Go 规范指南

说明

本小节是拓展内容，笔者会不定期更新 Go 规范指南，使该指南的内容尽可能全，并保证规范的实用性。

> 说明：本指南参考了网络上各种 REST 最佳实践，结合笔者的实际经验汇总而来。

Go 规范指南

1. 写完代码都必须格式化，保证代码优雅：gofmt goimports
2. 编译前先执行代码静态分析：go vet pathxxx/
3. package 名字：包名与目录保持一致，尽量有意义，简短，不和标准库冲突， 全小写，不要有下划线
4. 竞态检测：go build -race (测试环境编译时加上 -race 选项，生产环境必须去掉，因为 race 限制最多 goroutine 数量为 8192 个)
5. 每行长度约定：一行不要太长，超过请使用换行展示，尽量保持格式优雅；单个文件也不要太大，最好不要超过 500 行
6. 多返回值最多返回三个，超过三个请使用 struct
7. 变量名采用驼峰法，不要有下划线，不要全部大写
8. 在逻辑处理中禁用 panic，除非你知道你在做什么
9. 错误处理的原则就是不能丢弃任何有返回 err 的调用，不要采用_丢弃，必须全部处理。接收到错误，要么返回 err，要么实在不行就 panic，或者使用 log 记录下来。
   不要这样写:

```
if err != nil {

    // error handling

} else {

    // normal code

}
```

而应该是:

```
if err != nil {

    // error handling

    return // or continue, etc.
}

// normal code
```

10. 常用的首字母缩写名词，使用全小写或者全大写，如 UIN URL HTTP ID IP OK
11. Receiver:：用一两个字符，能够表示出类型，不要使用 me self this
12. 参数传递:

    - 对于少量数据，不要传递指针
    - 对于大量数据的 struct 可以考虑使用指针
    - 传入参数是 map，slice，chan，interface，string 不要传递指针

13. 每个基础库都必须有实际可运行的例子, 基础库的接口都要有单元测试用例

14. 不要在 for 循环里面使用 defer，defer只有在函数退出时才会执行
15. panic 捕获只能到goroutine最顶层，每个自己启动的goroutine，必须在入口处就捕获panic，并打印出详细的堆栈信息
16. Go 的内置类型slice、map、chan都是引用，初次使用前，都必须先用 make 分配好对象，不然会有空指针异常
17. 使用 map 时需要注意：map 初次使用，必须用 make 初始化；map 是引用，不用担心赋值内存拷贝；并发操作时，需要加锁；range 遍历时顺序不确定，不可依赖；不能使用 slice、map 和 func 作为 key
18. import 在多行的情况下，goimports 会自动帮你格式化，但是我们这里还是规范一下 import 的一些规范，如果你在一个文件里面引入了一个 package，还是建议采用如下格式：

```
import (
    "fmt"
)
```

如果你的包引入了三种类型的包，标准库包，程序内部包，第三方包，建议采用如下方式进行组织你的包：

```
import (
    "encoding/json"
    "strings"

    "myproject/models"
    "myproject/controller"
    "myproject/utils"

    "github.com/astaxie/beego"
    "github.com/go-sql-driver/mysql"
)
```

有顺序的引入包，不同的类型采用空格分离，第一种实标准库，第二是项目包，第三是第三方包。

19. 如果你的函数很短小，少于 10 行代码，那么可以使用，不然请直接使用类型，因为如果使用命名变量很容易引起隐藏的 bug。
    当然如果是有多个相同类型的参数返回，那么命名参数可能更清晰：

```
func (f *Foo) Location() (float64, float64,
error)
```

20. 长句子打印或者调用，使用参数进行格式化分行
    我们在调用 fmt.Sprint 或者 log.Sprint 之类的函数时，有时候会遇到很长的句子，我们需要在参数调用处进行多行分割：

下面是错误的方式：

```
log.Printf(“A long format string: %s %d %d %s”,
myStringParameter, len(a),
    expected.Size,
defrobnicate(“Anotherlongstringparameter”,
        expected.Growth.Nanoseconds() /1e6))
```

应该是如下的方式：

```
log.Printf(
    “A long format string: %s %d %d %s”,
    myStringParameter,
    len(a),
    expected.Size,
    defrobnicate(
        “Anotherlongstringparameter”,
        expected.Growth.Nanoseconds()/1e6,
    ),
)
```

21. 注意闭包的调用
    在循环中调用函数或者 goroutine 方法，一定要采用显示的变量调用，不要在闭包函数里调用循环的参数

```
fori:=0;i<limit;i++{
    go func(){ DoSomething(i) }() //错误的做法
    go func(i int){ DoSomething(i) }(i)//正确的做法
}
```

22. recieved 是值类型还是指针类型
    到底是采用值类型还是指针类型主要参考如下原则：

```
func(w Win) Tally(playerPlayer)int     //w不会有任何改变
func(w *Win) Tally(playerPlayer)int     //w会改变数据
```

23. struct 声明和初始化格式采用多行：
    定义如下：

```
type User struct{
    Username  string
    Email     string
}
```

初始化如下：

```
u := User{
    Username: "astaxie",
    Email:    "astaxie@gmail.com",
}
```

24. 变量命名

- 和结构体类似，变量名称一般遵循驼峰法，首字母根据访问控制原则大写或者小写，但遇到特有名词时，需要遵循以下规则：
    - 如果变量为私有，且特有名词为首个单词，则使用小写，如 apiClient
    - 其它情况都应当使用该名词原有的写法，如 APIClient、repoID、UserID
    - 错误示例：UrlArray，应该写成 urlArray 或者 URLArray
- 若变量类型为 bool 类型，则名称应以 Has、Is、Can 或 Allow 开头

```
var isExist bool
var hasConflict bool
var canManage bool
var allowGitHook bool
```

25. 常量命名
    常量均需使用全部大写字母组成，并使用下划线分词

```
const APP_VER = "1.0"
```