# Go语言单元测试覆盖率分析（本地环境）

## 说明

已测试可行

## 环境

基于linux或者windows环境，已安装go语言环境

## 前提

单元测试文件必须以 \*\_test.go 形式存在，测试函数必须必Test\*\*\*方法命名形式存在

## 工具

Go 语言自身提供的单元测试工具 go test

## 方法

*##运用go test命令生成当前目录下，所有单元测试文件的覆盖率输出结果，可将./…换成具体的单元测试文件所在目录*

*也可不换，它可递归查找当前目录下所有的单元测试文件*

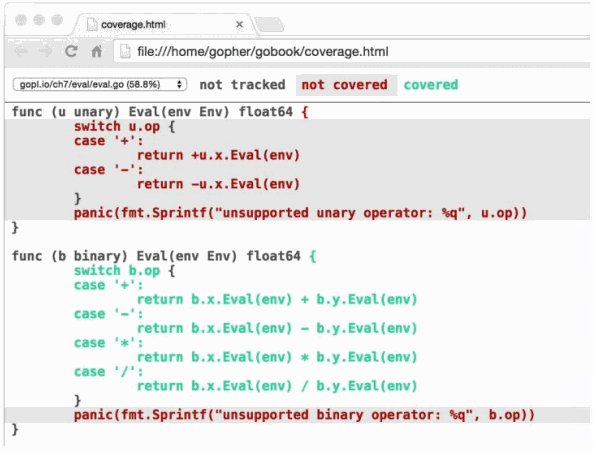
go test -v ./... -coverprofile=cover.out

*##运用go tool命令将生成的out转成html，方便覆盖结果查看*

go tool cover -html cover.out -o cover.html

## 查看分析结果

html类似如下：



# Go语言单元测试覆盖率分析（接入Sonar）

## 说明

已测试可行

## 环境

基于linux（ubuntu 18.04）环境，已安装go语言环境

## 前提

已下载最新项目代码到环境上,已安装sonarqube的server和scanner

## 工具

Go 语言自身提供的单元测试工具 go test

## 方法

*##配置Sonar-Scanner的配置文件sonar-scanner.properties，配置扫描仪扫描的源码、测试目录等*

简单配置如下：

总体要求，配置sonar-scanner扫描源码文件时，排除测试文件，扫描测试文件时，排除源码文件

，至于sonar. inclusions sonar.test.inclusions两项不用配置都行

#sonar安装的服务器地址，根据自己安装sonar服务器时的设置，自行设置ip和port

sonar.host.url=http://ip:port

#项目的独特关键字,maven 项目是 <groupId>:<artiactId>，go 项目自己定义就可以

sonar.projectKey=ap

#将在web界面上显示的名字

sonar.projectName=ap

#项目版本

sonar.projectVersion=1.0

#需要分析的源码目录的路径，因为一般是直接在项目根目录下执行命令，可配置为src/开头的目录

sonar.sources=src/main/

#分析测试代码时，包括在内在外的文件

sonar.inclusions=

#分析源码时，排除在外的文件

sonar.exclusions=src/main/test/\*\_test.go

#需要分析的测试目录的路径，同源码，可配置为src/开头的目录

sonar.tests= src/main/test

#分析测试代码时，包括在内在外的文件

sonar.test.inclusions=

#分析测试代码时，排除在外的文件

sonar.test.exclusions= src/main/

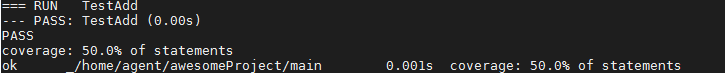
#单测覆盖率报告地址，下面的配置意思是直接在项目根目录下

sonar.go.coverage.reportPaths=cover.out

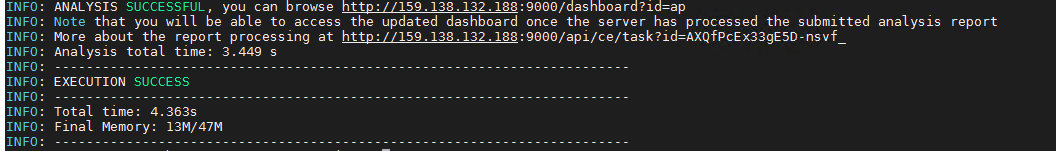
*##在项目的根目录下，执行go test命令生成覆盖率结果*

go test -v ./... -coverprofile=cover.out

*##会有类似的结果输出*



*##在项目的根目录下，执行sonar-scanner，得到输出结果结尾类似下面这样，代表运行成功*



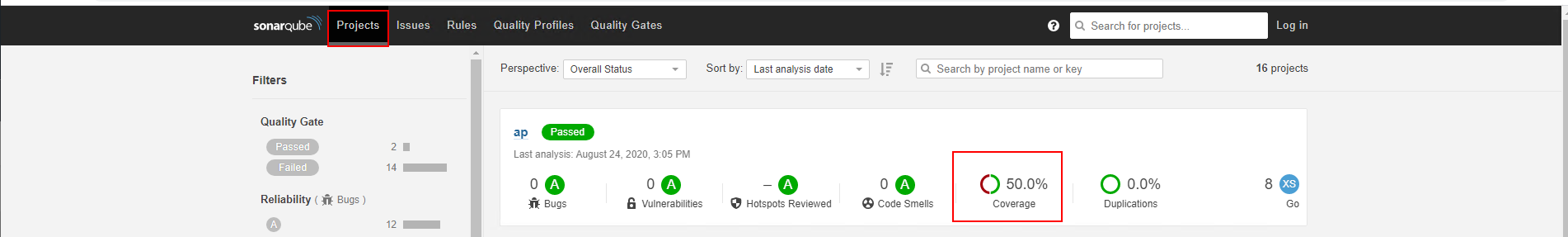
## 查看分析结果

登录sonarqube的界面，如上图中的，初始用户名和密码为admin,admin,当然sonarqube服务器端的ip和端口是安装sonar服务器端设置的，

也可以在sonar-scanner的配置文件sonar-scanner.properties中查看sonar.host.url的值

登录成功，选择project菜单，查找和我们在sonar-scanner.properties文件中设置的sonar.projectName的值同名的项目，查看覆盖率

类似如下：



# Go语言集成测试覆盖率分析（CI环境（jenkins）集成）

## 说明

未测试

## 环境

基于linux（ubuntu 18.04）环境，已安装go语言环境

## 前提

已下载最新项目代码到环境上

## 工具

go test（go语言自带）

gocov （需要安装的覆盖报告工具）

安装方法：go get github/axw/gocov/gocov（在159.138.132.188环境【sonarqube安装环境】安装未报错，但是使用命令报错，报找不到）

另外一种安装方法：

1. git clone [git@github.com:axw/gocov.git](mailto:git@github.com:axw/gocov.git)
2. unzip gocov-master.zip
3. cd gocov-master
4. go install github/axw/gocov/gocov（同go get安装未报错，但是使用命令报错，报找不到）

gocov-xml （一个简单的helper工具，用于为类似CI的Jenkins和其他gocov生成Cobertura格式的XML输出）

安装方法：go get github/AlekSi/gocov-xml（在159.138.132.188环境【sonarqube安装环境】安装未报错，但是使用命令报错，报找不到）

另外一种安装方法：

1. git clone [git@github.com:AlekSi/gocov-xml.git](mailto:git@github.com:AlekSi/gocov-xml.git)
2. unzip gocov-xml-master.zip
3. cd gocov-xml-master
4. go install github/AlekSi/gocov-xml（同go get安装未报错，但是使用命令报错，报找不到）

Cobertura Plugin（CI环境需安装此插件）

## 方法

1. 在测试目录下增加main\_test.go文件，编写函数如下：

func TestMainStart(t \*testing.T) {

var args []string

for \_, arg := range os.Args {

if !strings.HasPrefix(arg, "-test") {

args = append(args, arg)

}

}

os.Args = args

main()

}

main方法为项目入口，意思需要把main函数在此函数中复写一遍

1. 进入到项目的测试目录下

#第一步：执行集成测试，并将此函数编译成二进制文件

go test -coverpkg="./..." -c -o cover.test

#第二步：运行二进制文件，指定运行的测试方法是 TestMainStart，并将覆盖率报告输出

./cover.test -test.run "TestMainStart" -test.coverprofile=cover.out

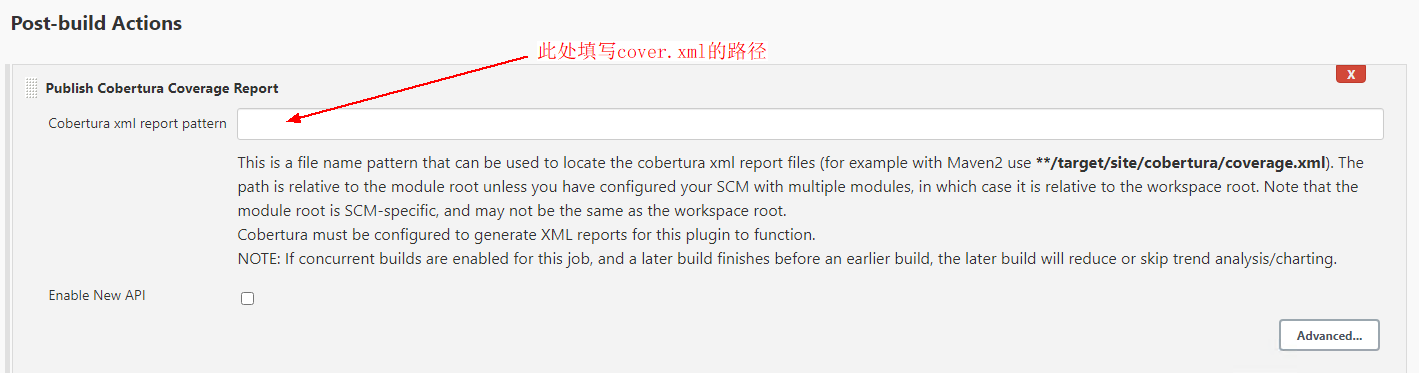
#第三步：将输出的覆盖率报告转换成 html 文件（html 文件查看效果比较好）

go tool cover -html cover.out -o cover.html

#第四步：生成 Cobertura 格式的 xml 文件

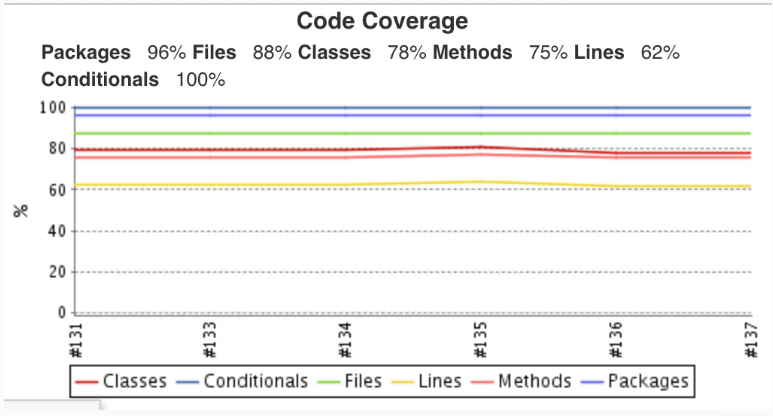
gocov convert cover.out | gocov-xml > cover.xml

1. 在CI环境配置项的Post-build Actions内添加Cobertura插件覆盖报告选项，编译项目，查看覆盖结果



## 查看分析结果

运行jenkins job查看运行结果，结果类似如下：



# Go语言集成测试增量覆盖率分析

借鉴此链接[Go语言集成测试增量覆盖分析](https://tech.youzan.com/you-zan-go-xiang-mu-dan-ce-ji-cheng-zeng-liang-fu-gai-lu-tong-ji-yu-fen-xi/)中的第四部分《集成测试增量覆盖率分析》

# 总结

Go语言无论是那种测试形式的覆盖率，都需要首先下载代码到本地，然后执行一些测试命令，生成覆盖率报告再进行分析。

就难度而言，本地环境和SonarQube集成会比较简单，并且容易实现，CI环境集成的过程就有些过于麻烦。

增量覆盖率分析对于我们来说没有必要，因为如果集成在SonarQube上的话，是会看到覆盖率细节的变化。