Go Code Review Comments(Go官方编程规范翻译)

2018-03-30 | Computer science , Golang

gofmt

所有代码在发布前均使用 `gofmt` 进行修正。

Comment Sentences（注释应当是一个完整的句子）

所有的注释都应该是一个完整的句子。句子应该以主语开头，句号结尾。

这样做，能使注释在转化成godoc时有一个不错的格式。

Declaring Empty Slices（声明空数组分片）

当你需要时，声明空的数组分片。

这是一个推荐的做法：

```
var t []string
```

这是不好的：

```
t := []string{}
```

原因是，前者能避免分配内存空间。有些时候，可能你从没向这个数组分片里面append元素

Doc Comments（文档注释）

Go提供两种注释风格，C的块注释风格/**/，C++的行注释风格//

- 每一个包都应该有包注释，位于文件的顶部，在包名出现之前。
  如果一个包有多个文件，包注释只需要出现在一个文件的顶部即可。
  包注释建议使用C注释风格，如果这个包特别简单，需要的注释很少，也可以选择使用C++注释风格。

- 每个public函数都应该有注释，注释句子应该以该函数名开头，如：

```
// Compile parses a regular expression and returns, if successful,
// a Regexp that can be used to match against text.
```

```
3  func Compile(str string) (*Regexp, error) {
```

这样做的好处是，但你要查找某个public函数的注释时， `grep` 函数名即可

Don't Panic（不要抛出panic）

尽量不要使用panic处理错误。函数应该设计成多返回值，其中包括返回相应的error类型

Error Strings（error提示）

错误提示不需要大写字母开头的单词，即使是句子的首字母也不需要。除非那是个专有名词或者缩写。
同时，错误提示也不需要以句号结尾，因为通常在打印完错误提示后还需要跟随别的提示信息。

Handle Errors（处理错误）

不要将error赋值给匿名变量 _ （因为你不可以使用匿名变量，当把error赋值给匿名变量后，相当于抛弃了这个error）。
如果一个函数返回error，一定要检查它是否为空，判断函数调用是否成功。如果不为空，说明发生了错误，一定要处理它。
直接将它返回给上级调用，进入一段错误处理逻辑，甚至panic（不建议这么做），都是可以的。

Imports

当import多个包时，应该对包进行分组。同一组的包之间不需要有空行，不同组之间的包需要一个空行。标准库的包应该放在第一组。

`goimports` 这个工具能直接帮你修正import包的规范。

以下是一个不错的import示例：

```
1  package main
2
3  import (
4      "fmt"
5      "hash/adler32"
6      "os"
7
8      "appengine/foo"
9      "appengine/user"
10
11     "code.google.com/p/x/y"
12     "github.com/foo/bar"
13 )
```

Import Dot（Golang特性：import . 包名）

在测试中，我们很可能会使用这个特性，该特性能让我们避免循环引用问题，思考一下下面的例子：

```
package foo_test

import (
    "bar/testutil" // also imports "foo"
    . "foo"
)
```

以上例子，该测试文件不能定义在于 `foo` 包里面，因为它import了 `bar/testutil` ，而 `bar/testutil` import了 `foo` ，这将构成循环引用。

所以我们需要将该测试文件定义在 `foo_test` 包中。使用了 `import . "foo"` 后，该测试文件内代码能直接调用 `foo` 里面的函数而不需要显式地写上包名。

但import dot这个特性，建议只在这种场景下使用，因为它会大大增加代码的理解难度。

Indent Error Flow（尽可能减少正常逻辑代码的缩进）

但函数调用返回错误时，我们需要判断错误是否为空，若不为空要进入错误处理的代码分支，结束后再进入正常逻辑代码。

应当尽可能减少正常逻辑代码的缩进，这有利于提高代码的可读性，便于快速分辨出哪些还是正常逻辑代码，例如：

这是一个不好的代码风格，正常逻辑代码被缩进在else分支里面：

```
if err != nil {
    // error handling
} else {
    // normal code
}
```

这是一个不错的代码风格，没有增加正常逻辑代码的缩进：

```
if err != nil {
    // error handling
    return // or continue, etc.
}
// normal code
```

另一种常见的情况，如果我们需要用函数的返回值来初始化某个变量，应该把这个函数调用单独写在一行，例如：

这是一个不好的代码风格，函数调用，初始化变量x，判断错误是否为空都在同一行，并增加了正常逻辑代码的缩进：

```
if x, err := f(); err != nil {
    // error handling
    return
} else {
    // use x
}
```

这是一个不错的代码风格，将函数调用，初始化变量x写在同一行，并且避免了正常逻辑代码的缩进：

```
x, err := f()
if err != nil {
    // error handling
    return
}
// use x
```

Initialisms（首字母大写和缩写）

当一个单词在代码中，可以是全小写的。也可以选择首字母大写，或者缩写。值得注意的是，一旦该单词选择了首字母大写或缩写的风格，
就应当在整份代码中保持这种风格，不要首字母大写和缩写两种风格混用。

以URL为例，如果选择了缩写 URL 这种风格，则应在整份代码中保持，以下命名都是不错的： `URLPony` ， `urlPony` ；切勿使用 `UrlPony` 这样的风格。

以ID为例子，如果选择了缩写 `ID` 这种风格，以下命名是不错的： `appleID` ；切勿使用 `appleId` 。

Line Length（代码行长度）

在Golang中，没有严格限制代码行长度，但我们应该尽量避免一行内写过长的代码，以及将长代码进行断行。

每行不超过80个字符，依然是一个不错的建议。

Mixed Caps（驼峰式命名）

Go建议使用驼峰式命名，不建议使用下划线命名。

大写开头表示public，小写开头表示private。

Named Result Parameters（给函数返回值命名）

给函数返回值命名，是一条适用于任何场景的建议，我们直接来看对比例子：

这是一个不好的代码风格，我们只知道函数返回的类型，但不知道每个返回值的名字：

```
func (n *Node) Parent1() *Node
func (n *Node) Parent2() (*Node, error)
```

这是一个不错的代码风格，我们准确知道每个返回值的名字：

```
func (n *Node) Parent1() (node *Node)
func (n *Node) Parent2() (node *Node, err error)
```

这条建议几乎不需要过多的解释。尤其对于一种场景，当你需要在函数结束的defer中对返回值做一些事情，给返回值名字实在是太必要了。

Package Names（包命名）

包名应该是全小写单词，不要使用下划线；包名应该尽可能简短，长单词并不有助于可读性。

Pass Values（传递值而不是指针）

不要为了节省一点空间就传递指针而不是传递值。

除非要传递的是一个庞大的结构体或者可预知在将来会变得非常庞大的结构体，指针是一个不错的选择。

Receiver Names（接受者命名）

结构体函数中，接受者的命名不应该采用 me，this，self等通用的名字，而应该采用简短的（1或2个字符）并且能反映出结构体名的命名风格。

例如，结构体名为 `Client` ，接受者可以命名为 `c` 或者 `cl` 。

这样做的好处是，当生成了godoc后，过长或者过于具体的命名，会影响搜索体验。

当接受者的名字足够简单和短时，它几乎可能出现在每一行中（例如 `c` ，无处不在），它不必像参数命名那么具体，因为我们几乎不关心接受者的名字。

另外，当一个结构体函数中的接受者命名要保持一致。上例中，如果使用了 `c` ，则不要使用 `cl`

Receiver Type（接受者类型）

编写结构体函数时，接受者的类型到底是选择值还是指针通常难以决定。

一条万能的建议：如果你不知道要使用哪种传递时，请选择指针传递吧！

以下是一些不错的建议：

- 当接受者是map, chan, func, 不要使用指针传递，因为它们本身就是引用类型。
- 当接受者是slice，而函数内部不会对slice进行切片或者重新分配空间，不要使用指针传递。
- 当函数内部需要修改接受者，必须使用指针传递。
- 当接受者是一个结构体，并且包含了 `sync.Mutex` 或者类似的用于同步的成员。必须使用指针传递，避免成员拷贝。
- 当接受者类型是一个结构体并且很庞大，或者是一个大数组，建议使用指针传递来提高性能。
- 当接受者是结构体，数组或slice，并且其中的元素是指针，并且函数内部可能修改这些元素，那么使用指针传递是个。不错的选择，这能使得函数的语义更加明确。
- 当接受者是小型结构体，小数组，并且不需要修改里面的元素，里面的元素又是一些基础类型，使用值传递是个不错的选择。

Variable Names（变量命名）

变量命名应该尽可能短，尤其是局部变量。

对于一些特殊的变量以及全局变量，我们可能需要对它有更多的描述，使用长命名是个不错的建议。