GitHook的执行的脚本位于项目.git/hooks文件夹下，默认未开启的脚本以.smaple结尾，如果要激活某个脚本，只需要将后缀删除即可。

这些能够执行的脚本分为两类，一种是在本地执行另一种是在服务器执行。

**本地执行（有序执行）：**

pre-commit：commit时最先执行，一般用来进行代码检测，可被git commit --no-verify命令绕过

prepare-commit-msg：一般用来生成模板commit，需要将生成的模板写入.git/COMMIT\_EDITMSG文件，输入的内容会替换git commit –m”备注”命令中输入的备注。

commit-msg：一般用来检测最终的commit，但此时仍然可以修改COMMIT\_EDITMSG文件

post-commit：commit成功后进行触发，一般用作通知

触发器示例：

**不允许代码量过大**

实际中还要判断文件是否是类文件，具体的类解析一般是执行一个python脚本或者运行check-style.jar来验证文件是否符合要求

.git/hooks/pre-commit

#!/bin/bash

diffFile=$(git diff --name-only)  
for file in $diffFile  
do  
 line="$(awk '{print NR}' $file|tail -n1)"  
 if [ $line -gt 500 ]  
 then  
 echo "$file 代码行数超过500行"  
 exit 1  
 fi  
done  
exit 0

checkstyle.jar检测：

1.下载checkstyle.jar <https://github.com/checkstyle/checkstyle/releases>

2.配置检测规则：

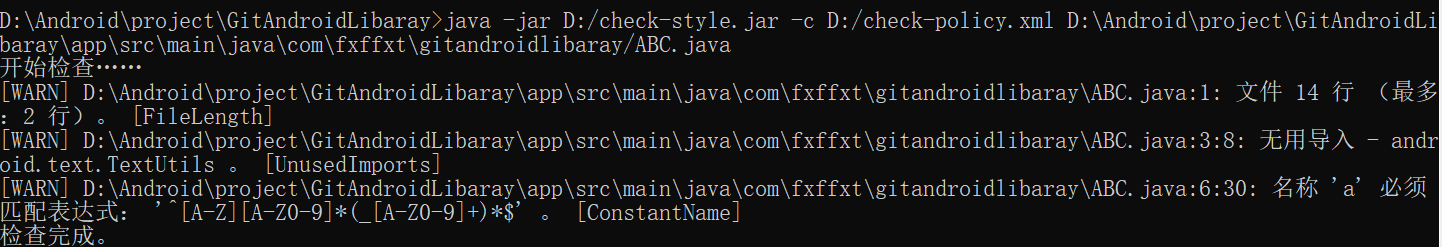
Checkstyle的检测规则是一个xml配置文档，可以根据需求进行配置（https://checkstyle.org/config.html）

<?xml version="1.0"?><!DOCTYPE module PUBLIC  
 "-//Puppy Crawl//DTD Check Configuration 1.3//EN"  
 "http://www.puppycrawl.com/dtds/configuration\_1\_3.dtd">  
  
<module name="Checker">  
 <property name="charset" value="utf-8" />  
 <property name="severity" value="warning" />  
 <property name="fileExtensions" value="java" />  
 <!—限制文件行数-->  
 <module name="FileLength">  
 <property name="max" value="2" />  
 </module>  
 <module name="TreeWalker">  
 <!-- 避免.\*,重复多余的和不使用的import-->  
 <module name="AvoidStarImport" />  
 <!-- 避免无效的import-->  
 <module name="UnusedImports" />  
 <!-- 常量全部用大写-->  
 <module name="ConstantName" />  
 <!-- 方法名称 -->  
 <module name="MethodName" />  
 </module>  
  
</module>

3.使用以下命令即可对文件进行检测：

java -jar checkstyle.jar -c 规则文件 待检测文件

检测结果样式：



可根据输出的结果来判断是否通过本次commit

如果内置的规则无法满足，则需要自定规则，比如MethodName规则中就无法排除对native方法的检测，需要自定义（待更新）

CheckStyle添加自定义规则：

首先需要了解DetailAST对象，DetailAST是一个类似于树的结构，该对象保存了整个Java文件的详细信息，可通过java -jar D:/check-style.jar –t xxx.java 的方式查看java文件信息

1. 定义CustomCommentCheck继承AbstractCheck，如果是对已有的规则进行加强，可以直接继承已有的规则，规则名+Check为具体规则实现类。
2. 确定要接收的类型：TokenTypes类中有类型常量

@Override  
public int[] getDefaultTokens() {//默认接收的类型  
 return getRequiredTokens();  
}  
  
@Override  
public int[] getAcceptableTokens() {  
 return getRequiredTokens();  
}  
  
@Override  
public int[] getRequiredTokens() {  
 return new int[]{TokenTypes.*SINGLE\_LINE\_COMMENT*};//接收单行注释  
}

@Override  
public boolean isCommentNodesRequired() {//开启注释接收[只在进行注释规则检测时使用]  
 return true;  
}

4．实现规则判断

@Override  
public void visitToken(DetailAST ast) {  
 DetailAST detailAST = ast.findFirstToken(TokenTypes.*COMMENT\_CONTENT*);  
 if (detailAST != null) {  
 String comment = detailAST.getText();  
 if (comment.length()>maxLength){  
 log(ast, "单行注释超过"+maxLength);  
 }  
 }  
}

以上实现了对单行注释(//)长度的限制

5．添加可通过外部配置的属性

CustomCommentCheck添加一个maxLength属性和对应的set方法

然后就可以在规则文件中使用了

<module name="com.example.check.CustomCheck">  
 <property name="maxLength" value="2"/>  
</module>

内部通过反射将property的值设置给CustomCheck

6．执行

将自定义的代码打包成myCheckStyle.jar

命令行运行：java –classpath ./checkstyle.jar;./myCheckStyle.jar com.puppycrawl.tools.checkstyle.Main –c ./checkstyle.xml ./JavaClassForCheck.java

7．还可以添加CustomChcek描述文件（非必要），可直接参考checkstyle.jar的内部实现

**组装commit模板**

可以对commit信息进行修饰，比如创建一个固定的commit格式

.git/hooks/prepare-commit-msg

#!/bin/bash  
echo "前缀 $(cat $1) 后缀" > $1  
exit 0

**commit信息进行检测**

对应一些不合适的commit信息可以驳回

./git/hooks/commit-msg

#!/bin/bash  
msg=$(cat $1)  
if [ ${#msg} -lt 8 ]  
then  
 echo "commit信息太短小于8个字符"  
 exit 1  
fi  
exit 0

补丁应用hook（用途待研究）

patch指某两次提交之间的差异，存储的是对代码的修改

applypathc-msg：接收单个参数， 包含请求合并信息的临时文件的名字，在执行git am xxx.patch后调用。可通过.git/rebase-apply/final-commit文件得到补丁的commit信息，同时可修改该文件的内容，产生新的commit提交信息，此时产生的提交不会走commit hook流程。

pre-applypatch：运行于应用补丁之后，产生提交之前，可以用它在提交前检查快照。

post-applypatch：用于通知，此时无法暂停工作流。

其他客户端hook（用途待研究）

pre-rebase：在变基操作之前执行，可以通过返回非零值来中止变基操作，。

post-merge：执行于git merge操作之后。

post-checkout：在进行切换分支或创建新分支时触发（git checkout）。可以根据项目环境来调整工作目录。 比如放入大文件、自动生成文档或进行其他操作。

pre-push：在git push操作成功前运行，可中止push流程。

服务端执行：

pre-receive：在服务器接收到更新推送时触发，可以检测推送信息是否合法，通过返回非零阻止推送更新到git仓库。

update：在分支更新时触发，如果一次提交有多个分支需要更新，会每个分支各执行一次，可中止流程。

post-update：整个更新结束后触发，无法中止流程。