Go编程规范(谢孟军版)

序言

看过很多方面的编码规范，可能每一家公司都有不同的规范，这份编码规范是写给我自己的，同时希望我们公司内部同事也能遵循这个规范来写Go代码。
如果你的代码没有办法找到下面的规范，那么就遵循标准库的规范，多阅读标准库的源码，标准库的代码可以说是我们写代码参考的标杆。

格式化规范

go默认已经有了gofmt工具，但是我们强烈建议使用goimport工具，这个在gofmt的基础上增加了自动删除和引入包.

```
go get golang.org/x/tools/cmd/goimports
```

不同的编辑器有不同的配置，sublime的配置教程：

http://michaelwhatcott.com/gosublime-goimports/

LiteIDE默认已经支持了goimports，如果你的不支持请点击属性配置->golangfmt->勾选goimports
保存之前自动fmt你的代码。

行长约定

一行最长不超过80个字符，超过的请使用换行展示，尽量保持格式优雅。

go vet

vet工具可以帮我们静态分析我们的源码存在的各种问题，例如多余的代码，提前return的逻辑，struct的tag是否符合标准等。

```
go get golang.org/x/tools/cmd/vet
```

使用如下：

```
go vet .
```

package名字

保持package的名字和目录保持一致，尽量采取有意义的包名，简短，有意义，尽量和标准库不要冲突。

import 规范

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。

在项目中不要使用相对路径引入包：

```
// 这是不好的导入
import "../net"

// 这是正确的做法
import "github.com/repo/proj/src/net"
```

变量申明

变量名采用驼峰标准，不要使用_来命名变量名，多个变量申明放在一起

```
var (
    Found bool
    count int
)
```

在函数外部申明必须使用var，不要采用:=，容易踩到变量的作用域的问题。

自定义类型的string循环问题

如果自定义的类型定义了String方法，那么在打印的时候会产生隐藏的一些bug

```
type MyInt int
func (m MyInt) String() string {
    return fmt.Sprint(m)    //BUG:死循环
}

func(m MyInt) String() string {
```

```
    return fmt.Sprint(int(m))    //这是安全的,因为我们内部进行了类型转换
}
```

避免返回命名的参数

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

```
func Foo(a int, b int) (string, ok){

}
```

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)
```

下面的代码就更清晰了：

```
// Location returns f's latitude and longitude.
// Negative values mean south and west, respectively.
func (f *Foo) Location() (lat, long float64, err error)
```

错误处理

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

error 信息

error的信息不要采用大写字母，尽量保持你的错误简短，但是要足够表达你的错误的意思。

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,
```

```
    ),
)
```

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，不要再闭包函数里面调用循环的参数

```
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
}
```

http://golang.org/doc/articles/race_detector.html#Race_on_loop_counter

在逻辑处理中禁用panic

在main包中只有当实在不可运行的情况采用panic，例如文件无法打开，数据库无法连接导致程序无法 正常运行，但是对于其他的package对外的接口不能有panic，只能在包内采用。

强烈建议在main包中使用log.Fatal来记录错误，这样就可以由log来结束程序。

注释规范

注释可以帮我们很好的完成文档的工作，写得好的注释可以方便我们以后的维护。详细的如何写注释可以 参考：

http://golang.org/doc/effective_go.html#commentary

bug注释

针对代码中出现的bug，可以采用如下教程使用特殊的注释，在godocs可以做到注释高亮：

```
// BUG(astaxie):This divides by zero.
var i float = 1/0
```

http://blog.golang.org/2011/03/godocdocumentinggocode.html

struct规范

struct申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",
}
```

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
```

更多的请参考：https://code.google.com/p/go-wiki/wiki/CodeReviewComments#Receiver_Type

带mutex的struct必须是指针receivers

如果你定义的struct中带有mutex,那么你的receivers必须是指针

参考资料

1. https://code.google.com/p/go-wiki/wiki/CodeReviewComments
2. http://golang.org/doc/effective_go.html