各独立部分可以同时运行，运行这些独立部分的就是**线程（thread）**
- 多线程运行：
  - 提升性能表现
  - 增加复杂性：无法保障各线程的执行顺序

多线程可导致的问题

- 竞争状态，线程以不一致的顺序访问数据或资源
- 死锁，两个线程彼此等待对方使用完所持有的资源，线程无法继续
- 只在某些情况下发生的 Bug，很难可靠地复制现象和修复

实现线程的方式

- 通过调用 OS 的 API 来创建线程：1:1 模型
  - 需要较小的运行时
- 语言自己实现的线程（绿色线程）：M:N 模型
  - 需要更大的运行时

- Rust：需要权衡运行时的支持
- Rust 标准库仅提供 1:1 模型的线程

通过 spawn 创建新线程

- 通过 thread::spawn 函数可以创建新线程：
  - 参数：一个闭包（在新线程里运行的代码）
- （例子）
- thread::sleep 会导致当前线程暂停执行

通过 join Handle 来等待所有线程的完成

- thread::spawn 函数的返回值类型是 JoinHandle
- JoinHandle 持有值的所有权
  - 调用其 join 方法，可以等待它线程的完成
- join 方法：调用 handle 的 join 方法会阻止当前运行线程的执行，直到 handle 所表示的这些线程终结。
- （例子）

使用 move 闭包

- move 闭包通常和 thread::spawn 函数一起使用，它允许你使用其它线程的数据
- 创建线程时，把值的所有权从一个线程转移到另一个线程
- （例子）

再见