Lecture 2 Lab1前置知识

1. 为什么使用GO?

1. GO的RPC（Remote Procedure Call）包完善且简单
2. GO类型安全且内存安全，有垃圾收集机制
3. 比C++精简，调试简单

2. 多线程

1. goroutine：协程/轻量级**线程**
2. 线程的原理详见操作系统
    - 线程仅拥有一些执行时必不可少的资源（stack，PC，register）
    - 多线程共享进程地址空间。并且不同线程的栈空间是不隔离的（可以相互访问）。
3. 使用线程的原因：
    1. IO concurrency IO异步
    2. Parallelsim 并行化
    3. Convenience （相比于单线程程序降低了编码难度）
4. 在MapReduce的实现中，我们需要为每一个RPC创建一个线程（在等待RPC返回时可以让进程进行别的工作）。
5. GoRoutine采用一种多对多的方式。单个内核级线程上可以有多个GoRoutine（用户级线程）。因此在进行线程调度时，首先需要操作系统选择合适的内核级线程，而后由Go来选择运行哪一个GoRoutine。
6. Race 竞争：两个线程同时访问一个资源。 可以通过加锁来解决。
7. Coordination Go提供的线程同步工具：
    - channel 用于线程间通信
    - condition variables 条件变量
    - waitGroup 启动和等待线程 **主要用于计数方式的同步！**

- 定义数据结构：var variable_name sync.WaitGroup
- 该数据结构的作用为进行协程同步
- 用法：

  variable.Add(n)

  variable.Wait()

  每一次使用变量variable调用add，会使得计数$+1$。协程完成的时候计数-1。当计数不为0时，调用Wait的协程阻塞，直到计数为0。

3. Go相关

Go 语言之旅

1. Map/slice 以引用形式传递，而基础数据类型/数组结构体都是以值的方式进行传递。
2. 关于结构体指针：按照C语言的写法应当写为 `(*struct_ptr).member` 或者写为 `struct_ptr->member`，而Go语言含有一个语法糖，结构体指针可以直接写为 `struct_ptr.member`（而其他数据类型仍然需要使用类似C语言的形式）
3. slice本质上是个结构体，里面存了slice的大小信息，指向slice数据首地址的指针。可以直接传递slice作为参数，在函数内对slice中内容的修改会影响函数外的slice。但是slice是结构体在函数内对slice进行append等可以影响slice大小的操作并不会影响函数外slice的大小，解决办法之一：传递slice指针，而不是slice本身。**切片的底层是数组的某一部分。**
4. 闭包 = 匿名函数：可以减少代码量

   写法：

   ```
   func (variable1, varialbe2 Type1 ...) {
       Body of the function
   } (arguments)
   ```

   函数本身也可以作为值，称为**函数值**，可以赋值给变量。因此匿名函数也可以直接赋值给某个变量，用 `Function_Variable(arguments)` 的方式进行调用。

   匿名函数的使用

5. 在方法的实现中的语法糖：对于接受者参数，既可以写成结构体本身，也可以写成结构体指针，并且在函数内使用该变量时也可以直接用 `.` 而不需要解引用指针。在传参时也可以不需

要取地址（接受者为指针，实参为结构体本身时）或者解引用（接受者形参为结构体本身，实参为结构体指针时）。

6. defer：推迟执行当前行的代码，直到函数返回前才进行执行。defer可以保证无论如何该条语句都会被执行（即使函数执行过程中发生了错误）。

7. Race监测机制：Go自带，使用动态方式检查，不分析源代码。若两段发生竞争的代码未被执行，则race检测器(race detector)不能检测到race的存在。并且race detector发现的竞争部分不一定真的会发生竞争。

   使用方式：`go run -race filename.go`

8. 线程创建很难预估上限，创建过多线程会影响性能（线程需要一定的开销）。解决方案：预先设定一个固定线程数的线程池。