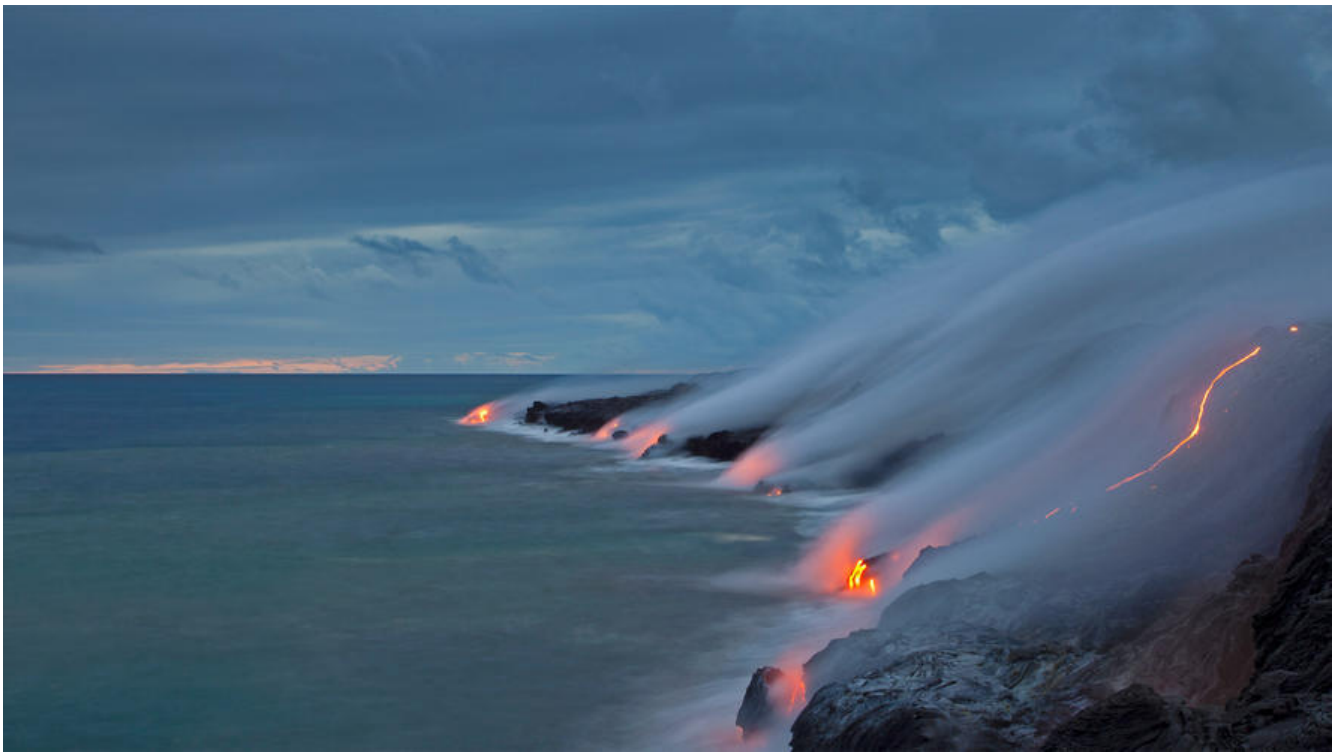
链滴

使用 Checkstyle 来规范我们的项目

作者：ymxfl

原文链接：https://ld246.com/article/1613814037505

来源网站：链滴



Checkstyle是什么

自从做了程序员，关于格式化的讨论就不曾中断过，到底什么才是正确的，什么才是错误的，到现在没有完整的定论。但随着时间发展，渐渐衍生出一套规范出来。没有什么绝对的正确和错误，关键在规范的定义。Checkstyle是一种开发工具，帮助程序员编写符合编码标准的Java代码。它自动化了检Java代码的过程，从而使人们避免了这项无聊(但重要)的任务。这使它成为希望实施编码标准的项目理想选择。Checkstyle是高度可配置的，并且可以支持几乎任何编码标准。提供了一个支持Sun代码范和谷歌Java风格的示例配置文件。

Checkstyle官网地址

https://checkstyle.sourceforge.io/index.html

在这里我们可以找到所有配置的详解。

checkstyle plugin与checkstyle的版本对应关系

http://maven.apache.org/plugins/maven-checkstyle-plugin/history.html

为什么要用，我们需要吗

无论我们是做产品还是做项目，养成一个良好的开发习惯对于每个程序员来说都是一种提升。代码写漂亮，自己看了也舒服，别人阅读起来也更方便。一个开发团队，少则三五人，多则数十人，每个人有不同的开发风格，有些新人写的代码规范性不强，看别人的代码都奇奇怪怪的。

如何使用

由于项目是使用maven来进行管理的，所以我在这里只写一下如何在maven中使用checkStyle。

对于maven中单模块应用

在pom中，添加 maven-checkstyle-plugin插件来进行代码样式校验。

```
<project>
  ...
  <reporting>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>3.1.0</version>
        <reportSets>
          <reportSet>
            <reports>
              <report>checkstyle</report>
            </reports>
          </reportSet>
        </reportSets>
      </plugin>
    </plugins>
  </reporting>
  ...
</project>
```

运行mvn site即可在目录下找到结果

target/site/checkstyle.html

也可以单独执行命令

mvn checkstyle:checkstyle

打开页面可以看到统计结果

最近更新: 2021-02-20 | 版本: 1.0.0

构建依靠 Maven

Checkstyle Results

The following document contains the results of Checkstyle 8.19 with /script/goose_checkstyle.xml ruleset.

Summary

Files	Info	Warnings	Errors
69	0	0	110

Files

File	I	W	E
cn/com/agree/goose/gateway/GooseGatewayApplication.java	0	0	2
cn/com/agree/goose/gateway/cache/ApiMappingDataCache.java	0	0	4
cn/com/agree/goose/gateway/httpclient/NettyHttpClientPlugin.java	0	0	3
cn/com/agree/goose/gateway/httpclient/WebClientPlugin.java	0	0	6
cn/com/agree/goose/gateway/httpclient/response/NettyClientResponsePlugin.java	0	0	2
cn/com/agree/goose/gateway/httpclient/response/WebClientResponsePlugin.java	0	0	2
cn/com/agree/goose/gateway/metrics/facade/MetricsTrackerFacade.java	0	0	2
cn/com/agree/goose/gateway/metrics/facade/executor/MetricsThreadPoolExecutor.java	0	0	1
cn/com/agree/goose/gateway/metrics/facade/handler/MetricsTrackerHandler.java	0	0	1
cn/com/agree/goose/gateway/metrics/spi/api/CounterMetricsTracker.java	0	0	2
cn/com/agree/goose/gateway/metrics/spi/api/GaugeMetricsTracker.java	0	0	2
cn/com/agree/goose/gateway/metrics/spi/api/HistogramMetricsTracker.java	0	0	2
cn/com/agree/goose/gateway/metrics/spi/api/SummaryMetricsTracker.java	0	0	2
cn/com/agree/goose/gateway/result/DefaultGooseResutEntity.java	0	0	1
cn/com/agree/goose/gateway/utils/StringParser.java	0	0	13
cn/com/agree/goose/gateway/utils/WebFluxResultUtils.java	0	0	6
cn/com/agree/goose/gateway/vo/StringParseResult.java	0	0	2
cn/com/agree/goose/gateway/web/configuration/GooseConfiguration.java	0	0	3
cn/com/agree/goose/gateway/web/filter/CrossFilter.java	0	0	5
cn/com/agree/goose/gateway/web/filter/DecryptAndSignFilter.java	0	0	35
cn/com/agree/goose/gateway/web/handler/GlobalErrorHandler.java	0	0	1
cn/com/agree/goose/gateway/websocket/WebsocketSyncDataService.java	0	0	2
cn/com/agree/goose/gateway/websocket/api/ApiMappingDataSubscriber.java	0	0	2
cn/com/agree/goose/gateway/websocket/handler/AbstractDataHandler.java	0	0	2
cn/com/agree/goose/gateway/websocket/handler/ApiMappingDataHandler.java	0	0	3
cn/com/agree/goose/gateway/websocket/handler/WebsocketDataHandler.java	0	0	2
cn/com/agree/goose/gateway/websocket/subscriber/ApiMappingDataAllSubscriber.java	0	0	2

原文链接：使用 Checkstyle 来规范我们的项目

Rules

Category	Rule	Violations	Severity
coding	FallThrough	2	Error
	ParameterAssignment	1	Error
	UnnecessaryParentheses	1	Error
	VariableDeclarationUsageDistance	1	Error
imports	AvoidStarImport	5	Error
	CustomImportOrder	11	Error
	UnusedImports	7	Error
indentation	Indentation	2	Error
javadoc	JavadocMethod • scope: "public" • ignoreMethodNamesRegex: "^assert.*$\|^verify.*$" • tokens: "METHOD_DEF, ANNOTATION_FIELD_DEF" • allowedAnnotations: "Override, Test, Before, After, BeforeClass, AfterClass, Parameterized, Parameters"	5	Error
	JavadocStyle	2	Error
	JavadocTagContinuationIndentation	1	Error
misc	FinalParameters	22	Error
naming	LocalFinalVariableName	6	Error
sizes	LineLength • max: "200"	1	Error
whitespace	EmptyLineSeparator • allowMultipleEmptyLines: "false" • allowMultipleEmptyLinesInsideClassMembers: "false"	24	Error
	SingleSpaceSeparator	1	Error
	WhitespaceAfter	7	Error
	WhitespaceAround	11	Error

原文链接：使用 Checkstyle 来规范我们的项目

Details

cn/com/agree/goose/gateway/GooseGatewayApplication.java

Severity	Category	Rule	Message	Line
Error	whitespace	EmptyLineSeparator	'package 前应有空行。	7
Error	javadoc	JavadocStyle	Javadoc 首句应以句号结尾。	18

cn/com/agree/goose/gateway/cache/ApiMappingDataCache.java

Severity	Category	Rule	Message	Line
Error	coding	FallThrough	从 switch 块的前一分支落入。	84
Error	sizes	LineLength	本行字符数 204个，最多：200个。	104
Error	coding	FallThrough	从 switch 块的前一分支落入。	105
Error	javadoc	JavadocMethod	缺少 Javadoc 。	150

cn/com/agree/goose/gateway/httpclient/NettyHttpClientPlugin.java

Severity	Category	Rule	Message	Line
Error	naming	LocalFinalVariableName	名称 'GooseContext' 必须匹配表达式： '^[a-z][a-zA-Z0-9]*$' 。	79
Error	whitespace	WhitespaceAfter	类型转换后应有空格。	122
Error	naming	LocalFinalVariableName	名称 'GooseContext' 必须匹配表达式： '^[a-z][a-zA-Z0-9]*$' 。	138

cn/com/agree/goose/gateway/httpclient/WebClientPlugin.java

Severity	Category	Rule	Message	Line
Error	naming	LocalFinalVariableName	名称 'GooseContext' 必须匹配表达式： '^[a-z][a-zA-Z0-9]*$' 。	69
Error	whitespace	WhitespaceAfter	类型转换后应有空格。	76
Error	indentation	Indentation	'method def' 子元素缩进了10个缩进符，应为8个。	79
Error	naming	LocalFinalVariableName	名称 'GooseContext' 必须匹配表达式： '^[a-z][a-zA-Z0-9]*$' 。	95
Error	whitespace	WhitespaceAround	'+' 前应有空格。	115
Error	whitespace	WhitespaceAround	'+' 后应有空格。	115

cn/com/agree/goose/gateway/httpclient/response/NettyClientResponsePlugin.java

Severity	Category	Rule	Message	Line
Error	whitespace	WhitespaceAfter	',' 后应有空格。	65
Error	naming	LocalFinalVariableName	名称 'GooseContext' 必须匹配表达式： '^[a-z][a-zA-Z0-9]*$' 。	94

cn/com/agree/goose/gateway/httpclient/response/WebClientResponsePlugin.java

Severity	Category	Rule	Message	Line
Error	whitespace	WhitespaceAfter	',' 后应有空格。	72
Error	naming	LocalFinalVariableName	名称 'GooseContext' 必须匹配表达式： '^[a-z][a-zA-Z0-9]*$' 。	84

Maven多模块的checkstyle配置

大多数情况下，我们会把项目的逻辑按照模块拆分出来，便于分离和解耦，项目脉络也更加清晰。在种情况下，我们为每个模块创建checkstyle任务，需要放到parent的pom里。

在父pom里添加：

```
<project>
  ...
    <build>
```

原文链接：使用 Checkstyle 来规范我们的项目

```
<pluginManagement>
  <plugins>
    <!-- compiler在maven声明周期内置，所以后面不用声明也可使用 -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.7.0</version>
      <configuration>
        <source>${java.version}</source>
        <target>${java.version}</target>
        <encoding>${project.build.sourceEncoding}</encoding>
      </configuration>
    </plugin>
<!-- 公共checkstyle标准配置，可以在子模块中覆盖，修改自定义选项 -->
<plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-checkstyle-plugin</artifactId>
      <version>3.1.0</version>
      <configuration>
        <configLocation>/script/goose_checkstyle.xml</configLocation>
        <includeTestSourceDirectory>false</includeTestSourceDirectory>
        <consoleOutput>true</consoleOutput>
        <encoding>${project.build.sourceEncoding}</encoding>
        <skip>false</skip>
        <!--这里是说当前这个配置是什么错误级别。如果配置的是error，那么扫描到不符合
件的，就是打印error。checkstyle里允许的错误级别有error, warning, info. -->
        <violationSeverity>error</violationSeverity>
        <!-- 是否打断命令执行，错误的时候会停止。否则，错误会生成报告，但不会阻止命
执行。 -->
        <failsOnError>false</failsOnError>
      </configuration>
      <executions>
        <execution>
          <id>validate</id>
          <phase>validate</phase>
          <goals>
            <goal>checkstyle</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
  </plugins>
</pluginManagement>
<plugins>
  <!--所有子模块都要执行的plugin-->
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-checkstyle-plugin</artifactId>
  </plugin>
  <plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-site-plugin</artifactId>
    <version>3.7.1</version>
  </plugin>
```

```
    </plugins>
  </build>
  <reporting>
  <!-- 所有子模块都要执行的报告 -->
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-checkstyle-plugin</artifactId>
        <version>3.1.0</version>
        <reportSets>
          <reportSet>
            <reports>
              <report>checkstyle</report>
            </reports>
          </reportSet>
        </reportSets>
      </plugin>
    </plugins>
  </reporting>
 ...
</project>
```

这里，maven-compiler-plugin不是必须的，事实上，maven会在项目生命周期中自动执行，我添这个插件的原因是在idea里的java编译级别需要根据这里来指定。

checkstyle plugin的配置有点多，需要仔细理解一下maven中plugin的含义。

build

在maven指令执行的时候会读取这个节点的配置，决定哪个plugin应该执行，怎么执行。

pluginManagement

这个是版本和共同配置的节点, 同 dependencyManagement, 为了约束子项目使用共同的配置。不的是，这个是指plugin。

plugin

这个表示一个插件，maven执行命令都可以用插件来理解。

plugin>configuration

对于plugin的配置，具体有哪些配置项要看具体的plugin。

executions>

execution

plugin应该什么时候执行

```
<execution>
  <id>validate</id>
  <phase>validate</phase>
  <goals>
    <goal>checkstyle</goal>
  </goals>
</execution>
```

id可以随意，phase则是需要绑定到lifecycle的phase中的哪个命令上，这里是绑定到validate上，即执行mvn validate的时候会执行本plugin。

goals>goal

一个plugin有多个goals，即任务，是指绑定执行哪个任务。这里是绑定 checkstyle。

checkstyle的错误级别

在checkstyle.xml的配置文件中，有

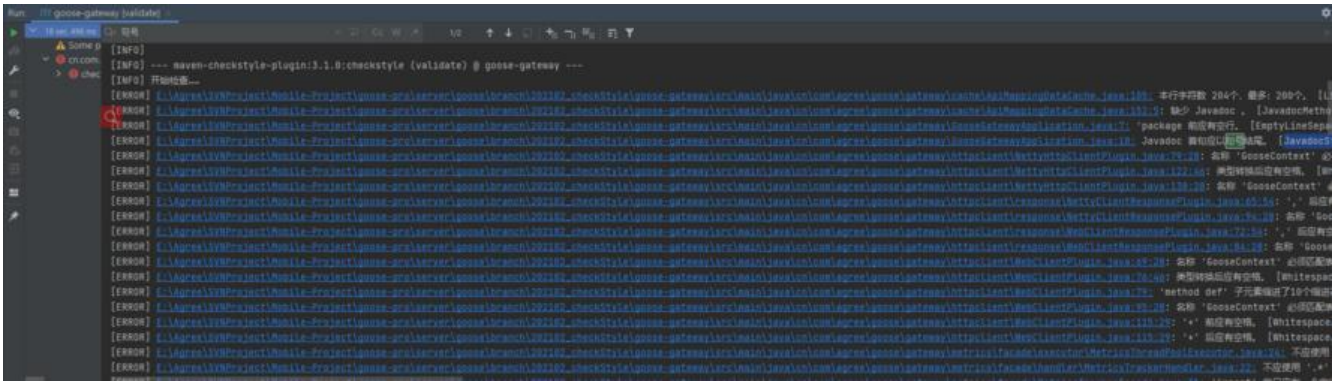
```
<property name="severity" value="error"/>
```

这里是说当前这个配置是什么错误级别。如果配置的是error，那么扫描到不符合条件的，就是打印err r。对于配置了

```
<failsOnError>true</failsOnError>
```

则会打断命令执行，错误的时候会停止。否则，错误会生成报告，但不

会阻止命令执行。



checkstyle里允许的错误级别有error, warning, info. 只有error并配置了failsOnError会打断命令执行。打断执行后会在对应的子模块的target下生成

```
target/checkstyle-result.xml
```

但不能生成html，或者可以选择声明plain，这个更不好看。

##附带一份我配置的checkstyle.xml

```
<?xml version="1.0"?>
<!--
  ~ Licensed to the Apache Software Foundation (ASF) under one or more
  ~ contributor license agreements.  See the NOTICE file distributed with
  ~ this work for additional information regarding copyright ownership.
  ~ The ASF licenses this file to You under the Apache License, Version 2.0
  ~ (the "License"); you may not use this file except in compliance with
  ~ the License.  You may obtain a copy of the License at
  ~
  ~     http://www.apache.org/licenses/LICENSE-2.0
  ~
  ~ Unless required by applicable law or agreed to in writing, software
  ~ distributed under the License is distributed on an "AS IS" BASIS,
  ~ WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
  ~ See the License for the specific language governing permissions and
  ~ limitations under the License.
```

```
 -->
<!--配置详解参考 https://checkstyle.sourceforge.io/config_coding.html -->
<!DOCTYPE module PUBLIC "-//Puppy Crawl//DTD Check Configuration 1.3//EN"
       "http://checkstyle.sourceforge.net/dtds/configuration_1_3.dtd">
<module name="Checker">
    <property name="charset" value="UTF-8"/>
    <property name="severity" value="error"/>
    <property name="fileExtensions" value="java, properties, xml"/>
    <module name="Header">
        <!-- 指定要处理的文件的文件类型扩展名为java。 -->
        <property name="fileExtensions" value="java"/>
    </module>
    <!-- 检查文件中是否含有'\t' -->
    <module name="FileTabCharacter">
        <property name="eachLine" value="true"/>
    </module>
    <!-- 文件长度不超过1500行 -->
    <module name="FileLength">
        <property name="max" value="1500"/>
    </module>
    <!-- 检查文件是否以一个\n结束 -->
    <module name="NewlineAtEndOfFile">
        <property name="lineSeparator" value="lf"/>
    </module>
    <!-- 检查property文件中是否有相同的key -->
    <module name="Translation"/>
    <module name="UniqueProperties"/>

    <module name="SeverityMatchFilter"/>
    <!-- 每个java文件一个语法树 -->
    <module name="TreeWalker">
        <!-- 命名规范 -->
        <!-- 包名的检查（只允许小写字母），默认^[a-z]+(\.[a-zA-Z_][a-zA-Z_0-9_]*)*$ -->
        <module name="PackageName">
            <property name="format" value="^[a-z]+(\.[a-z][a-z0-9]*)*$"/>
            <message key="name.invalidPattern" value="包名 ''{0}'' 要符合 ''{1}''格式."/>
        </module>
        <!-- Class或Interface名检查，默认^[A-Z][a-zA-Z0-9]*$-->
        <module name="TypeName">
            <property name="severity" value="warning"/>
            <message key="name.invalidPattern" value="名称 ''{0}'' 要符合 ''{1}''格式."/>
        </module>
        <!-- 检查方法名称是否符合指定的模式，默认"^[a-z][a-zA-Z0-9]*$" -->
        <module name="MethodName"/>
        <!-- 检查接口类型参数名是否符合指定的模式，默认"^[A-Z]$" -->
        <module name="InterfaceTypeParameterName"/>
        <!-- 检查类类型参数名是否符合指定的模式，默认"^[A-Z]$" -->
        <module name="ClassTypeParameterName"/>
        <!-- 检查方法类型参数名称是否符合指定的模式，默认"^[A-Z]$"-->
        <module name="MethodTypeParameterName"/>
        <!-- 检查常量名称是否符合指定的模式，默认"^[A-Z][A-Z0-9]*(_[A-Z0-9]+)*$" -->
        <module name="ConstantName"/>
        <!-- 检查静态的非final变量名是否符合指定的模式，默认"^[a-z][a-zA-Z0-9]*$" -->
        <module name="StaticVariableName"/>
```

```
        <!-- 检查实例变量名称是否符合指定的模式，默认"^[a-z][a-zA-Z0-9]*$" -->
        <module name="MemberName"/>
        <!-- 检查局部的非final变量名是否符合指定的模式。catch参数被认为是一个局部变量，默认"^[
-z][a-zA-Z0-9]*$" -->
        <module name="LocalVariableName"/>
        <!-- 检查局部final变量名称是否符合指定的模式。try语句中的catch参数和资源被认为是局部
、最终的变量，默认"^[a-z][a-zA-Z0-9]*$"-->
        <module name="LocalFinalVariableName"/>
        <!-- 检查方法参数名称是否符合指定的模式，默认"^[a-z][a-zA-Z0-9]*$" -->
        <module name="ParameterName"/>
        <!-- 检查catch参数名是否符合指定的模式，默认"^(e|t|ex|[a-z][a-z][a-zA-Z]+)$" -->
        <module name="CatchParameterName"/>
        <!-- 验证标识符名称中的缩写(连续大写字母)长度。-->
        <module name="AbbreviationAsWordInName">
            <!-- 指出目标标识符(类、接口、变量和方法名称中的缩写，等等)中允许连续大写字母的数
。 -->
            <property name="allowedAbbreviationLength" value="6"/>
        </module>

        <!-- 长度检查 -->
        <!-- 检查长匿名内部类长度，默认最大20 -->
        <module name="AnonInnerLength"/>
        <!-- 检查长方法和构造函数，默认最大150 -->
        <module name="MethodLength"/>
        <!-- 每行不超过200个字符 -->
        <module name="LineLength">
            <property name="max" value="200"/>
        </module>
        <!-- 检查在文件的外部(或根)级别声明的类型的数量,默认最大1 -->
        <module name="OuterTypeNumber"/>

        <!-- 检查初始化式的空白填充。默认非空 -->
        <module name="EmptyForInitializerPad"/>
        <!-- 检查迭代器的空填充，默认非空 -->
        <module name="EmptyForIteratorPad"/>
        <!-- 检查方法定义、构造函数定义、方法调用或构造函数调用的标识符之间的填充;以及参数列
的左括号。
        也就是说，如果标识符和左括号在同一行，检查标识符后面是否需要空格，或者这样的空格是禁
的。
        如果它们不在同一行上，报告违规，除非配置为允许换行。要允许在标识符之后换行，将属性all
wLineBreaks设置为true。 -->
        <module name="MethodParamPad"/>
        <!-- 检查圆括号内填充的策略;也就是左括号和右括号之前是否需要空格，默认非空格 -->
        <module name="ParenPad"/>
        <!-- 在类型转换时，不允许左圆括号右边有空格，也不允许与右圆括号左边有空格 -->
        <module name="TypecastParenPad"/>
        <!-- 检查所选语句是否没有行换行。默认情况下，此检查限制包装import和package语句，但
可以检查任何语句。 -->
        <module name="NoLineWrap"/>
        <!-- 检查如何在操作符上换行的策略。默认nl -->
        <module name="OperatorWrap"/>
        <!-- 用分隔符检查行包装。 -->
        <module name="SeparatorWrap">
            <property name="id" value="SeparatorWrapDot"/>
```

```
            <property name="tokens" value="DOT"/>
            <property name="option" value="nl"/>
        </module>
        <module name="SeparatorWrap">
            <property name="id" value="SeparatorWrapComma"/>
            <property name="tokens" value="COMMA"/>
            <property name="option" value="EOL"/>
        </module>
        <module name="SeparatorWrap">
            <property name="id" value="SeparatorWrapEllipsis"/>
            <property name="tokens" value="ELLIPSIS"/>
            <property name="option" value="EOL"/>
        </module>
        <module name="SeparatorWrap">
            <property name="id" value="SeparatorWrapArrayDeclarator"/>
            <property name="tokens" value="ARRAY_DECLARATOR"/>
            <property name="option" value="EOL"/>
        </module>
        <module name="SeparatorWrap">
            <property name="id" value="SeparatorWrapMethodRef"/>
            <property name="tokens" value="METHOD_REF"/>
            <property name="option" value="nl"/>
        </module>
        <!-- 检查通用标记周围的空格(尖括号)"<"和">"是正确的典型约定。该约定是不可配置的。 -->
        <module name="GenericWhitespace"/>
        <!-- 检查标记之前是否有空格。更具体地说，它检查前面是否有空格，或者(如果允许换行)前面
所有字符都是空格。 -->
        <module name="NoWhitespaceBefore"/>
        <!-- 检查标记后是否有空格。更具体地说，它检查后面是否有空格，或者(如果允许换行)后面的
中的所有字符都是空格。 -->
        <module name="NoWhitespaceAfter"/>
        <!-- 检查标记是否被空格包围。空构造函数，方法，类，枚举，接口，循环体(块)，形式的lamb
as -->
        <module name="WhitespaceAround"/>
        <!-- 检查标记后面是否有空格，但不检查空迭代器的分号后面是否有空格。 -->
        <module name="WhitespaceAfter"/>
        <!-- 检查非空白字符之间的空格不能超过一个。 -->
        <module name="SingleSpaceSeparator"/>
        <!-- 检查头、包、所有导入声明、字段、构造函数、方法、嵌套类、静态初始化器和实例初始
器之后的空行分隔符。 -->
        <module name="EmptyLineSeparator">
            <!-- 允许在类成员之间有多个空行，默认true，此处为false-->
            <property name="allowMultipleEmptyLines" value="false"/>
            <!-- 允许在类成员中有多个空行,默认true，此处为false-->
            <property name="allowMultipleEmptyLinesInsideClassMembers" value="false"/>
        </module>

        <!-- import检查-->
        <!-- 避免使用* -->
        <module name="AvoidStarImport"/>
        <!-- 避免使用静态导入 -->
        <module name="AvoidStaticImport">
            <property name="excludes"
                      value="org.junit.Assert.*,org.hamcrest.CoreMatchers.*,org.mockito.Mockito.*,org
```

```
mockito.ArgumentMatchers.*"/>
        </module>
        <!-- 检查是否从非法的包中导入了类 -->
        <module name="IllegalImport"/>
        <!-- 检查冗余的导入语句。 -->
        <module name="RedundantImport"/>
        <!-- 没用的import检查，比如：1.没有被用到2.重复的3.import java.lang的4.import 与该类在一个package的 -->
        <module name="UnusedImports"/>
        <!-- 检查导入声明组是否按用户指定的顺序出现。如果有一个导入，但是在配置中没有指定它组，那么这个导入应该放在导入列表的末尾。 -->
        <module name="CustomImportOrder"/>

        <!-- 检查语言元素上注解的位置。默认情况下，检查强制以在文档块之后和目标元素之前定位解，注解应该位于与目标元素分开的行中。如果注解不在同一行上，该检查还将验证注解是否与注释素处于同一缩进级别。 -->
        <module name="AnnotationLocation">
            <property name="id" value="AnnotationLocationMostCases"/>
            <property name="tokens" value="CLASS_DEF, INTERFACE_DEF, ENUM_DEF, METHOD DEF, CTOR_DEF"/>
        </module>
        <module name="AnnotationLocation">
            <property name="id" value="AnnotationLocationVariables"/>
            <property name="tokens" value="VARIABLE_DEF"/>
            <!-- 允许注解与目标元素位于同一行。 -->
            <property name="allowSamelineMultipleAnnotations" value="true"/>
        </module>
        <!-- 检查注释中元素的样式。 -->
        <module name="AnnotationUseStyle"/>
        <!-- 验证当@inheritDoc javadoc标记存在时是否存在@Override注释。 -->
        <module name="MissingOverride"/>
        <!-- 允许指定@SuppressWarnings不允许抑制哪些警告。您还可以指定一个tokenttype列表配置的警告不能被抑制。 -->
        <module name="SuppressWarnings"/>
        <!-- 验证注释@Deprecated和Javadoc标记@Deprecated是否同时存在。 -->
        <module name="MissingDeprecated"/>
        <!-- 维护一组来自@SuppressWarnings注释的检查抑制。它允许防止Checkstyle报告来自注为@SuppressWarnings的代码部分的违规，并使用被排除的检查名称。还可以为需要抑制的检查名定义别名。 -->
        <module name="SuppressWarningsHolder"/>

        <!-- 修饰符检查 -->
        <!-- 检查修饰符的顺序是否遵照java语言规范，默认public、protected、private、abstract、efault、static、final、transient、volatile、synchronized、native、strictfp -->
        <module name="ModifierOrder"/>
        <!-- 检查接口和annotation中是否有多余修饰符，如接口方法不必使用public -->
        <module name="RedundantModifier"/>

        <!-- Coding -->
        <!-- 检查数组初始化是否包含末尾逗号。 -->
        <module name="ArrayTrailingComma"/>
        <!-- 检查类是否覆盖了equals(java.lang.Object)。 -->
        <module name="CovariantEquals"/>
        <!-- 检查default是否在switch语句中的所有case之后。 -->
```

```
        <module name="DefaultComesLast"/>
        <!-- 检查类、记录或接口声明的各个部分是否按照Java编程语言的代码约定建议的顺序出现。--

        <module name="DeclarationOrder"/>
        <!-- 检查空的代码段 -->
        <module name="EmptyStatement"/>
        <!-- 检查字符串字面值的任何组合是否位于equals()比较的左侧。还检查分配给某些字段的字符
字面量(例如someString。= (anotherString = “文本”))。 -->
        <module name="EqualsAvoidNull"/>
        <!-- 检查任何类或对象成员的类型值是否显式初始化为默认值(对象引用为null，数值类型为0,b
olean类型为char, false)。 -->
        <module name="ExplicitInitialization"/>
        <!-- 检查switch代码的case中是否缺少break，return，throw和continue。 -->
        <module name="FallThrough"/>
        <!-- 检查优先使用工厂方法的非法实例化。 -->
        <module name="IllegalInstantiation"/>
        <!-- 检查某些异常类型不会出现在catch语句中。 -->
        <module name="IllegalCatch">
            <!-- 指定要拒绝的异常类名。 -->
            <property name="illegalClassNames" value="Error,Throwable,java.lang.Error,java.lang
Throwable"/>
        </module>
        <!-- 检查指定的类型是否声明为要抛出。声明一个方法会引发java.lang.Error或java.lang.Runti
eException几乎是不可接受的。 -->
        <module name="IllegalThrows"/>
        <!-- 检查特定的类或接口是否从未被使用。-->
        <module name="IllegalType">
            <!-- 方法、参数、变量-->
            <property name="tokens" value="METHOD_DEF,PARAMETER_DEF,VARIABLE_DEF"/>
        </module>
        <!-- 检查非法的分隔符的下个字符。-->
        <module name="IllegalTokenText">
            <property name="tokens" value="STRING_LITERAL, CHAR_LITERAL"/>
            <property name="format"
                      value="\\u00(09|0(a|A)|0(c|C)|0(d|D)|22|27|5(C|c))|\\(0(10|11|12|14|15|42|47)|134)
/>
            <property name="message"
                      value="考虑使用特殊转义序列来代替八进制值或Unicode转义值."/>
        </module>
        <!-- 检查switch语句是否有default -->
        <module name="MissingSwitchDefault"/>
        <!-- 检查for循环控制变量是否在for块中被修改。-->
        <module name="ModifiedControlVariable"/>
        <!-- 检查每个变量声明是否在自己的语句中并在自己的行中。-->
        <module name="MultipleVariableDeclarations"/>
        <!-- 将嵌套的if-else块限制到指定的深度。 -->
        <module name="NestedIfDepth">
            <property name="max" value="3"/>
        </module>
        <!-- 将嵌套的try-catch-finally块限制到指定的深度。默认1 -->
        <module name="NestedTryDepth"/>
        <!-- 检查是否从Object 类中重写克隆方法。 -->
        <module name="NoClone"/>
        <!-- 检查没有带零个参数的方法finalize。 -->
```

```
        <module name="NoFinalizer"/>
        <!-- 检查覆盖的clone()方法是否调用super.clone()。不检查native方法，因为它们没有可能的ja
a定义的实现。-->
        <module name="SuperClone"/>
        <!-- 检查覆盖的finalize()方法是否调用super.finalize()。不检查native方法，因为它们没有可
的java定义的实现。 -->
        <module name="SuperFinalize"/>
        <!-- 检查每行是否只有一条语句。-->
        <module name="OneStatementPerLine"/>
        <!-- 检查是否将重载方法分组在一起。重载的方法具有相同的名称但不同的签名，其中签名可
因输入参数的数量或输入参数的类型或两者的数量而不同。-->
        <module name="OverloadMethodsDeclarationOrder"/>
        <!-- 确保类有包声明，以及(可选的)包名是否与源文件的目录名匹配。-->
        <module name="PackageDeclaration"/>
        <!-- 禁止给参数赋值。-->
        <module name="ParameterAssignment"/>
        <!-- 检查过于复杂的布尔表达式。目前发现代码，如if (b == true)， b || true， !false，等等。
->
        <module name="SimplifyBooleanExpression"/>
        <!-- 检查过于复杂的布尔返回语句。 -->
        <module name="SimplifyBooleanReturn"/>
        <!-- 检查字符串字面值是否与==或!=一起使用。因为==将比较对象引用，而不是字符串的实
值，所以应该使用String.equals()。-->
        <module name="StringLiteralEquality"/>
        <!-- 检查语句或表达式中是否使用了不必要的括号。-->
        <module name="UnnecessaryParentheses"/>
        <!-- 检查变量声明与第一次使用之间的距离。注意:变量声明/初始化语句在计算长度时不被计算
-->
        <module name="VariableDeclarationUsageDistance"/>

        <!-- 代码块检查 -->
        <!-- 检查空块。此检查不验证顺序块。 -->
        <module name="EmptyBlock"/>
        <!-- 检查空的catch块。默认情况下，check允许包含任何注释的空catch块。 -->
        <module name="EmptyCatchBlock">
            <!-- 如果需要异常的变量名，或者忽略，或者有任何注释，则配置检查来抑制空catch块-->
            <property name="exceptionVariableName" value="expected|ignore"/>
        </module>
        <!-- 查找嵌套块(代码中自由使用的块)。 -->
        <module name="AvoidNestedBlocks"/>
        <!-- 检查代码块周围的括号。 -->
        <module name="NeedBraces"/>
        <!-- 检查左大括号位置 -->
        <module name="LeftCurly"/>
        <!-- 检查代码块的左花括号('{')位置。 -->
        <module name="RightCurly"/>

        <!-- 类设计检查 -->
        <!-- 检查只有private构造函数的类是否声明为final -->
        <module name="FinalClass"/>
        <!-- cannot recognize for lombok @NoArgsConstructor(access = AccessLevel.PRIVATE),
ust ignore -->
        <!--<module name="HideUtilityClassConstructor"/>-->
        <!-- 检查每个顶级类、接口、枚举或注释是否驻留在自己的源文件中。-->
```

```
        <module name="OneTopLevelClass"/>
        <!-- 检查接口是否仅定义类型 -->
        <module name="InterfaceIsType"/>
        <!-- 嵌套的(内部的)类/接口在初始化和静态初始化块、方法、构造函数和字段声明之后，在主(级)类的底部声明。-->
        <module name="InnerTypeLast"/>
        <!-- 检查类成员的可见性。只有static final、不可变或由指定注释成员注释的才可以是public;非设置了protectedAllowed或packageAllowed属性，否则其他类成员必须是私有的。-->
        <module name="VisibilityModifier"/>
        <!-- 确保异常类(名称与某些正则表达式一致的类以及名称与其他正则表达式一致的显式扩展类)不可变的，也就是说，它们只有final字段。-->
        <module name="MutableException"/>
        <!-- 将抛出语句限制为指定的计数。带有"Override"或"java.lang.Override "的方法。从验证中过注释，因为当前类不能更改这些方法的签名。-->
        <module name="ThrowsCount"/>

        <!-- 定义检查 -->
        <!-- 检查数组类型定义的样式 -->
        <module name="ArrayTypeStyle"/>
        <!-- 检查long型定义是否有大写的“L” -->
        <module name="UpperEll"/>
        <!-- 限制使用Unicode转义(例如/u221e)。允许对不可打印的控制字符使用转义。此外，可以此检查配置为如果存在跟踪注释，则允许使用转义。通过这个选项，如果字面只包含转义，可以允许用转义。-->
        <module name="AvoidEscapedUnicodeCharacters"/>
        <!-- 检查其他令牌下面的受限令牌。-->
        <module name="DescendantToken"/>
        <!-- 检查方法、构造函数、catch和for-each块的参数是否为final。没有检查接口、抽象和nativ方法:final关键字对于接口、抽象和native方法参数没有意义，因为没有代码可以修改参数。 -->
<!--        <module name="FinalParameters"/>-->
        <!-- 检查Java代码的正确缩进。-->
        <module name="Indentation"/>
        <!-- 检查外部类型名称和文件名是否匹配。例如，类Foo必须在一个名为Foo.java的文件中。-->
        <module name="OuterTypeFilename"/>
        <!-- 检查TODO:注释。实际上，它是Java注释上的一个通用正则表达式匹配器。要检查Java注中的其他模式，设置format属性。-->
        <module name="TodoComment"/>
        <!-- 检查以确保要求注释是一行上唯一的东西。-->
        <module name="TrailingComment"/>

        <!-- Javadoc Comments -->
        <!-- 检查javadoc块标记或javadoc标记的顺序。-->
        <module name="AtclauseOrder">
            <property name="tagOrder" value="@param, @return, @throws, @deprecated"/>
        </module>
        <!-- 检查块标签后面是否有描述。-->
        <module name="NonEmptyAtclauseDescription"/>
        <!-- 检查Javadoc段落。-->
        <module name="JavadocParagraph"/>
        <!-- 验证Javadoc注释，以确保它们格式良好。 -->
        <module name="JavadocStyle"/>
        <!-- 检查块标记中延续行的缩进。-->
        <module name="JavadocTagContinuationIndentation"/>
        <!-- 检查Javadoc块是否可以放在一行中，并且不包含块标记。至少包含一个块标记的Javadoc
```

```
释应该用几行进行格式化。-->
        <module name="SingleLineJavadoc"/>
<!--        <module name="SummaryJavadoc"/>-->
        <!-- 检查方法和构造函数的javadoc -->
        <module name="JavadocMethod">
            <property name="scope" value="public"/>
            <property name="allowedAnnotations"
                      value="Override, Test, Before, After, BeforeClass, AfterClass, Parameterized, Par
meters"/>
            <property name="ignoreMethodNamesRegex" value="^assert.*$|^verify.*$"/>
            <property name="tokens" value="METHOD_DEF, ANNOTATION_FIELD_DEF"/>
        </module>

        <!-- Filters -->
        <module name="SuppressionCommentFilter"/>
        <module name="SuppressWithNearbyCommentFilter"/>
    </module>
</module>
```