Gitlab的Hooks(钩子)做Push代码检测

# 背景简介：

项目越来越大，团队越来越大，代码越来越来多，然后就是各种五花八门的代码格式、代码规范。要做到代码规范我们可以采取代码交叉检视、IDEA/Eclipse安装插件自检、以及代码管理服务端做校验，且不说各种方法的利弊，本文直接采用服务端做校验(Gitlab)。

# 2. 服务端Gitlab校验原理简要说明

利用Gitlab的 (钩子)hooks机制，关于钩子有客户端、服务端的，本文仅描述服务端的钩子，更多关于钩子的资料参考：

<https://git-scm.com/book/zh/v2/%E8%87%AA%E5%AE%9A%E4%B9%89-Git-Git-%E9%92%A9%E5%AD%90>。

Gitlab服务端会有三个主要钩子：

**pre-receive**(处理客户端push动作时最先被调用的脚本，以非0值退出拒绝Push，可以用来做注释标准化、代码标准化等)；

**update**(与pre-receive功能类似，为每个准备更新的分之各运行一次)；

**post-receive**(是在push之后执行的脚本，可以用来调用后续的持续集成、发邮件通知等)。

而本文的用到的做代码规则校验的就是pre-receive的钩子，通过此钩子执行脚本进行规则校验，然后检查如不符合规范返回错误信息到客户端(Idea，Git，Bash等)。

# 3 Gitlab Hooks配置

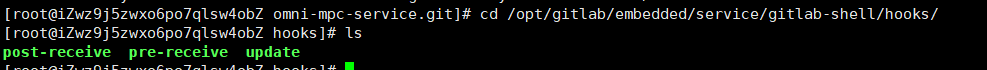
Gitlab hooks配置分全局(所有项目)和项目配置

官方文档说明地址：

<https://docs.gitlab.com/ee/administration/custom_hooks.html#setup>

**全局配置路径：**

/opt/gitlab/embedded/service/gitlab-shell/hooks/

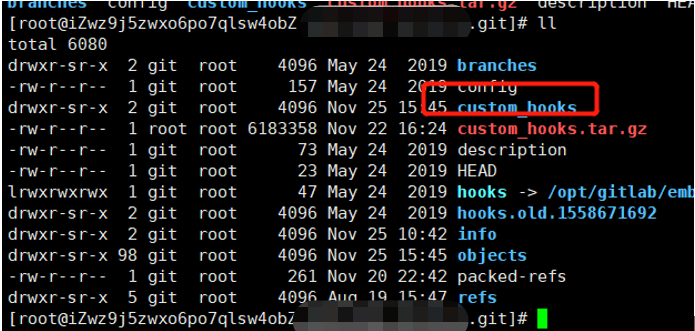


**针对项目的配置路径：**

/var/opt/gitlab/git-data/repositories/${group}/${project\_name}.git/

需要在该文件下创建custom\_hooks目录，不要试图直接修改hooks目录下的东西，是针对全局的。

且custom\_hooks目录的权限必须是git:git



本次以项目为单位配置代码校验，即在custom\_hooks

# 4. 使用CheckStyle校验

## 4.1 下载CheckStyle-all-xx.jar包

<https://github.com/checkstyle/checkstyle/releases>

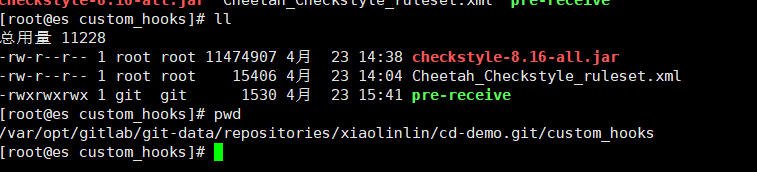


*tips：请使用对应的与checkStyle.xml文件对应的版本，比如这里使用的是8.16*

## 4.2 获取checkStyle.xml检测文件



## 4.3 checkStyle和checkStyle-all的jar包拷贝到custom\_hooks目录下：



## 4.4 提交检测结果：



# 5. 阿里云P3C-PMD校验

## 5.1. 获取P3C-PMD文件

P3C阿里云的插件，Github地址：

<https://github.com/alibaba/p3c>

此处我们用到p3c-pmd插件，需要打包成jar文件，以便Gitlab服务端使用。

### 5.1.1 打包p3c-pmd，

使用最新2.0.0版本，Windows环境构建

1）准备gradle环境和git环境(可选)。

请自行百度安装gradle文档

1）github下载源码：

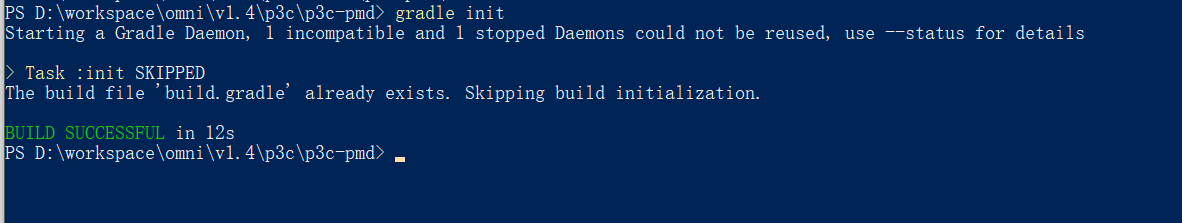
git clone <https://github.com/alibaba/p3c.git>

或者下载github上download zip文件

3） 进入p3c-pmd目录：



4）执行gradle init



5）编辑build.gradle文件

在文件末尾添加：

jar {

from {

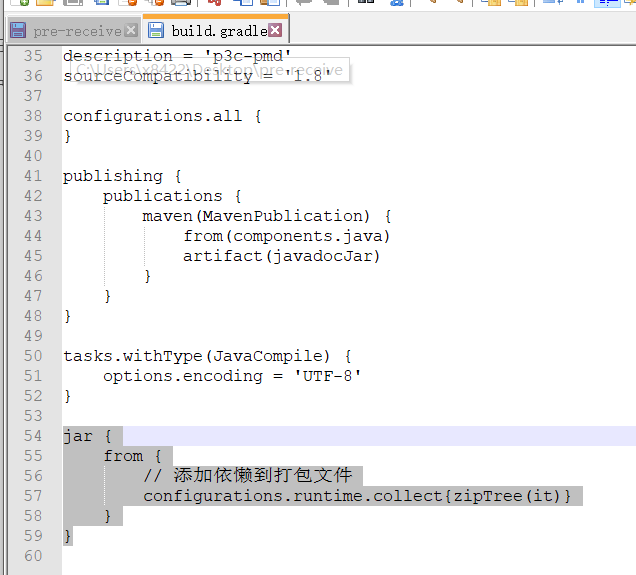
// 添加依懒到打包文件

configurations.runtime.collect{zipTree(it)}

}

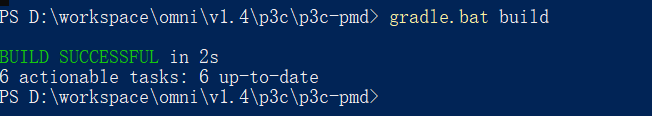
}

如下图所示：

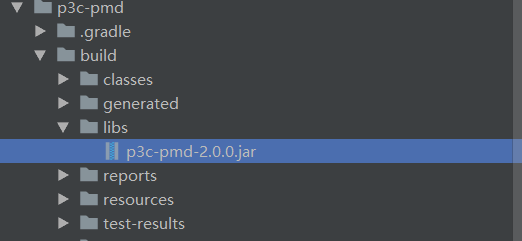


6）开始构建gradle

执行gradle.bat build



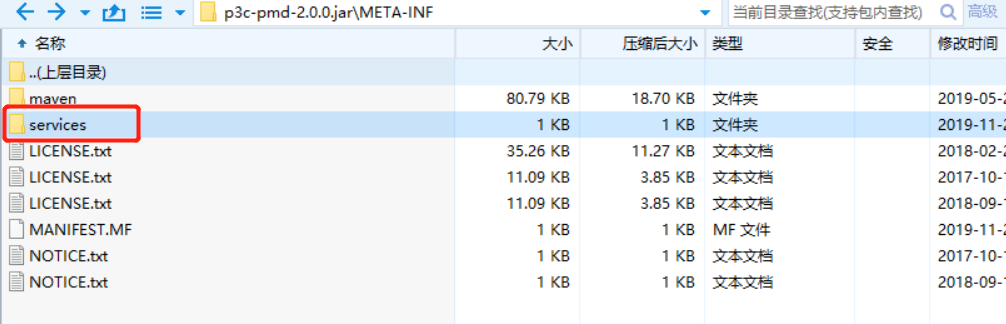
构建成功后会在 build\libs生成p3c-pmd-2.0.0.jar文件



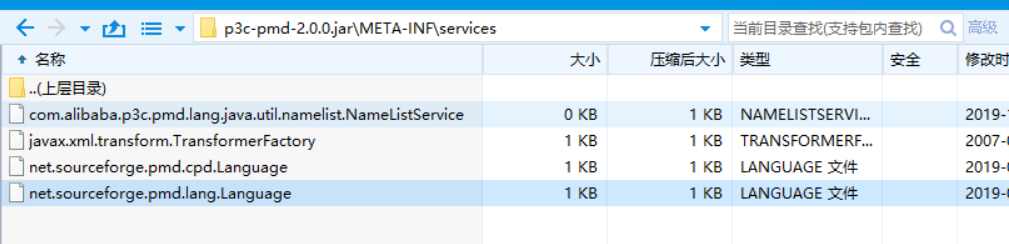
7）修改p3c-pmd-2.0.0.jar中META-INFO文件内容

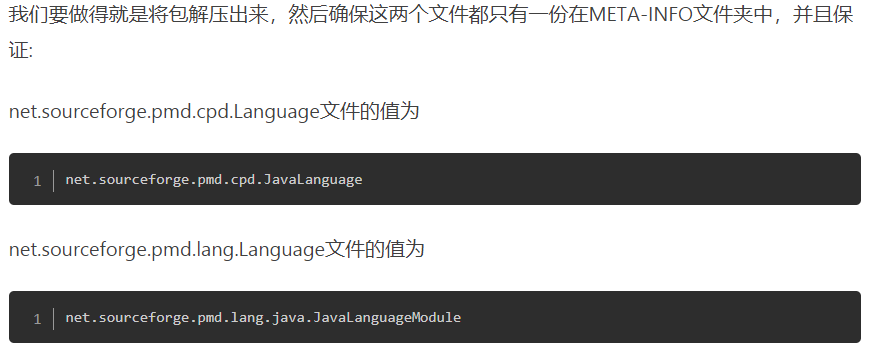
A.用压缩文件打开.jar文件：



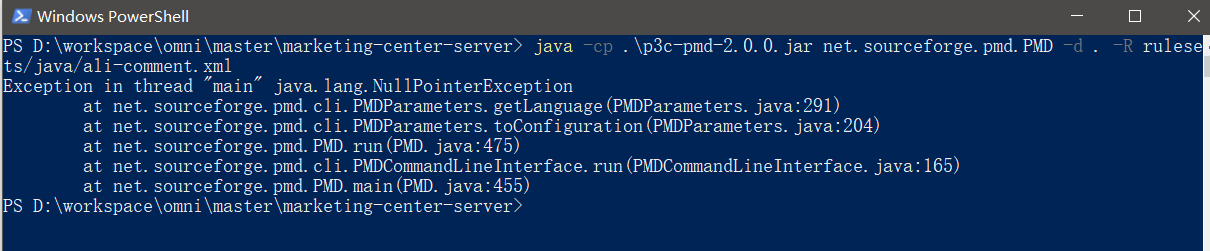


B.删掉重复的文件并确保里面的内容是正确的





C.主要用于解决：

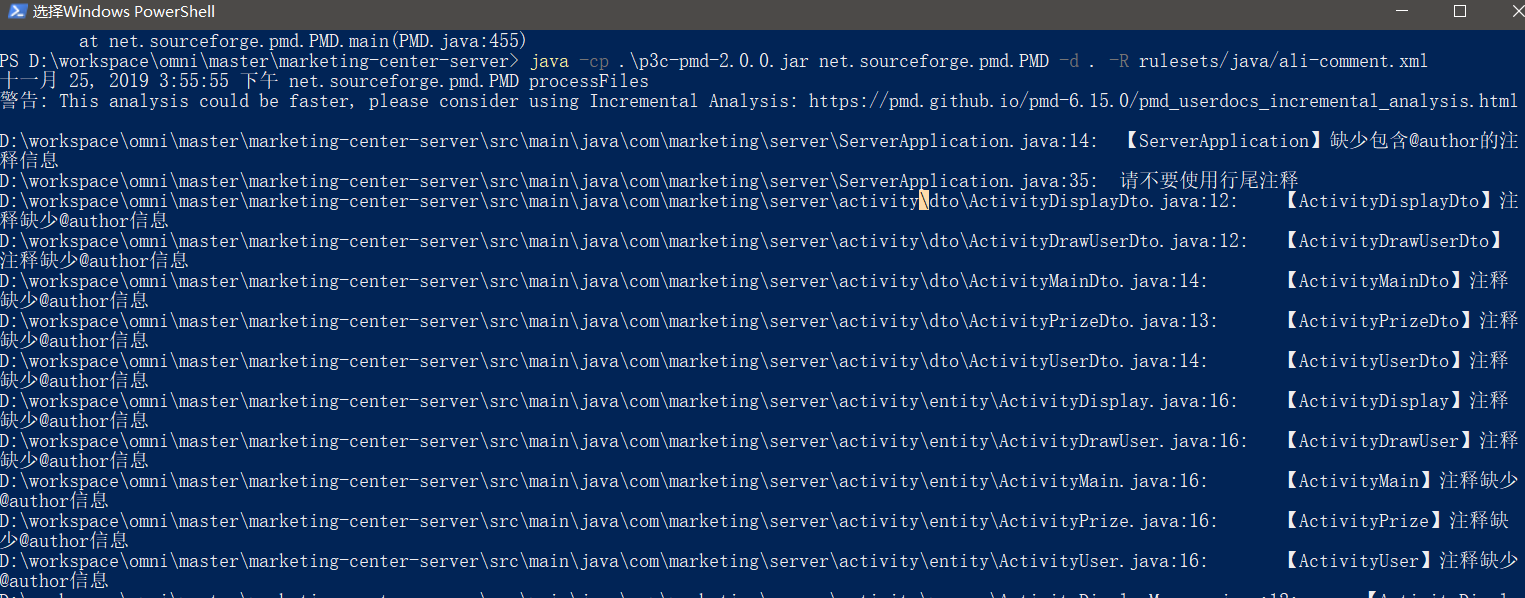




### 5.1.2 执行jar校验是否正确

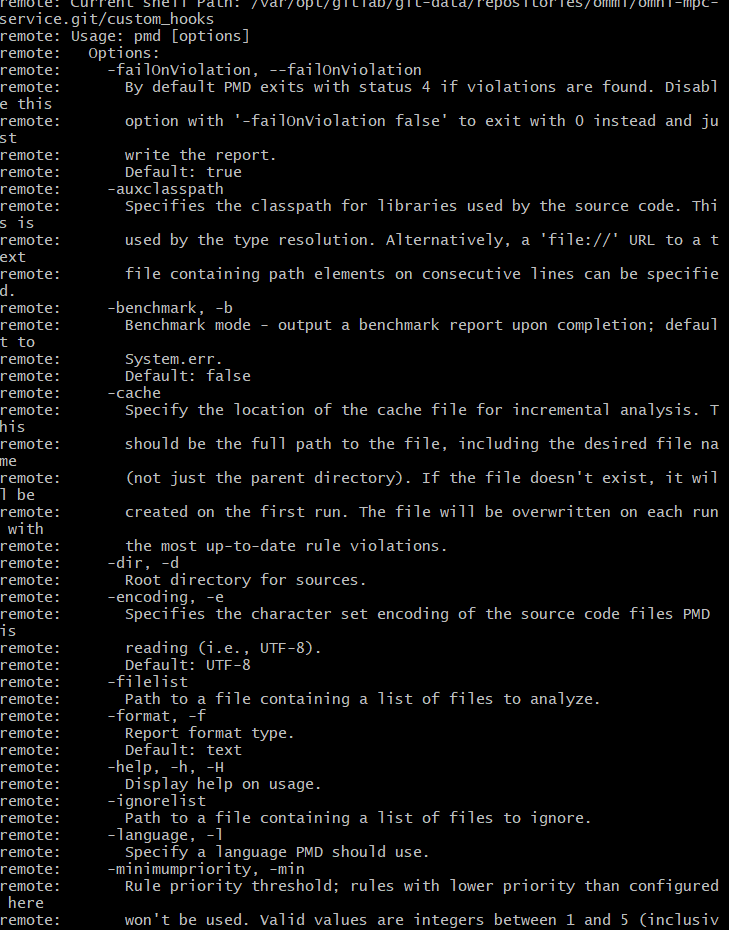
java -cp p3c-pmd-2.0.0.jar net.sourceforge.pmd.PMD -d . -R rulesets/java/ali-comment.xml

得到如下提示，表名正确可用：



p3c-pmd 几个参数说明：-d 待校验的源文件目录，-R 校验规则，-f 检测结果输出格式。

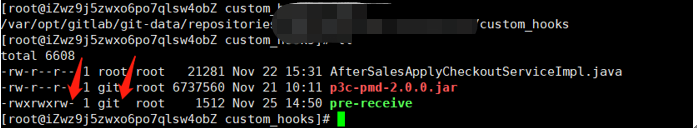
Github没有明确说明各种参数，只不过执行命令错误会提示各种参数以及使用方法：



## 5.3 配置p3c-pmd-2.0.2.jar

5.3.1拷贝3步骤所生成的p3c-pmd-2.0.0.jar文件到 custom\_hooks目录

5.3.2 pre-receive文件，并保持776可执行权限以及仅属于git用户



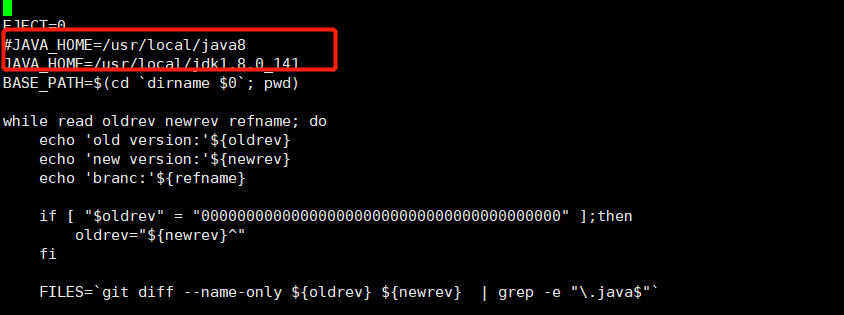
5.3.3 编辑里面的内容：

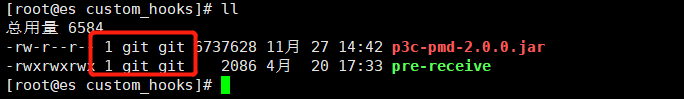
|  |
| --- |
| EJECT=0  #JAVA\_HOME=/usr/local/java8  JAVA\_HOME=/usr/local/jdk1.8.0\_141  BASE\_PATH=$(cd `dirname $0`; pwd)  while read oldrev newrev refname; do  echo 'old version:'${oldrev}  echo 'new version:'${newrev}  echo 'branc:'${refname}  if [ "$oldrev" = "0000000000000000000000000000000000000000" ];then  oldrev="${newrev}^"  fi  FILES=`git diff --name-only ${oldrev} ${newrev} | grep -e "\.java$"`  if [ -n "$FILES" ]; then  TEMPDIR=$BASE\_PATH/"tmp"  for FILE in ${FILES}; do  mkdir -p "${TEMPDIR}/`dirname ${FILE}`" >/dev/null  git show $newrev:$FILE > ${TEMPDIR}/${FILE}  done;  MAIN\_JAVA\_PATH=$TEMPDIR'/src/main'  #echo 'Temp update files path: '$MAIN\_JAVA\_PATH  #FILES\_TO\_CHECK=`find $MAIN\_JAVA\_PATH -name '\*.java'`  #echo 'Check files:'${FILES\_TO\_CHECK}  echo 'Aliyun p3c-pmd check starting.....'  #echo 'Current shell Path:' $BASE\_PATH  #echo 'JAVA\_HOME:' $JAVA\_HOME  #echo 'Root directory for java sources: '$MAIN\_JAVA\_PATH  $JAVA\_HOME/bin/java -Dpmd.language=en -cp $BASE\_PATH/p3c-pmd-2.0.0.jar net.sourceforge.pmd.PMD -d $MAIN\_JAVA\_PATH -R rulesets/java/ali-comment.xml,rulesets/java/ali-concurrent.xml,rulesets/java/ali-constant.xml,rulesets/java/ali-exception.xml,rulesets/java/ali-flowcontrol.xml,rulesets/java/ali-naming.xml,rulesets/java/ali-oop.xml,rulesets/java/ali-other.xml,rulesets/java/ali-set.xml -f text  REJECT=$?  echo 'Aliyun p3c-pmd check end...'  #REJECT=1  echo $REJECT  rm -rf $TEMPDIR  fi  done  exit $REJECT |

参考文件pre-recive文件和jar:

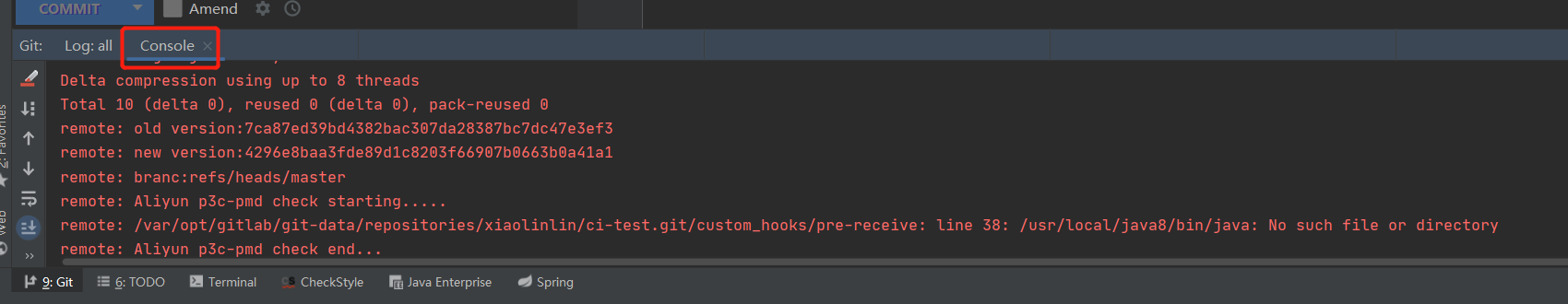


***注意的坑：JAVA\_HOME的设置、BASE\_PATH对比更新后的待检查临时文件的目录、以及Dpmd.language=en参数设置成英文(暂无法处理返回到客户端乱码)，以上两个环境变量配置错误需要自己调试该这个校验的命令***



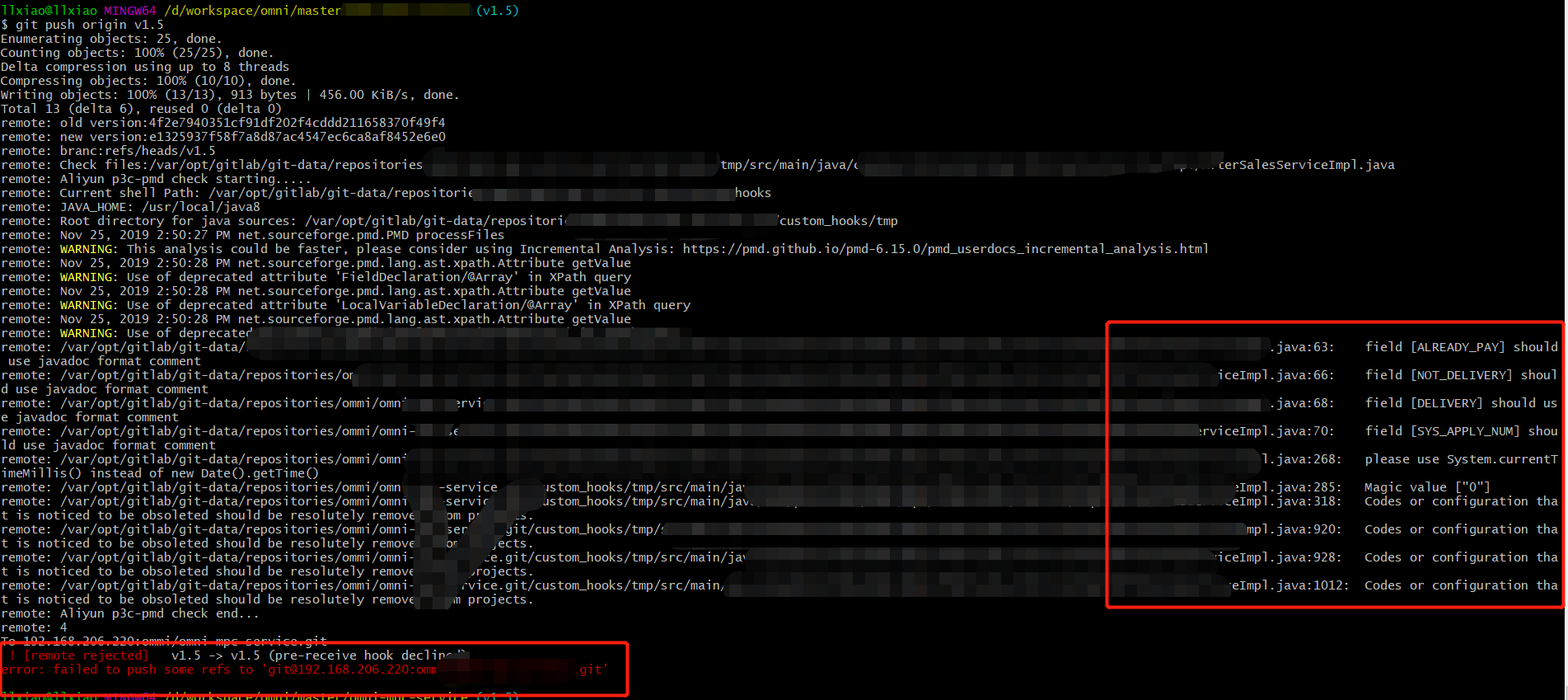


***客户端调试信息：***

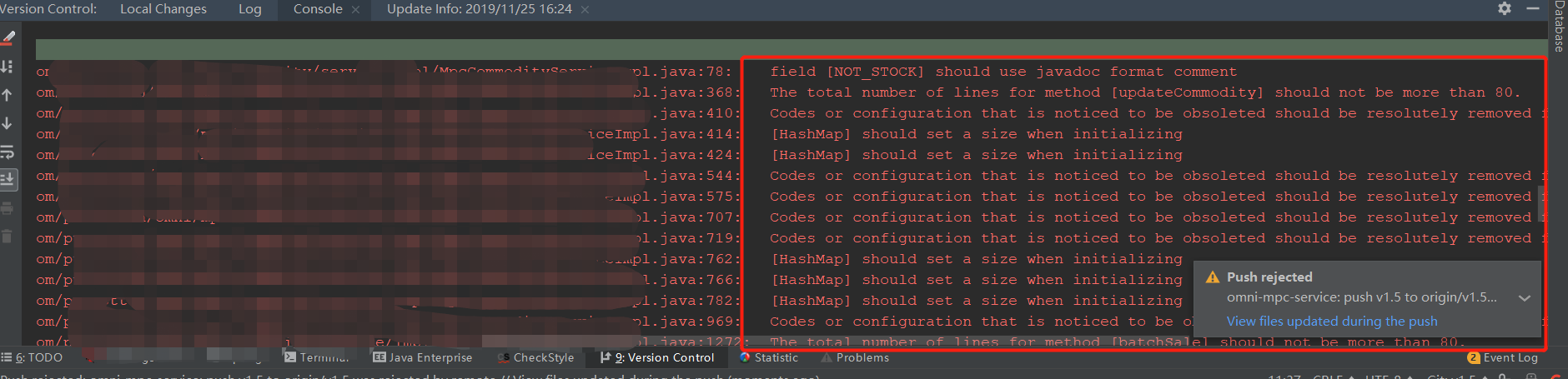


## 5.4 客户端提交测试

Git bash提交测试服返回结果：



IDEA 返回结果：



修改所有返回错误后再次push返回成功的

## 5.5 自定义或修改规则

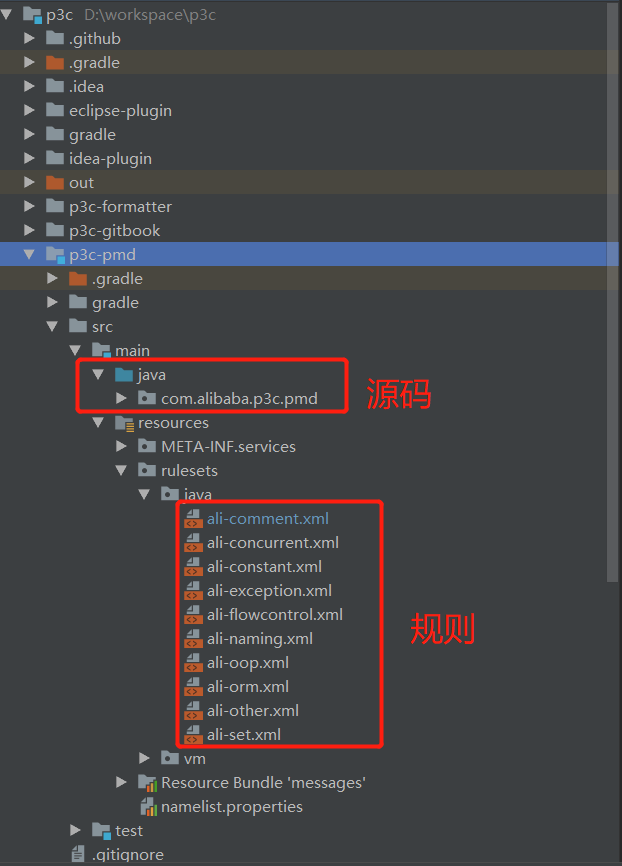
### 5.5.1背景：

理论上是我们是希望严格遵守这个阿里的p3c的规则，但是难免会有些搞特殊化的，比如方法80行的限制，假如我们有个数据库表对象超过80个字段，做getset的时候就会超过80行，因此我们需要在一些特殊的规则上做一些调整。

### 5.5.2 原理：其实就是修改源码

此处需要准备：Idea IDE、JAVA、Gradle环境，安装步骤都自行百度

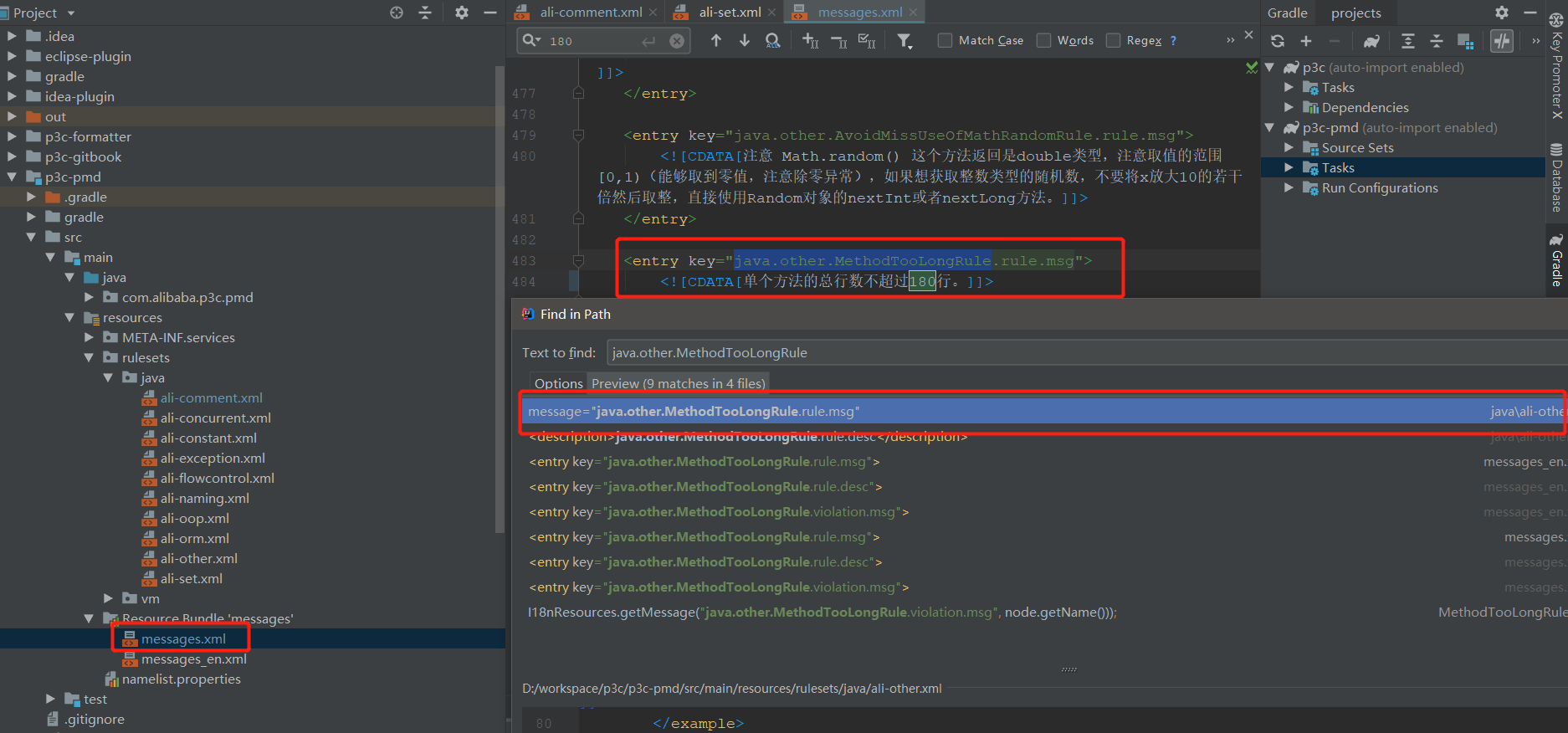
导入idea 的gradle工程大概目录如下:



每一个规则其实都对应的一个java类，调整规则或者java类。比如我们调整80行代码为例：

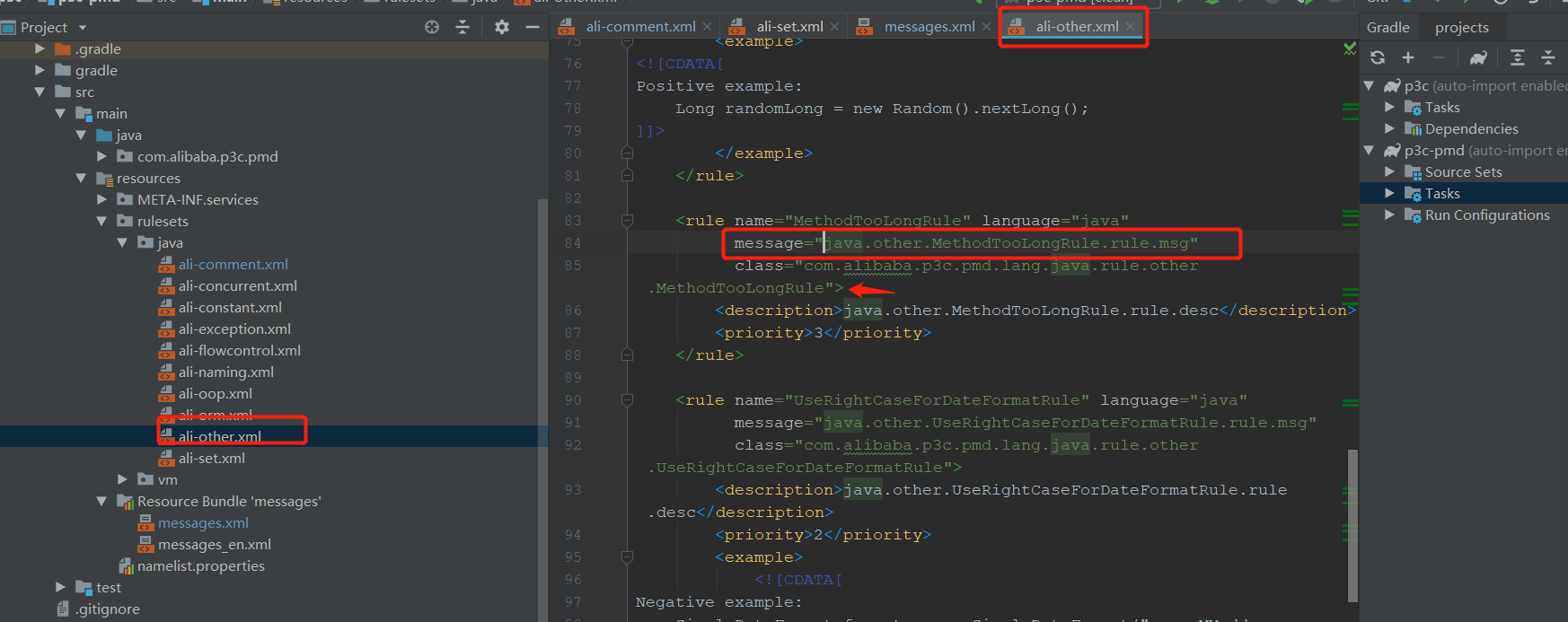
### 5.5.3找到80行代码限制的规则所在

我们以message提示为入口突破

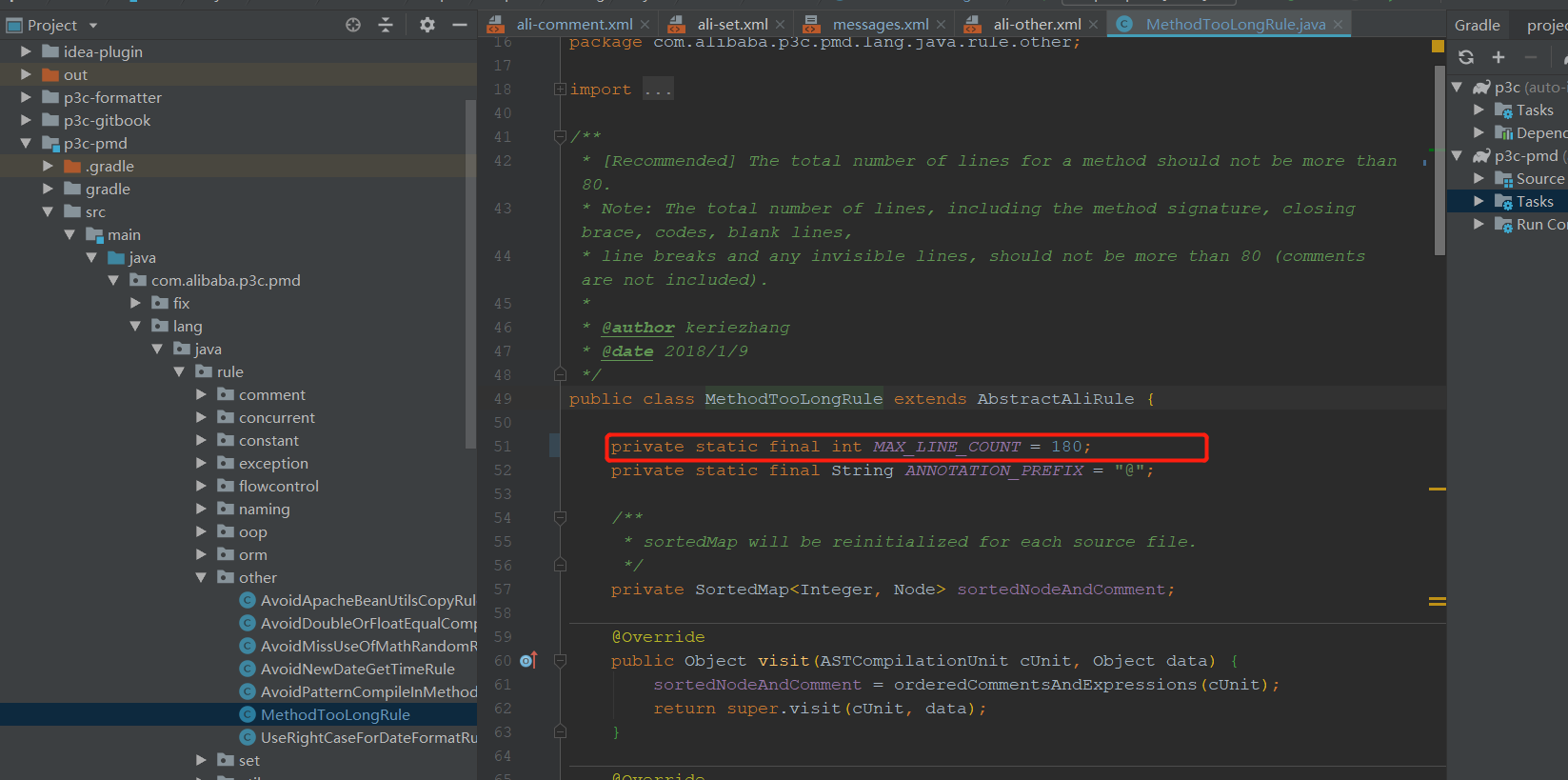


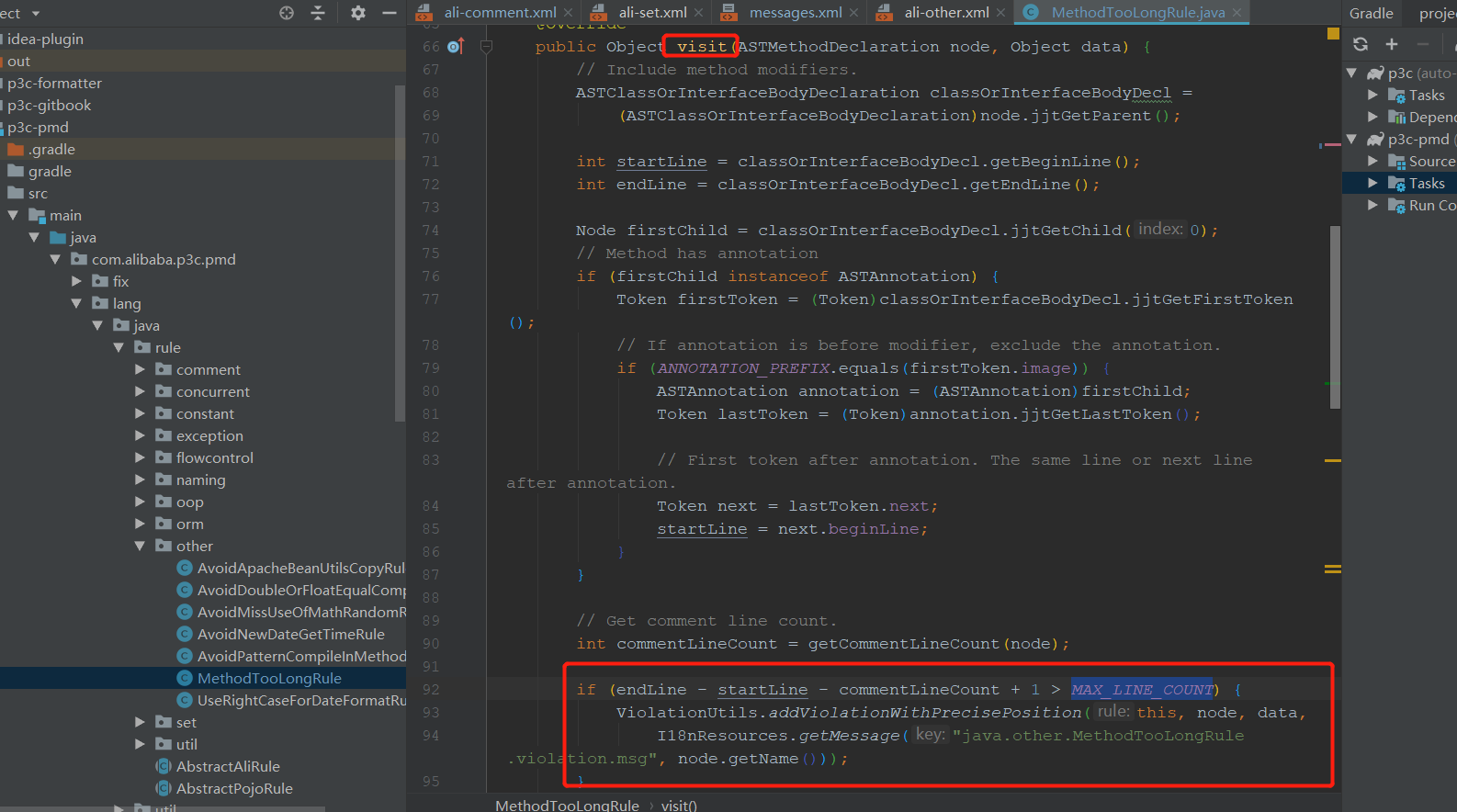
***Tips: ctrl+shift+F：idea ide 里面万能搜***

### 5.5.4找到所在的规则xml，如：java\ali-other.xml



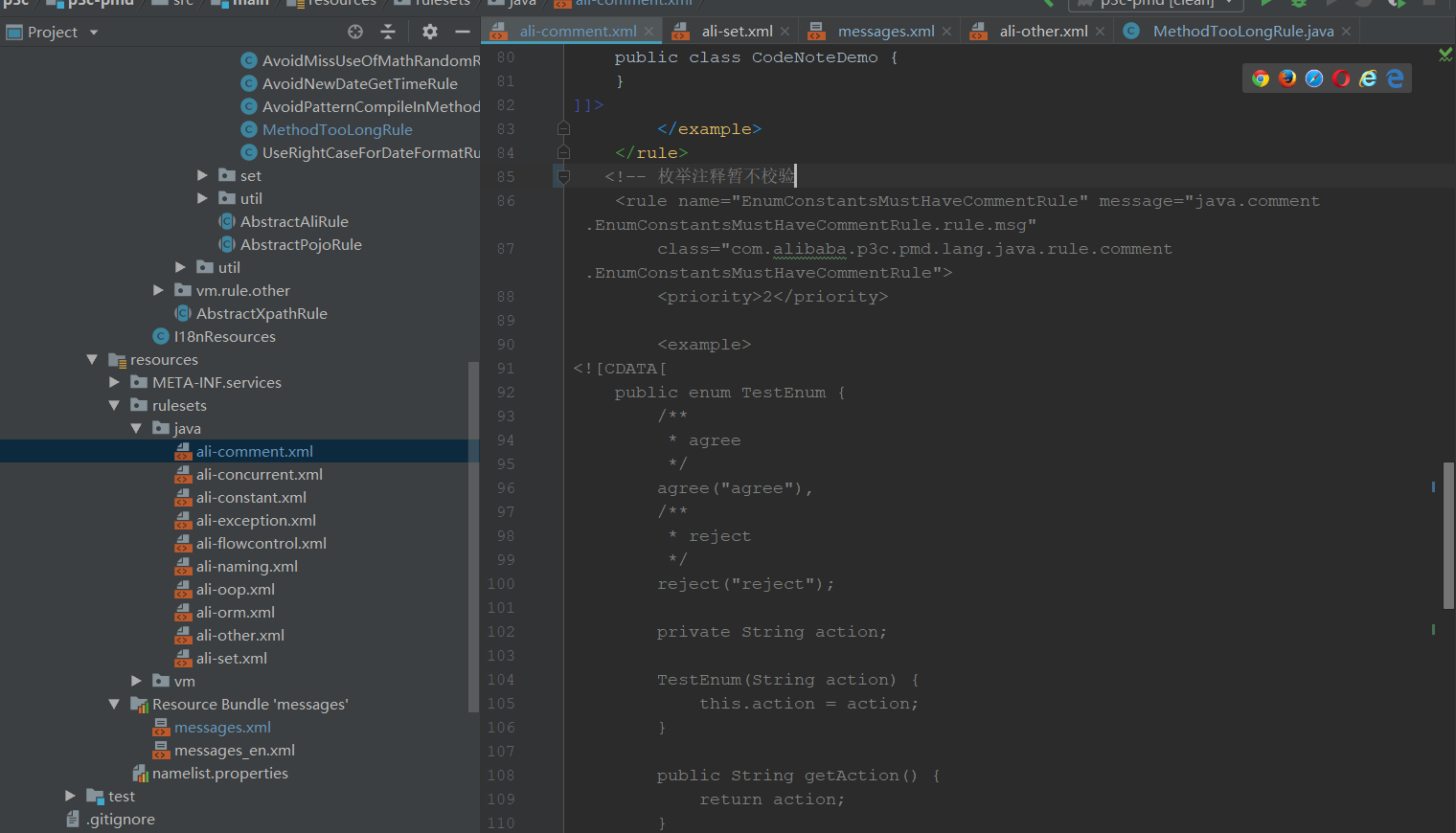
### 5.5.5 进入这个class，修改校验规则增加长度





### 5.5.6 此外可以取消掉一些注释

在对应的规则 ali-\*.xml文件中注释对应的 <rule></rule>节点，如注释掉枚举校验：

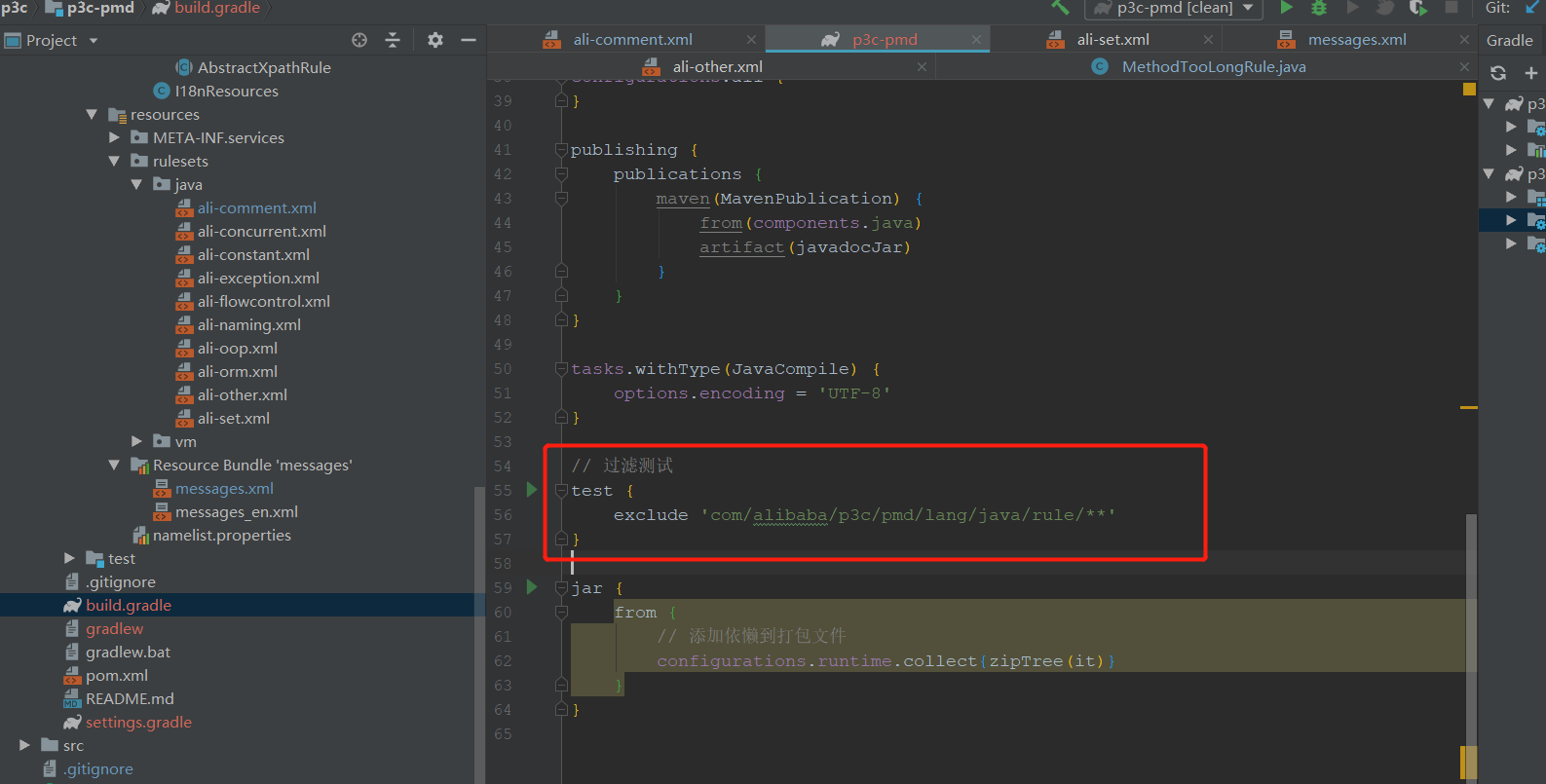


### 5.5.7 去除build的时候的执行test

gradle build的时候会执行所有的test，阿里P3C已经将所有的规则都有对应的test用例，这样做了调整你也需要做对应的测试用例调整，否则你在build的时候会构建不成功。我这里采用暴力做法，直接过滤掉测试：

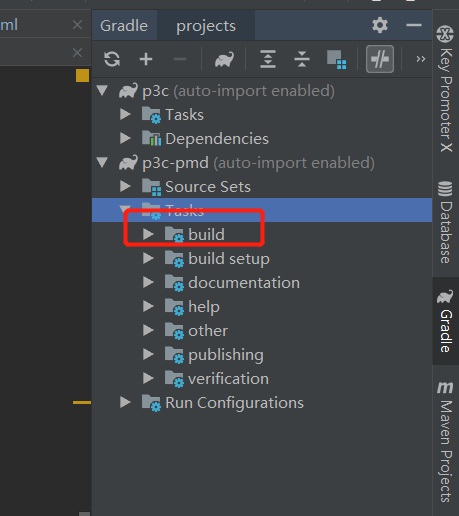
在build.gradle文件中添加：

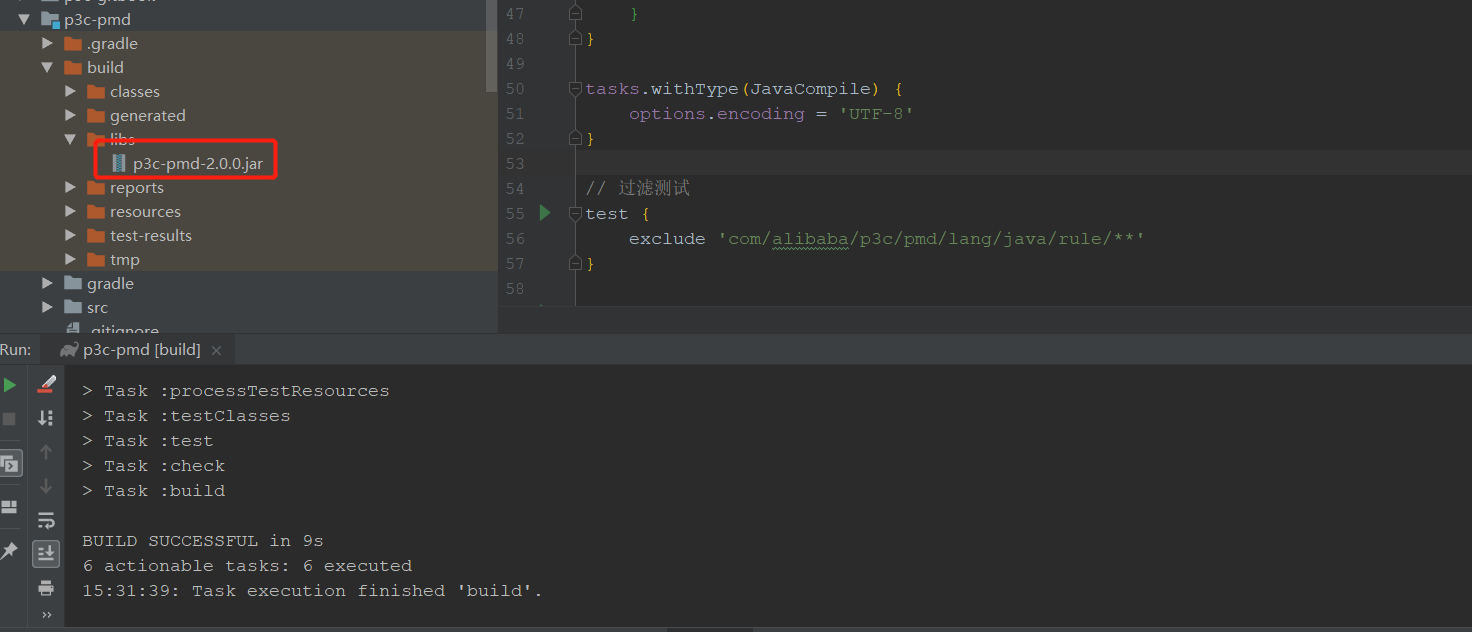
|  |
| --- |
| // 过滤测试 test *{* exclude 'com/alibaba/p3c/pmd/lang/java/rule/\*\*' *}* |



### 5.5.8 打包，参考[5.1章节](#_5.1._获取P3C-PMD文件)，

或者在idea中task执行build





# 参考文献：

<https://www.jianshu.com/p/b87ca8615c9c>

<https://my.oschina.net/leelance/blog/1922709>