GO参数传递：指针还是值

假设有以下三个方法

```
type MyStruct struct {
    Val int
}

func myfunc() MyStruct {
    return MyStruct{Val: 1}
}

func myfunc() *MyStruct {
    return &MyStruct{}
}

func myfunc(s *MyStruct) {
    s.Val = 1
}
```

- 第一个方法返回 `MyStruct{Val:1}` 的复制
- 第二个方法在函数内创建 `MyStruct{Val:1}` 的指针，将其返回
- 第三个方法需要传入一个 `MyStruct` ，然后函数会覆盖其值
  这三种情况在各大源码中都有出现，但其作用相同，那么在这三种方式中进行选择的最佳实践是什么呢？

简要回答

1. 大多数情况下使用指针传递。google开发组的经验是:“当你不知道改选择什么进行传递时，就使用指针吧”
2. Slices, maps, channels, strings, function values, 和 interface values本身就是使用指针实现的，所以再使用指针传递是多余的
3. 对于打的结构体，或者不得不修改的结构体传指针，否则传值，因为由于传递指针导致结构体被修改的问题很难被排查

详细分析

大多数情况下都要使用指针传递

- 因为函数经常要修改传递过来的参数，所以在go语言指导中，推荐使用指针传递，但也有极少部分需要使用值传递，如：
  - copyfighter 使用值来传递

还有一些情况不需要使用指针

如果安装了桌面端（非ChromeApp），点击开启快速跳转。 没有安装，不再显示

- google代码审查小组提倡对不需要函数进行修改的、小的结构体中使用值传递，例如 `type Point struct { latitude, longitude float64 }`
  - 值传递可以避免由于别名而带来的值的修改错误
  - 有时使用小的结构体的值进行参数传递可以避免缓存未命中或者重新分配堆空间
- 对于slice，是没有必要使用指针传递的，参考Go数据结构探究

大结构体和小结构体如何区分
没有相关知道，需要自己判断

参考

Pointers vs. values in parameters and return values

如果安装了桌面端（非ChromeApp），点击开启快速跳转。 没有安装，不再显示