如何编写Go代码

1. 简介

本文档会介绍如何编写一个新的包，以及如何测试代码。本文档假设读者已经根据安装指南成功地安装了Go。

在着手修改已有的包或是新建包之前，一定要先把自己的想法发到邮件列表。这样可以避免重复劳动，并在动手写代码之前让设计经过充分的讨论。

2. 社区资源

寻求实时帮助，可以在Freenode IRC服务器的#go-nuts频道里找到其他的用户或是开发人员。

Go语言的官方邮件列表是Go Nuts。

报告Bug可以使用Go问题追踪器。

对于想及时了解开发进度的读者，可以加入另一个邮件列表golang-chenkins，这样在有人往Go代码库中checkin新代码时就会收到一封简要的邮件。

3. 新建一个包

根据一般约定，导入路径为x/y的包的源代码应放在$GOROOT/src/pkg/x/y目录中。

3.1 Makefile

如果能有专门针对Go的工具能检测源代码文件，决定编译顺序就好了，但现在，我们还只能用GNU的make。所以，新建包首先要新建的文件就是Makefile。如果是在Go源代码树中，其

基本格式可参照src/pkg/container/vector/Makefile：

```
include ../../../Make.inc

TARG=container/vector
GOFILES=\
      intvector.go\
      stringvector.go\
      vector.go\

include ../../../Make.pkg
```

在Go的源代码树之外（个人包），标准的格式则是：

```
include $(GOROOT)/src/Make.inc

TARG=mypackage
GOFILES=\
my1.go\
my2.go\

include $(GOROOT)/src/Make.pkg
```

第一行和最后一行分别导入了标准定义和规则。Go源代码树中所维护的包使用相对路径（代替$(GOROOT)/src），所以即使是$(GOROOT) 中含有空格也可以正常使用。这无疑简化了程序员尝试Go的难度。

如果没有设置$GOROOT环境变量，在运行gomake时就必须使用第二种makefile。即使系统中的GNU Make的名字是gmake而不是make，Gomake也能正常的调用它。

TARG是这个包的目标安装路径，就是客户用来导入这个包的字符串。在Go的源代码树中，这个字符串必须跟Makefile中的目录保持一致，不需要$GOROOT/src/pkg/前缀。在Go的源代码树之外，则可以使用任何跟标准Go包名称不冲突的TARG。一个常见的规则是用一个独有的名称把自己的包组合在一起，例如：myname/tree、 myname/filter等。注意，即使包的源代码是放在Go源代码树外部，为了便于编译器找到你的包，运行make install之后最好也把编译后的包放到标准位置，即$GOROOT/pkg。

GOFILES是创建包所需要编译的源代码文件清单。用反斜杠符号\就能将这份清单分成多行，方便排序。

如果在Go的源代码树中新建包目录，只需要将其添加到$GOROOT/src/pkg/Makefile的清单中，就能将其包含在标准构建中。然后运行：

```
cd $GOROOT/src/pkg
./deps.bash
```

这是更新以来文件Make.deps。（每次运行all.bash或make.bash时都会自动执行此操作。）

如果是修改一个已有的包，就不需要编辑$GOROOT/src/pkg/Makefile，不过运行deps.bash还是必须的。

3.2 Go源文件

对于每个源代码文件，在Makefile中的命令首先是包的名称，该名称也是导入包的默认名称。（同一个包中所有文件必须使用同一个名称。）Go的规则是，包的名称是导入路径的最后一个元素，例如以“crypto/rot13”为导入路径的包的名称应该是rot13。现在，Go工具还要求链接到同一个二进制文件的所有包的名称都应该是唯一的，但这个限制很快就会被移除。

Go会一次性编译包中所有的源代码文件，所以源代码中可以试用其它文件中的常量、变量、类型和函数，而无需特别的安排或声明。

编写简洁易懂的Go代码超出了本文档的范围。Effective Go对此有介绍。

4. 测试

Go有一个名为gotest的轻量级测试框架。编写测试首先要创建一个文件名以_test.go结尾的文件，然后在其中加入名为TestXXX且签名是(t *testing.T)的函数。测试框架会逐个地运行此类函数；如果函数调用了失败函数，例如t.Error或t.Fail，测试就会失败。gotest命令的文档和testing包的文档中有关于测试的详细信息。

不需要在Makefile中列出*_test.go文件。

运行make test或gotest就能运行测试（两个命令是等价的）。如果只需要运行单个测试文件中的测试，例如one_test.go，执行gotest one_test.go即可。

如果关心程序的性能，可以添加一个Benchmark函数（详见gotest命令文档），然后用gotest -benchmarks=.运行该函数。

代码通过测试后，就可以提交给别人审查了。

5. 一个带测试的演示包

这个演示用的包numbers含有一个函数Double，其参数为一个整数，返回值则是将该整数乘以2。这个包由三个文件组成：

第一个，包的实现，numbers.go:

```
package numbers
```

```
func Double(i int) int {
      return i * 2
}
```

接下来是测试，numbers_test.go：

```
package numbers

import (
"testing"
)

type doubleTest struct {
in, out int
}

var doubleTests = []doubleTest{
      doubleTest{1, 2},
      doubleTest{2, 4},
      doubleTest{-5, -10},
}

func TestDouble(t *testing.T) {
for _, dt := range doubleTests {
      v := Double(dt.in)
      if v != dt.out {
            t.Errorf("Double(%d) = %d, want %d.", dt.in, v, dt.out)
      }
}
}
```

最后是Makefile：

include $(GOROOT)/src/Make.inc

```
TARG=numbers
GOFILES=\
numbers.go\

include $(GOROOT)/src/Make.pkg
```

运行gomake构建并将这个包安装到$GOROOT/pkg/（可供系统上的所有程序使用）。

执行gotest测试会重建这个包，包括numbers_test.go文件，然后运行TextDouble函数。输出“PASS”表示所有测试都成功通过。把实现中的乘数从2改成3就能看到测试失败的报告。

更多细节请参考gotest文档和testing包的文档。