代码质量监控平台

**简述：**

Gitlab cicd 结合sonarqube构建代码质量监控平台

**实验环境：**

Gitlab11.1.0

GitRunner

Sonarqube7.4

Mysql5.7

Jdk 8

Maven3.6.0

# 1.基础环境准备

## 1.1 Maven+jdk8+mysql安装步骤省略

## 1.2 Mysql用于sonarqube 存放质量监控数据

或使用现有的mysql数据创建一个sonar账号即可，创建语法：

> CREATE DATABASE sonar CHARACTER SET utf8 COLLATE utf8\_general\_ci;> grant all privileges on sonar.\* to "sonar"@"%" identified by "sonar";> flush privileges;

另sonarqube7.9以上版本不在支持mysql,支持es和postgresql、msSql

# 2.Gitlab安装

参考《Gitlab安装手册.doc》

# 3.Sonarqube安装

## 3.1 创建sonar用户

useradd sonar;echo "123456"|passwd --stdin sonar

*tips: 内置es不能以root用户启动*

## 3.2 安装sonarqube

wget <https://binaries.sonarsource.com/Distribution/sonarqube/sonarqube-7.4.zip>

unzip sonarqube-7.4.zip

chown -R sonar:sonar sonarqube-7.4/

## 3.3 修改系统配置，内置es需要

echo -e "\* soft nofile 65536\n\* hard nofile 131072\n\* soft nproc 2048\n\* hard nproc 4096\n" >>/etc/security/limits.conf

echo "vm.max\_map\_count=655360" >>/etc/sysctl.conf

sysctl -p

## 3.4 配置sonarqube

vi sonarqube-7.4/conf/sonar.properties

#jdbc连接账号密码

sonar.jdbc.username=sonar

sonar.jdbc.password=sonar

# 数据连接

sonar.jdbc.url=jdbc:mysql://172.16.68.164:3306/sonar?useUnicode=true&characterEncoding=utf8&rewriteBatchedStatements=true&useConfigs=maxPerformance&useSSL=false

# web配置绑定IP和端口

sonar.web.host=172.16.68.164

sonar.web.port=9000

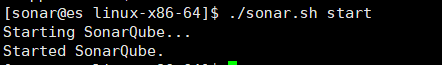
## 3.5 启动

su - sonar

*Tips: 一定要切换成sonar用户，内置的es不允许root启动*

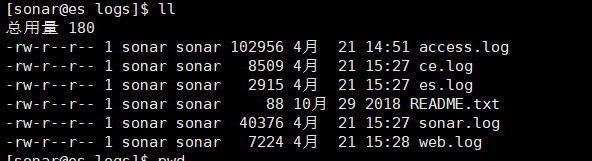
cd ${sonarqube\_home}/ bin/linux-x86-64/

./sonar.sh start

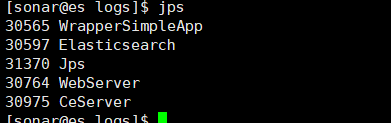


查看日志目录启动情况：

cd ${sonarqube\_home}/logs



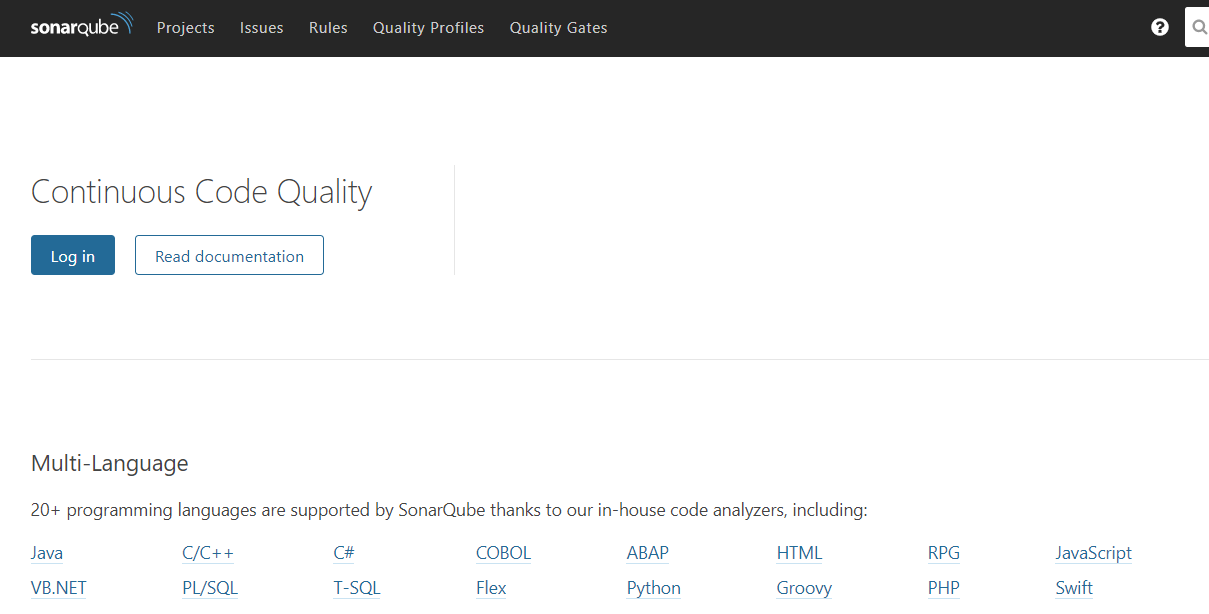
主要查看sonar.log和web.log的日志文件



## 3.6 启动验证

<http://172.16.68.164:9000/>

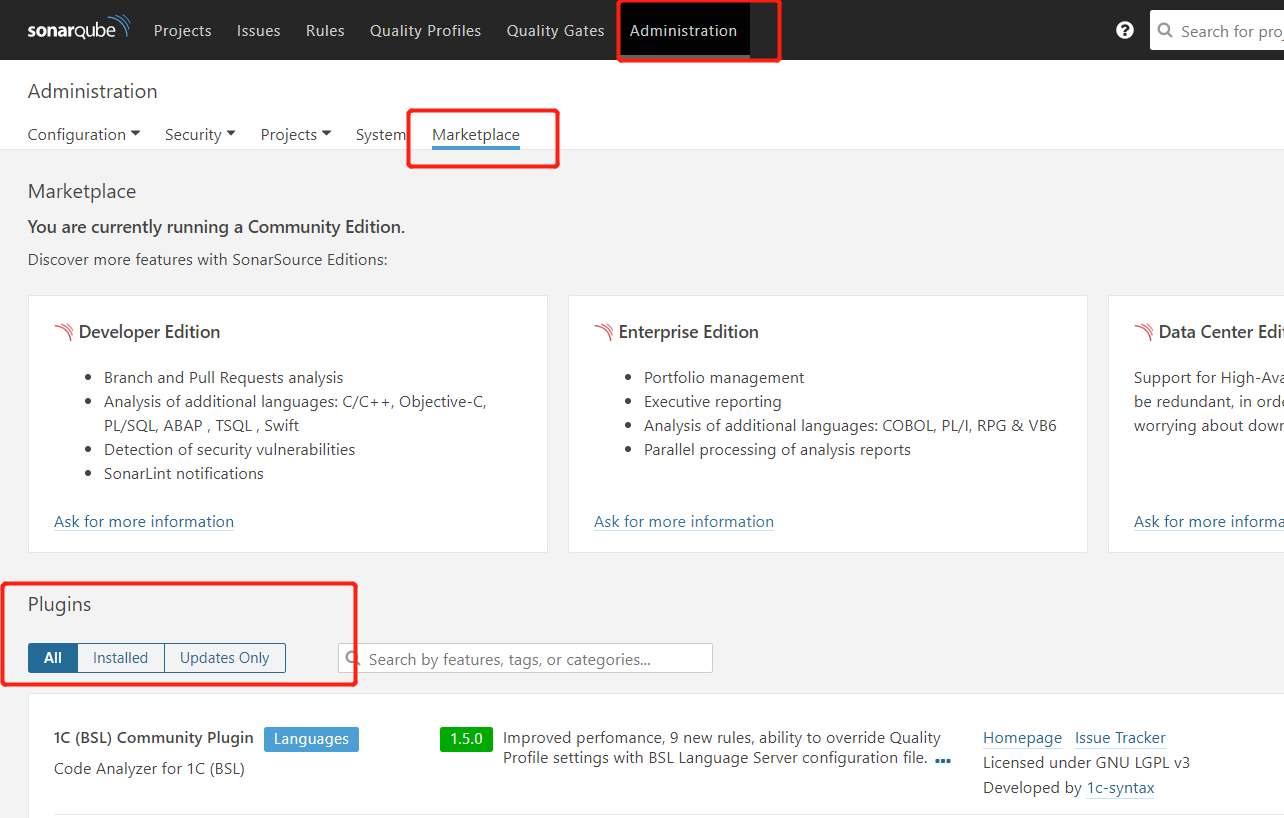
默认账号密码admin/admin



*Tips:如无法访问确定是否启动以及端口是否开放*

## 3.7 汉化以及插件安装

在线安装方式：administration--->Marketplace--->Plugins



版本兼容问题我们采取离线安装方式，需要安装的离线插件：

**A.离线安装一定要找到对应版本**

**B.下载离线安装包.jar文件放到${sonarqube}/ extensions/plugins目录下**

1)汉化包sonar-l10n-zh 1.24

https://github.com/SonarQubeCommunity/sonar-l10n-zh/releases

2)sonar-gitlab-plugin 4.0.0

https://github.com/gabrie-allaigre/sonar-gitlab-plugin/releases

3)pmd 3.0.1

https://github.com/jensgerdes/sonar-pmd/releases

4)findbugs 3.9.1

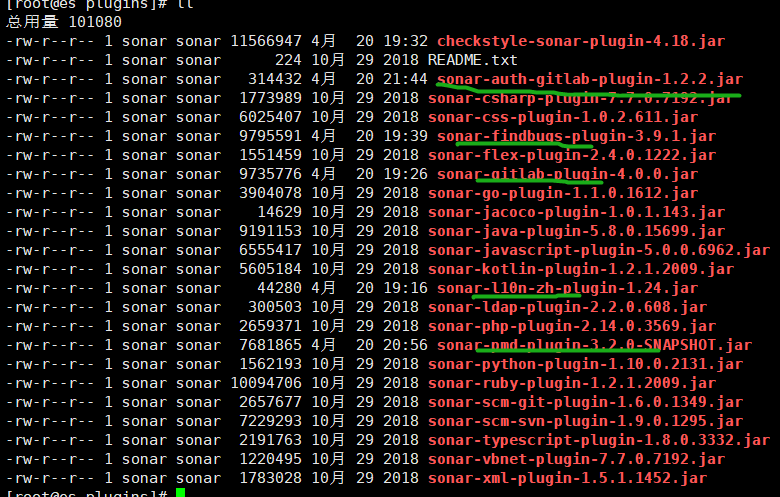
https://github.com/spotbugs/sonar-findbugs/releases

5)sonar-auth-gitlab-plugin 1.2.2

https://github.com/gabrie-allaigre/sonar-auth-gitlab-plugin/releases

6)p3c-pmd 3.2.0-SNAPSHOT

https://github.com/rhinoceros/sonar-p3c-pmd/releases

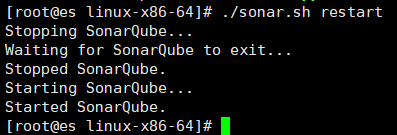


*Tips：请注意插件包的权限*

重启sonarqube

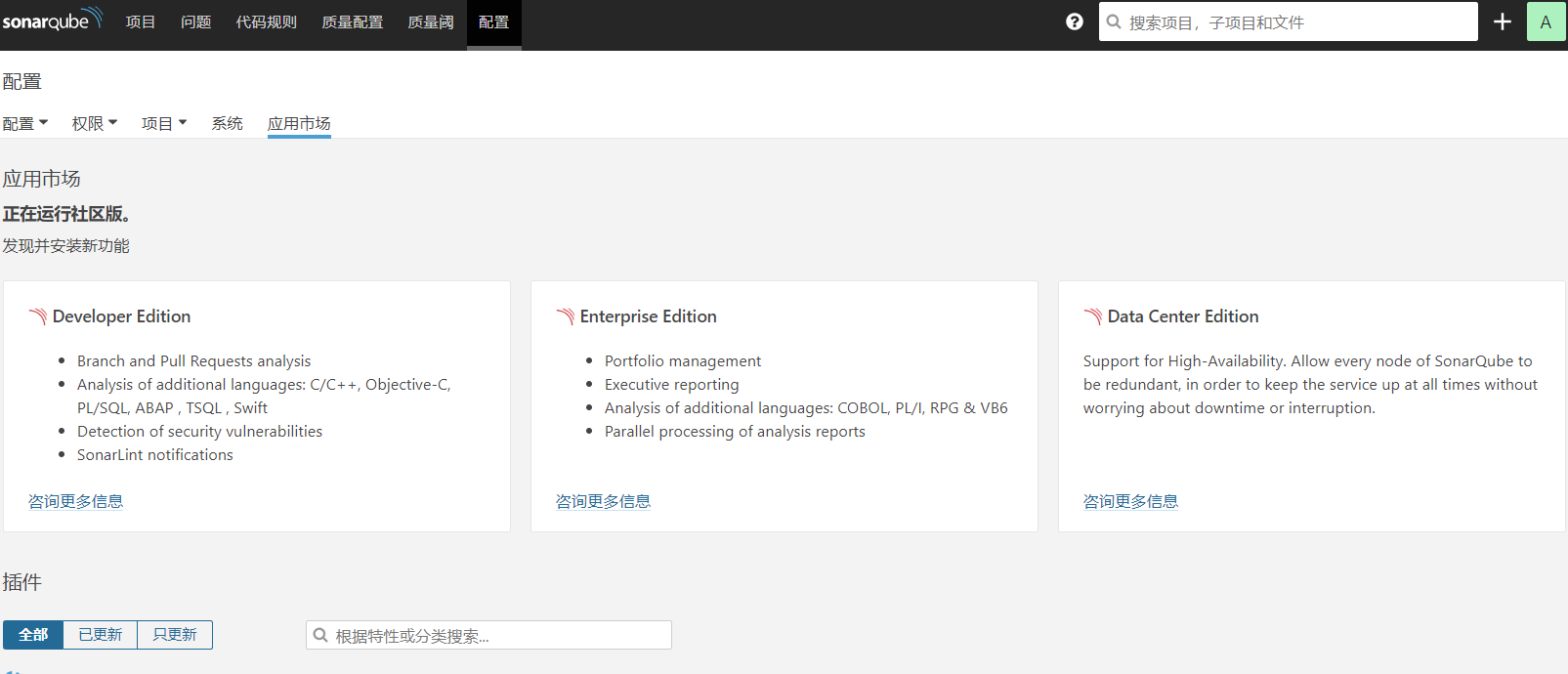
cd ${sonarqube\_home}/ bin/linux-x86-64/

./sonar.sh restart



验证插件安装：

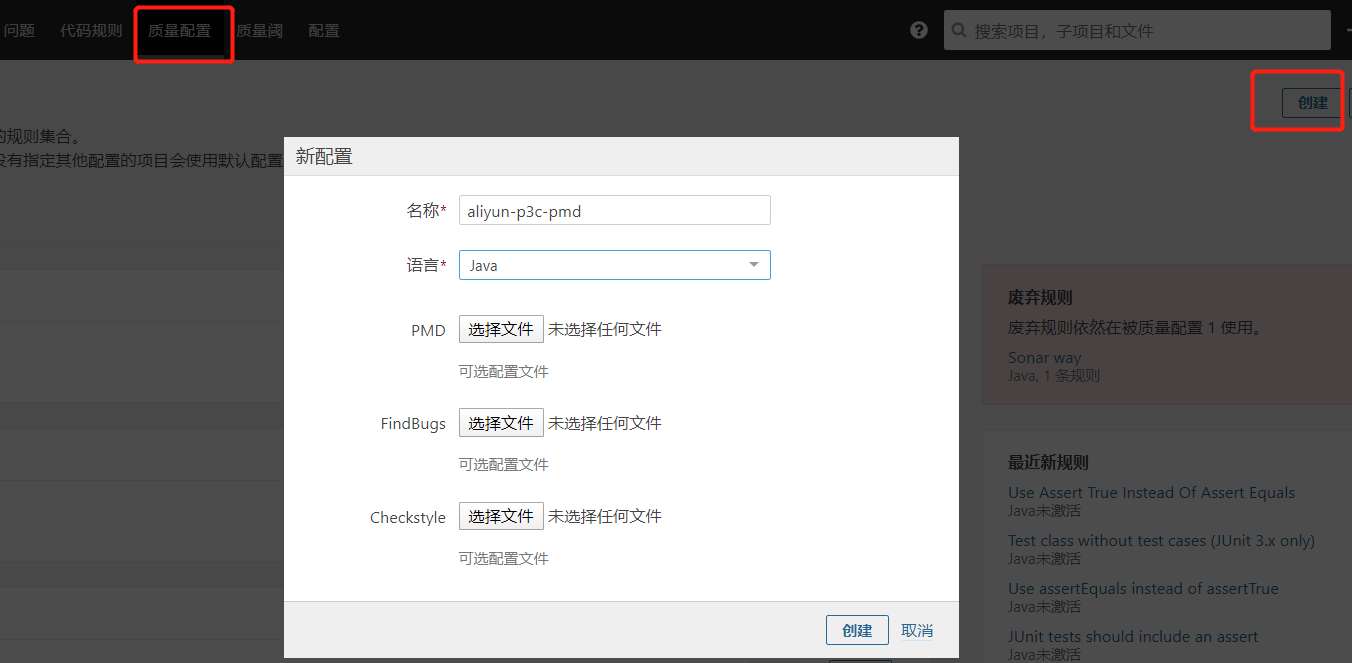
<http://172.16.68.164:9000/>



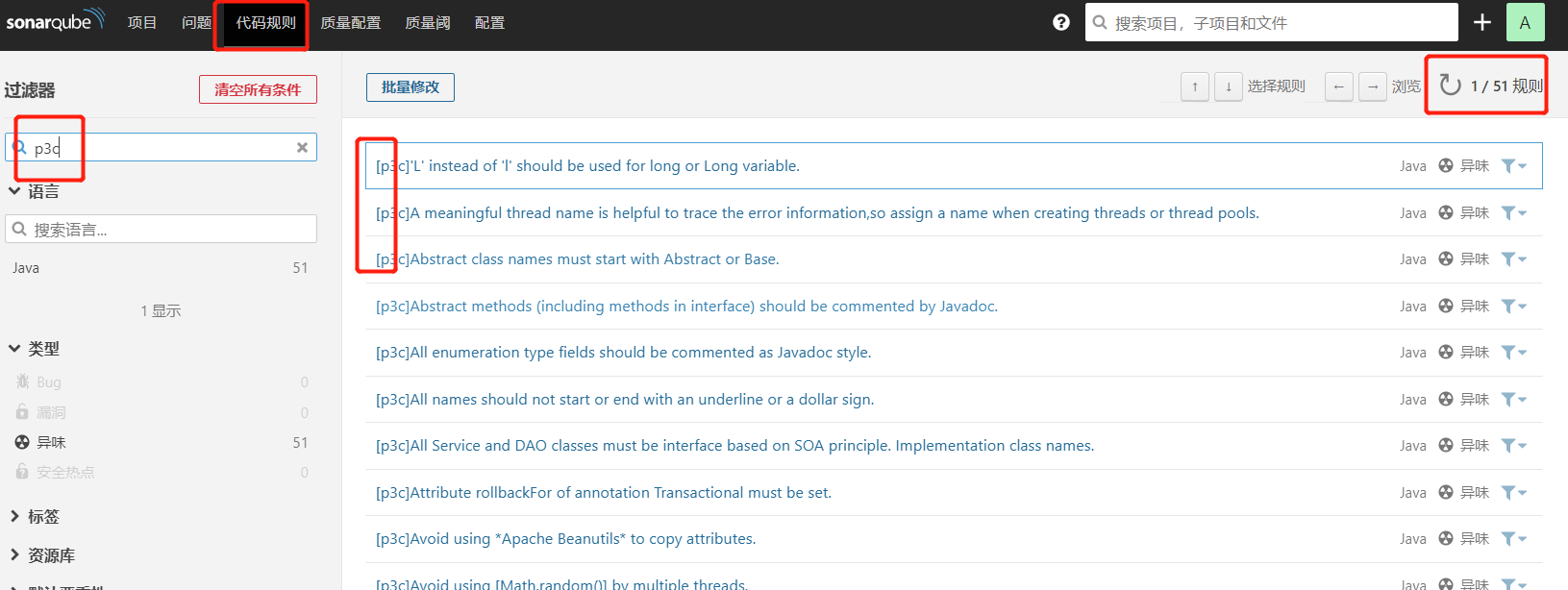
## 3.8 使用阿里云P3C质量检测规则

### 3.8.1 创建新的质量配置规则

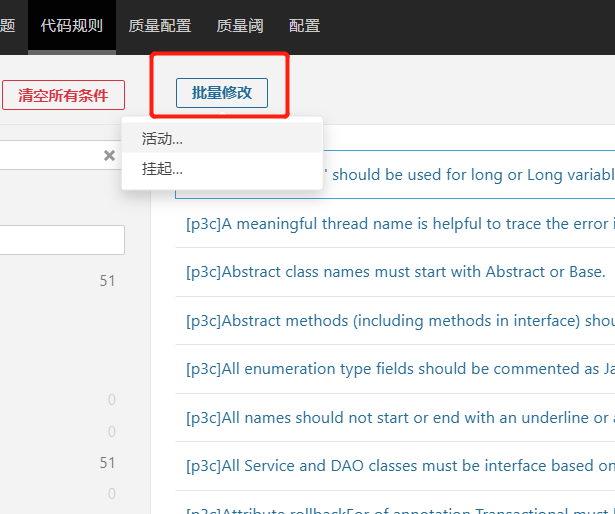
质量配置-创建

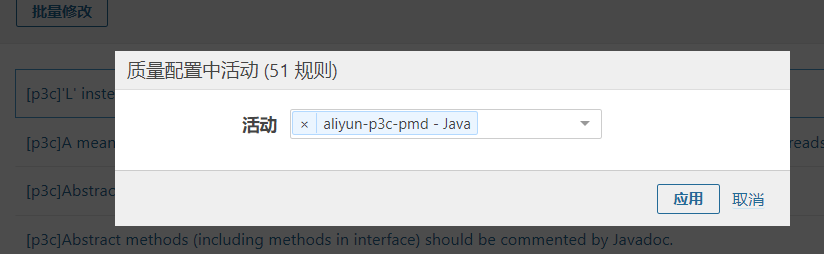
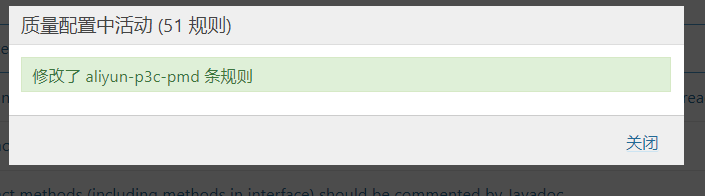


### 3.8.2 添加p3c的规则到新创建的配置中 51条规则

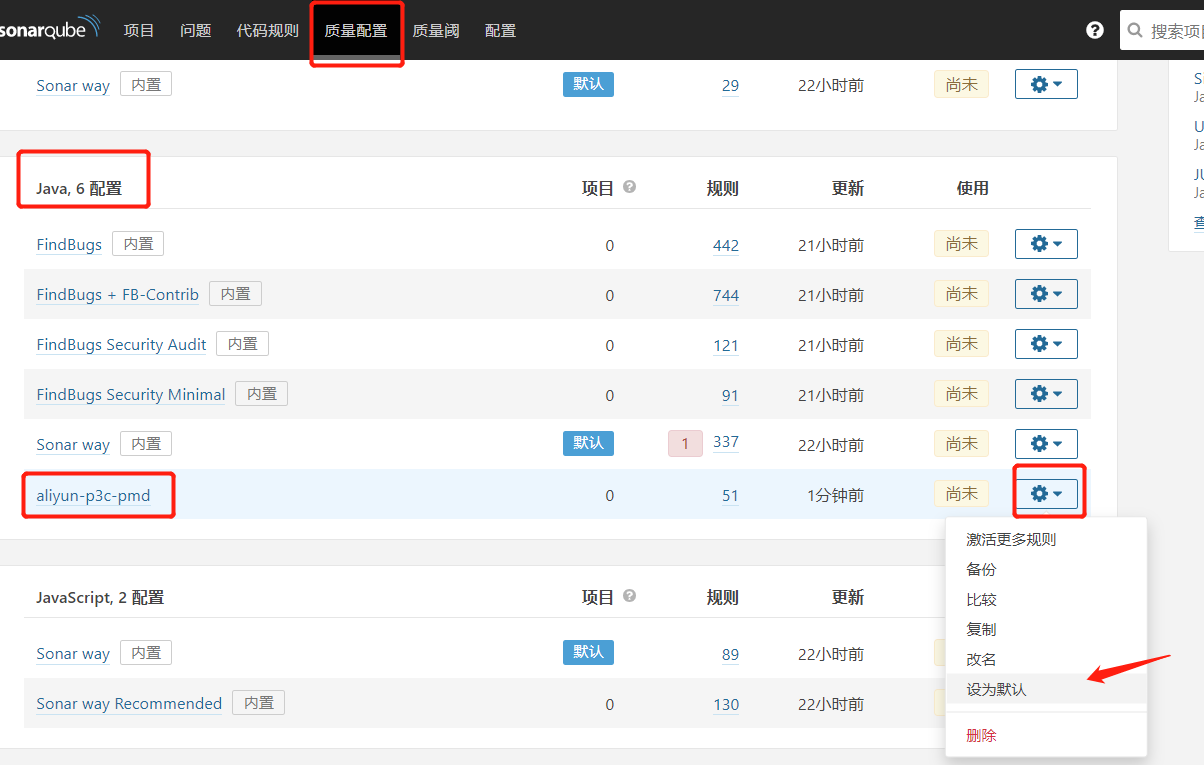


点击批量修--活动添加到新建的规则中



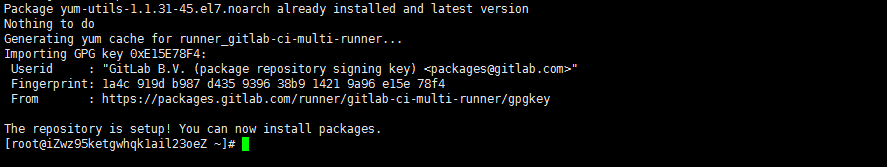
### 3.8.3 设置P3C为默认



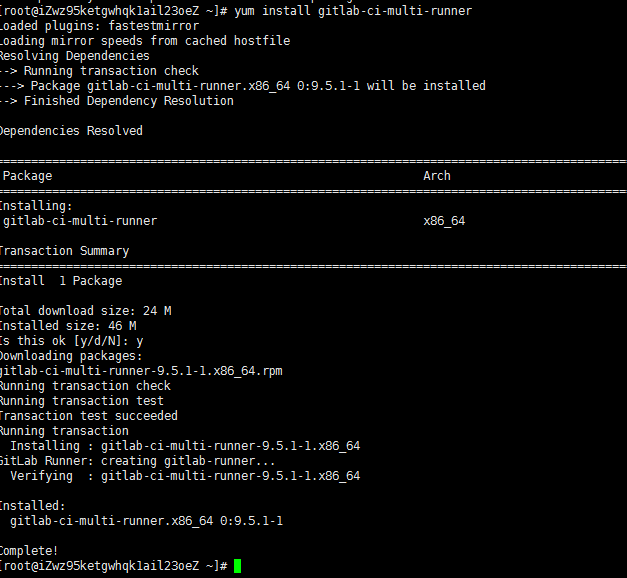
# 4.Git-runner安装

## 4.1 安装

[root@iZwz95ketgwhqk1ail23oeZ ~]# curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-ci-multi-runner/script.rpm.sh | sudo bash



[root@iZwz95ketgwhqk1ail23oeZ ~]# yum install gitlab-ci-multi-runner

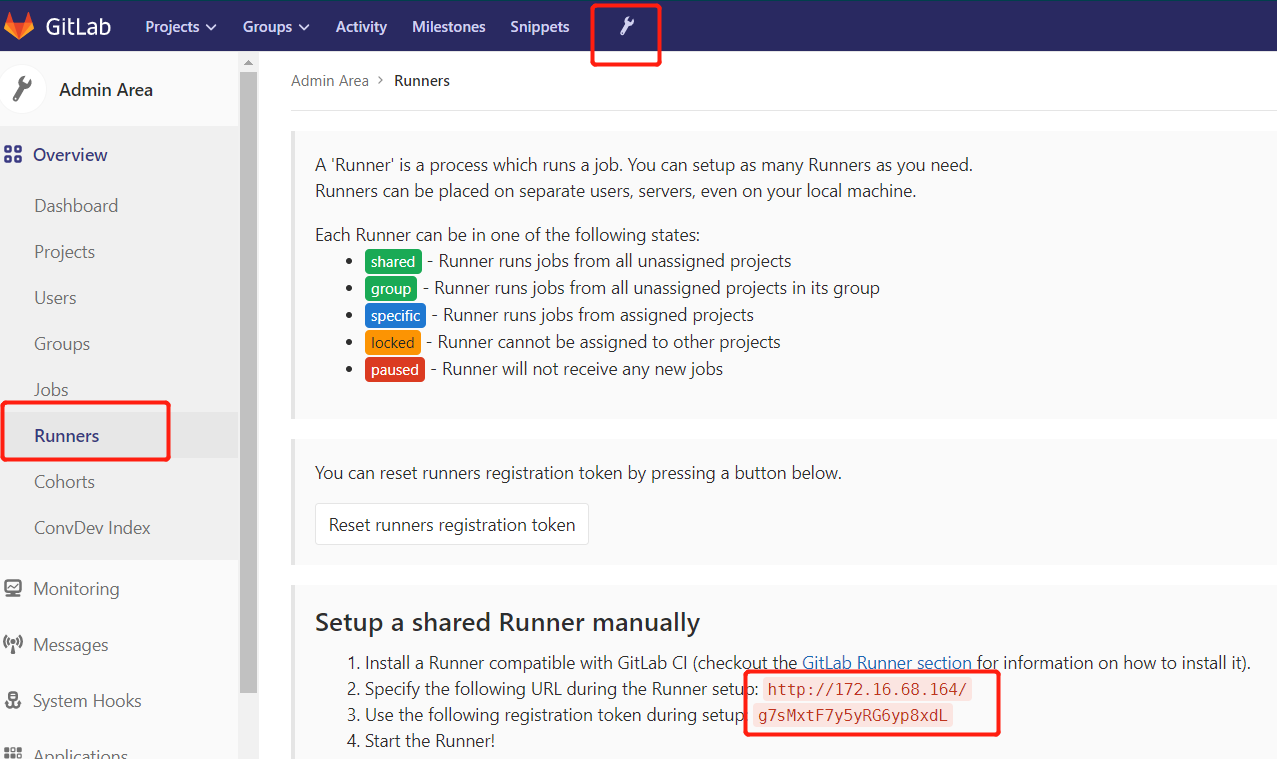


## 4.2 gitlab注册 runner

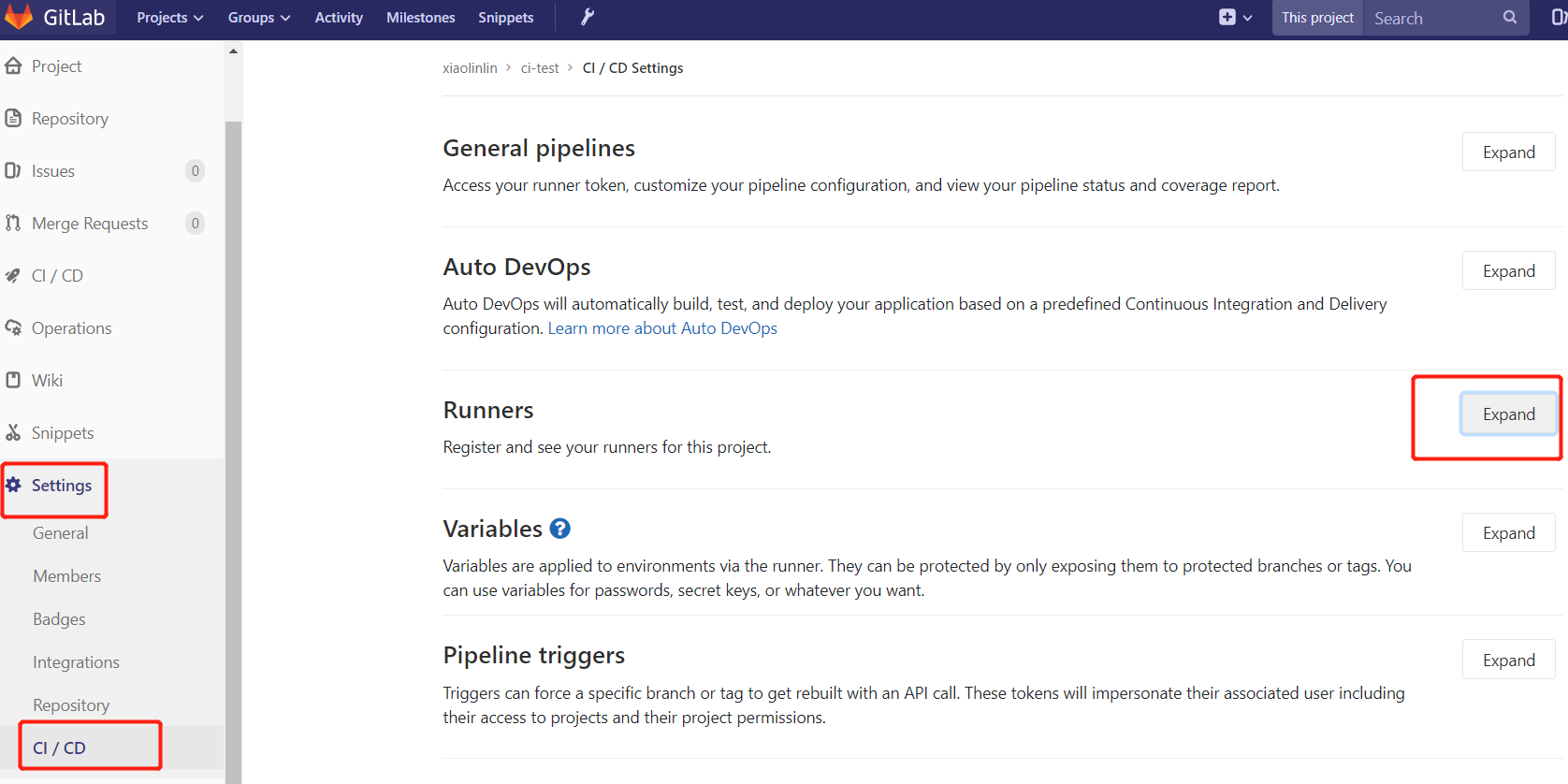
分全局共享runner和项目独享runner,二者注册方式一样，获取的token方式不一样

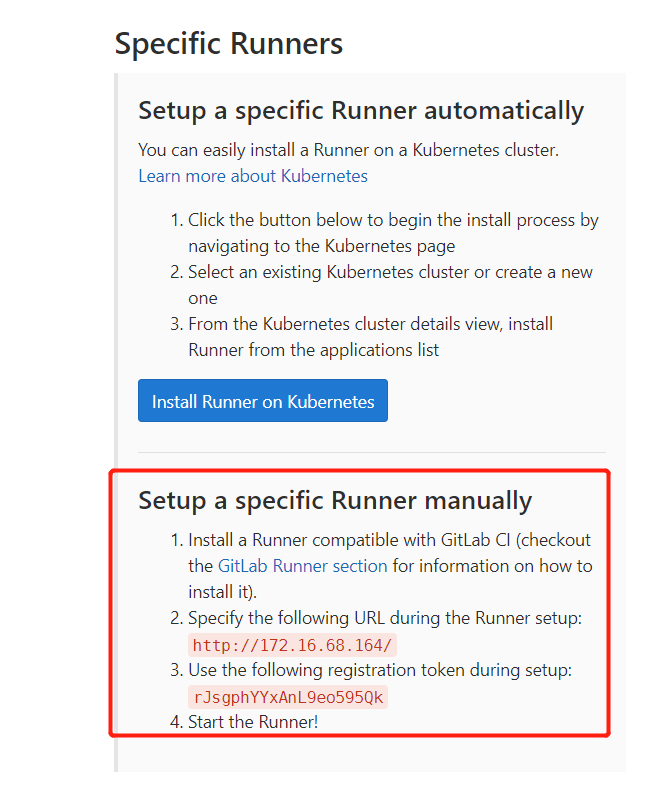
### 4.2.1 全局共享runner获取token

Tips:需要gitlab管理员权限



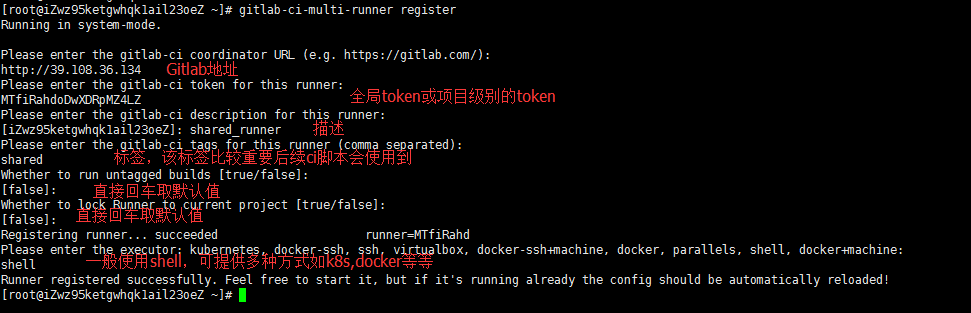
### 4.2.2 项目独享runner获取token



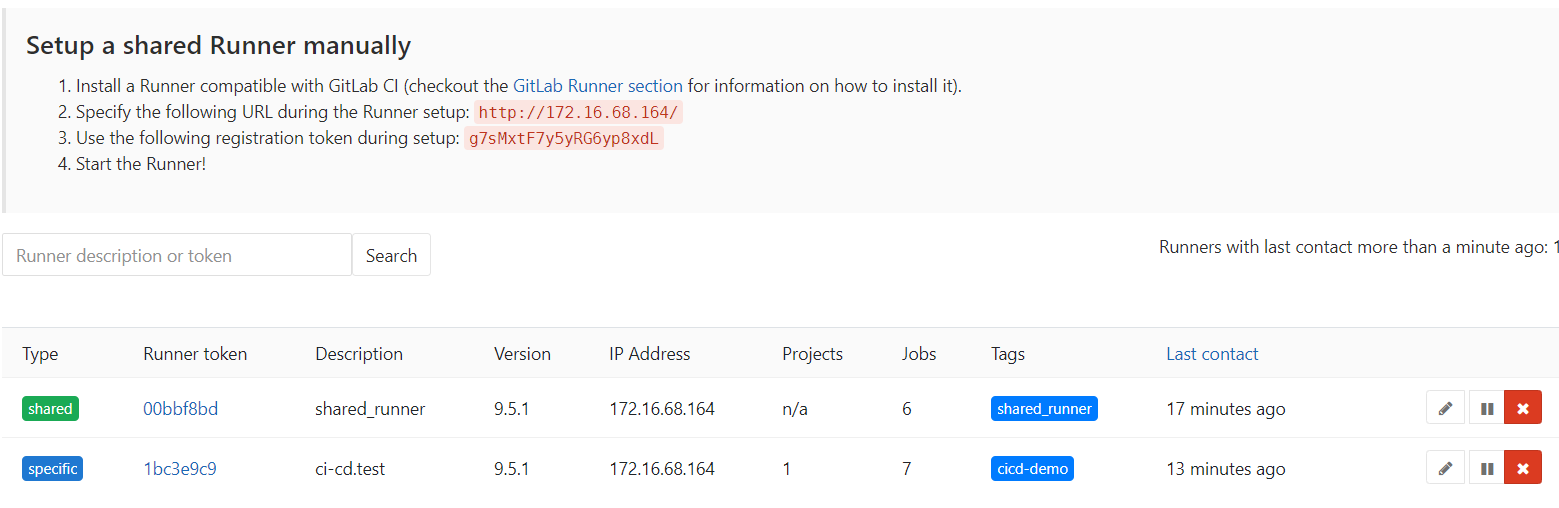


### 4.3.3 注册

[root@iZwz95ketgwhqk1ail23oeZ ~]# gitlab-ci-multi-runner register



全局注册成功后可以全局runner看到runner列表：

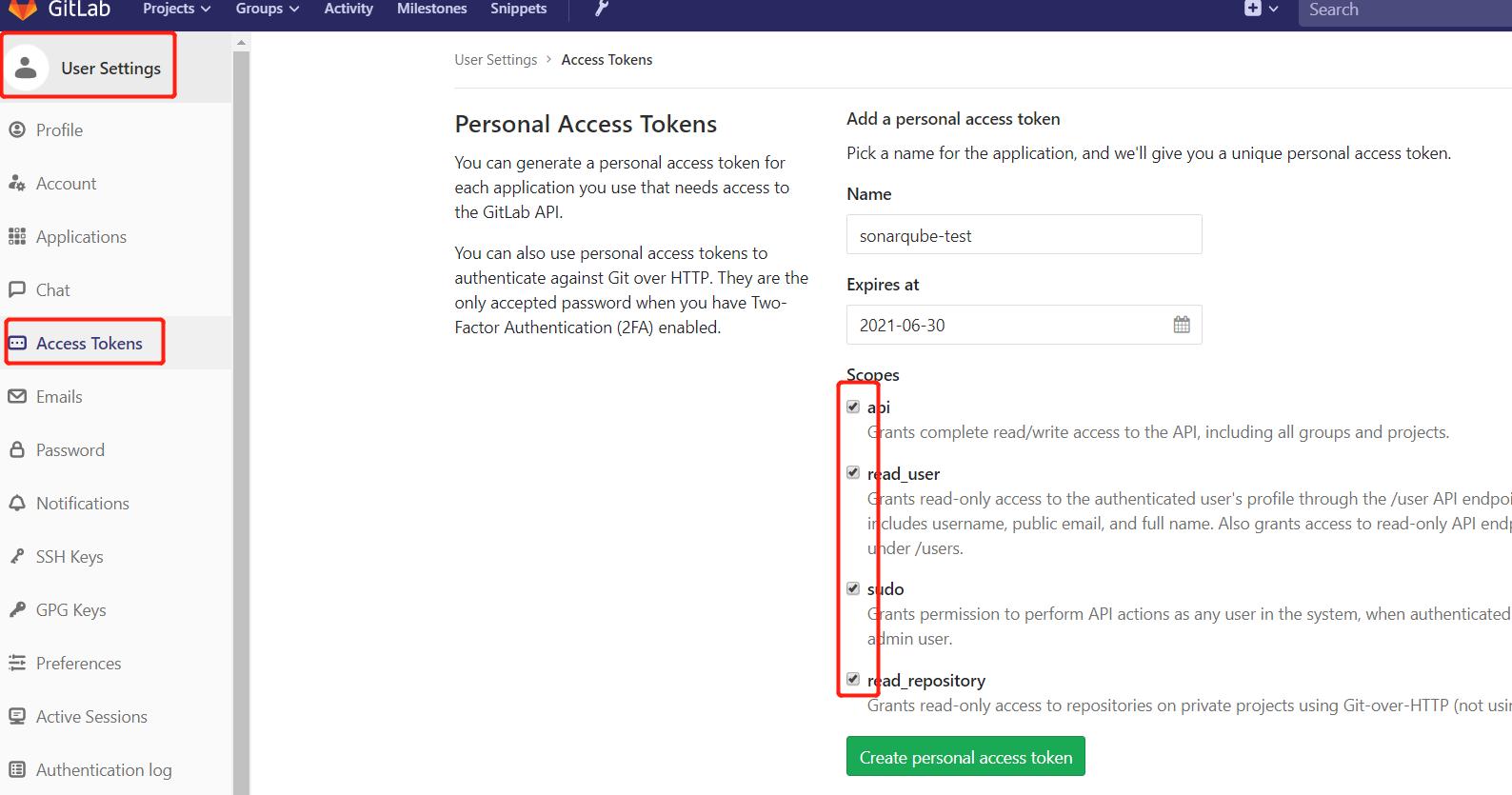


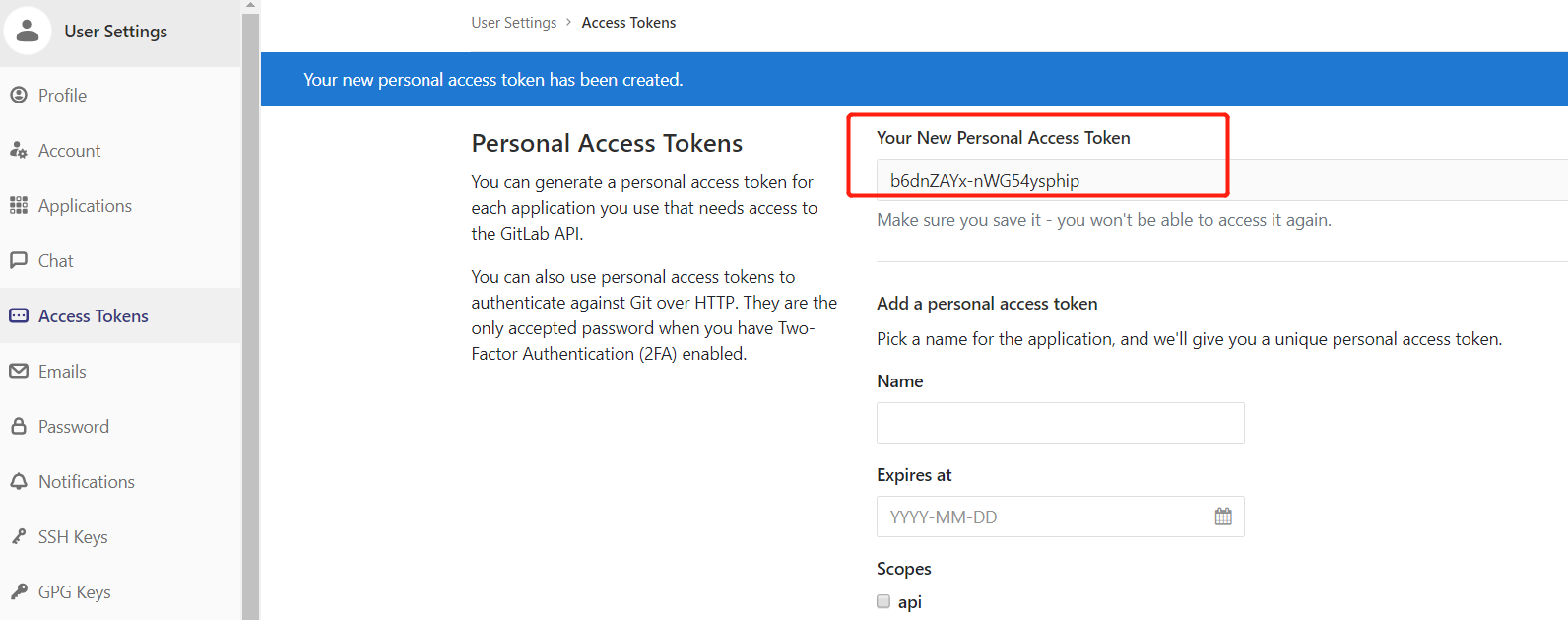
# 5.Gitlab-CICD和Sonarqube配合使用

## 5.1 Sonarqube Gitlab配置

### 5.1.1 gitlab中获取权限

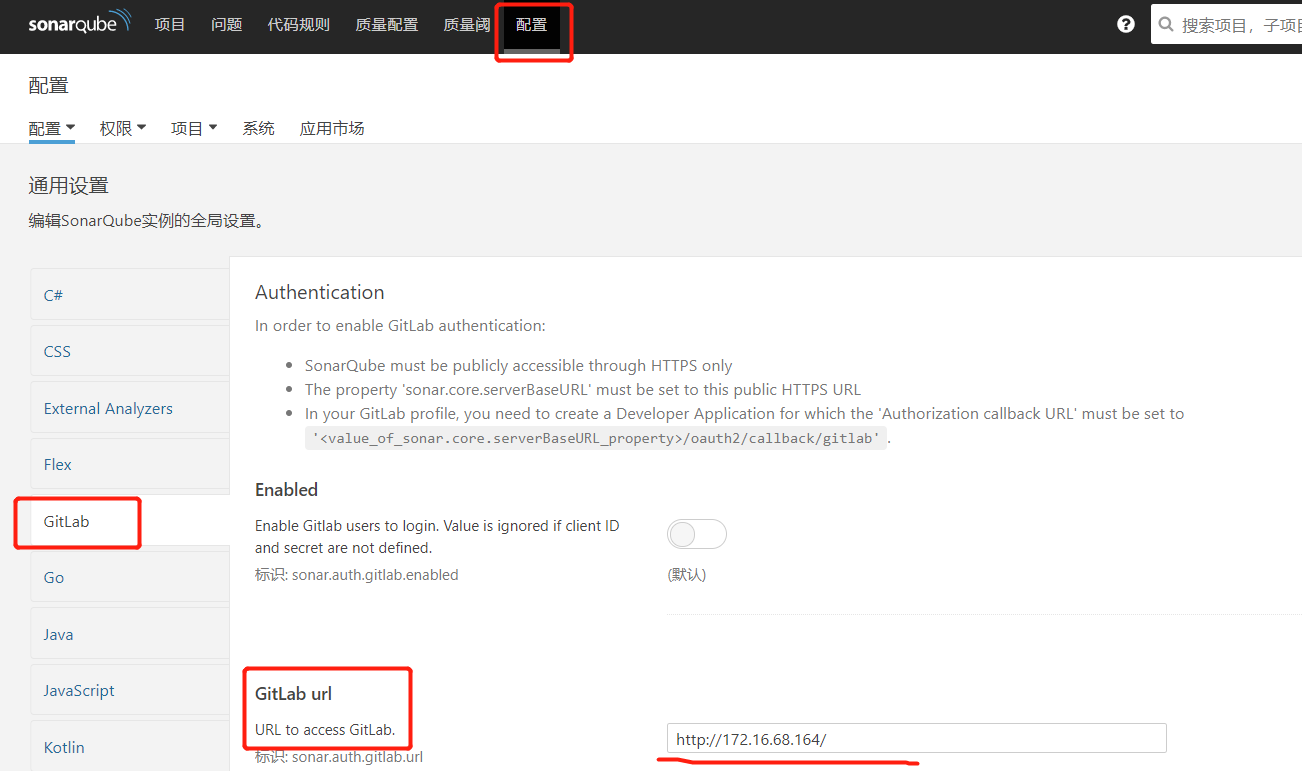
User==>Setting==>AccessTokens

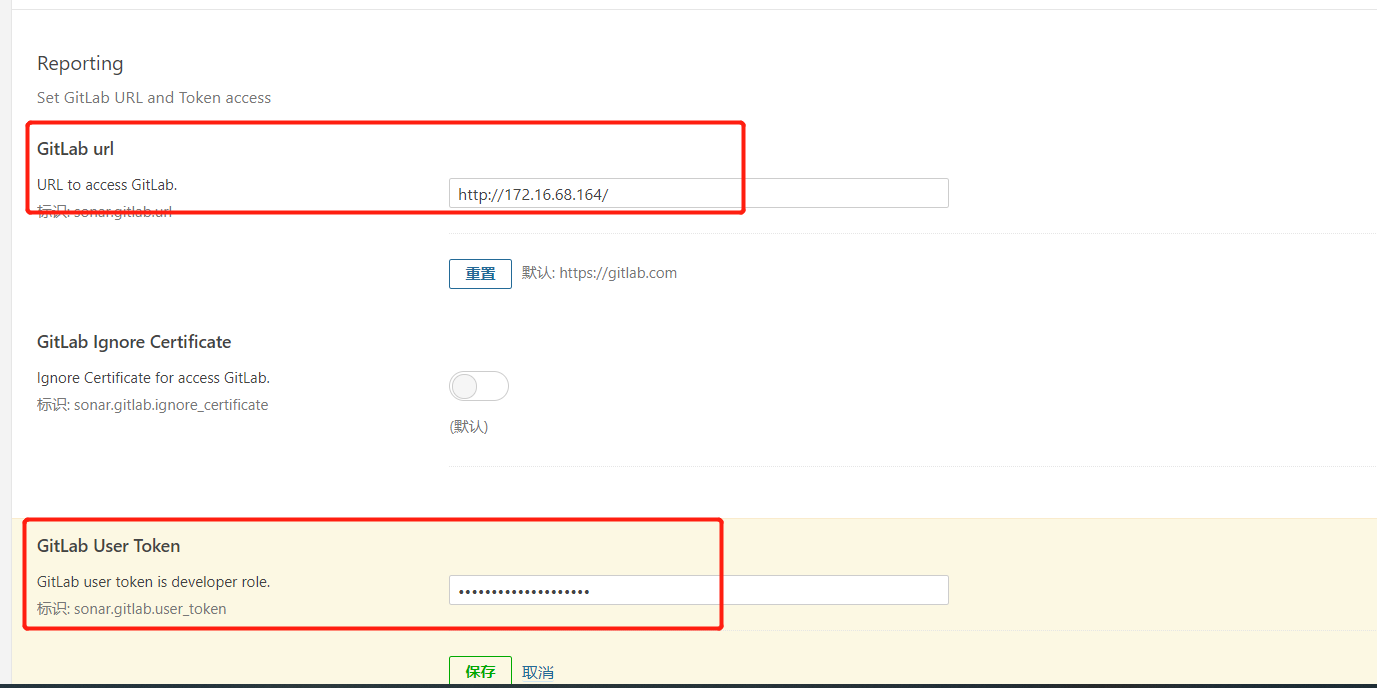




Tips：记住这个Token需要到sonarquebe中配置

### 5.1.2 sonarqube配置gitlab信息

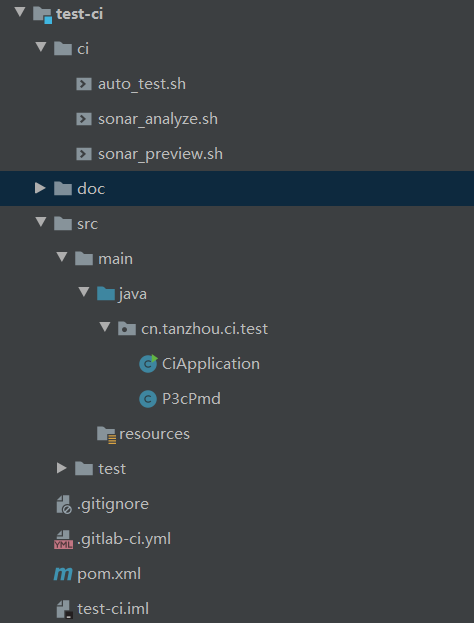




## 5.2 CICD配置

### 5.2.1 Gitlab上创建项目

目录结构大致如下：



### 5.2.2 添加CICD文件 .gitlab-ci.yml 内容：

|  |
| --- |
| variables:  SCRIPT\_PATH: $CI\_PROJECT\_DIR/ci #环境变量，获取当前构建的项目路径  sonar\_preview:  stage: build  script:  - chmod -R 755 $SCRIPT\_PATH #给脚本目录可执行权限  - dos2unix $SCRIPT\_PATH/\* #将所有.sh脚本 转成unix格式  - ci/sonar\_preview.sh #执行脚本  except:  - master  tags:  - cicd-demo  auto\_test:  stage: build  script:  - chmod -R 755 $SCRIPT\_PATH  - dos2unix $SCRIPT\_PATH/\*  - ci/auto\_test.sh  except:  - master  tags:  - cicd-demo  sonar\_analyze:  stage: build  script:  - chmod -R 755 $SCRIPT\_PATH  - dos2unix $SCRIPT\_PATH/\*  - ci/sonar\_analyze.sh  only:  - master  tags:  - cicd-demo |

主要是Maven Build阶段执行ci目录下sonar\_preview sonar\_analyze auto\_test三个脚本，注意dos2unix命令默认linux未安装



### 5.2.3 脚本内容：

sonar\_preview.sh 此脚本仅将非master分之代码保留在commit中

|  |
| --- |
| #!/bin/bash  mvn clean verify sonar:sonar \  -Dsonar.host.url=http://172.16.68.164:9000 \  -Dsonar.login=55b3358419c9543d761fb808731288df5cc8ed98  -Dsonar.analysis.mode=preview \  -Dsonar.gitlab.project\_id=$CI\_PROJECT\_ID \  -Dsonar.gitlab.commit\_sha=$CI\_COMMIT\_SHA \  -Dsonar.gitlab.ref\_name=$CI\_COMMIT\_REF\_NAME  if [ $? -eq 0 ]; then  echo "sonarqube code-analyze-preview over."  fi |



sonar\_analyze.sh 生成报告上报到sonarqube服务器

|  |
| --- |
| #!/bin/bash  echo "sonarqube code-analyze...."  mvn clean install -DskipTests  mvn sonar:sonar \  -Dsonar.host.url=http://172.16.68.164:9000 \  -Dsonar.login=55b3358419c9543d761fb808731288df5cc8ed98  -Dsonar.issuesReport.html.enable=true \  -Dsonar.analysis.mode=preview \  -Dsonar.preview.excludePlugins=issueassign,scmstats  if [ $? -eq 0 ]; then  echo "sonarqube code-analyze over."  fi |



auto\_test.sh 模拟自动化测试

|  |
| --- |
| #!/bin/bash  COMMITTER=$(git log -1 --format=%cE)  echo ${COMMITTER}  if [ $? -eq 0 ]; then  echo "do something for auto\_test here."  echo "auto\_test over."  fi |

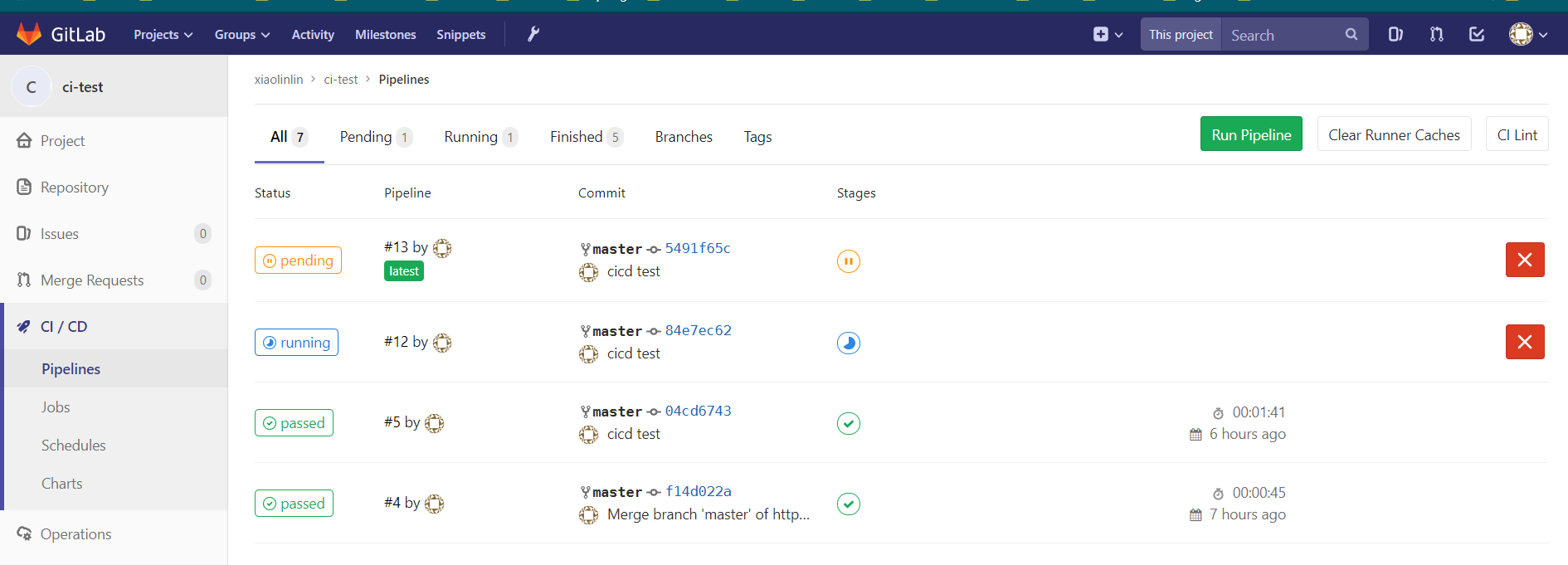


### 5.2.4 添加java文件内容触发gitlab提交：

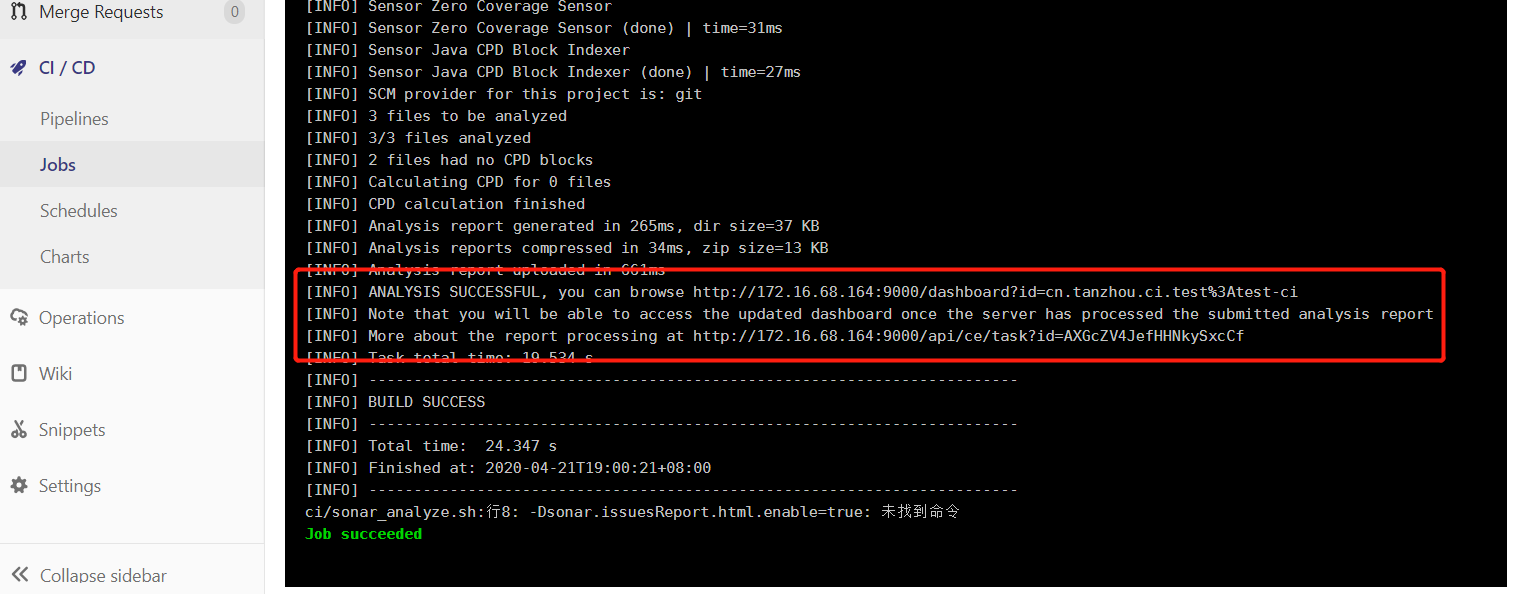
|  |
| --- |
| /\*\*  \* [简要描述]: ci 测试  \* [详细描述]:  \*  \* @author xiaolinlin  \* @version 1.0, 2020/4/20 16:47  \* @since JDK 1.8  \*/  public class CiApplication  {  public static final String testfinal = "00001";  public static void main(String[] args)  {  System.out.printf("CI-test");  }  } |

### 5.2.5 提交代码触发CICD

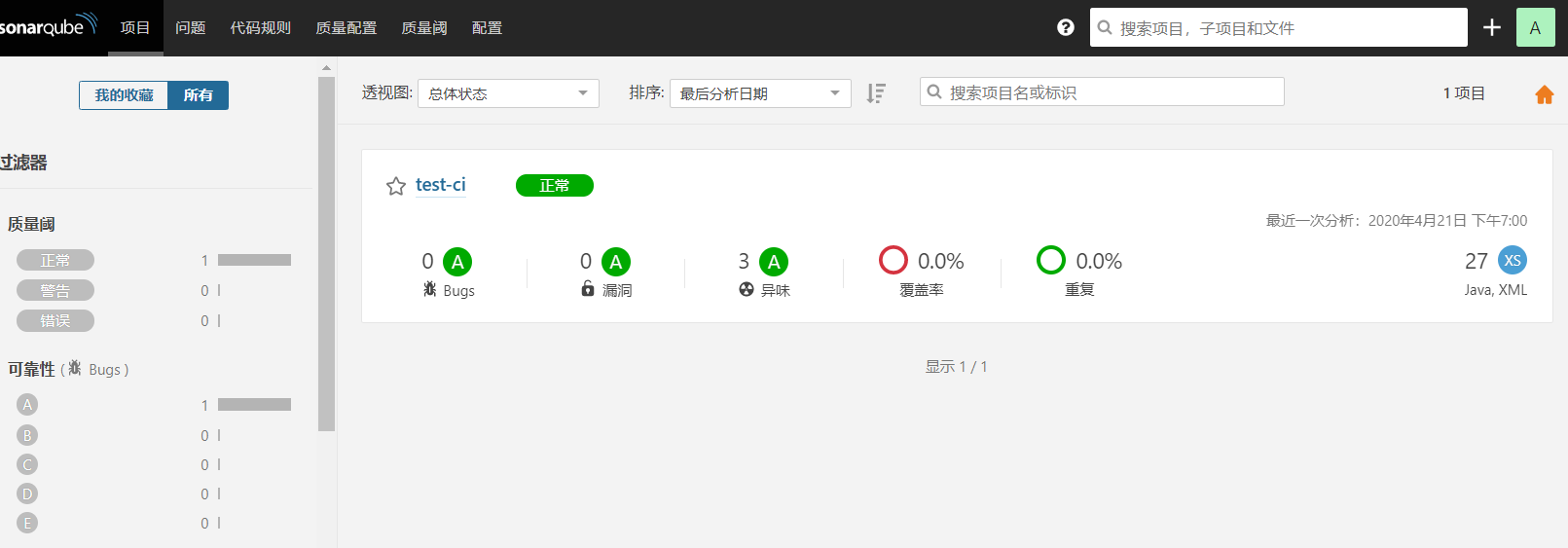
Gitlab触发CICD

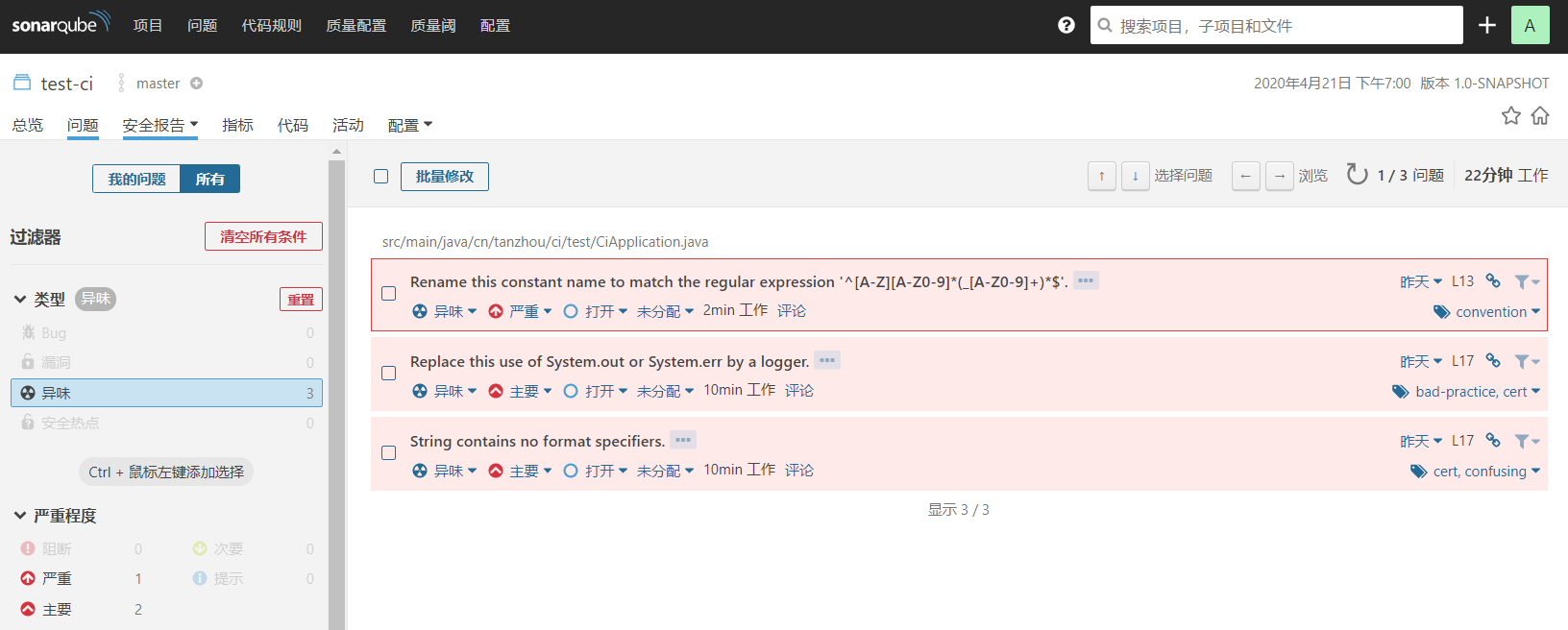


点击running，可以查看执行日志



### 5.2.6 sonarqube代码展示(自动创建项目)





## 5.3 Gitlab-ci.yml配置详解

|  |
| --- |
| before\_script:  - echo "每个job之前都会执行"  after\_script:  - echo "每个job之后都会执行"  #全局变量定义，在job中使用 ${variableName}  #内置变量：CI\_COMMIT\_REF\_NAME（项目所在的分支或标签名称）、CI\_JOB\_NAME（任务名称）、CI\_JOB\_STAGE（任务阶段）  variables:  # 定义拉取代码的方式，有3种：clone/fetch/none，默认为clone，速度最慢，每步job都会重新clone一次代码。我们一般将它设置为none，  # 在具体任务里设置为fetch就可以满足需求，毕竟不是每步都需要新代码，那也不符合我们测试的流程  GIT\_STRATEGY: "none" #GIT策略，  # 来指定需要在job之间缓存的文件或目录  cache:  #因为缓存为不同管道和任务间共享，可能会覆盖，所以有时需要设置key  key: ${CI\_COMMIT\_REF\_NAME} # 启用每分支缓存  #key: "$CI\_JOB\_NAME/$CI\_COMMIT\_REF\_NAME" # 启用每个任务和每个分支缓存  untracked: true #缓存所有Git未跟踪的文件  #paths: #缓存哪些文件/文件夹  #- node\_modules/  #- dist/  #表示构建阶段，主要包含 build、test、deploy三个阶段，任务将按此顺序执行  # build阶段job并行，所有build执行ok并行test阶段job，所有test执行ok并行deploy，全部ok标记success，否则标记failure  # yml文件中如未明确定义stages，默认取值就是：build、test、deploy，job中未指定stage，默认取值test  stages:  - build  - test  - deploy  # Jobs 表示构建工作，表示某个 Stage 里面执行的工作,可以在 Stages 里面定义多个 Jobs  # 定义job任务，job\_name 全局唯一且不能是关键字  test-job1:  # test阶段执行  stage: test  tags:  - xx #注册runner的标签  only:  - dev # 只有dev分支提交代码才会执行这个任务  - /^future-.\*$/ #正则匹配，只有future-开头的分支执行  script: #要执行的内容，脚本  - echo "I am job1"  - echo "Just from dev&future-\* branch and test phase!"  # 代码覆盖率设置  coverage: '/Code coverage: \d+\.\d+/'  # 指定该任务执行完毕后，哪些目录或文件需要保留，打成zip包供后续使用  artifacts:  path:  - public  # 定义此作业完成部署的环境名称，job将会执行部署到production环境  #environment:  #name: production  test-job2:  # 默认stage阶段为 test  stage: test  only:  - master # 只有Master分支有提交代码才会执行  script:  - echo "Trigger for master branch!"  allow\_failure: true #允许失败，即不影响下一步构建  build-job:  stage: build  except:  - dev # 除dev环境，其他分之提交点都会执行  script:  - echo "Trigger without dev branc!"  # when触发条件  # on\_success、on\_failure、always、manual(手动)  # 不管前面步骤是否成功，都会执行  when: always  ############################ yml高级用法###########################  # .开头的job忽略执行，做继承、模板方法等  # &(锚)使用隐藏的key,\*(aliases)别名做job模板，后续job可以<<(继承)  .general: &job\_template  stage: deploy  script:  - echo 'Job template'  - path/shell.sh #执行一个脚本  only:  - master  tags:  - runnerTag  # 生产job  .prod: &prod  # 继承general  <<: \*job\_template  # 重写script  script:  - echo 'Prod branch!'  # 各环境配置  .test: &test  <<: \*job\_template  script:  - echo 'Dev branch!'  only:  - test  .dev: &dev  <<: \*job\_template  script:  - echo 'Dev branch!'  only:  - dev  - /^feture-\*$/  prod-job:  <<: \*prod  test-job:  <<: \*test  dev-job:  <<: \*dev |

