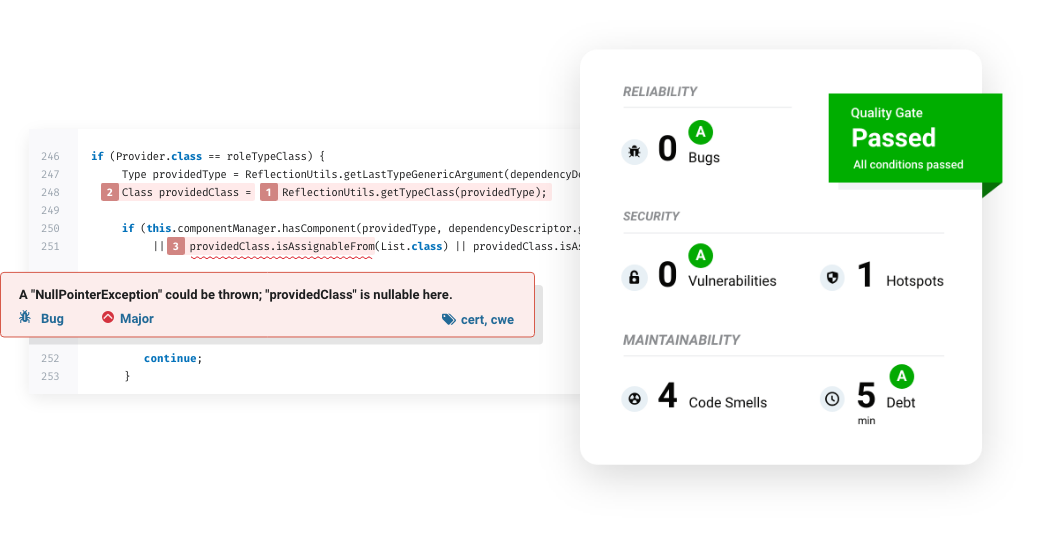
**Day4-07月03日-代码质量平台实践**

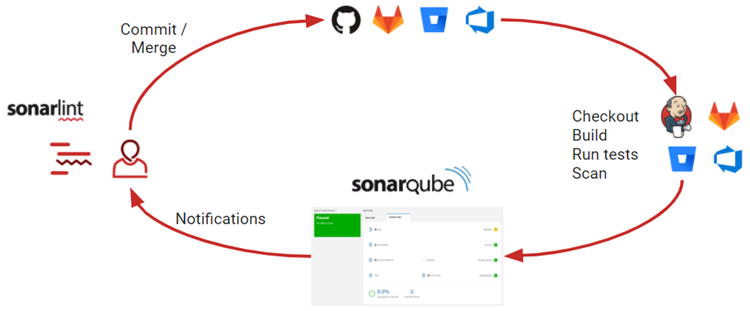
|  |  |  |
| --- | --- | --- |
| **时间** | **标题** | **内容** |
| **上午**  **下午** | **代码质量平台实践**  （SonarQube） | 1. SonarQube平台的简介与工作原理 2. SonarQube平台的安装部署（LTS 7.9.x） 3. 常用的SonarScanner工具使用方法 4. 使用SonarScanner进行代码质量检查 5. 扫描报告指标含义及如何配置 |
| 1. CI流水线集成Sonar代码扫描阶段 2. SonarQube开源版本存在的问题 3. 使用SonarAPI完成项目的创建 4. 使用SonarAPI完成质量规则与质量阈配置 |
|
|
|
|
|  | 课程信息 | * ✅SonarQube版本=LTS 8.9.1 * ✅课程实验sublime代码： [📎day4.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1625309517663-a3f34a87-5f86-4068-a2aa-c938e6d2aa2b.zip) * ✅day4共享库代码： [📎jenkinslib-day4.zip](https://www.yuque.com/attachments/yuque/0/2021/zip/2584012/1625309903717-21b8bf24-f6a7-439c-aab2-418def8c656a.zip) |
|  | SonarQube 插件   * ✅官网： http://sonarqube.org * ✅中文插件： <https://github.com/xuhuisheng/sonar-l10n-zh/tree/master> * ✅scanner下载：<https://docs.sonarqube.org/8.9/analysis/overview/> * ✅插件源：<https://update.sonarsource.org/> * ✅API文档： <http://192.168.1.200:9000/web_api> * ✅多分支插件： <https://github.com/mc1arke/sonarqube-community-branch-plugin/releases> * ✅关联提交：<https://github.com/gabrie-allaigre/sonar-gitlab-plugin/tree/4.1.0-SNAPSHOT> | |

**1. 代码质量平台实践**



**1.1 SonarQube 简介**

[SonarQube®](http://www.sonarqube.org/)是一种自动代码审查工具，可检测代码中的**错误，漏洞和代码味道。**它可以与您现有的工作流程集成，以实现**跨项目分支**和**拉取请求的持续代码检查。**

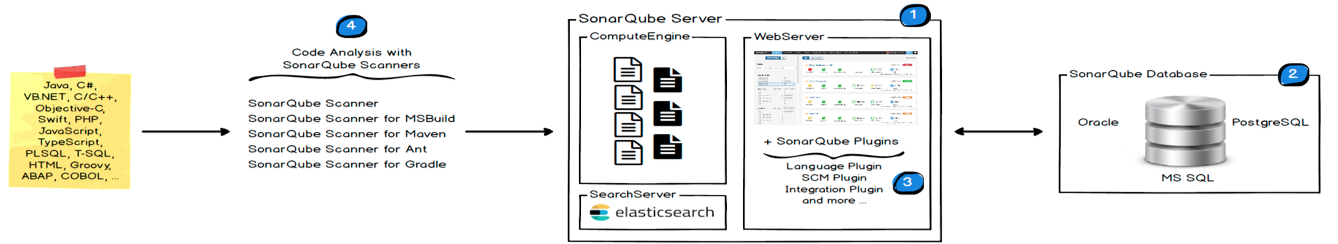


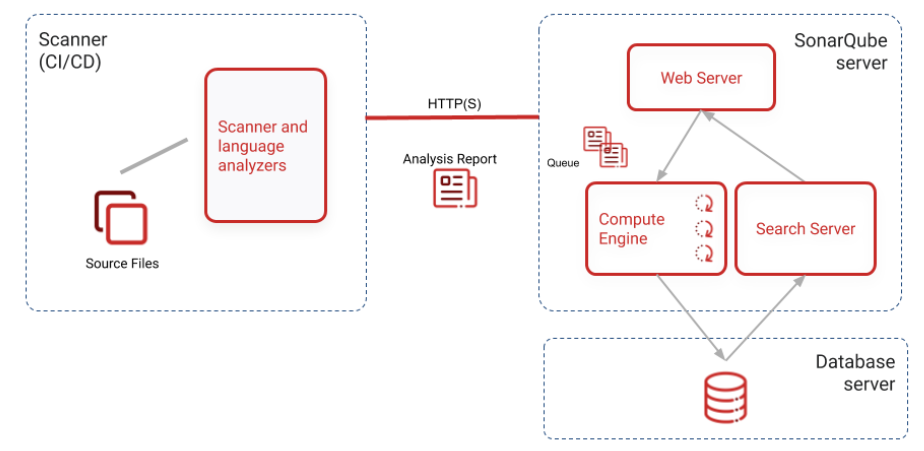
开发人员在IDE开发代码，可以安装**SonarLint**插件进行提交前代码扫描 当开发人员提交代码到版本控制系统中，自动触发jenkins进行代码扫描。

**1.2 SonarQube平台安装配置**

**1.2.1 组件与服务组成**

* **SonarQube Server**启动3个主要进程：
  + **Web服务器**，供开发人员，管理人员浏览高质量的快照并配置SonarQube实例
  + 基于Elasticsearch的**Search Server**从UI进行搜索服务。
  + **Compute Engine**服务器，负责处理代码分析报告并将其保存在SonarQube数据库中。
* **SonarQube数据库**要存储：SonarQube实例的配置（安全，插件设置等）项目，视图质量快照。
* 服务器上安装了多个**SonarQube插件**，可能包括语言，SCM，集成，身份验证和管理插件。
* 在持续集成服务器上运行一个或多个**SonarScanner**，以分析项目。





**1.2.2 安装配置**

* Java (Oracle JRE 11 or OpenJDK 11)

下面是基于Docker的方式安装部署SonaQube：

## 创建数据目录

mkdir -p /data/cicd2/sonarqube/{sonarqube\_conf,sonarqube\_extensions,sonarqube\_logs,sonarqube\_data}

chmod 777 -R /data/cicd2/sonarqube/

## 运行

docker run -itd --name sonarqube \

-p 9000:9000 \

-v /data/cicd2/sonarqube/sonarqube\_conf:/opt/sonarqube/conf \

-v /data/cicd2/sonarqube/sonarqube\_extensions:/opt/sonarqube/extensions \

-v /data/cicd2/sonarqube/sonarqube\_logs:/opt/sonarqube/logs \

-v /data/cicd2/sonarqube/sonarqube\_data:/opt/sonarqube/data \

sonarqube:8.9.1-community

## 验证

docker logs -f sonarqube

## lib目录（低于SonarQube8.9.1可以配置持久化，以支持多分支插件。）

mkdir -p /data/cicd/sonarqube/sonarqube\_lib

cd /data/cicd/sonarqube/sonarqube\_lib

docker cp sonarqube:/opt/sonarqube/lib/\* ./

docker run -itd --name sonarqube \

-p 9000:9000 \

-v /data/cicd2/sonarqube/sonarqube\_conf:/opt/sonarqube/conf \

-v /data/cicd2/sonarqube/sonarqube\_extensions:/opt/sonarqube/extensions \

-v /data/cicd2/sonarqube/sonarqube\_logs:/opt/sonarqube/logs \

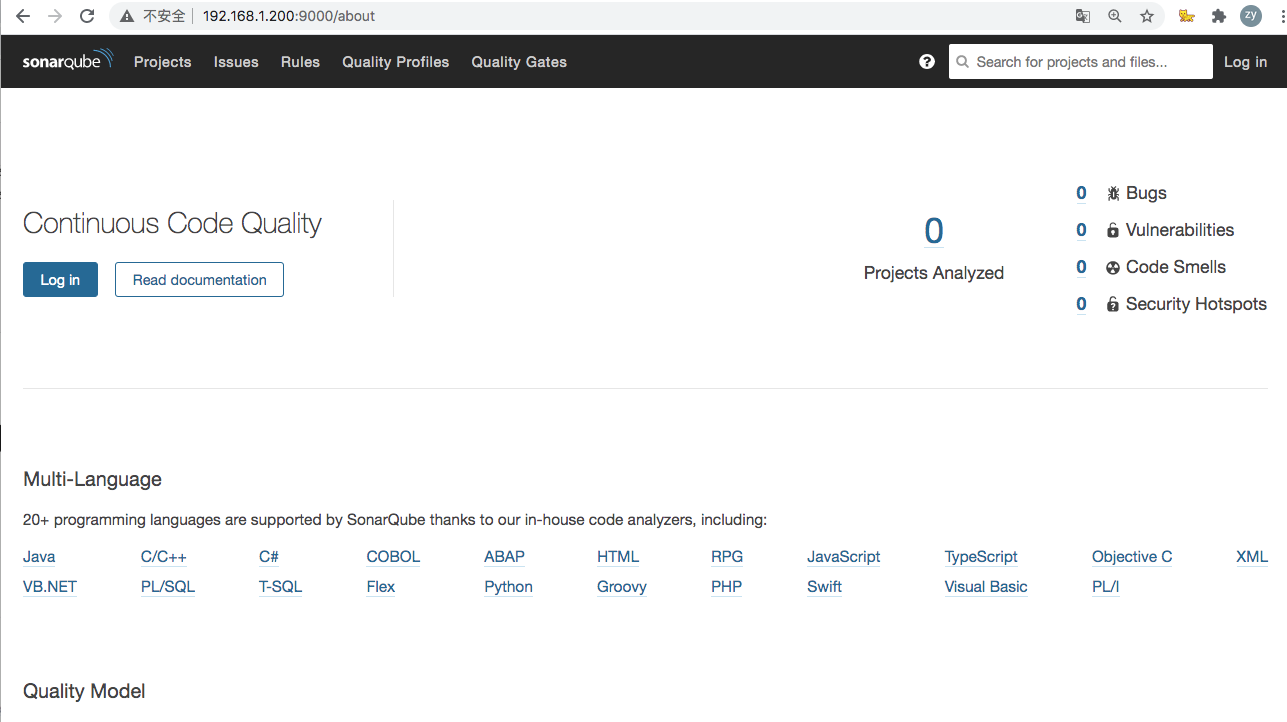
-v /data/cicd2/sonarqube/sonarqube\_data:/opt/sonarqube/data \

-v /data/cicd2/sonarqube/sonarqube\_lib:/opt/sonarqube/lib \

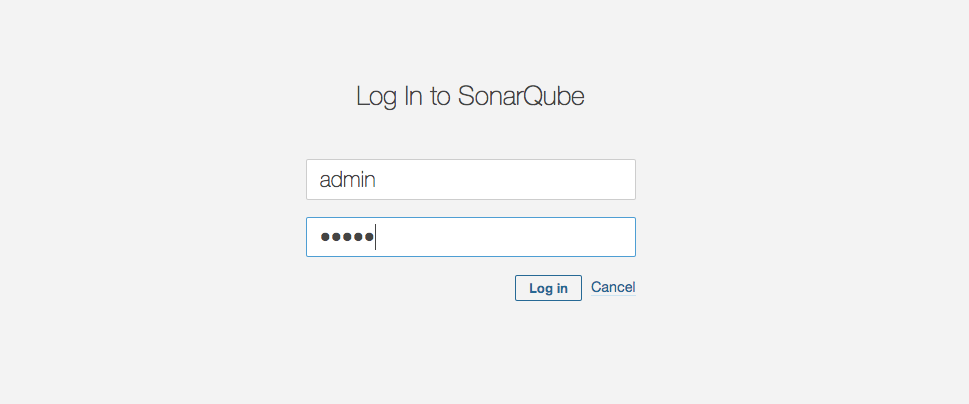
sonarqube:8.9.1-community

访问： [http://192.168.1.200:9000/](http://192.168.1.200:9000/about) admin admin1234

这是SonarQubeLTS 7.9.6 的默认首页：



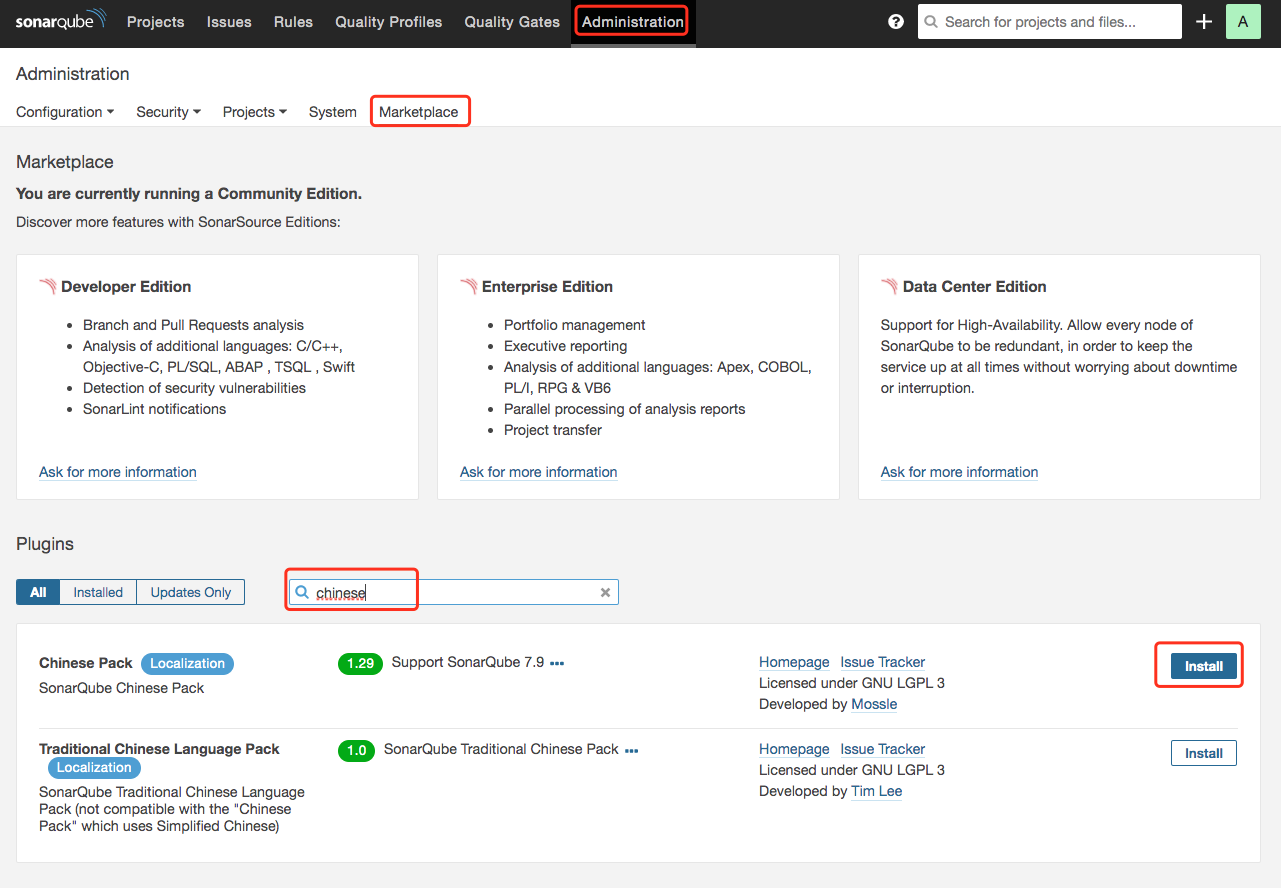
**这是SonarQubeLTS 8.9.1 的默认首页：**



**1.2.3 安装中文插件**

在线安装或者手动上传插件

**在线安装：操作如下图**

阅

**1.2.4 手动安装插件**

为什么需要手动安装？ 由于不同区域的不同网络原因，可能在线源会出现下载插件失败的情况，超时。本次以中文插件为例，为大家讲述如何手动离线安装一次插件。

Caused by: org.sonar.api.utils.SonarException: Fail to download: https://github.com/xuhuisheng/sonar-l10n-zh/releases/download/sonar-l10n-zh-plugin-1.29/sonar-l10n-zh-plugin-1.29.jar (no proxy)

at org.sonar.core.util.DefaultHttpDownloader.failToDownload(DefaultHttpDownloader.java:155)

at org.sonar.core.util.DefaultHttpDownloader.download(DefaultHttpDownloader.java:150)

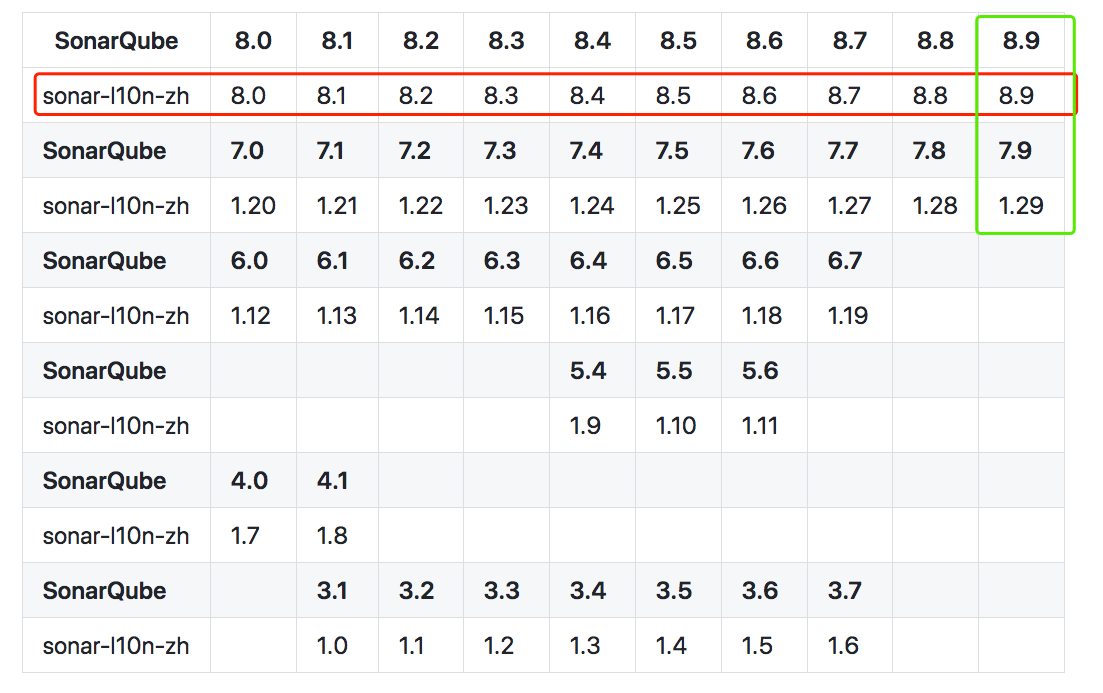
at org.sonar.server.plugins.PluginDownloader.downloadRelease(PluginDownloader.java:152)

at org.sonar.server.plugins.PluginDownloader.download(PluginDownloader.java:126)

... 47 common frames omitted

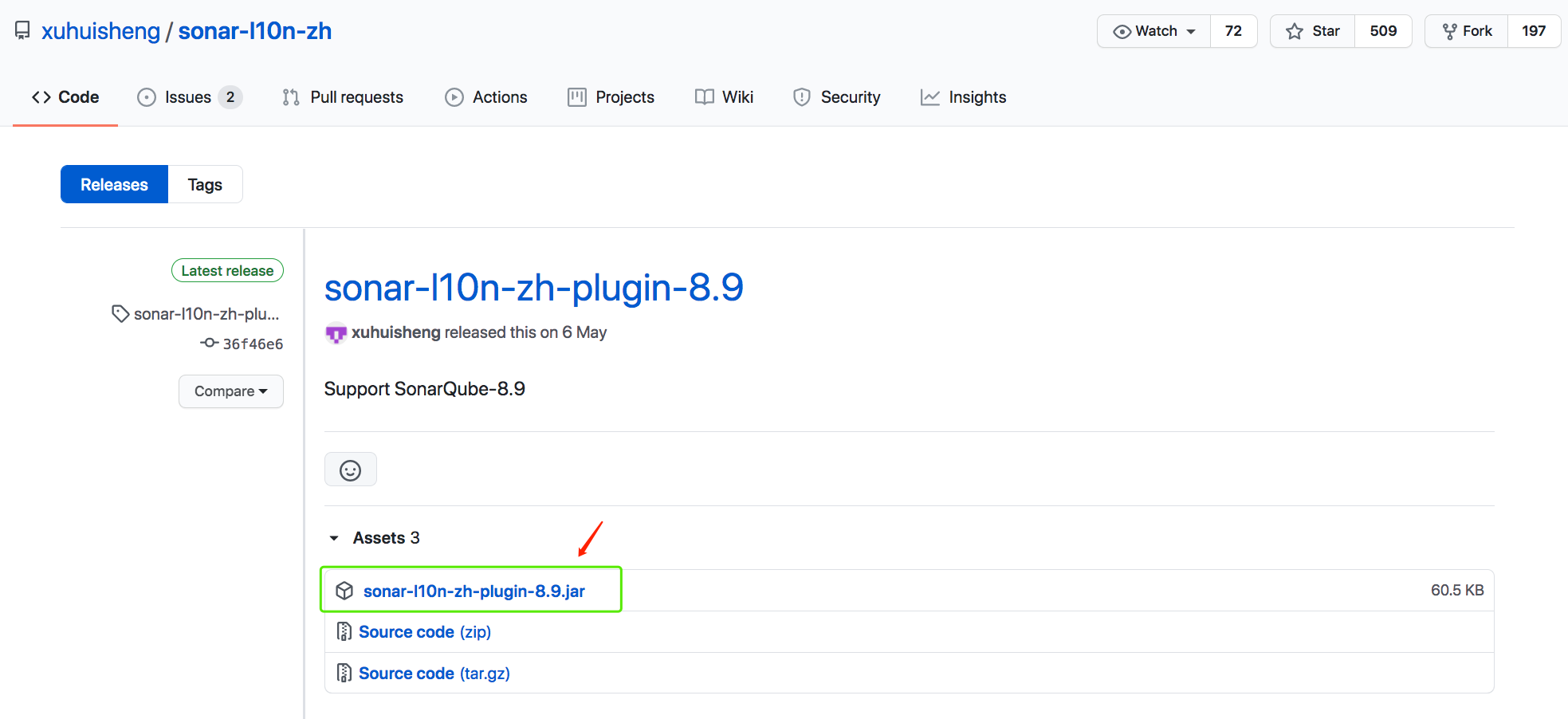
Caused by: java.net.SocketTimeoutException: Read timed out

阅读插件说明文档：<https://github.com/xuhuisheng/sonar-l10n-zh/tree/master> 找到sonarqube 版本对应的插件版本。



选择Releases 找到当前SonarQube对应的版本， 我的版本是8.9.1， 所以下载 :

<https://github.com/xuhuisheng/sonar-l10n-zh/releases/download/sonar-l10n-zh-plugin-8.9/sonar-l10n-zh-plugin-8.9.jar>



cd /data/cicd/sonarqube/sonarqube\_extensions/downloads

wget https://github.com/xuhuisheng/sonar-l10n-zh/releases/download/sonar-l10n-zh-plugin-1.29/sonar-l10n-zh-plugin-1.29.jar

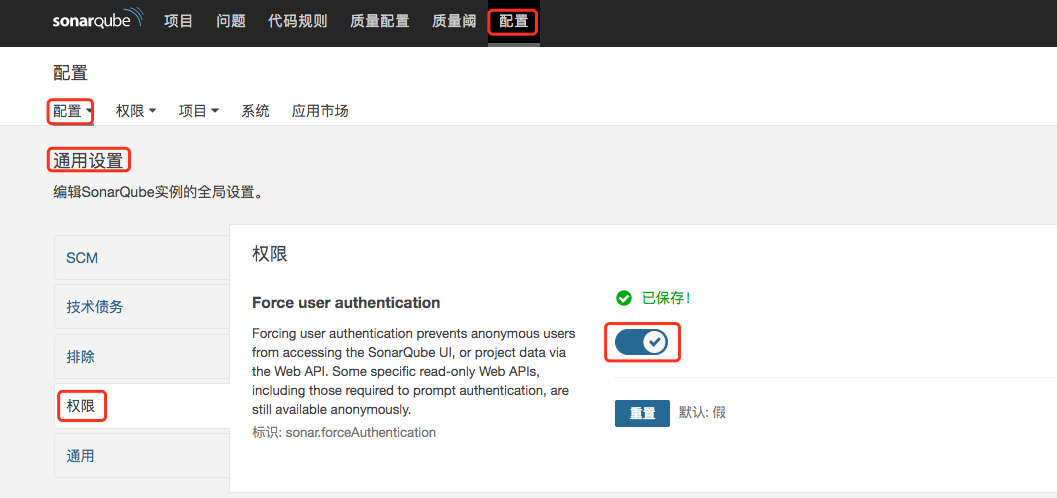
chmod +x sonar-l10n-zh-plugin-1.29.jar

docker restart sonarqube

插件也可以直接在插件源：<https://update.sonarsource.org/>中下载的。

**1.2.5 配置强制登录**

默认所有项目都可以公开的查看，在企业内肯定要配置成私有的。只有登录后才能查看。(***sonarqube8.9.1 默认已经开启了***，低于该版本的可以手动开启)



**1.2.6 配置Scanner**

scanner的类型有很多， 可以通过官网：<https://docs.sonarqube.org/8.9/analysis/overview/> 获取支持的列表。

* 变更项目代码： 可以使用**构建工具**进行扫描。例如maven、ant、gradle可以在配置文件中引入对应的配置。
* 不变更项目代码配置： 可以使用Jenkins或其他平台的扩展插件， 以及使用命令行进行扫描。
* Gradle - [SonarScanner for Gradle](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-gradle/)
* .NET - [SonarScanner for .NET](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-msbuild/)
* Maven - use the [SonarScanner for Maven](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-maven/)
* Jenkins - [SonarScanner for Jenkins](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-jenkins/) （Jenkins 集成插件）
* Azure DevOps - [SonarQube Extension for Azure DevOps](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-azure-devops/) （devops平台扩展插件）
* Ant - [SonarScanner for Ant](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-ant/)
* anything else (CLI) - [SonarScanner](https://docs.sonarqube.org/latest/analysis/scan/sonarscanner/)（更加通用的扫描命令行）

使用更加灵活的命令行工具进行扫描， 可以在不更改项目配置的前提下进行代码扫描：

## 进入实验目录

cd /data/cicd2

## 下载包

wget https://binaries.sonarsource.com/Distribution/sonar-scanner-cli/sonar-scanner-cli-4.6.2.2472-linux.zip

## 解压

tar zxf sonar-scanner-xxxx.tar.gz

unzip sonar-scanner-cli-4.6.2.2472-linux.zip

## 设置环境变量

vim /etc/profile

export SONAR\_SCANNER\_HOME=/data/cicd2/buildtools/sonar-scanner-4.6.2.2472-linux

export PATH=$SONAR\_SCANNER\_HOME/bin:$PATH

source /etc/profile

## 测试生效

[root@zeyang-nuc-service sonar-scanner-4.6.2.2472-linux]# sonar-scanner -v

INFO: Scanner configuration file: /data/cicd2/buildtools/sonar-scanner-4.6.2.2472-linux/conf/sonar-scanner.properties

INFO: Project root configuration file: NONE

INFO: SonarScanner 4.6.2.2472

INFO: Java 11.0.11 AdoptOpenJDK (64-bit)

INFO: Linux 4.18.0-80.el8.x86\_64 amd64

这里请注意JDK版本的问题， 默认安装的sonarscanner是使用自带的jdk。通过上面的显示可以发现是jdk1.11版本， 如果想使用系统默认的jdk版本，可以修改scanner可执行程序的配置。如下所示：

[root@zeyang-nuc-service sonar-scanner-4.6.0.2311-linux]# vi bin/sonar-scanner

[root@zeyang-nuc-service sonar-scanner-4.6.0.2311-linux]#

### 编辑约第42行， true改成false

42 use\_embedded\_jre=false

43 if [ "$use\_embedded\_jre" = true ]; then

44 export JAVA\_HOME="$sonar\_scanner\_home/jre"

45 fi

### 更改后，再次验证，发现变成了系统jdk版本。 更改生效。

[root@zeyang-nuc-service sonar-scanner-4.6.0.2311-linux]# sonar-scanner -v

INFO: Scanner configuration file: /usr/local/sonar-scanner-4.6.0.2311-linux/conf/sonar-scanner.properties

INFO: Project root configuration file: NONE

INFO: SonarScanner 4.6.0.2311

INFO: Java 1.8.0\_282 AdoptOpenJDK (64-bit)

INFO: Linux 4.18.0-80.el8.x86\_64 amd64

**1.3 SonarScanner使用方法**

代码扫描过程： 本地（构建节点）安装配置SonarScanner环境，然后通过设置sonar的一系列参数进行扫描分析。

* 配置文件方式读取扫描参数
* 命令行方式读取扫描参数

一个基本的sonar-project.properties配置文件的参数：

# 定义唯一的关键字

sonar.projectKey=devops-hello-service

# 定义项目名称

sonar.projectName=My project

# 定义项目的版本信息

sonar.projectVersion=1.0

# 指定扫描代码的目录位置（多个逗号分隔）

sonar.sources=.

# 执行项目编码

sonar.sourceEncoding=UTF-8

sonar.host.url=

sonar.login

sonar.password

这些配置项都是统一的，目前sonar支持将扫描参数以文件的方式存放或者以命令行传参的方式读取。 文件方式：可以将扫描参数放到项目的根目录或者sonar-scanner的配置文件目录等自定义的目录中， 命令行传参则可以直接将变量传递给sonarsacnner cli -Dsonar.projectKey=xxx 。

# 指定配置文件

sonar-scanner -Dproject.settings=myproject.properties

# 命令行传参

sonar-scanner -Dsonar.projectKey=myproject -Dsonar.sources=src1

**扩展-Docker运行sonarscanner**

docker run \

--rm \

-e SONAR\_HOST\_URL="http://${SONARQUBE\_URL}" \

-e SONAR\_LOGIN="myAuthenticationToken" \

-v "${YOUR\_REPO}:/usr/src" \

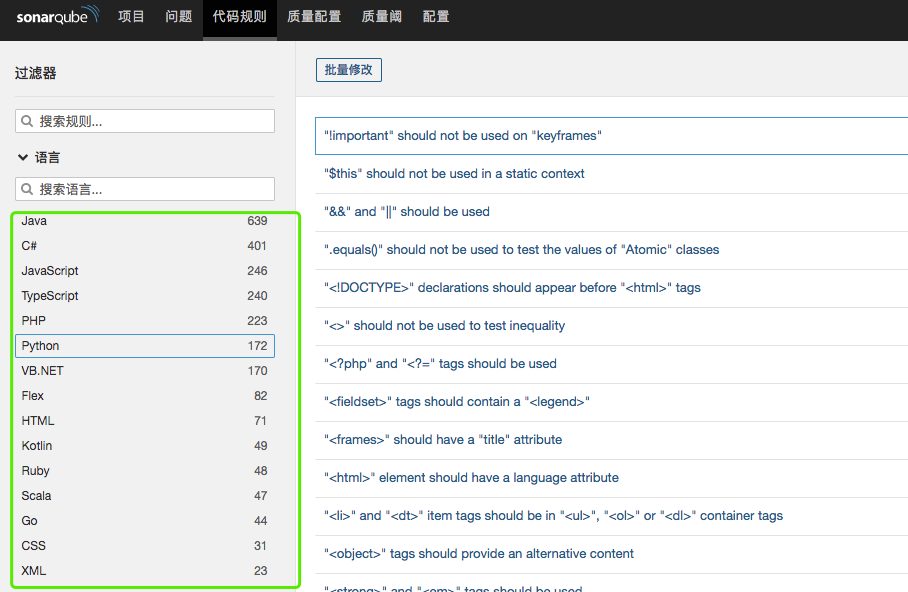
sonarsource/sonar-scanner-cli

关于项目参数可以参考：<https://docs.sonarqube.org/latest/analysis/analysis-parameters/>

各种语言的扫描示例：[https://docs.sonarqube.org/latest/analysis/languages/](https://docs.sonarqube.org/latest/analysis/languages/go/)

SonarQube中各种语言的扫描规则都是**以jar包的方式**。在SonarQube8.9.1之前版本，默认没有安装语言规则插件， 需要手动安装。 服务端安装**Java Code Quality and Security**SonarJS SonarGO 插件，并重启服务器。（如果这里由于网速原因下载不了插件，可以使用课程提供的压缩包，解压到downloads目录下然后重启sonarqube）

***由于本次实验使用的是SonarQube8.9.1版本，所以不用在手动安装语言插件了。***



如果低于该版本，可以参考如下操作。（可选，跳过）

[root@zeyang-nuc-service ~]# cd /data/cicd/plugin-sonar/

[root@zeyang-nuc-service plugin-sonar]# ls

sonar-go-plugin-1.6.0.719.jar sonar-l10n-zh-plugin-1.29.jar

sonar-java-plugin-6.3.2.22818.jar sonar-typescript-plugin-2.1.0.4359.jar

sonar-javascript-plugin-6.2.2.13315.jar

[root@zeyang-nuc-service plugin-sonar]# cp sonar-go-plugin-1.6.0.719.jar sonar-java-plugin-6.3.2.22818.jar sonar-javascript-plugin-6.2.2.13315.jar sonar-typescript-plugin-2.1.0.4359.jar /data/cicd/sonarqube/sonarqube\_extensions/downloads/

[root@zeyang-nuc-service plugin-sonar]# ls /data/cicd/sonarqube/sonarqube\_extensions/downloads/

sonar-go-plugin-1.6.0.719.jar sonar-javascript-plugin-6.2.2.13315.jar

sonar-java-plugin-6.3.2.22818.jar sonar-typescript-plugin-2.1.0.4359.jar

[root@zeyang-nuc-service plugin-sonar]# chmod +x /data/cicd/sonarqube/sonarqube\_extensions/downloads/\*

[root@zeyang-nuc-service plugin-sonar]# ls /data/cicd/sonarqube/sonarqube\_extensions/downloads/

sonar-go-plugin-1.6.0.719.jar sonar-javascript-plugin-6.2.2.13315.jar

sonar-java-plugin-6.3.2.22818.jar sonar-typescript-plugin-2.1.0.4359.jar

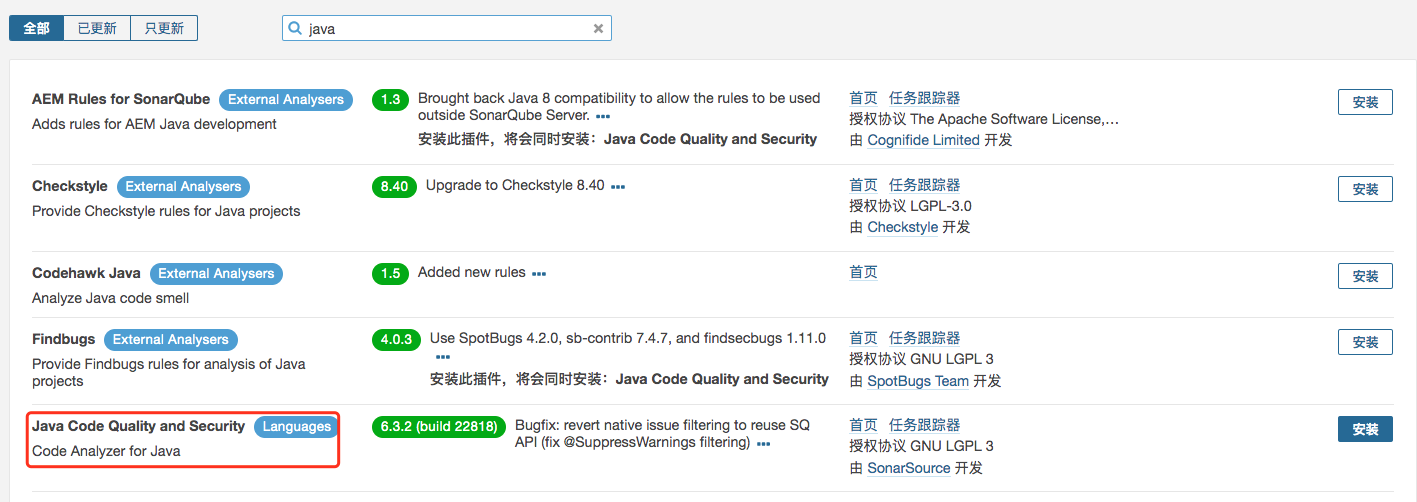
[root@zeyang-nuc-service plugin-sonar]# docker restart sonarqube

sonarqube



**1.3.1 Java项目扫描**

sonarqube服务器端需要安装Java语言规则插件



sonar.projectKey 指定项目的关键字，sonar.host.url指定服务器地址（可以直接在配置文件中写死）,projectName指定项目的名称, projectVersion指定项目的版本(可以用构建时间和构建ID定义),login指定登录用户名,password指定登录用户密码, projectDescription指定项目的描述信息, links.homepage指定项目的主页（超链接）, sources指定扫描的目录, sourceEncoding指定扫描时的编码, **java.binaries指定编译后的类文件目录(必填), java.test.binaries指定编译后的测试类目录,java.surefire.report指定测试报告目录。**

sonar-scanner -Dsonar.host.url=http://192.168.1.200:9000 \

-Dsonar.projectKey=devops-maven-service \

-Dsonar.projectName=devops-maven-service \

-Dsonar.projectVersion=1.0 \

-Dsonar.login=admin \

-Dsonar.password=admin \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=http://192.168.1.200/devops/devops-maven-service \

-Dsonar.links.ci=http://192.168.1.200:8080/job/demo-pipeline-service/ \

-Dsonar.sources=src \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports

**1.3.2 Web前端项目扫描**



sonar-scanner \

-Dsonar.projectKey=demo-devops-ui \

-Dsonar.projectName=demo-devops-ui \

-Dsonar.sources=src \

-Dsonar.host.url=http://192.168.1.200:9000 \

-Dsonar.login=0809881d71f2b06b64786ae3f81a9acf22078e8b \

-Dsonar.projectVersion=2.0 \

-Dsonar.ws.timeout=30 \

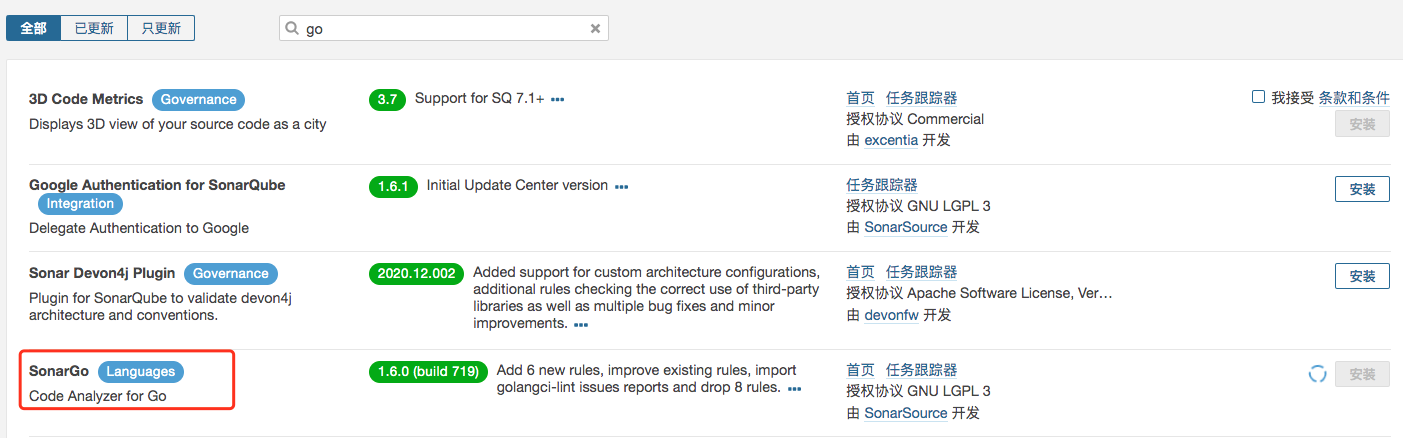
-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=http://192.168.1.200/devops/devops-maven-service \

-Dsonar.links.ci=http://192.168.1.200:8080/job/demo-pipeline-service/ \

-Dsonar.sourceEncoding=UTF-8

**1.3.4 Golang项目扫描**



sonar-scanner -Dsonar.projectKey=devops-golang-service \

-Dsonar.projectName=devops-golang-service \

-Dsonar.sources=src \

-Dsonar.login=admin \

-Dsonar.password=admin \

-Dsonar.host.url=http://192.168.1.200:9000

## 有测试用例的情况

sonar.exclusions=\*\*/\*\_test.go

sonar.tests=.

sonar.test.inclusions=\*\*/\*\_test.go

**1.4 CI流水线集成**

本次集成重点演示两种方式： 1. 使用命令行方式 2. 使用Jenkins扩展插件的方式。

**1.4.1 命令行方式**

流水线中添加代码扫描阶段， 然后在script标签中定义一段脚本。 （其实这段脚本就是我们手动在服务器上面执行的sonar-scanner的命令和参数组成的）【可以先运行该段代码确保扫描成功，然后进一步优化】

stage("SonarScan"){

steps{

script{

sh """

sonar-scanner \

-Dsonar.projectKey=demo-devops-service \

-Dsonar.projectName=demo-devops-service \

-Dsonar.sources=src \

-Dsonar.host.url=http://192.168.1.200:9000 \

-Dsonar.login=0809881d71f2b06b64786ae3f81a9acf22078e8b \

-Dsonar.projectVersion=1.0 \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=http://192.168.1.200/devops/devops-maven-service \

-Dsonar.links.ci=http://192.168.1.200:8080/job/demo-pipeline-service/ \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports

"""

}

}

}

}

通过上面的代码可以发现一些问题：

* 扫描参数值都是写死的，无法使其他项目通用。
* 代码中存在敏感数据信息。

既然想通用，就要制定规范。 例如：

* 我们统一使用Jenkins的作业名称做为SonarQube项目名称。
* 将Sonar在进行扫描时用到的认证信息，也存储到Jenkins的系统凭据中（Secret Text类型），避免流水线中存在敏感信息。然后使用withCredentials将凭据的值内容赋值给变量SONAR\_TOKEN引用。
* 使用BUILD\_NUMBER作为sonarqube项目版本（可以使用时间戳、版本号、commitid）。
* -Dsonar.links.ci=${BUILD\_URL} SonarQube的扩展链接， 方便在系统中跳转。
* -Dsonar.links.homepage=${env.srcUrl} SonarQube的扩展链接， 方便在系统中跳转。

优化后：

// 凭据列表

credentials = ["sonar" : '06bf5ee4-f571-4fe4-9b52-d17190ce54e5']

//服务器列表

servers = ["sonar": 'http://192.168.1.200:9000']

pipeline {

...

stage("CodeScan"){

steps{

script {

withCredentials([string(credentialsId: "${credentials['sonar']}", variable: 'SONAR\_TOKEN')]) {

sh """

sonar-scanner \

-Dsonar.projectKey=${JOB\_NAME.split('/')[-1]} \

-Dsonar.projectName=${JOB\_NAME.split('/')[-1]} \

-Dsonar.sources=src \

-Dsonar.host.url=${servers['sonar']} \

-Dsonar.login=${SONAR\_TOKEN} \

-Dsonar.projectVersion=${BUILD\_NUMBER} \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=${env.srcUrl} \

-Dsonar.links.ci=${BUILD\_URL} \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports

"""

}

}

}

}

}

上面我们完成了，一个Java类型项目的扫描参数设置。 但是是否想过在企业中存在***多种语言类型的项目***？ 此时我们要让代码扫描变的更加灵活一些，支持多种类型的代码扫描。

* 根据不同的构建工具，使用不同的代码扫描参数；
* 例如： maven 对应java类型项目， npm对应前端类型项目

withCredentials([string(credentialsId: "${credentials['sonar']}", variable: 'SONAR\_TOKEN')]) {

switch(buildType) {

case "mvn":

sh """

sonar-scanner \

-Dsonar.projectKey=${JOB\_NAME.split('/')[-1]} \

-Dsonar.projectName=${JOB\_NAME.split('/')[-1]} \

-Dsonar.sources=src \

-Dsonar.host.url=${servers['sonar']} \

-Dsonar.login=${SONAR\_TOKEN} \

-Dsonar.projectVersion=${BUILD\_NUMBER} \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=${env.srcUrl} \

-Dsonar.links.ci=${BUILD\_URL} \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports

"""

break

case "npm":

sh """

sonar-scanner \

-Dsonar.projectKey=${JOB\_NAME.split('/')[-1]} \

-Dsonar.projectName=${JOB\_NAME.split('/')[-1]} \

-Dsonar.sources=src \

-Dsonar.host.url=${servers['sonar']} \

-Dsonar.login=${SONAR\_TOKEN} \

-Dsonar.projectVersion=${BUILD\_NUMBER} \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=${env.srcUrl} \

-Dsonar.links.ci=${BUILD\_URL} \

-Dsonar.sourceEncoding=UTF-8 \

"""

break

default:

println("buildTools choice error![mvn|npm]")

break;

}

}

最终将上述代码，纳入共享库。创建sonar.groovy。

## sonar.groovy

package org.devops

// 代码扫描

def SonarScan(credentials,buildType,servers){

withCredentials([string(credentialsId: "${credentials['sonar']}", variable: 'SONAR\_TOKEN')]) {

switch(buildType) {

case "mvn":

sh """

sonar-scanner \

-Dsonar.projectKey=${JOB\_NAME.split('/')[-1]} \

-Dsonar.projectName=${JOB\_NAME.split('/')[-1]} \

-Dsonar.sources=src \

-Dsonar.host.url=${servers['sonar']} \

-Dsonar.login=${SONAR\_TOKEN} \

-Dsonar.projectVersion=${BUILD\_NUMBER} \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=${env.srcUrl} \

-Dsonar.links.ci=${BUILD\_URL} \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports

"""

break

case "npm":

sh """

sonar-scanner \

-Dsonar.projectKey=${JOB\_NAME.split('/')[-1]} \

-Dsonar.projectName=${JOB\_NAME.split('/')[-1]} \

-Dsonar.sources=src \

-Dsonar.host.url=${servers['sonar']} \

-Dsonar.login=${SONAR\_TOKEN} \

-Dsonar.projectVersion=${BUILD\_NUMBER} \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=${env.srcUrl} \

-Dsonar.links.ci=${BUILD\_URL} \

-Dsonar.sourceEncoding=UTF-8 \

"""

break

default:

println("buildTools choice error![mvn|npm]")

break;

}

}

}

Jenkinsfile内容：

def sonar = new org.devops.sonar()

stage("CodeScan"){

steps{

script{

sonar.SonarScan(credentials,buildType,servers)

}

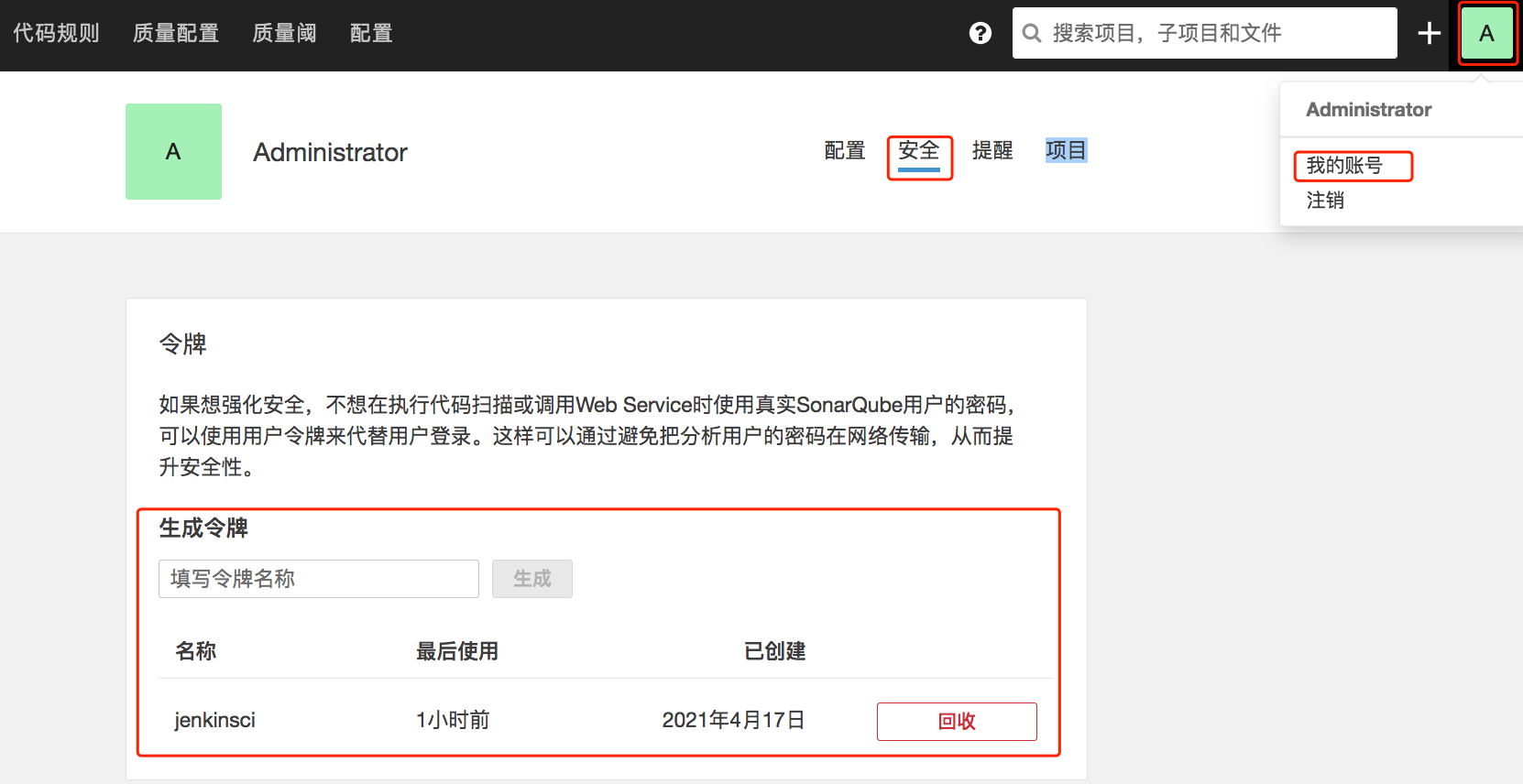
}

}

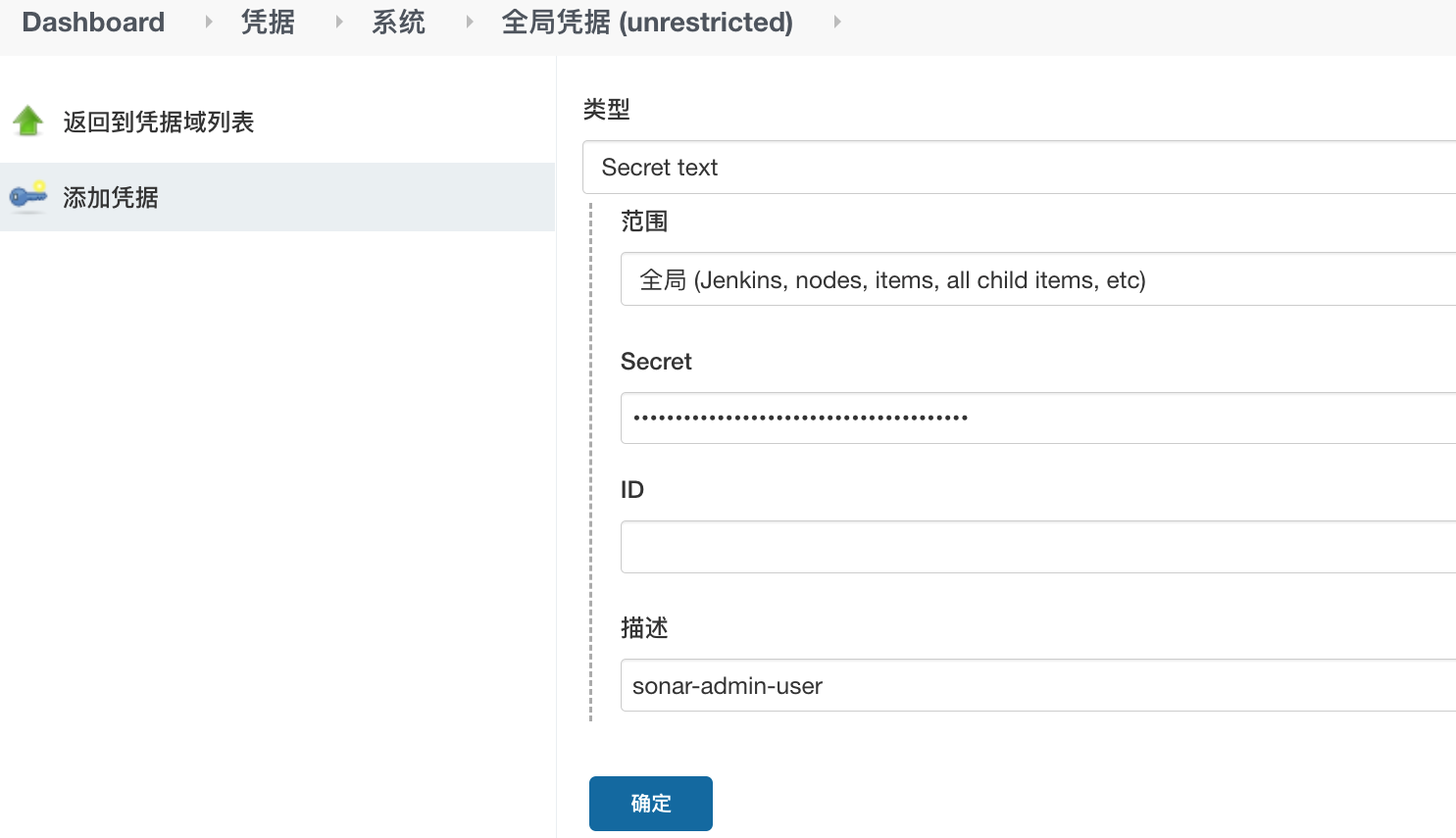
**1.4.2 插件方式**

参考：<https://docs.sonarqube.org/latest/analysis/scan/sonarscanner-for-jenkins/>

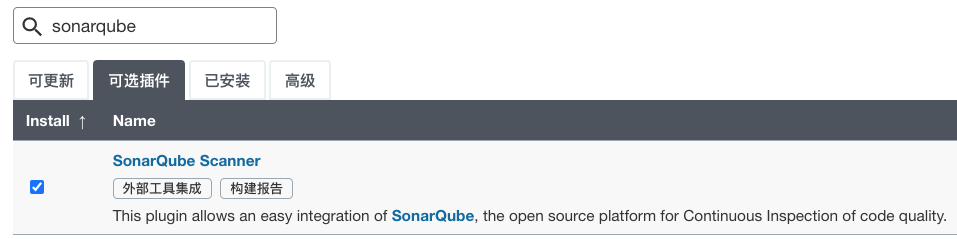
创建SonaQube的账户token



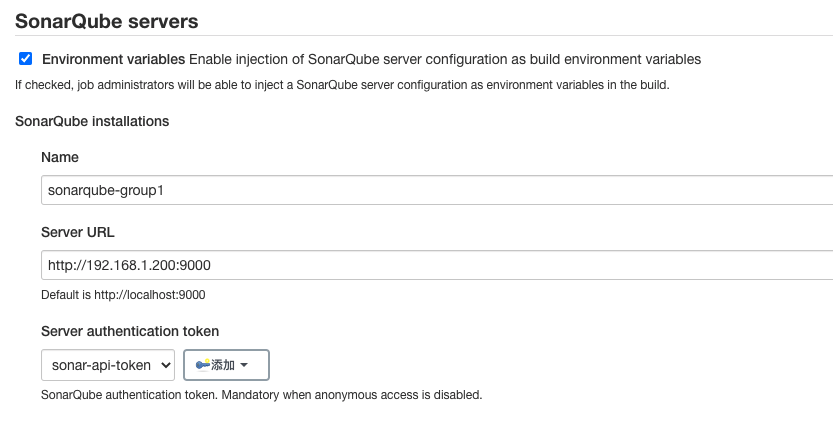
将token保存到Jenkins凭据中



在Jenkins中安装插件sonarqube scanner。



转到"**管理Jenkins>系统配置**"，向下滚动到**SonarQube配置**部分，单击Add SonarQube，添加服务器，选择凭据。

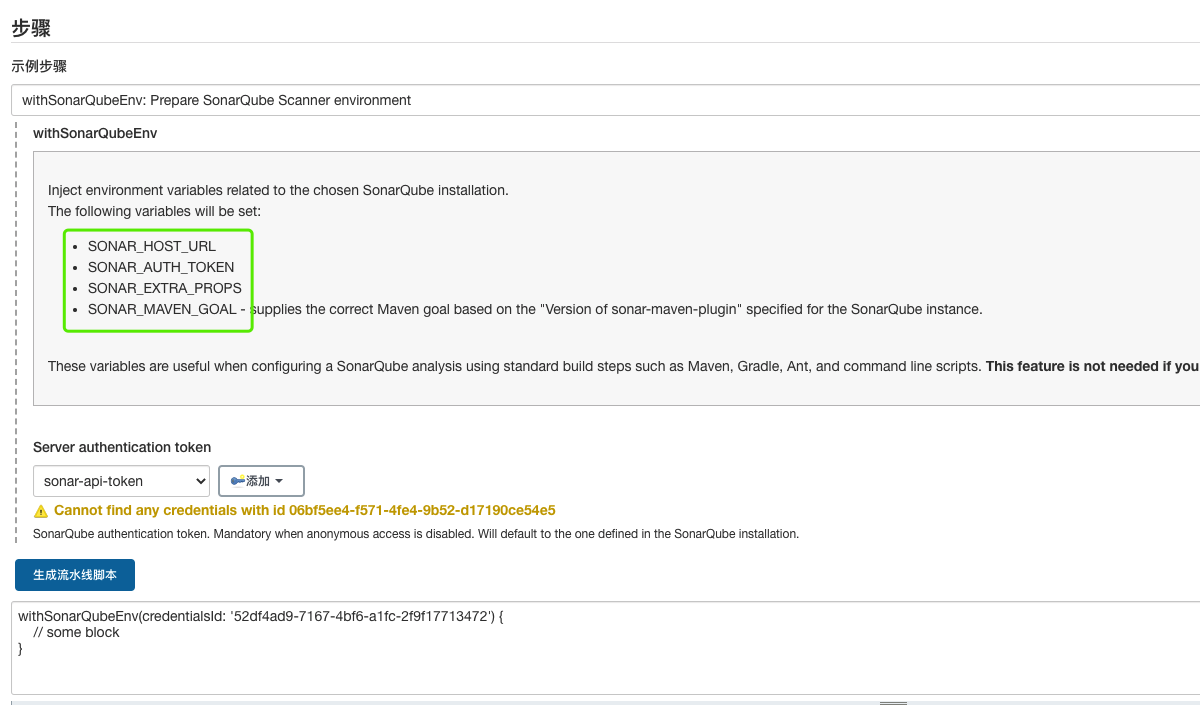


在片段生成器中查看用法, 注入与所选SonarQube 安装相关的环境变量。将设置以下变量：

SONAR\_HOST\_URL ## 在jenkins管理页面配置的sonar地址

SONAR\_AUTH\_TOKEN ## 在jenkins管理页面配置的sonar认证信息

此处生成的代码有问题， 推荐直接看下我提供的模板。



使用withSonarQubeEnv DSL引入在Jenkins中配置的sonar环境。

## 括号中的`sonarqube-group1`一定要与Jenkins设置页面定义的一致。

withSonarQubeEnv("sonarqube-group1") {

sh """

sonar-scanner \

-Dsonar.projectKey=${JOB\_NAME.split('/')[-1]} \

-Dsonar.projectName=${JOB\_NAME.split('/')[-1]} \

-Dsonar.sources=src \

-Dsonar.host.url=${SONAR\_HOST\_URL} \

-Dsonar.login=${SONAR\_AUTH\_TOKEN} \

-Dsonar.projectVersion=${BUILD\_NUMBER} \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=${env.srcUrl} \

-Dsonar.links.ci=${BUILD\_URL} \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports

"""

}

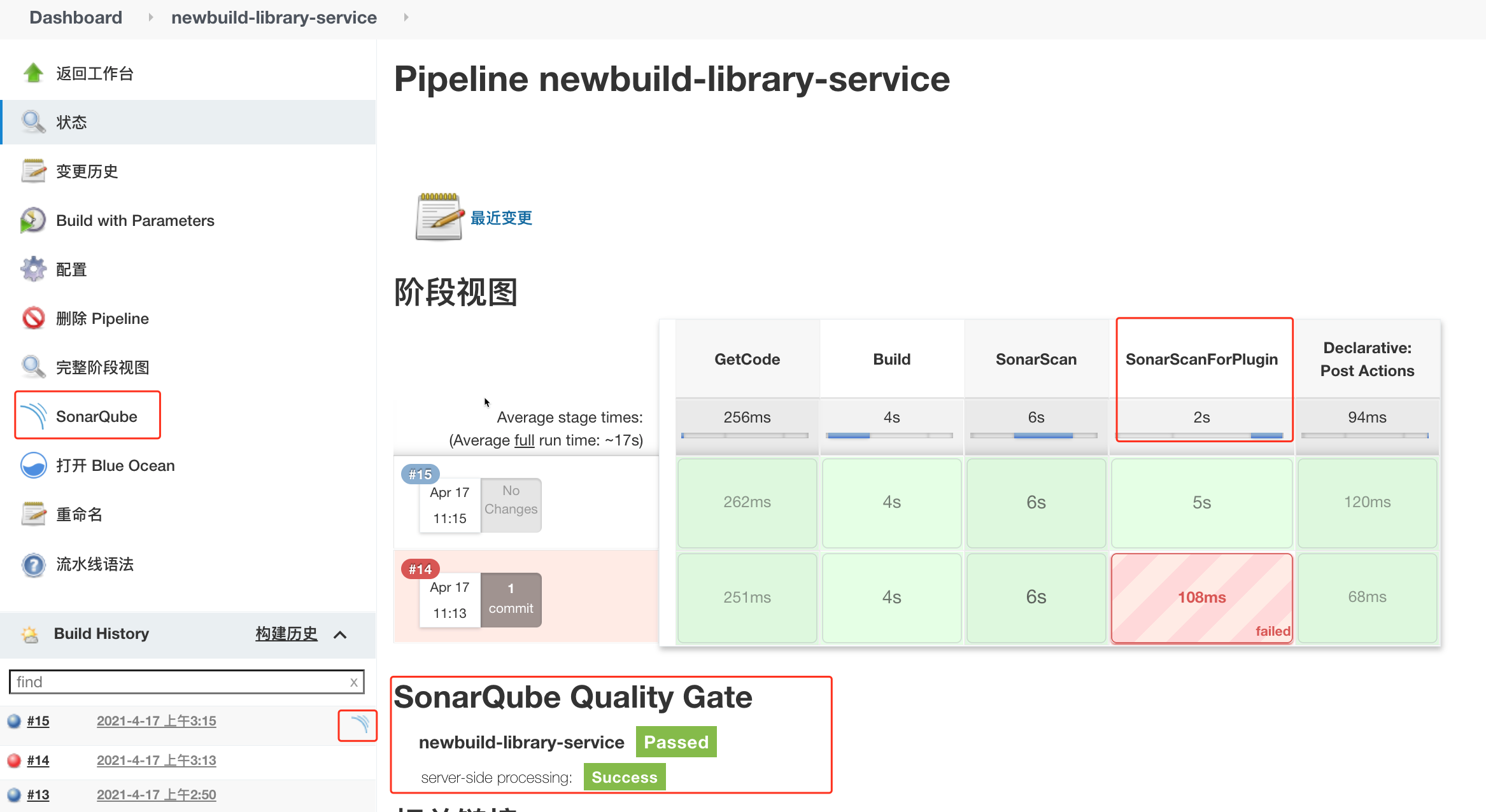
FAQ： sonar服务器名称错误，需要与系统设置中配置的一致。

ERROR: SonarQube installation defined in this job (mysonarserver) does not match any configured installation. Number of installations that can be configured: 1.

If you want to reassign jobs to a different SonarQube installation, check the documentation under https://redirect.sonarsource.com/plugins/jenkins.html

最终的效果如下：

**相比命令行只是增加了一些扩展链接， 对于经常操作Jenkins的用户方便些。**



最后课程中， 在sonar.groovy文件中。同时定义了SonarScanWithPlugin(插件扫描) 和SonarScan(命令行方式扫描)。 用法没有太大的区别，***使用插件扫描的共享库代码部分，可以参考使用命令行方式的讲解。***

**1.5 SonarQube REST API实践**

SonarQube系统的API文档： <http://192.168.1.200:9000/web_api>

//查找项目

api/projects/search?projects=${projectName}"

//创建项目

api/projects/create?name=${projectName}&project=${projectName}"

//更新语言规则集

api/qualityprofiles/add\_project?language=${language}&qualityProfile=${qualityProfile}&project=${projectName}"

//项目授权

api/permissions/apply\_template?projectKey=${projectKey}&templateName=${templateName}"

//更新质量阈

api/qualitygates/select?projectKey=${projectKey}&gateId=${gateId}"

SonarQube API的请求方法

curl --location \

--request GET \

'http://192.168.1.200:9000/api/projects/search?projects=day4-maven2-service' \

--header 'Authorization: Basic YWRtaW46YWRtaW4xMjM0'

Jenkins代码中不要存在敏感信息， 将base64格式的SonarQube 用户token YWRtaW46YWRtaW4xMjM0存储到Jenkins凭据中(Secret Text类型)，后续使用withCredentials将值赋值给变量SONAR\_TOKEN 。

考虑到Api的URL都具有相同部分<http://192.168.1.200:9000/api>所以单独复制给变量sonarApi。每个接口返回的都是JSON类型的数据， 这里使用readJSON进行解析和处理。【所以有了下面的代码】

def SonarRequest(apiUrl,method){

withCredentials([string(credentialsId: "52df4ad9-7167-4bf6-a1fc-2f9f17713472", variable: 'SONAR\_TOKEN')]) {

sonarApi = "http://192.168.1.200:9000/api"

response = sh returnStdout: true,

script: """

curl --location \

--request ${method} \

"${sonarApi}/${apiUrl}" \

--header "Authorization: Basic ${SONAR\_TOKEN}"

"""

try {

response = readJSON text: """ ${response - "\n"} """

} catch(e){

response = readJSON text: """{"errors" : true}"""

}

return response

}

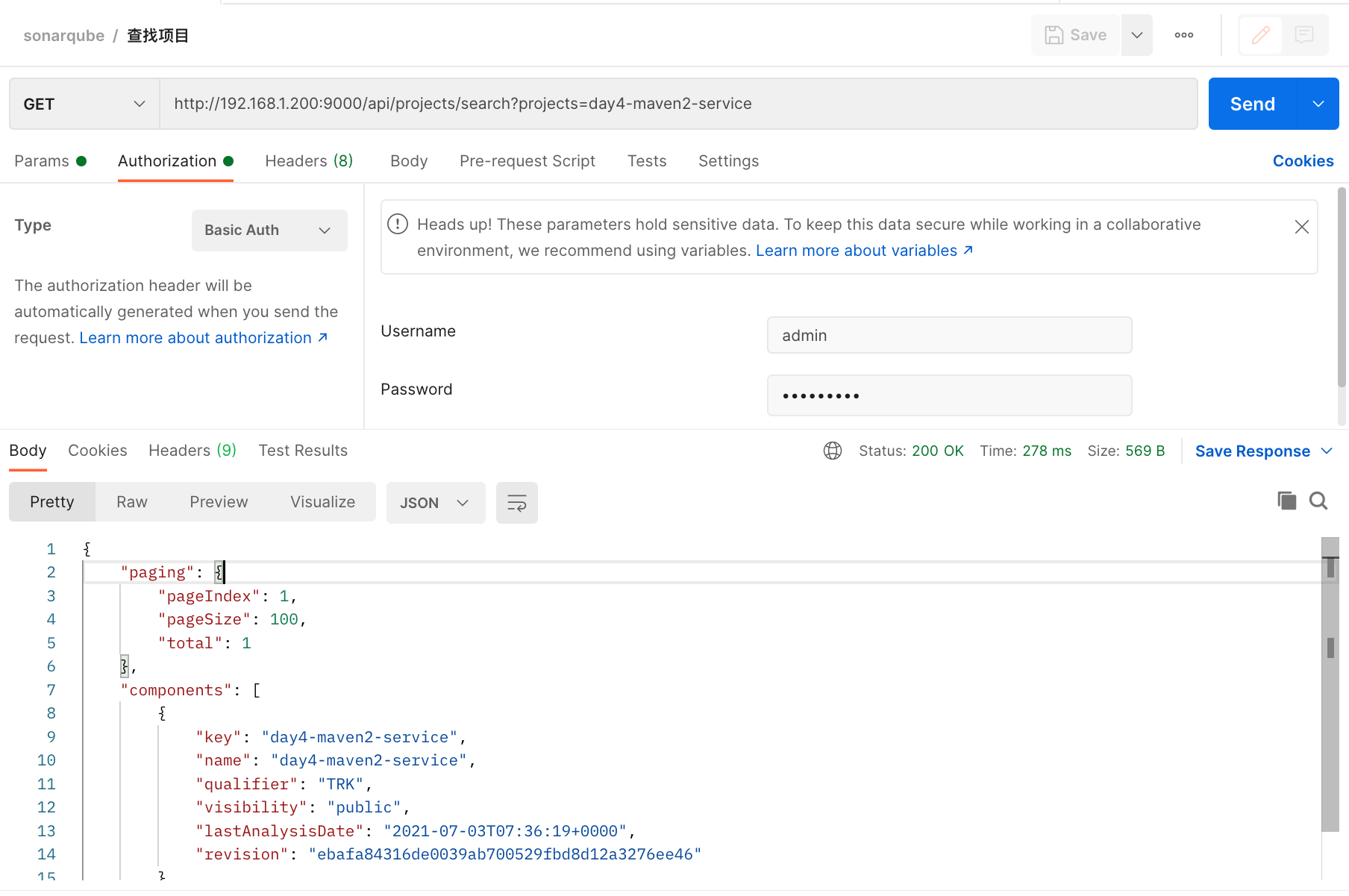
}

**1.5.1 查找项目**

接口地址和参数: <http://192.168.1.200:9000/api/projects/search?projects=day4-maven2-service>

请求类型： GET

postman调试：



转换cURL命令：

curl --location --request GET 'http://192.168.1.200:9000/api/projects/search?projects=day4-maven2-service' \

--header 'Authorization: Basic YWRtaW46YWRtaW4xMjM0'

Jenkins Pipeline

// 查找项目

def ProjectSearch(projectName){

apiUrl = "projects/search?projects=${projectName}"

response = SonarRequest(apiUrl,"GET")

if (response.paging.total == 0){

println("Project not found!.....")

return false

}

return true

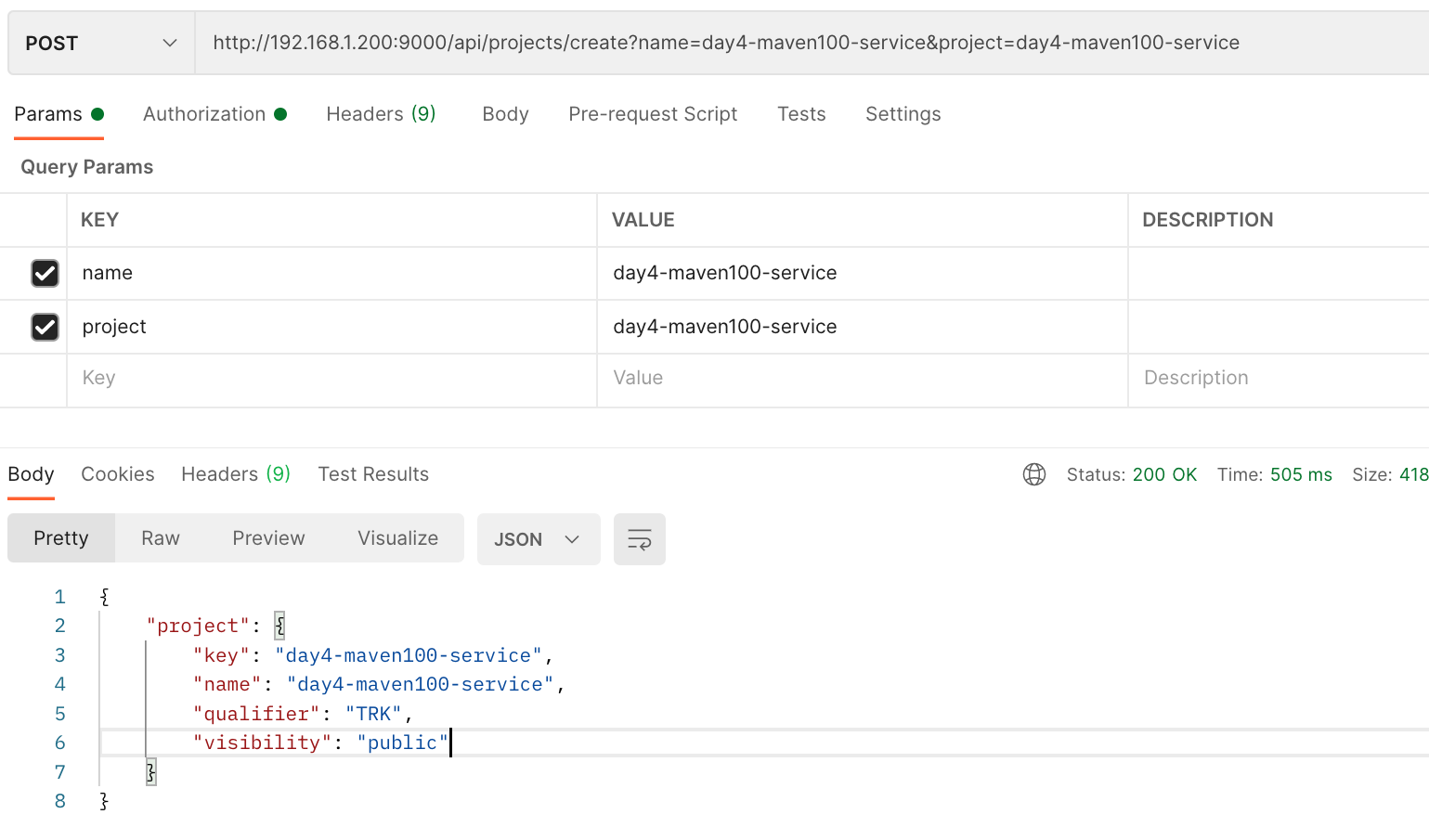
}

**1.5.2 创建项目**

接口地址和参数: <http://192.168.1.200:9000/api/projects/create?name=day4-maven4-service&project=day4-maven4-service>

请求类型：POST

postman调试：



cURL

curl --location --request POST 'http://192.168.1.200:9000/api/projects/create?name=day4-maven100-service&project=day4-maven100-service' \

--header 'Authorization: Basic YWRtaW46YWRtaW4xMjM0'

Jenkins Pipeline

// 创建项目

def CreateProject(projectName){

apiUrl = "projects/create?name=${projectName}&project=${projectName}"

response = SonarRequest(apiUrl,"POST")

try{

if (response.project.key == projectName ) {

println("Project Create success!...")

return true

}

}catch(e){

println(response.errors)

return false

}

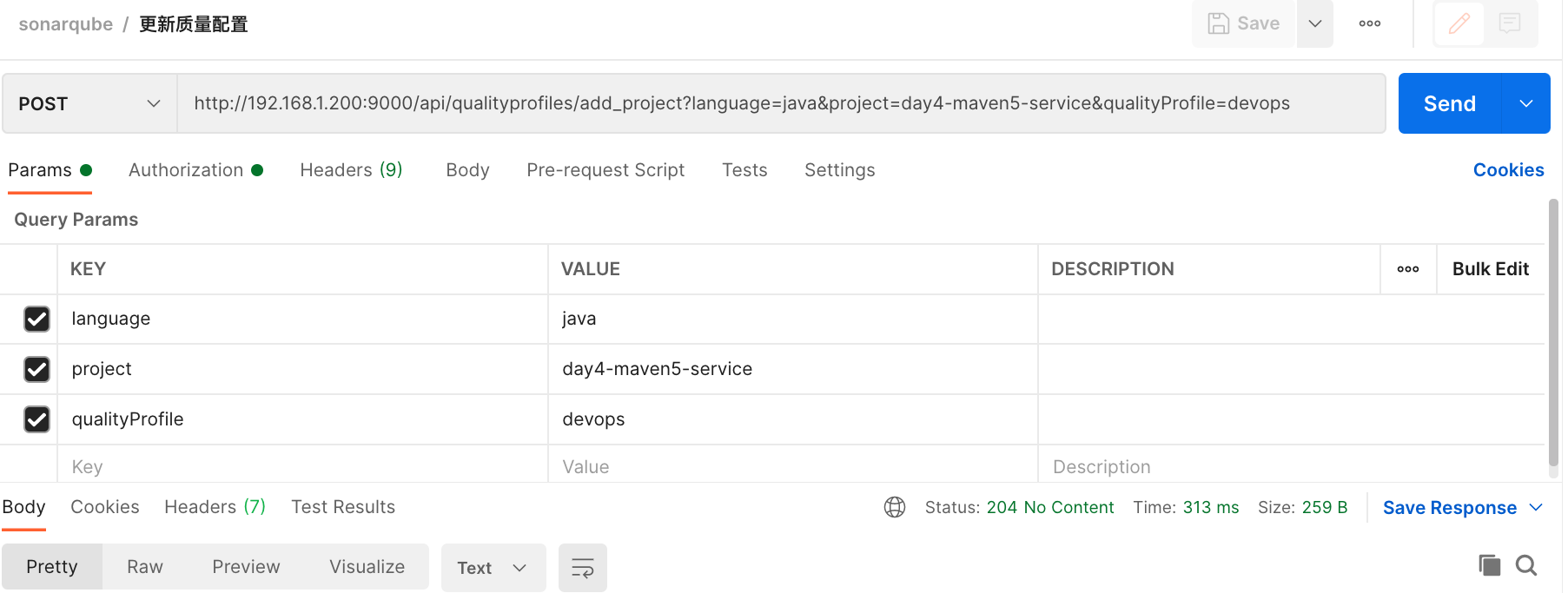
}

**1.5.3 更新项目质量配置**

接口地址和参数: <http://192.168.1.200:9000/api/qualityprofiles/add_project?language=java&project=day4-maven5-service&qualityProfile=devop>

请求类型：POST

postman调试：



cURL

curl --location --request POST 'http://192.168.1.200:9000/api/qualityprofiles/add\_project?language=java&project=day4-maven5-service&qualityProfile=devops' \

--header 'Authorization: Basic YWRtaW46YWRtaW4xMjM0'

Jenkins Pipeline

// 更新质量阈

def UpdateQualityProfiles(lang, projectName, profileName){

apiUrl = "qualityprofiles/add\_project?language=${lang}&project=${projectName}&qualityProfile=${profileName}"

response = SonarRequest(apiUrl,"POST")

if (response.errors != true){

println("ERROR: UpdateQualityProfiles ${response.errors}...")

return false

} else {

println("SUCCESS: UpdateQualityProfiles ${lang} > ${projectName} > ${profileName}" )

return true

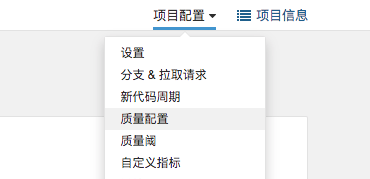
}

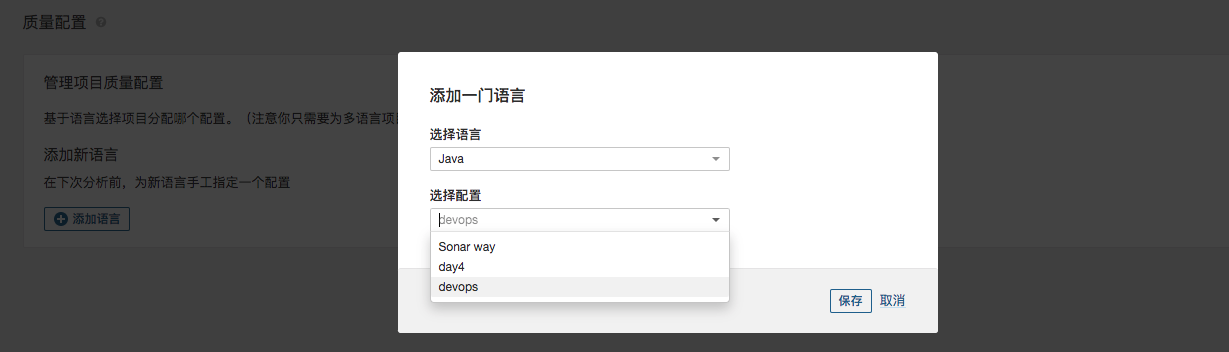
}

**1.5.4 控制逻辑(实践)**

* 项目没有配置质量， 默认使用sonarway（内置质量）。
* 默认直接使用sonarscanner，扫描的项目，使用的内置的默认质量。

如果具有单独的质量配置，例如每个组织一个质量配置。 此时就需要先手动在web页面创建一个空的项目，然后在项目的配置中设置目标质量配置。(下面是手动的操作步骤)





自动化实现方式 ：

* 查询项目是否存在
  + 存在： 直接更新质量配置
  + 不存在： 创建空的项目然后更新质量配置

stage("sonartest"){

steps{

script{

// 判断项目是否存在

sonarProjectName = "${JOB\_NAME.split('/')[-1]}"

result = ProjectSearch(sonarProjectName)

println(result)

if (result != true){

println("Create Sonar Project!...")

CreateProject(sonarProjectName)

}

// 指定项目的配置

UpdateQualityProfiles("java", sonarProjectName, "${sonarProjectName.split('-')[0]}")

}

}

}

完整的jenkinsfile代码

pipeline {

agent {

label "build"

}

stages{

stage("sonartest"){

steps{

script{

// 判断项目是否存在

sonarProjectName = "${JOB\_NAME.split('/')[-1]}"

result = ProjectSearch(sonarProjectName)

println(result)

if (result != true){

println("Create Sonar Project!...")

CreateProject(sonarProjectName)

}

// 指定项目的配置

UpdateQualityProfiles("java", sonarProjectName, "${sonarProjectName.split('-')[0]}")

}

}

}

}

}

def SonarRequest(apiUrl,method){

withCredentials([string(credentialsId: "52df4ad9-7167-4bf6-a1fc-2f9f17713472", variable: 'SONAR\_TOKEN')]) {

sonarApi = "http://192.168.1.200:9000/api"

response = sh returnStdout: true,

script: """

curl --location \

--request ${method} \

"${sonarApi}/${apiUrl}" \

--header "Authorization: Basic ${SONAR\_TOKEN}"

"""

try {

response = readJSON text: """ ${response - "\n"} """

} catch(e){

response = readJSON text: """{"errors" : true}"""

}

return response

}

}

// 更新质量阈

def UpdateQualityProfiles(lang, projectName, profileName){

apiUrl = "qualityprofiles/add\_project?language=${lang}&project=${projectName}&qualityProfile=${profileName}"

response = SonarRequest(apiUrl,"POST")

if (response.errors != true){

println("ERROR: UpdateQualityProfiles ${response.errors}...")

return false

} else {

println("SUCCESS: UpdateQualityProfiles ${lang} > ${projectName} > ${profileName}" )

return true

}

}

// 创建项目

def CreateProject(projectName){

apiUrl = "projects/create?name=${projectName}&project=${projectName}"

response = SonarRequest(apiUrl,"POST")

try{

if (response.project.key == projectName ) {

println("Project Create success!...")

return true

}

}catch(e){

println(response.errors)

return false

}

}

// 查找项目

def ProjectSearch(projectName){

apiUrl = "projects/search?projects=${projectName}"

response = SonarRequest(apiUrl,"GET")

if (response.paging.total == 0){

println("Project not found!.....")

return false

}

return true

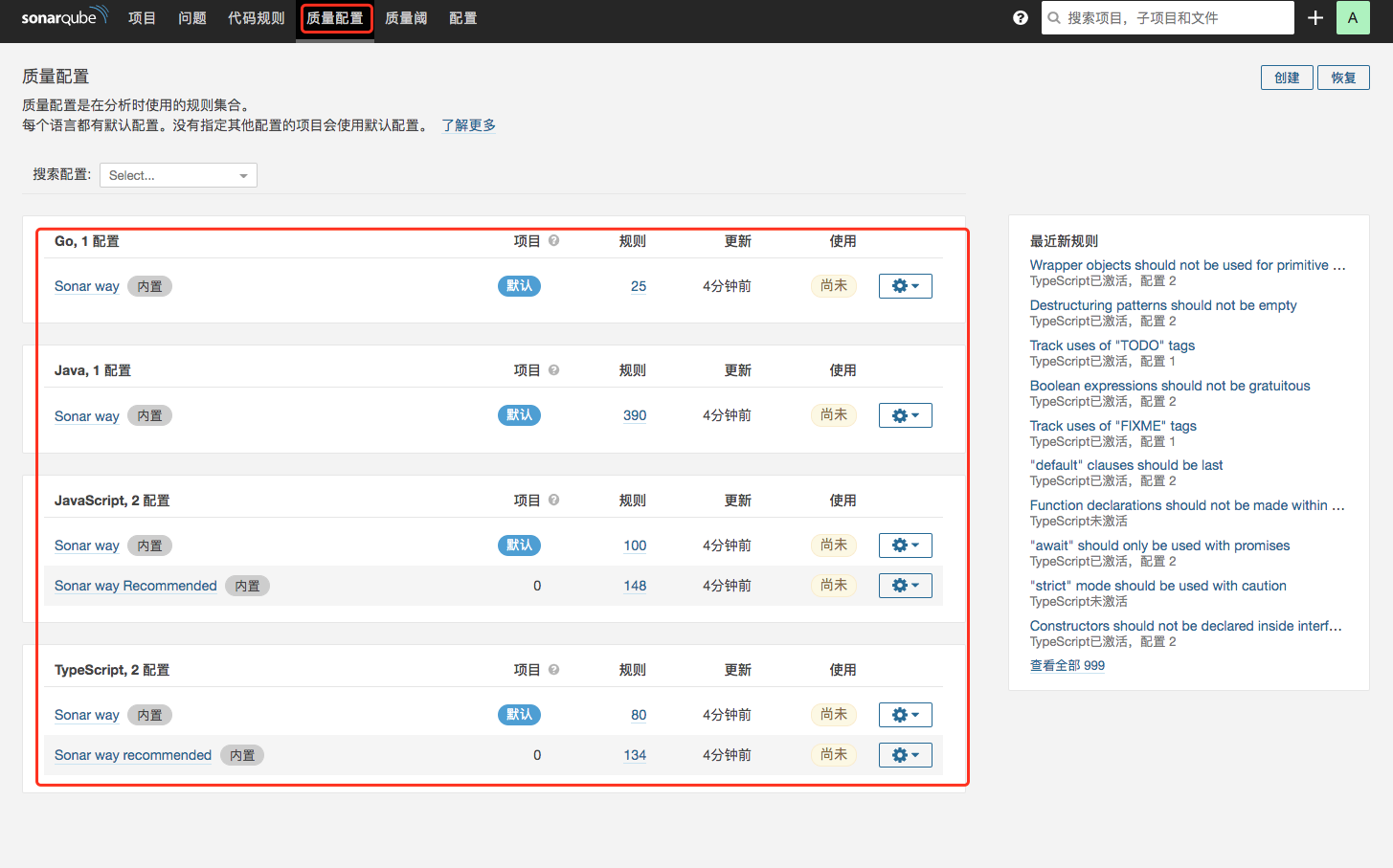
}

**1.6 其他日常使用问题**

**1.6.1 规则的禁用与启用**

目的： 掌握默认规则中的一部分规则如何激活和禁用。

进入质量配置页面， 可以看到所有的语言规则配置。在这里可以看到规则的使用情况。



这里假设我要调整Go语言的规则配置， 点击规则数量数字。





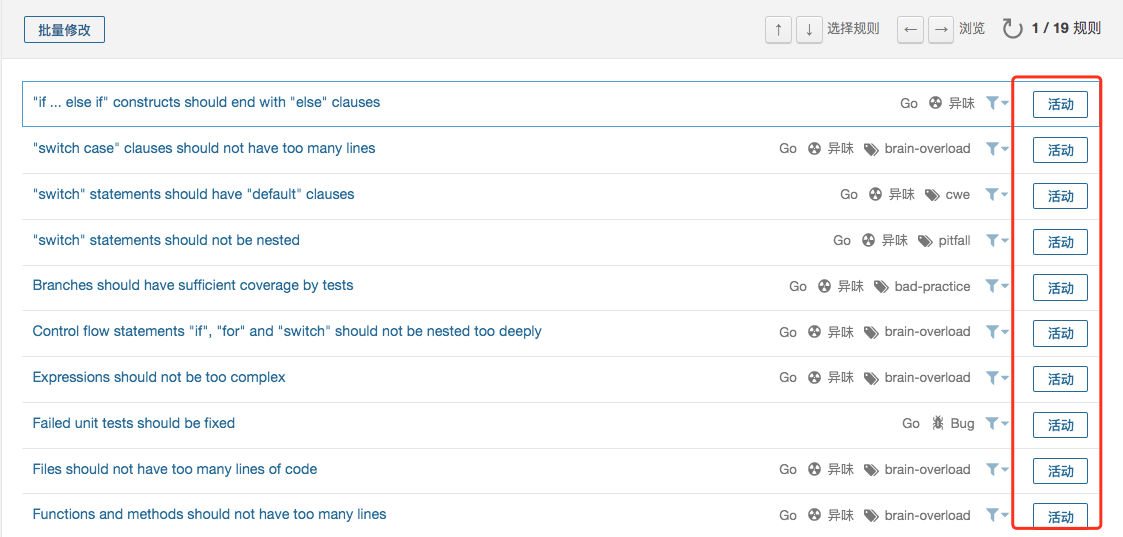
创建新的规则集



点击更多激活规则， 进入规则设置页面。



激活或者下线规则。（活动/挂起）



使用规则： 先在页面配置项目，然后使用SonarScanner扫描。

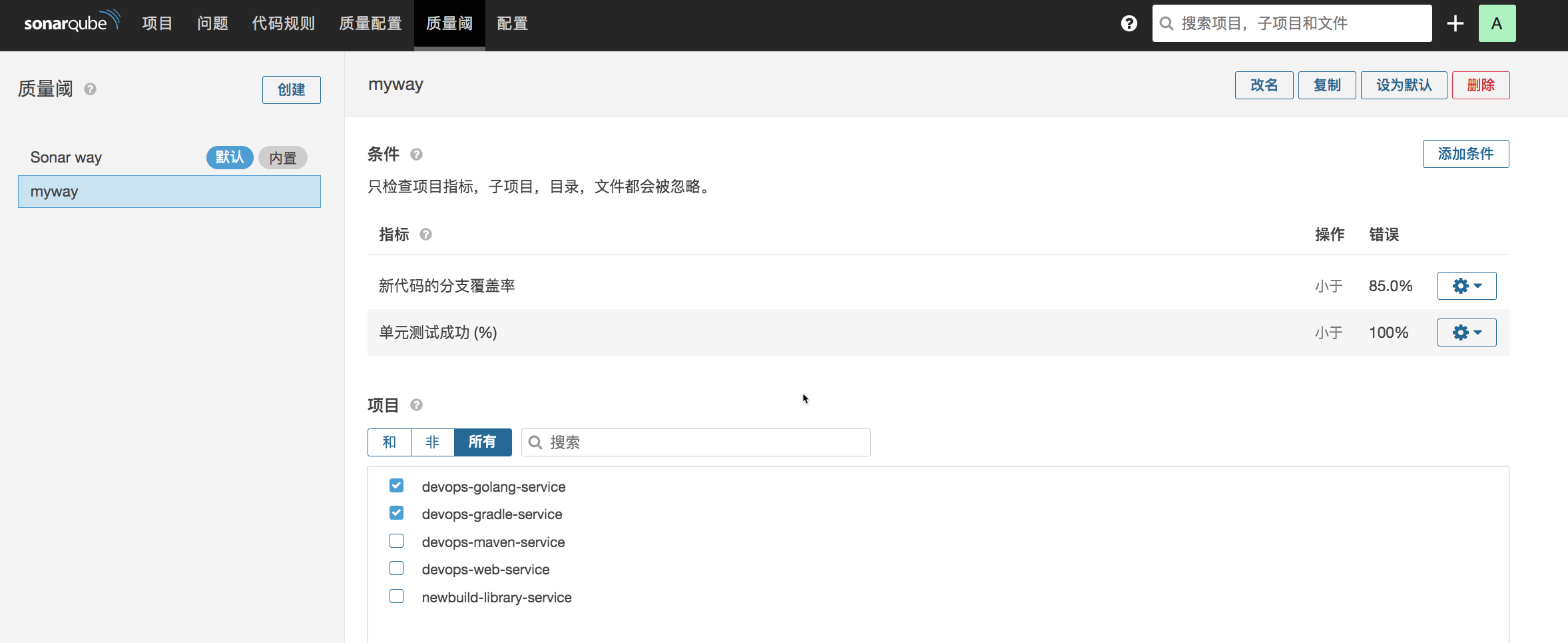




**1.6.2 质量阈的配置**

目的： 适用于以质量门禁作为交付关卡。





**1.6.3 代码覆盖率统计**

找一个具有大量单元测试的项目， 然后集成jacoco插件，生成覆盖率报告，最后由sonar收集。

**Maven集成Jacoco**

添加jacoco-maven-plugin 和junit插件。

<dependencies>

<dependency>

<groupId>org.jacoco</groupId>

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

<version>0.8.2</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.12</version>

<scope>test</scope>

</dependency>

</dependencies>

添加插件

<plugin>

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

<artifactId>maven-compiler-plugin</artifactId>

<version>3.6.1</version>

<configuration>

<skipMain>true</skipMain>

<skip>true</skip>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

<groupId>org.jacoco</groupId>

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

<version>0.7.5.201505241946</version>

<executions>

<execution>

<id>prepare-agent</id>

<goals>

<goal>prepare-agent</goal>

</goals>

</execution>

<execution>

<id>report</id>

<phase>prepare-package</phase>

<goals>

<goal>report</goal>

</goals>

</execution>

<execution>

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

<phase>test</phase>

<goals>

<goal>report</goal>

</goals>

<configuration>

<dataFile>target/jacoco.exec</dataFile>

<outputDirectory>target/jacoco-reports</outputDirectory>

</configuration>

</execution>

</executions>

<configuration>

<systemPropertyVariables>

<jacoco-agent.destfile>target/jacoco.exec</jacoco-agent.destfile>

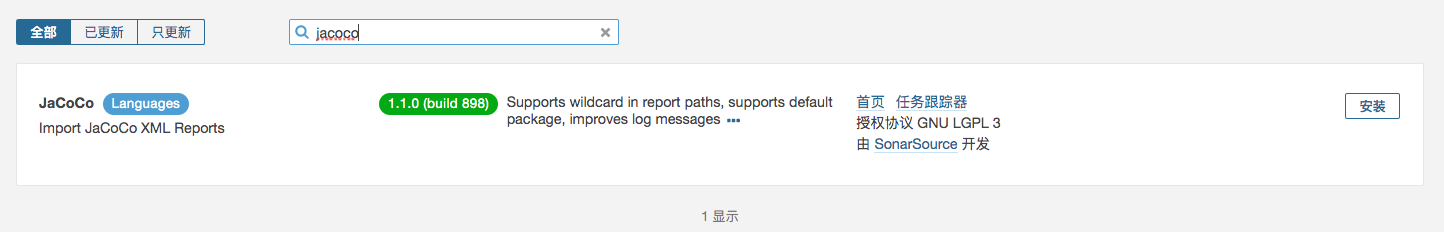
</systemPropertyVariables>

</configuration>

</plugin>

SonarQube安装Jacoco插件（**8.9.1 版本可以跳过，已经集成**）

<https://github.com/SonarSource/sonar-jacoco/releases/download/1.1.0.898/sonar-jacoco-plugin-1.1.0.898.jar>



# 指定代码覆盖率工具为jacoco

sonar.core.codeCoveragePlugin=jacoco

# 指定exec二进制文件存放路径

sonar.jacoco.reportPaths=target/jacoco.exec

cd devops-jacoco-service/

sonar-scanner -Dsonar.host.url=http://192.168.1.200:9000 \

-Dsonar.projectKey=devops-jacoco-service \

-Dsonar.projectName=devops-jacoco-service \

-Dsonar.projectVersion=1.0 \

-Dsonar.login=admin \

-Dsonar.password=admin \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=http://www.baidu.com \

-Dsonar.sources=src \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports \

-Dsonar.core.codeCoveragePlugin=jacoco \

-Dsonar.jacoco.reportPaths=target/jacoco.exec

**1.6.4 多分支代码扫描**

<https://github.com/mc1arke/sonarqube-community-branch-plugin/releases>



**1.6.4.1 Sonar 7.9.6版本**

将插件放到extensions/plugins 和 lib/common目录中，然后重启sonar。

## 临时方案

docker exec -it sonarqube bash

cd /opt/sonarqube/lib/common

cp ../../extensions/plugins/sonarqube-community-branch-plugin-1.3.2 ./

exit

docker restart sonarqube

## 持久化lib目录后

[root@zeyang-nuc-service sonarqube]# ls

sonarqube\_conf sonarqube\_data sonarqube\_extensions sonarqube\_lib sonarqube\_logs

[root@zeyang-nuc-service sonarqube]# cp /root/sonarqube-community-branch-plugin-1.3.2.jar sonarqube\_extensions/plugins/

[root@zeyang-nuc-service sonarqube]# chmod +x sonarqube\_extensions/plugins/sonarqube-community-branch-plugin-1.3.2.jar

[root@zeyang-nuc-service sonarqube]#

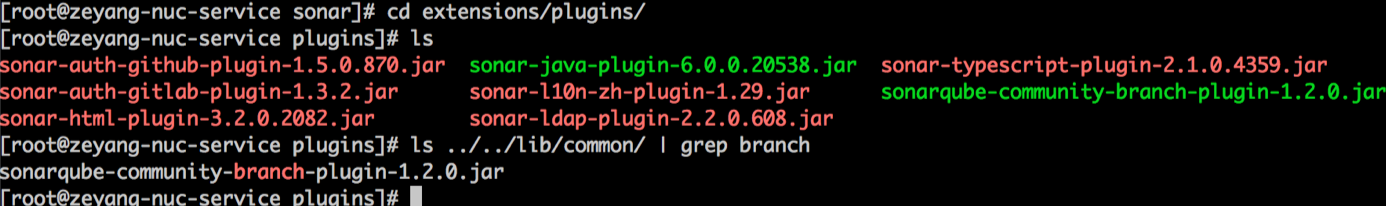
[root@zeyang-nuc-service sonarqube]#

[root@zeyang-nuc-service sonarqube]# cp /root/sonarqube-community-branch-plugin-1.3.2.jar sonarqube\_lib/common/

[root@zeyang-nuc-service sonarqube]# chmod +x sonarqube\_lib/common/sonarqube-community-branch-plugin-1.3.2.jar

[root@zeyang-nuc-service sonarqube]#

docker restart sonarqube



##扫描参数增加 –Dsonar.branch.name=

sonar-scanner -Dsonar.host.url=http://192.168.1.200:9000 \

-Dsonar.projectKey=devops-maven2-service \

-Dsonar.projectName=devops-maven2-service \

-Dsonar.projectVersion=1.0 \

-Dsonar.login=admin \

-Dsonar.password=admin \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=http://192.168.1.200/devops/devops-maven-service \

-Dsonar.links.ci=http://192.168.1.200:8080/job/demo-pipeline-service/ \

-Dsonar.sources=src \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports \

-Dsonar.branch.name=release-1.1.1

FAQ: 需要先把主分支扫描一遍，不然会报错。

ERROR: Error during SonarScanner execution

ERROR: No branches currently exist in this project. Please scan the main branch without passing any branch parameters.

ERROR:

ERROR: Re-run SonarScanner using the -X switch to enable full debug logging.

**1.6.4.2 Sonar 8.9.1 版本**

新版本插件的配置有变化，效果和使用方式不变。

1. 将插件下载到extensions/plugins/目录。
2. 更新sonar服务端的配置文件。
3. 重启docker restart sonarqube 。

# cd /data/cicd2/sonarqube/

# ls

sonarqube\_conf sonarqube\_data sonarqube\_extensions sonarqube\_logs

# cat sonarqube\_conf/sonar.properties

sonar.web.javaAdditionalOpts=-javaagent:./extensions/plugins/sonarqube-community-branch-plugin-1.8.0.jar=web

sonar.ce.javaAdditionalOpts=-javaagent:./extensions/plugins/sonarqube-community-branch-plugin-1.8.0.jar=ce

# ls sonarqube\_extensions/plugins/

sonar-gitlab-plugin-4.1.0-SNAPSHOT.jar sonar-l10n-zh-plugin-8.9.jar sonarqube-community-branch-plugin-1.8.0.jar

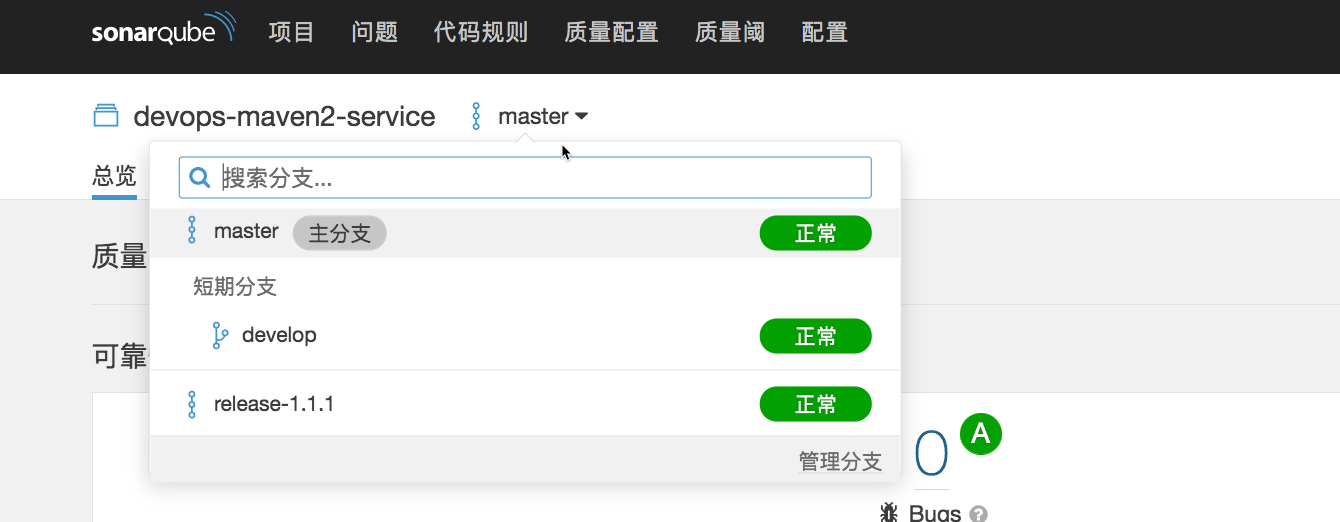
原文档描述：

1. Copy the plugin JAR file to the extensions/plugins/ directory of your SonarQube instance

2. Add -javaagent:./extensions/plugins/sonarqube-community-branch-plugin-${version}.jar=web to the sonar.web.javaAdditionalOptions property in your Sonarqube installation's config/sonar.properties file, e.g. sonar.web.javaAdditionalOpts=-javaagent:./extensions/plugins/sonarqube-community-branch-plugin-1.8.0.jar=web

3. Add -javaagent:./extensions/plugins/sonarqube-community-branch-plugin-${version}.jar=ce to the sonar.ce.javaAdditionalOptions property in your Sonarqube installation's config/sonar.properties file, e.g. sonar.ce.javaAdditionalOpts=-javaagent:./extensions/plugins/sonarqube-community-branch-plugin-1.8.0.jar=ce

4. Start Sonarqube, and accept the warning about using third-party plugins



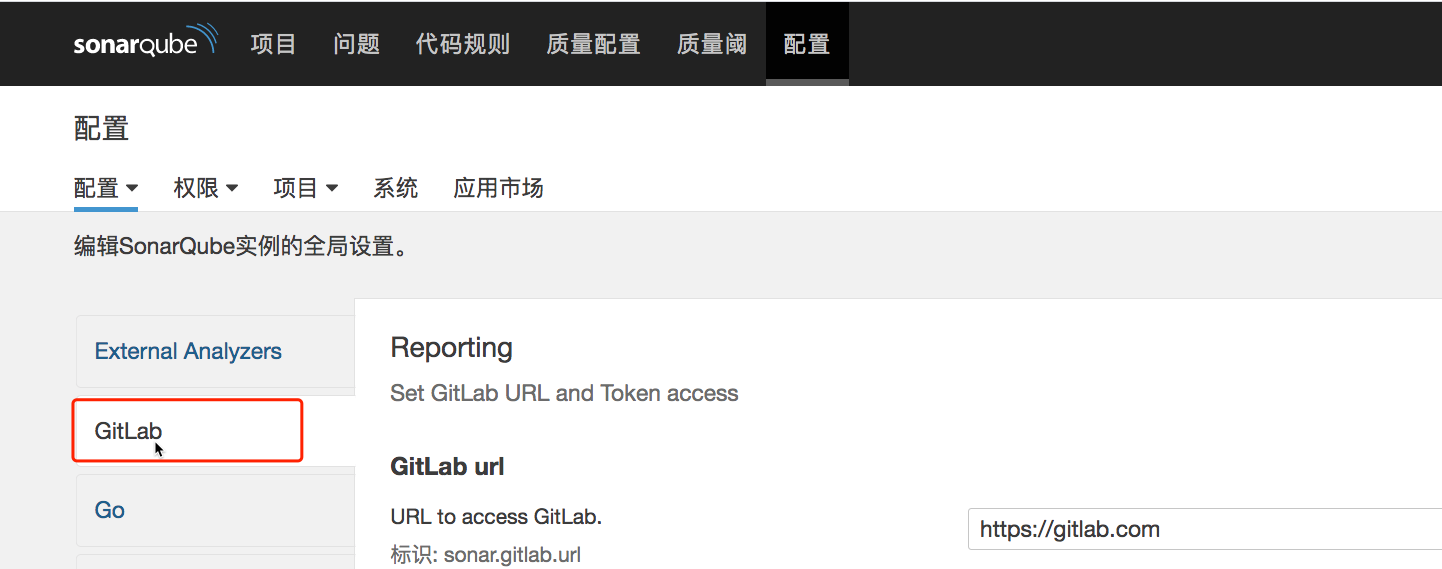
**1.6.5 扫描结果关联commitid**

提前装好插件：<https://github.com/gabrie-allaigre/sonar-gitlab-plugin/tree/4.1.0-SNAPSHOT>插件的说明文档查看该插件的Readme文档。

# cp sonar-gitlab-plugin-4.1.0-SNAPSHOT.jar /data/cicd/sonarqube/sonarqube\_extensions/plugins/

# chmod +x /data/cicd/sonarqube/sonarqube\_extensions/plugins/sonar-gitlab-plugin-4.1.0-SNAPSHOT.jar

# docker restart sonarqube



-Dsonar.gitlab.failure\_notification\_mode 值为commit-status表示更改提交状态， 值为nothing不做任何动作。

sonar-scanner -Dsonar.host.url=http://192.168.1.200:9000 \

-Dsonar.projectKey=devops-jacoco-service \

-Dsonar.projectName=devops-jacoco-service \

-Dsonar.projectVersion=1.0 \

-Dsonar.login=0809881d71f2b06b64786ae3f81a9acf22078e8b \

-Dsonar.ws.timeout=30 \

-Dsonar.projectDescription="my first project!" \

-Dsonar.links.homepage=http://www.baidu.com \

-Dsonar.sources=src \

-Dsonar.sourceEncoding=UTF-8 \

-Dsonar.java.binaries=target/classes \

-Dsonar.java.test.binaries=target/test-classes \

-Dsonar.java.surefire.report=target/surefire-reports \

-Dsonar.core.codeCoveragePlugin=jacoco \

-Dsonar.jacoco.reportPaths=coverage/jacoco.exec \

-Dsonar.gitlab.commit\_sha=1632c729e8f78f913cbf0925baa2a8c893e4473b \

-Dsonar.gitlab.ref\_name=master \

-Dsonar.gitlab.project\_id=16 \

-Dsonar.dynamicAnalysis=reuseReports \

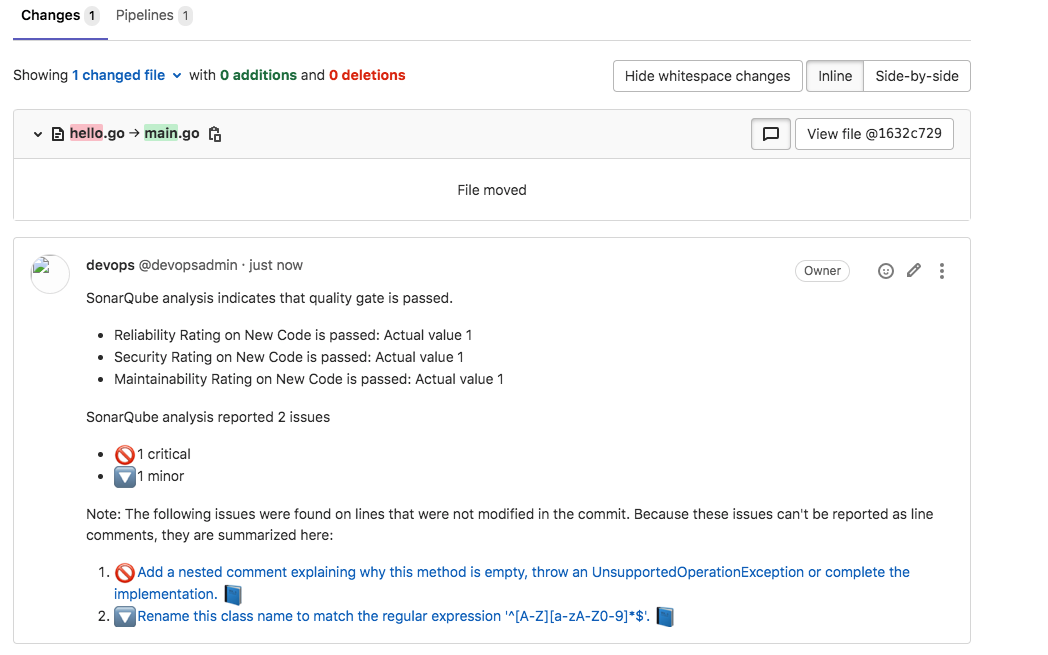
-Dsonar.gitlab.failure\_notification\_mode=commit-status \

-Dsonar.gitlab.url=http://192.168.1.200 \

-Dsonar.gitlab.user\_token=k8xLe6dYTzdtoewSysmy \

-Dsonar.gitlab.api\_version=v4

效果



**1.6.6 控制代码扫描步骤运行**

stage("SonarScan"){

when {

environment name: 'skipSonar', value: 'false'

}

}

