Jenkins配置使用说明

# 简介

jenkins 是一款流行的开源持续集成工具，广泛用于项目开发，具有自动化构建、测试和部署等功能。本文以防伪认证项目为例，以此介绍在jenkins中配置 Java 项目（SVN+Maven+Sonar+Jacoco），实现自动化编译、代码质量检查、单元测试覆盖率检测、部署项目，真正提升整个团队效率，实现devops理念。

# 前提

* jenkins、sonarqube服务已经搭建完成
* 构建机--项目可以正常构建编译的环境,包括安装sonar scanner，并且需要将该机器配置到jenkins节点上
* 发布机--项目部署的机器

# jenkins构建触发器定时任务

## 3.1 定时器语法

\* \* \* \* \*

星号中间用空格隔开

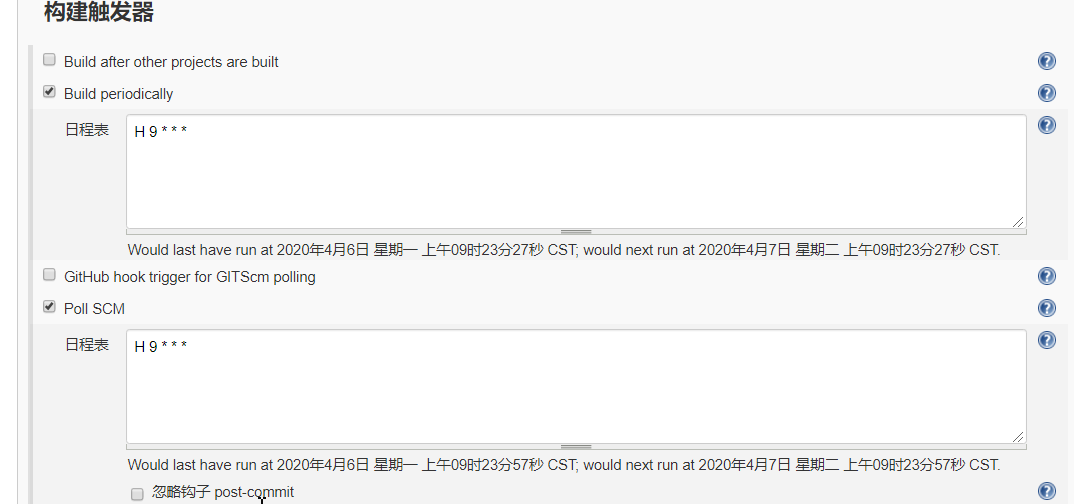
* 第一个\*表示分钟，取值0~59
* 第二个\*表示小时，取值0~23
* 第三个\*表示一个月的第几天，取值1~31
* 第四个\*表示第几月，取值1~12
* 第五个\*表示一周中的第几天，取值0~7，其中0和7代表的都是周日

用法举例：

* 每30分钟构建一次：H代表形参  
  H/30 \* \* \* \*
* 每2个小时构建一次:  
  H H/2 \* \* \*
* 每天的8点，12点，22点，一天构建3次： (多个时间点中间用逗号隔开)  
  0 8,12,22 \* \* \*
* 每天早上8点到晚上6点每三小时检查一次  
  H 8-18/3 \* \* \*

## 3.2 具体实例

这里主要解释（Build periodically，Poll SCM）



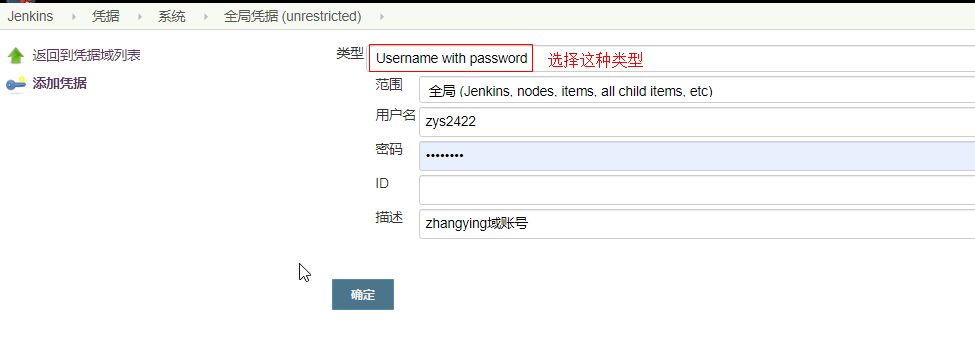
* Build periodically（定时构建）：周期进行项目构建（它不care源码是否发生变化）
* Poll SCM（轮训 SCM）：定时检查源码变更（根据SCM软件的版本号），如果有更新就checkout最新code下来，然后执行构建动作。
* GitHub hook trigger for GITScm polling： 这个是管理github上代码有变动时构建。这个用的比较少，大家了解下就行。

# Jenkins获取代码配置

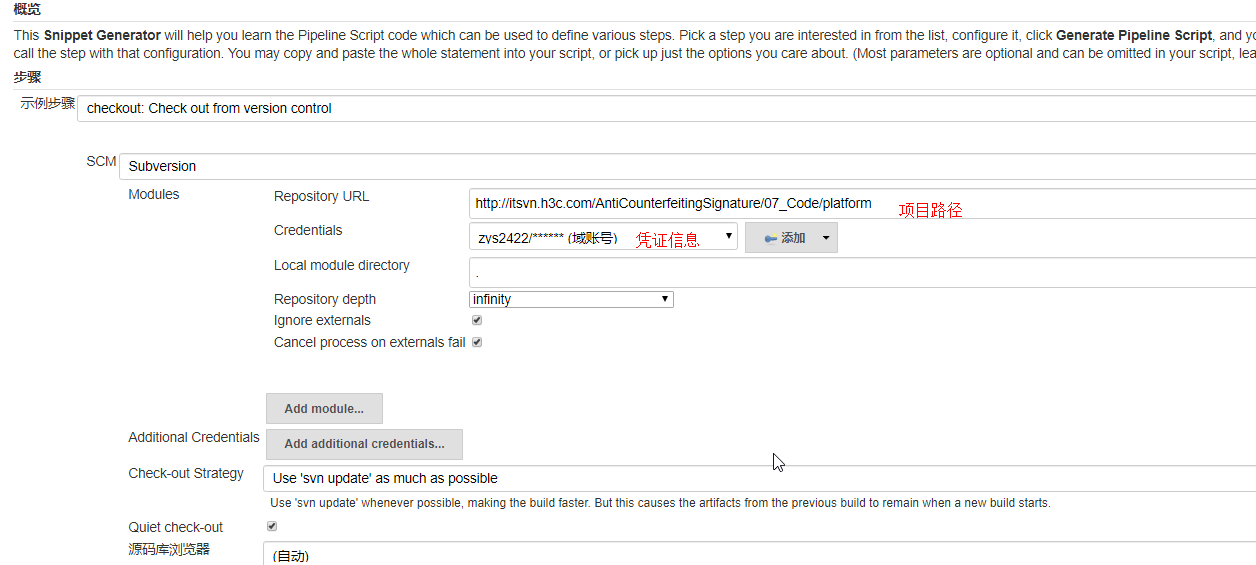
## 4.1 jenkins获取svn代码

jenkins获取svn代码配置的配置步骤如下：

1. 在构建机上安装SVN客户端，默认安装路径即可
2. 在jenkins上添加凭证，这里添加的是svn项目有权限Checkout的人员域账号信息，如下图



1. 在jenkins项目中流水线语法中配置如下图：



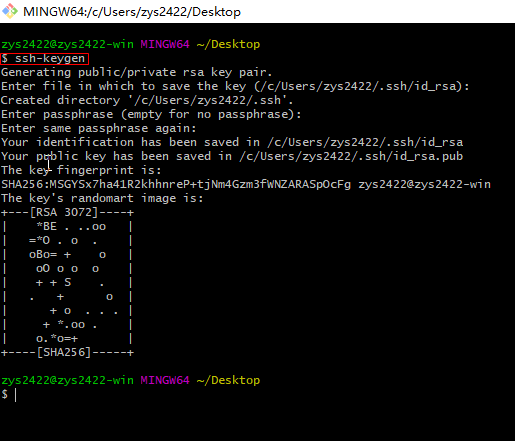
最后生成的流水线脚本是：

checkout([$class: 'SubversionSCM', additionalCredentials: [], excludedCommitMessages: '', excludedRegions: '', excludedRevprop: '', excludedUsers: '', filterChangelog: false, ignoreDirPropChanges: false, includedRegions: '', locations: [[cancelProcessOnExternalsFail: true, credentialsId: '156c2c1f-9ac6-4f9d-97a0-4cda8a3b8ad6', depthOption: 'infinity', ignoreExternalsOption: true, local: '.', remote: 'http://itsvn.h3c.com/AntiCounterfeitingSignature/07\_Code/platform']], quietOperation: true, workspaceUpdater: [$class: 'UpdateUpdater']])

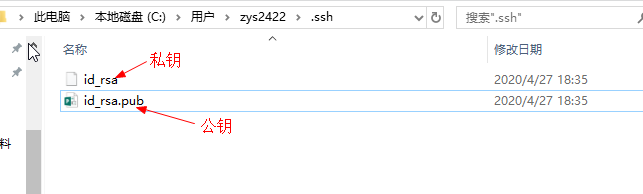
## 4.2 jenkins获取git代码

jenkins获取git代码配置的配置步骤如下：

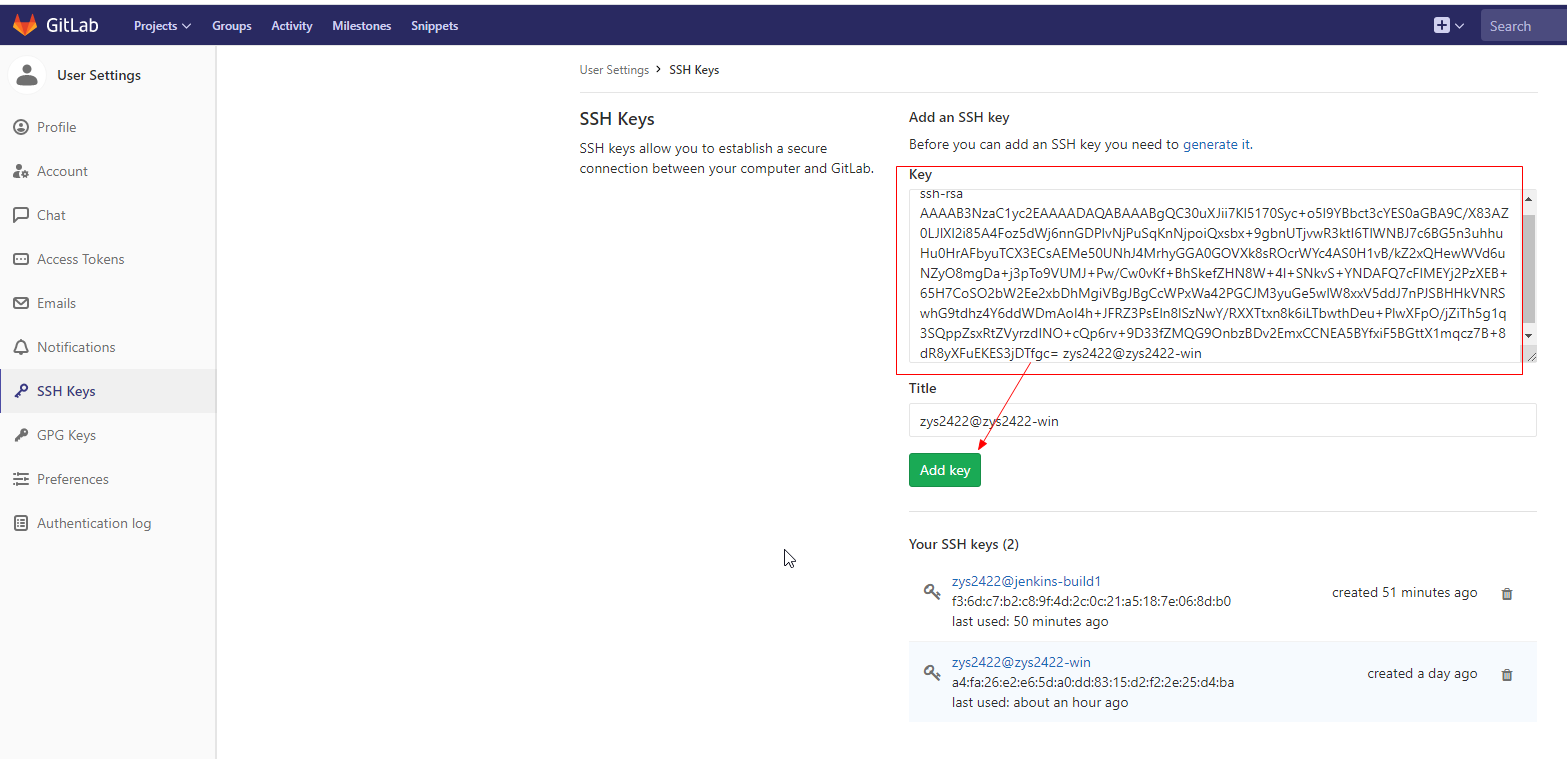
1. 执行机安装Git客户端，统一安装位置：C:\Git\
2. 在执行机上生成SSH key，执行命令ssh-keygen,后面一路回车即可



1. 查看生成的公钥、私钥 文件位置默认在C:\Users\用户名\.ssh下



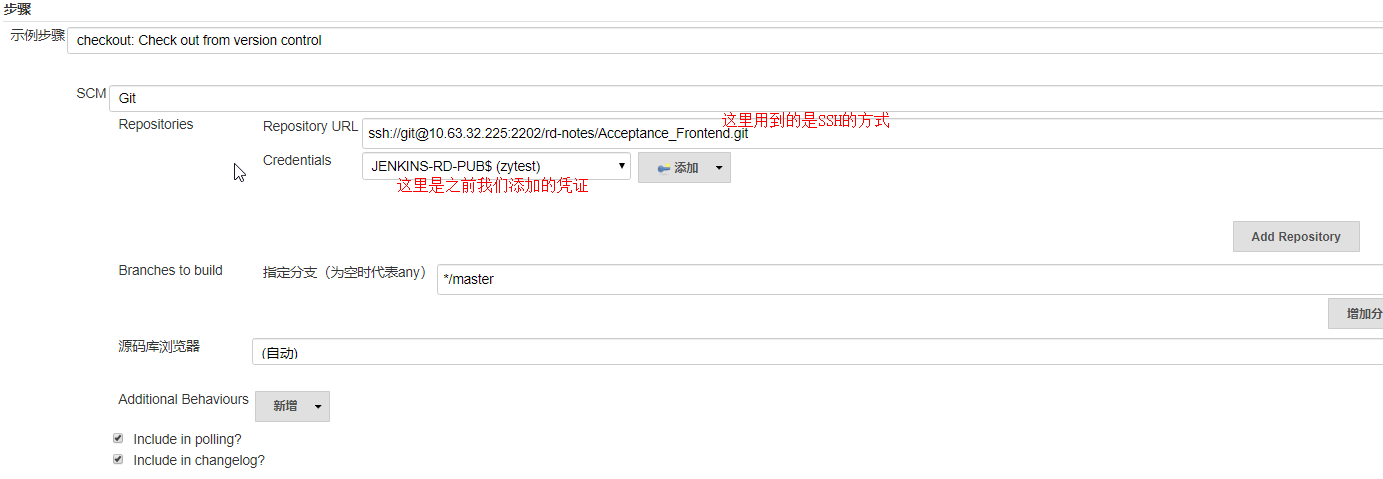
1. 在git服务器上添加SSH key ,注意，我们这里添加的是公钥



1. 在jenkins上添加凭证，注意这里添加的是私钥



1. 在jenkins的流水线语法中设置



最后生成的流水线脚本是：

checkout([$class: 'GitSCM', branches: [[name: '\*/master']], doGenerateSubmoduleConfigurations: false, extensions: [], submoduleCfg: [], userRemoteConfigs: [[credentialsId: '1d2aa288-0746-4166-95fb-a569fa6e880b', url: 'ssh://git@10.63.32.225:2202/rd-notes/Acceptance\_Frontend.git']]])

友情提示，在git中添加的公钥都是跟域账号权限下相关的项目，如果获取时没有权限，这里就需要确认下是不是配置人员没有相关项目的权限。这时候需要有权限的用户登录构建机按照上面的配置执行即可。

# 核心模块配置

## 5.1 Jenkins集成Sonar进行代码质量检测

### 5.1.1安装插件

Jenkins需要安装sonar插件：SonarQube Scanner for Jenkins

### **5.1.2 项目配置**

在项目根目录下新建sonar-project.properties（固定名称）文件，配置如下

sonar.projectKey=platform #sonar平台中相对应项目的key

sonar.projectName=platform #sonar平台中相对应项目的名字

sonar.projectVersion=1.0 #sonar平台中相对应项目的版本

sonar.sources=./src/main/java #sonar检测源文件目录，多个用’,’分开

sonar.tests=./src/test/java #sonar检测的单元测试的目录

sonar.exclusions #设置分析除指定文件以外的所有文件

sonar.inclusions #设置仅仅分析指定的文件

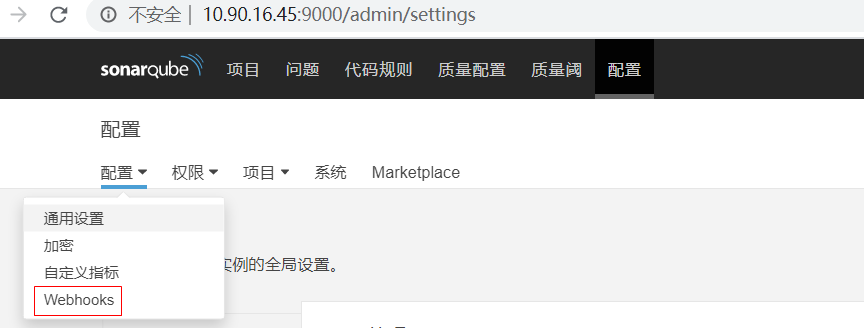
sonar.language=java #sonar检测的是Java文件

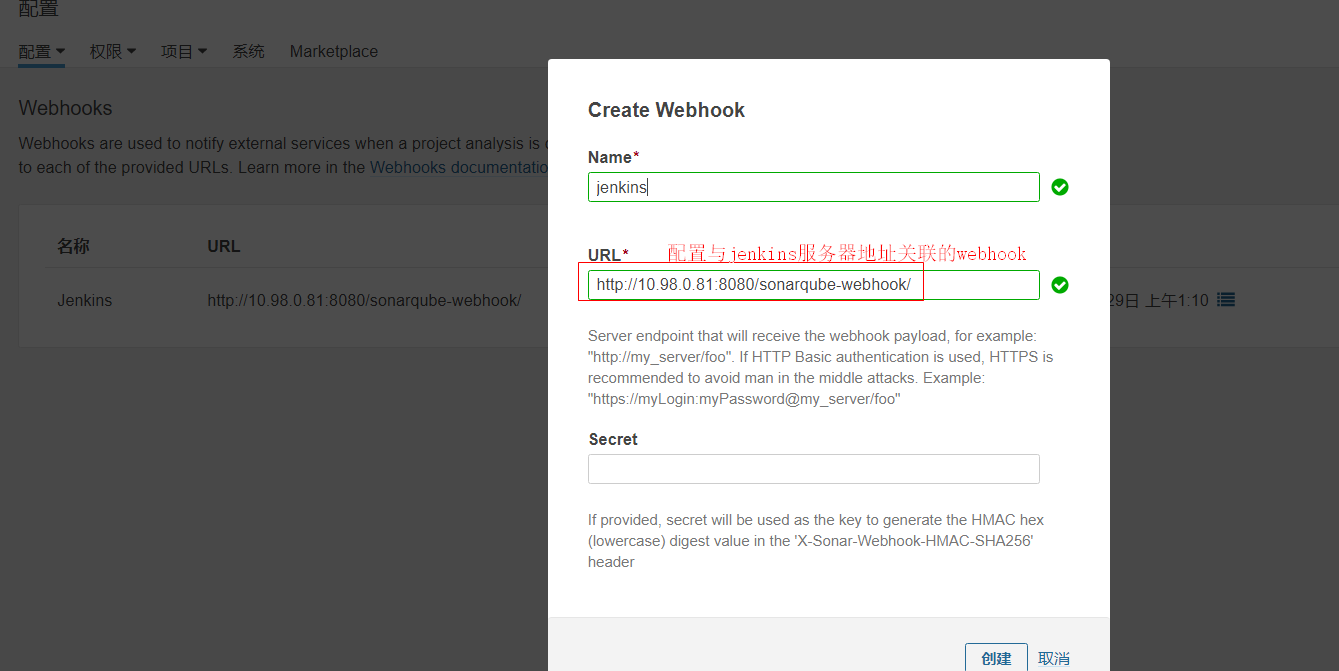
sonar.java.binaries=./target/classes #编译后的class位置

sonar.java.coveragePlugin=jacoco #sonar所用的Java的代码覆盖率插件

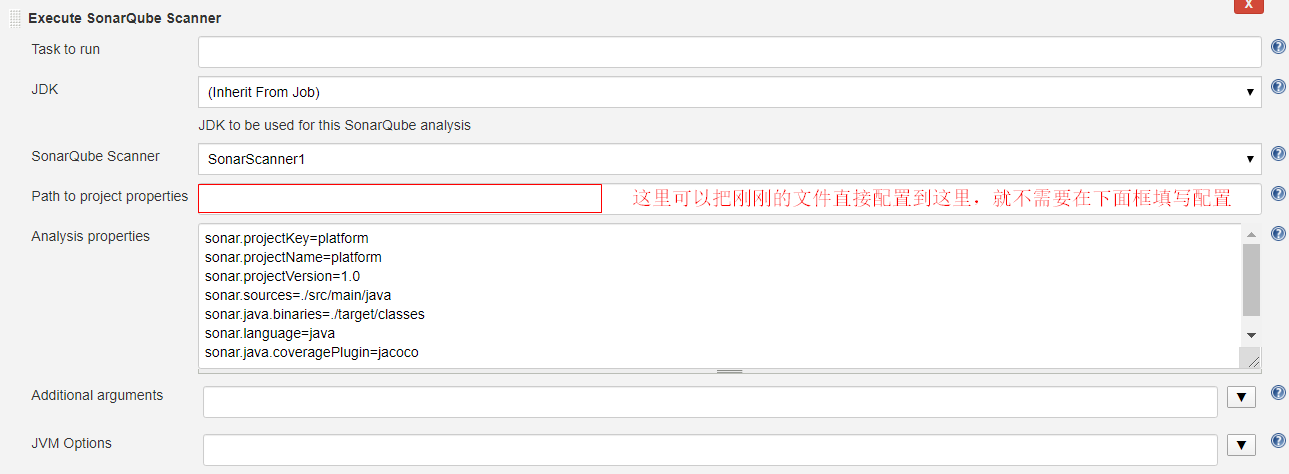
### **5.1.3 sonar配置webhooks**

Jenkins执行过程，需要实时获取持续集成-静态代码扫描的结果，以确定扫描结果是否符合既定的要求。通过在SonarQube中设定与Jenkins的WebHook，即可解决这个问题。

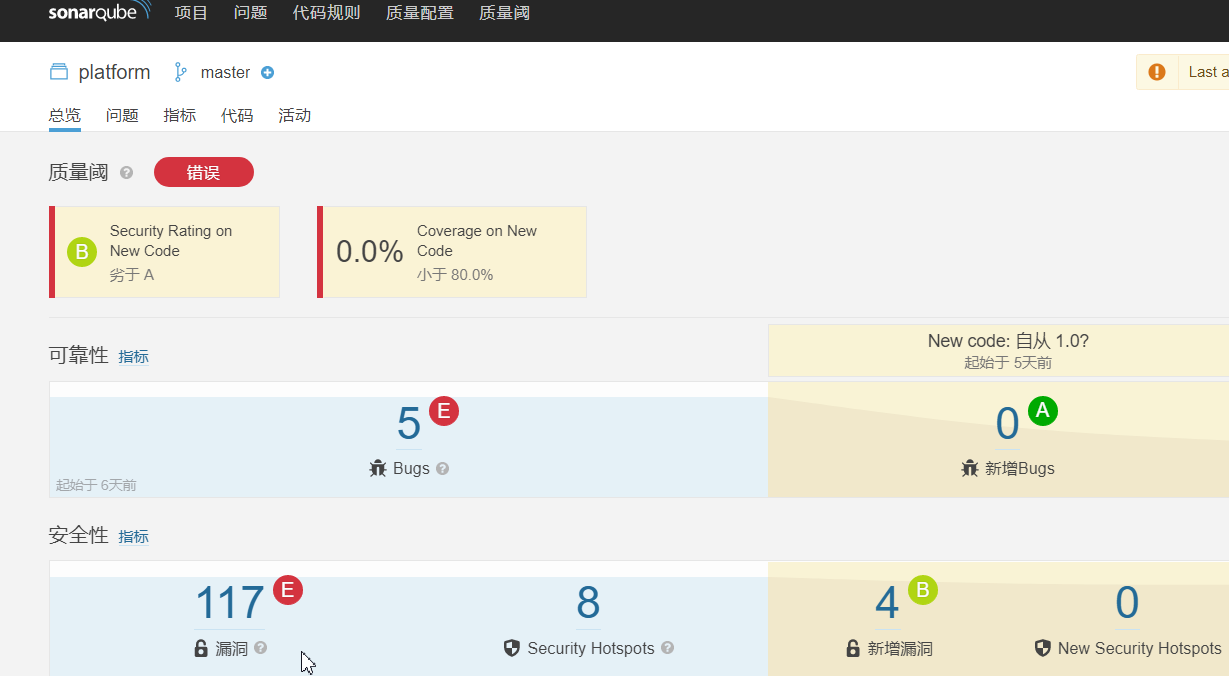


在代码扫描成功后，扫描结果需要回调Jenkins。 添加的Jenkins的Webhook结构为： http://[jenkins\_url]/sonarqube-webhook/

### **5.1.4 jenkins配置代码质量检测**



在构建完成后jenkins中，可以点击sonarqube图标，跳转到配置的sonarqube服务器上查看具体的检测详细



## **5.2 Jenkins集成Jacoco生成单元测试覆盖率报告**

### **5.2.1 安装插件**

Jenkins需要安装插件： [JaCoCo plugin](http://updates.jenkins-ci.org/latest/jacoco.hpi)

### **5.2.2 配置pom.xml**

调整pom.xml（跳过测试设置也应该关闭）--这里需要开发配置并提交代码

<plugin>

<groupId>org.jacoco</groupId>

<artifactId>jacoco-maven-plugin</artifactId>

<version>0.8.5</version>

<executions>

<execution>

<id>pre-unit-test</id>

<goals>

<goal>prepare-agent</goal>

</goals>

<configuration>

<propertyName>jacocoArgLine</propertyName>

</configuration>

</execution>

<execution>

<id>post-unit-test</id>

<phase>package</phase>

<goals>

<goal>report</goal>

</goals>

<configuration>

<!--这里可以设置单测覆盖率检测的一些文件，不设置默认检测所有-->

<includes>

<include>\*\*/com/h3c/platform/manufacturer/\*\*/\*.\*</include>

</includes>

</configuration>

</execution>

</executions>

</plugin>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<!-- maven-surefire-plugin是maven里执行测试用例的插件-->

<artifactId>maven-surefire-plugin</artifactId>

<version>2.9</version>

<configuration>

<argLine>-Xmx256M ${jacocoArgLine}</argLine>

<skip>false</skip>

<testFailureIgnore>false</testFailureIgnore>

<!--这里可以设置单测文件-->

<includes>

<include>\*\*/com/h3c/platform/manufacturer/serviceimpl/\*.\*</include>

</includes>

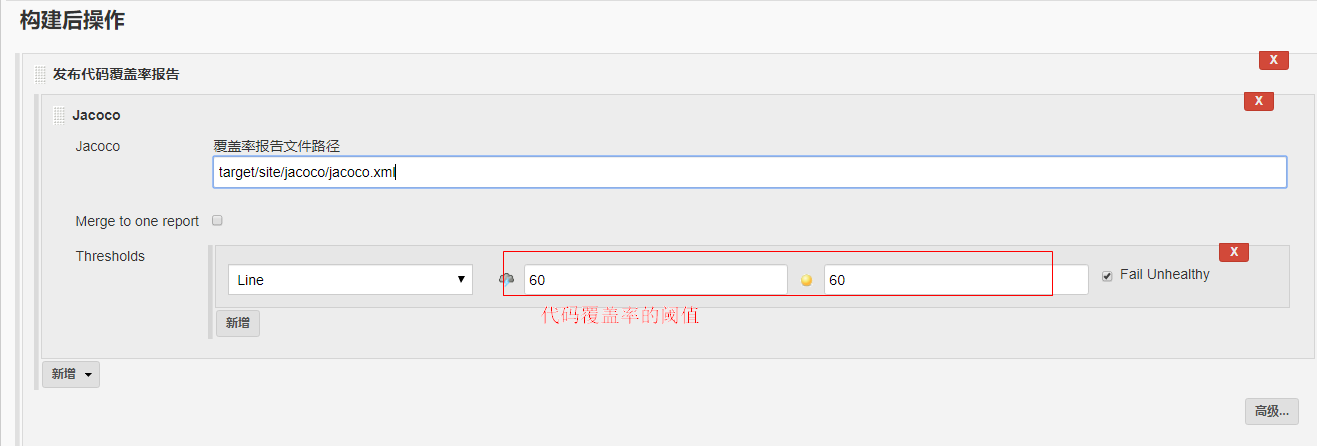
</configuration>

</plugin>

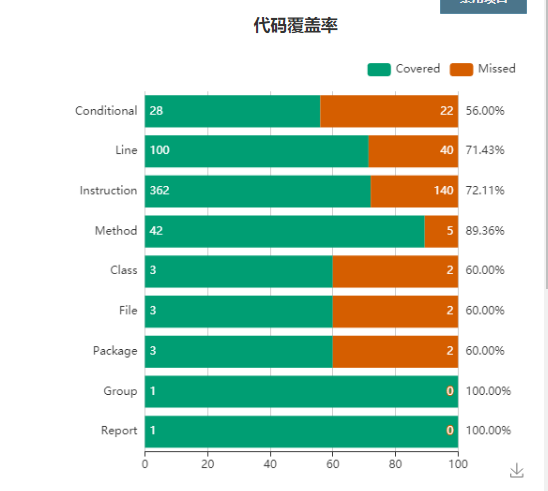
### **5.2.3 在jenkins中添加单元测试覆盖报告**

项目编译后会再targer文件夹下生成一些覆盖率相关的文件，我们可以把相关的报告展示在jenkins中。具体操作

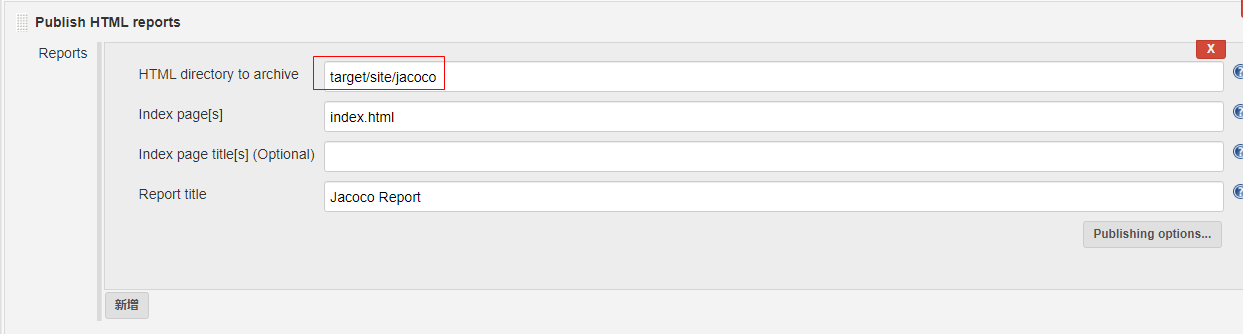
1.在项目的构建后操作中增加发布代码覆盖率报告，并设置期望的覆盖率阈值。



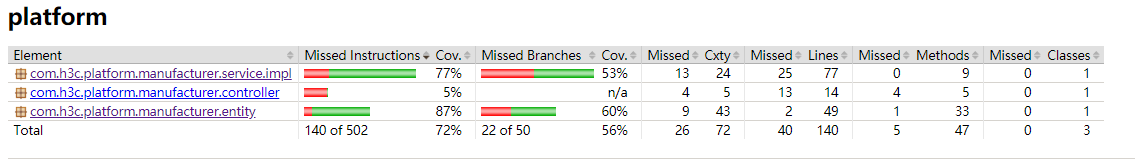
在jenkins构建后展示如下



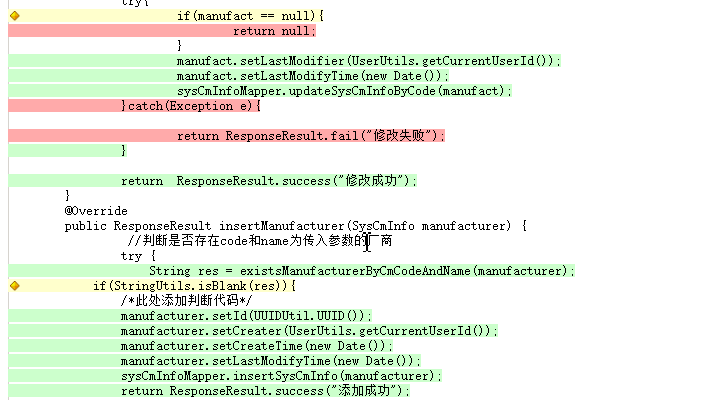
2. 由于上步展示的仅仅是结果，不能方便开发查看详细的代码覆盖率，此时我们在jenkins中添加HTML报告



在jenkins构建后点击jacoco报告



我们可以查看到具体到每一行代码覆盖的情况



## 5.3 jenkins集成jmeter进行接口自动化测试

### 5.3.1安装插件

Jenkins需要安装插件：performance plugin

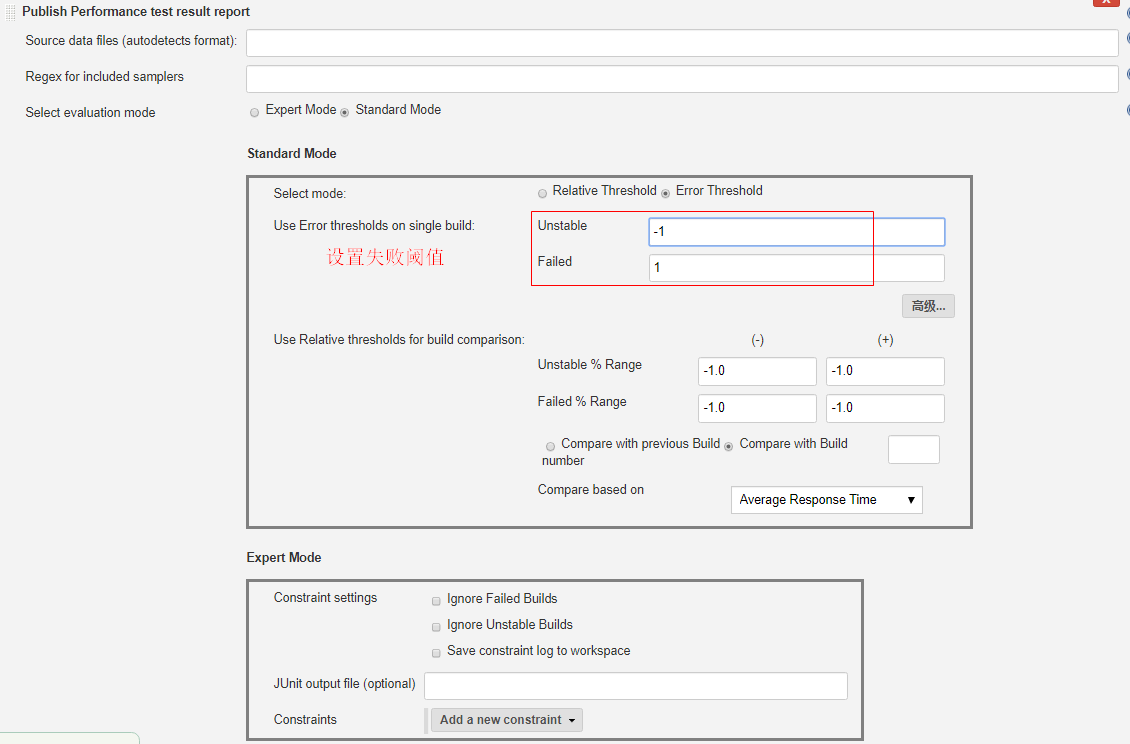
### 5.3.2项目配置

通过接口测试工具jmeter将项目已完成的接口脚本文件提交到svn上（注意，文件名请使用全英文无空格名称）。

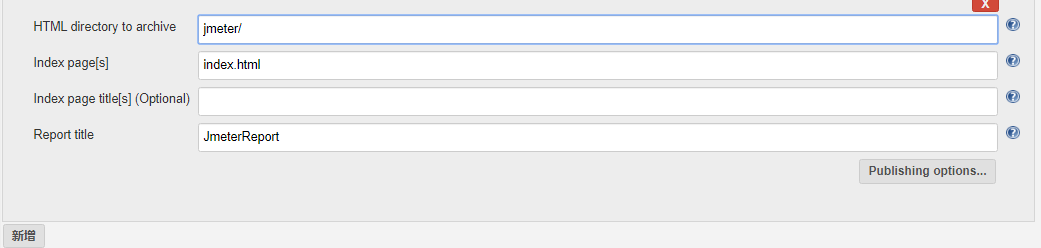
### 5.3.3 在jenkins中配置接口测试

通过jmeter命令执行接口脚本测试后，这时候可以通过jenkins的performance插件获取相关的执行结果，并且可以将测试报告以HTML的方式展示在jenkins中。具体操作

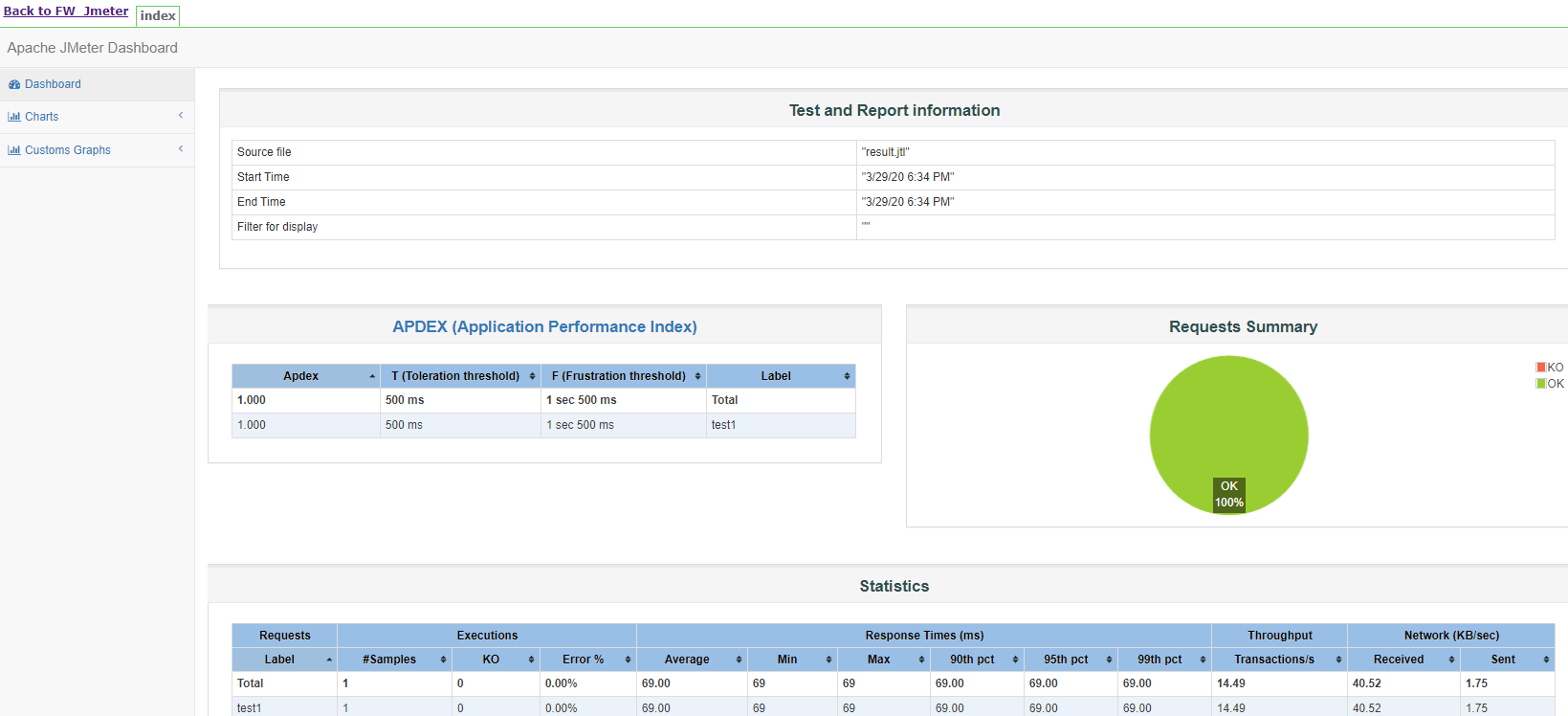
1.在项目的构建后操作中增加publish performance test result report，并设置期望的覆盖率阈值。



2. 上步展示的仅仅是结果，不能方便开发查看具体接口测试情况，此时我们在jenkins中添加HTML报告



Jenkins构建完后，可以在界面点击JmeterReport



# Pipeline自动化实现整个devops

jenkins的流水线是用户定义的一个CD流水线模型 。流水线的代码定义了整个的构建过程, 他通常包括构建, 测试和交付应用程序的阶段 ，它清晰的展示构建的每一个步骤，也便于维护自动化构建。

### 6.1 Pipeline语法

Pipeline能使用两种语法进行编写 - 声明式和脚本化，下面例子是声明式语法编写的，我们只需要记住下面这种特定的格式，具体步骤脚本大部门可以通过jenkins pipeline语法帮我们自动生成，当然，有些复杂的还需要我们去查一些相关资料去编写。

pipeline {

agent any

stages {

stage('每一阶段名称') {

steps {

echo '具体步骤!'

}

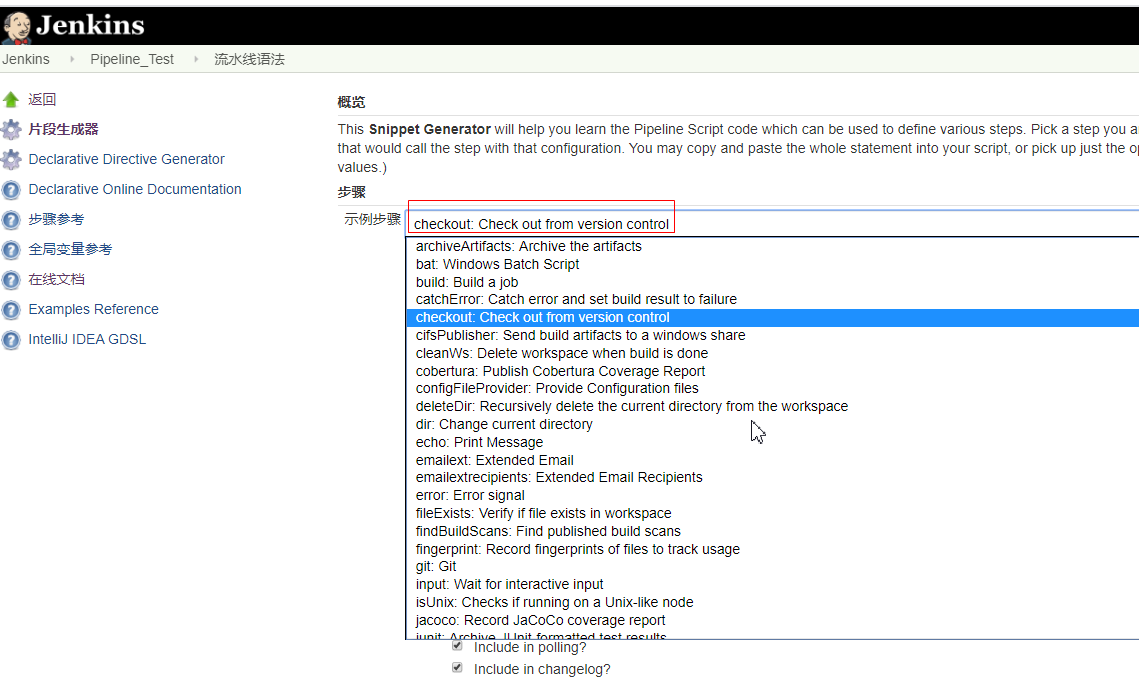
}

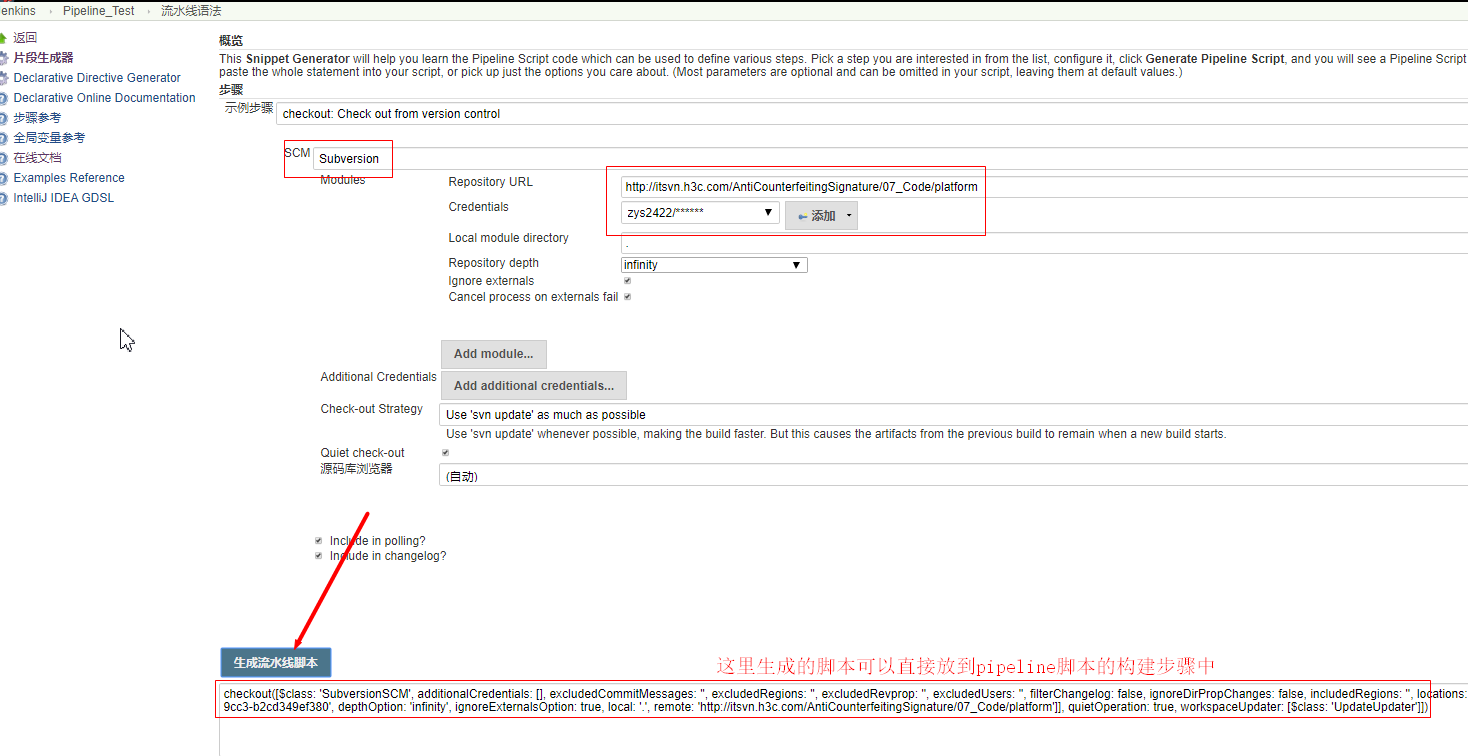
}

}

Jenkins pipeline语法

1.选择要执行的步骤





### 6.2 Pipeline完整脚本例子

下面是完整的构建步骤通过pipeline实现：代码获取—编译项目—代码质量检测—单元覆盖率检测—发布部署—发送邮件，脚本如下

pipeline {

agent {

node {

//这个是通过标签名称选择我们项目构建的机器

label 'UAT\_JAVA\_Build'

}

}

options {

//设置本次执行超时时长，超过该时长会自动停止构建

timeout(time: 60, unit: 'MINUTES')

}

stages {

stage('CheckOut') {

steps {

checkout([$class: 'SubversionSCM', additionalCredentials: [], excludedCommitMessages: '', excludedRegions: '', excludedRevprop: '', excludedUsers: '', filterChangelog: false, [[cancelProcessOnExternalsFail: true, credentialsId: 'c3bd2241-0fb6-4880-9cc3-b2cd349ef380', depthOption: 'infinity', ignoreExternalsOption: true, local: '.', remote: 'http://itsvn.h3c.com/AntiCounterfeitingSignature/07\_Code/platform']],

quietOperation: true, workspaceUpdater: [$class: 'UpdateUpdater']])

}

}

stage('Build'){

steps {

bat label: '', script: 'mvn clean package'

}

}

stage('SonarQube analysis'){

steps {

withSonarQubeEnv('sonarserver') {

bat label: '', script: 'E:/softpackage/sonar-scanner-4.2.0.1873-windows/bin/sonar-scanner'

}

script {

timeout(1) {

//这里设置超时时间1分钟，不会出现一直卡在检查状态

/\*\*利用sonar webhook功能通知pipeline代码检测结果，

未通过质量阈，pipeline将会fail \*\*/

def qg = waitForQualityGate('sonarserver')

/\*\*注意：这里waitForQualityGate()中的参数也要与之前SonarQube servers中Name的配置相同\*\*/

if (qg.status != 'OK') {

error "未通过Sonarqube的代码质量阈检查，请及时修改！failure: ${qg.status}"

}

}

}

}

}

stage('UnitTestCoverage'){

steps{

publishCoverage adapters: [jacocoAdapter

(path: 'target/site/jacoco/jacoco.xml',thresholds:

[[thresholdTarget: 'Line', unhealthyThreshold: 60.0,

unstableThreshold: 60.0],

[thresholdTarget: 'Report'], [thresholdTarget: 'Aggregated Report']])],

sourceFileResolver: sourceFiles('NEVER\_STORE')

publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false,

reportDir: 'target/site/jacoco', reportFiles: 'index.html', reportName: 'Jacoco Report', reportTitles: ''])

}

}

stage('deploy'){

steps{

sshPublisher(publishers: [sshPublisherDesc(configName: '10.98.0.127',

sshCredentials: [encryptedPassphrase: '{AQAAABAAAAAQDDuu10gAsEo8sK6iqy/3v4FM5jJNMfvH3XmQJ7CMybg=}',

key: '', keyPath: '', username: 'root'], transfers: [sshTransfer(cleanRemote: false, excludes: '',

execCommand: '''source /etc/profile

#!/bin/bash

cd /app/platform

sh ./stop.sh

sh ./start.sh''', execTimeout: 120000, flatten: false, makeEmptyDirs: false, noDefaultExcludes: false,

patternSeparator: '[, ]+', remoteDirectory: '/app/platform', remoteDirectorySDF: false, removePrefix: '',

sourceFiles: '\*\*/platform.jar')], usePromotionTimestamp: false, useWorkspaceInPromotion: false, verbose: false)])

}

}

stage('JmeterAPITest') {

steps {

bat label: '',returnStdout: true,script: "call ${jmeterhome}/jmeter.bat -n -t jemeter\\anfifake-sign.jmx -l jemeter\\result.jtl -e -o jemeter\\report"

perfReport errorFailedThreshold: 1, filterRegex: '', sourceDataFiles: 'jemeter\\result.jtl'

publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false, reportDir: 'jemeter\\report', reportFiles: 'index.html', reportName: 'JmeterReport', reportTitles: ''])

}

}

stage('PostmanAPITest') {

steps {

bat label: '',returnStdout: true,script: "newman run 脚本路径/XXXX.postman\_collection.json –e脚本路径/XXXX.postman\_environment.json -d脚本路径/XXXX.postman\_data.json –reporters html –reporter-html-export 输出报告路径\XXXX\_Report.html"

publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false, reportDir: '', reportFiles: 'index.html', reportName: 'PostmanReport', reportTitles: ''])

}

}

stage('UITest\_Katalon\_studio') {

steps {

//这里的例子是采用katalon studio软件编写的UI自动化脚本

bat label: '',returnStdout: true,script: "cd C:\soft\Katalon\_Studio\_Windows\_64-6.3.3

katalon -noSplash -runMode=console -projectPath="脚本路径\XXX.prj" -retry=0 -testSuitePath="Test Suites/TaskSuite" -executionProfile="default" -browserType="Chrome" -reportFolder="输出报告路径\" -reportFileName="报告名称" -apiKey="69c9fddb-ca60-47b7-9218-f08f512c0e2d"

}

}

stage('UITest\_Selenuim') {

steps {

//执行UI自动化测试，技术：Java+Maven+TestNG+Selenium

bat label: '',returnStdout: true,script: "cd %workspace%/automation/core

 mvn clean install

cd %workspace%/automation/autoprojectName

mvn clean install -Djenkins\_plan=%jenkins\_plan% -Djenkins\_timestamp=%BUILD\_TIMESTAMP%"

//展示UI测试报告

publishHTML([allowMissing: false, alwaysLinkToLastBuild: false, keepAll: false, reportDir: '${workspace}/automation/autoprojectName/report', reportFiles: 'Report${BUILD\_TIMESTAMP}.html', reportName: 'UITestReport', reportTitles: ''])

}

}

}

post {

//当项目在任何环节构建失败都会发送邮件通知

unsuccessful{

//这里主要是设置发送邮件内容模板，需要修改的是最后的to:收件人

configFileProvider([configFile(fileId:

'1d799a29-a5f9-4ba3-8f2d-7fd2bec9c94b', targetLocation: 'email.html', variable: 'content')]) {

script {

template = readFile encoding: 'UTF-8', file: "${content}"

emailext(

subject: "Job [${env.JOB\_NAME}] -Status:${currentBuild.result?: 'success'}",

body: """${template}""",

to: "YS.zhangying@h3c.com"

)

}

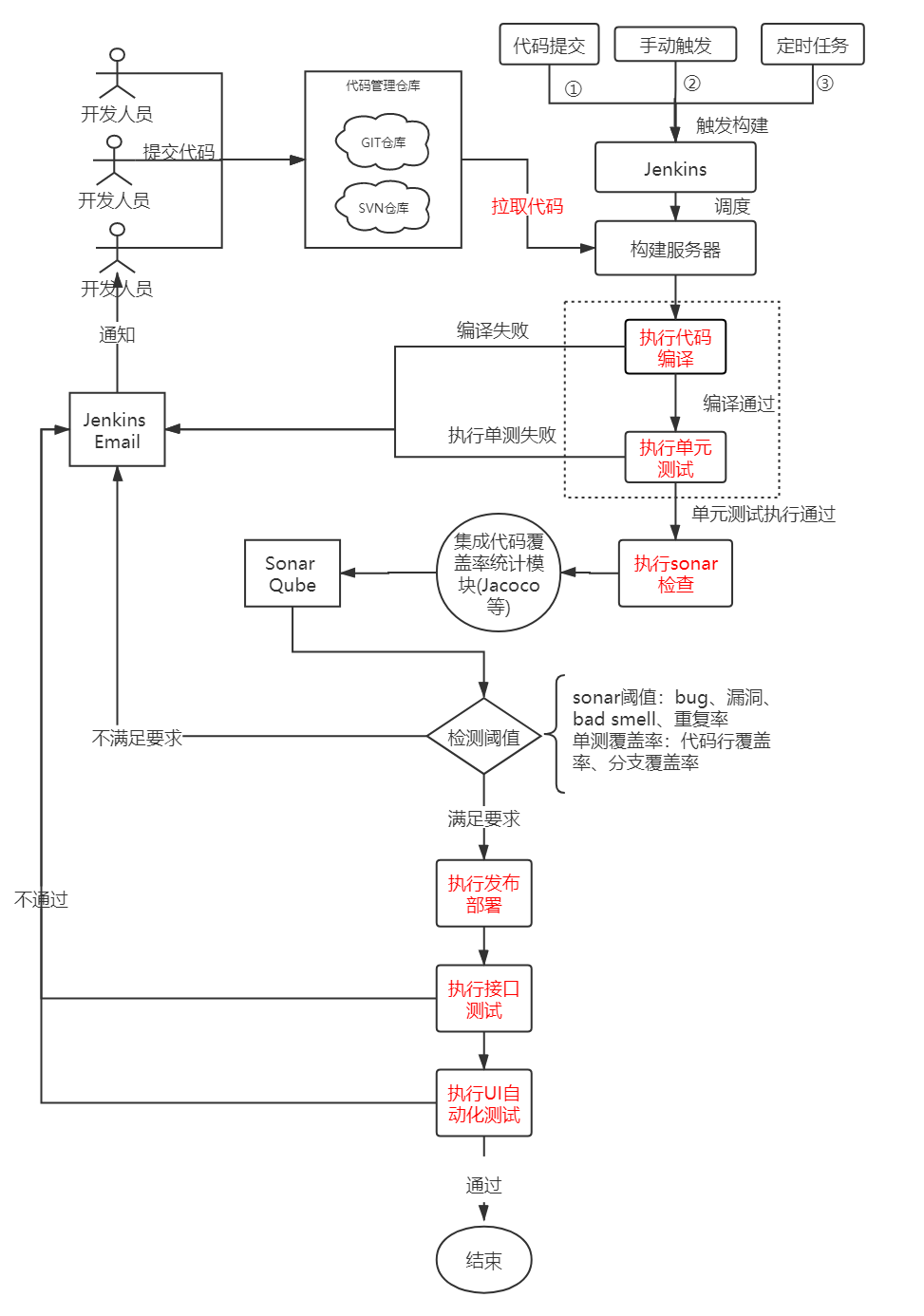
}

}

}

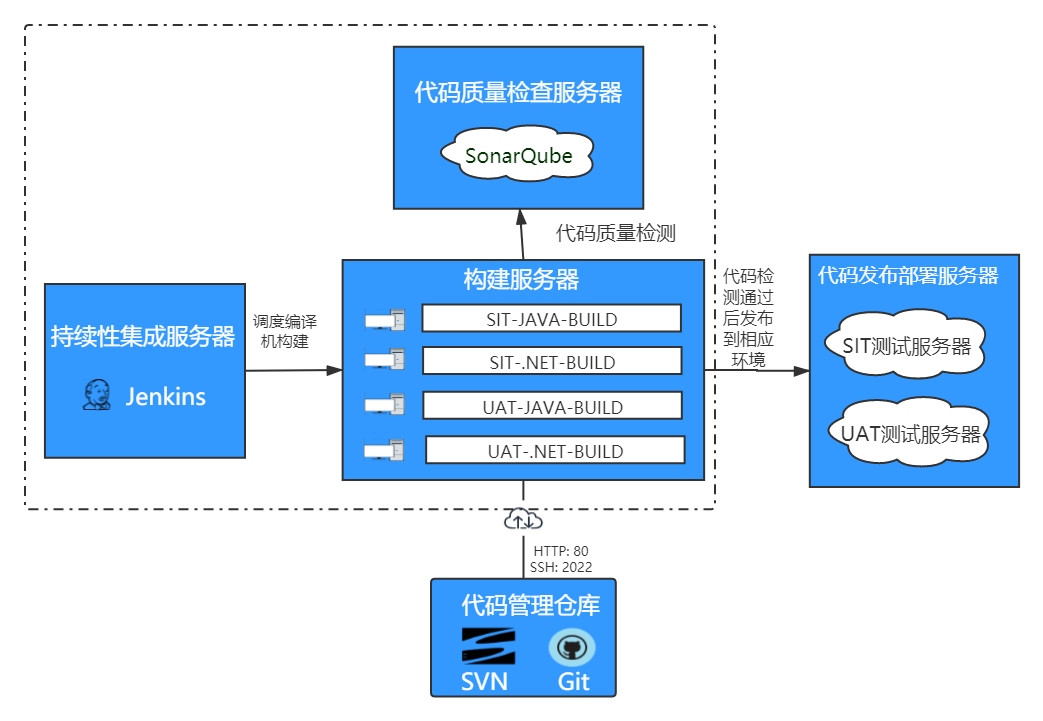
}

### 6.3 devops流程图



# 附件

### 7.1 devops物理架构图



### 7.2 物理机清单

|  |  |
| --- | --- |
| Ip | 服务器说明 |
| 10.98.0.81 | Jenkins服务器 |
| 10.90.16.45 | Sonar服务器 |
| 10.98.0.144 | Jenkins-编译机器（UAT-Java项目） |
| 10.98.0.145 | Jenkins-编译机器（UAT-NET项目） |
| 10.90.16.138 | Jenkins-编译机器（SIT-Java项目） |
| 10.90.16.107 | Jenkins-编译机器（SIT-NET项目） |
| itsvn.h3c.com | svn服务器 |
| 10.63.32.225 | git服务器 |