介绍

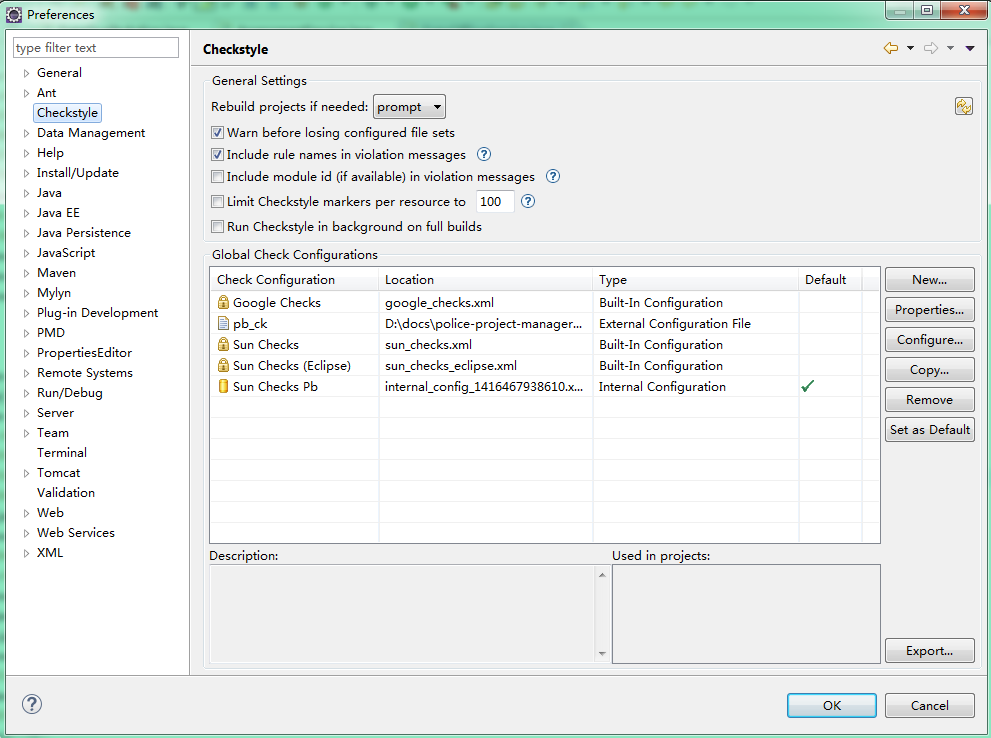
CheckStyle是SourceForge下的一个项目，提供了一个帮助JAVA开发人员遵守某些编码规范的工具。它能够自动化代码规范检查过程，从而使得开发人员从这项重要，但是枯燥的任务中解脱出来。   
CheckStyle默认提供一下主要检查内容：   
　　•Javadoc注释   
　　•命名约定   
　　•标题   
　　•Import语句   
　　•体积大小   
　　•空白   
　　•修饰符   
　  •块   
　　•代码问题   
　　•类设计   
　　•混合检查（包活一些有用的比如非必须的 System.out和printstackTrace）   
　　从上面可以看出，CheckStyle提供了大部分功能都是对于代码规范的检查，而没有提供象PMD和Jalopy那么多的增强代码质量和修改代码的功能。但是，对于团队开发，尤其是强调代码规范的公司来说，它的功能已经足够强大。

# 安装

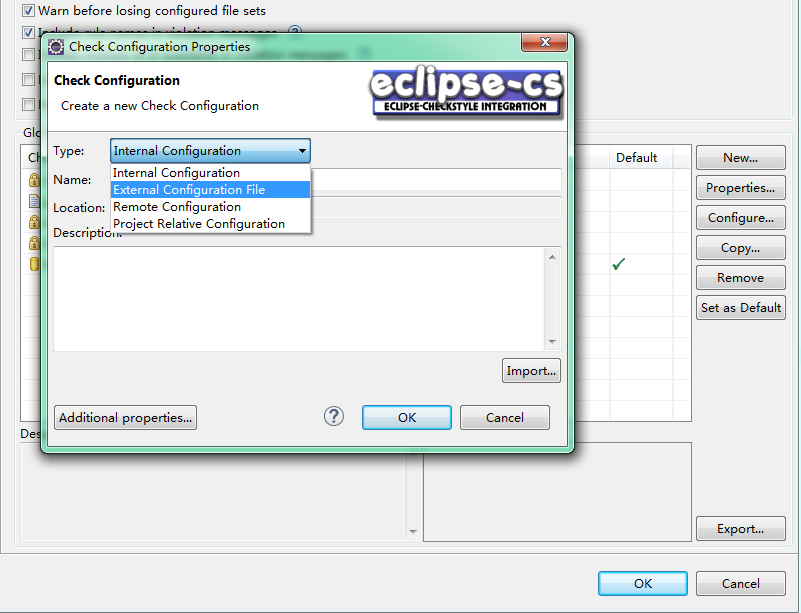
Eclipse插件安装和使用

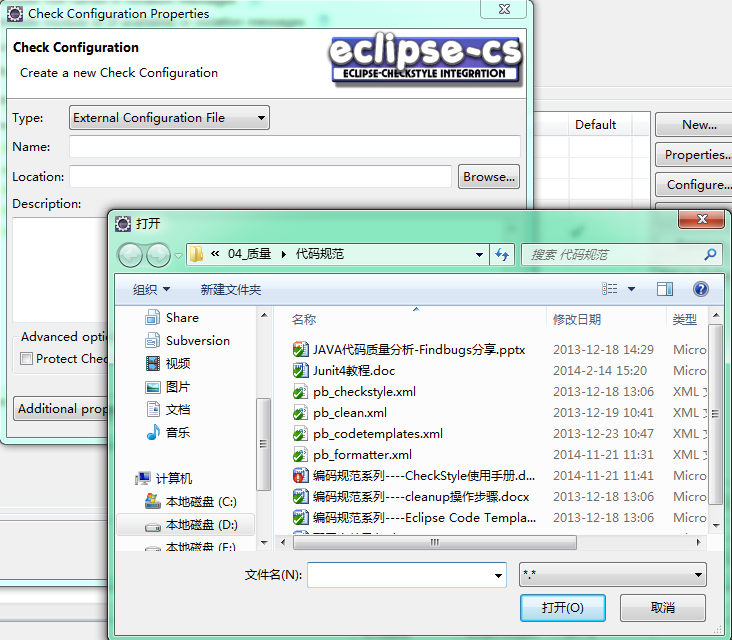
Eclipse install site:

<http://sourceforge.net/projects/checkclipse/files/checkclipse/>

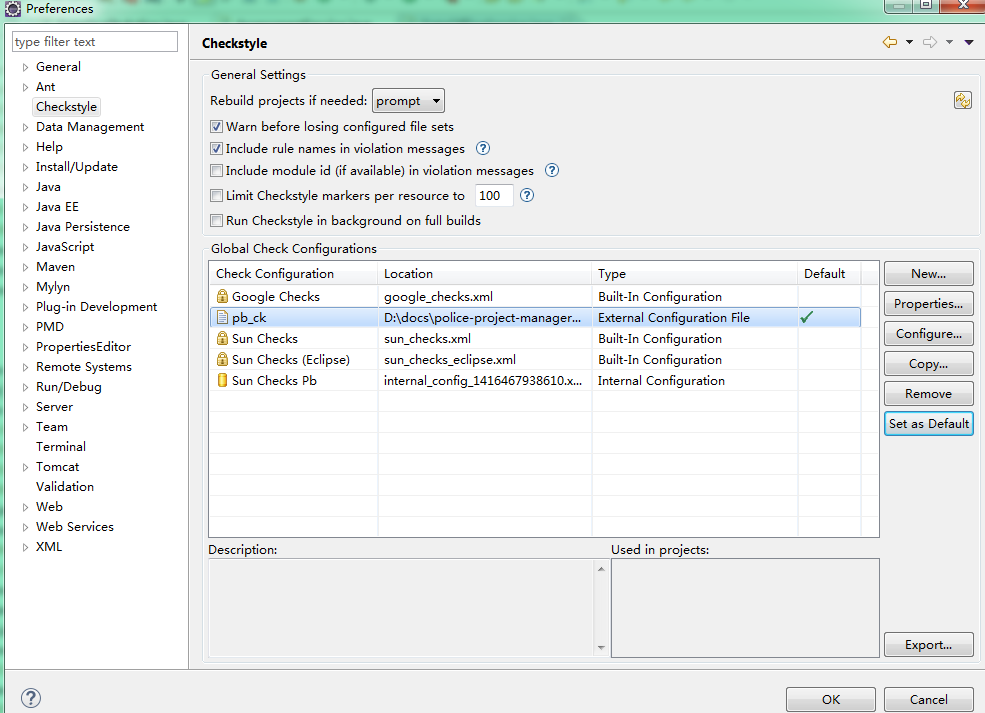
步骤一：http://sourceforge.net/projects/checkclipse/下载checkstyle的eclipse插件checkclipse。下载后，将包放入eclipse的plugins文件夹下，然后重启eclipse。在Windows—>preferences下找到checkclipse。如下图:   


导入配置文件

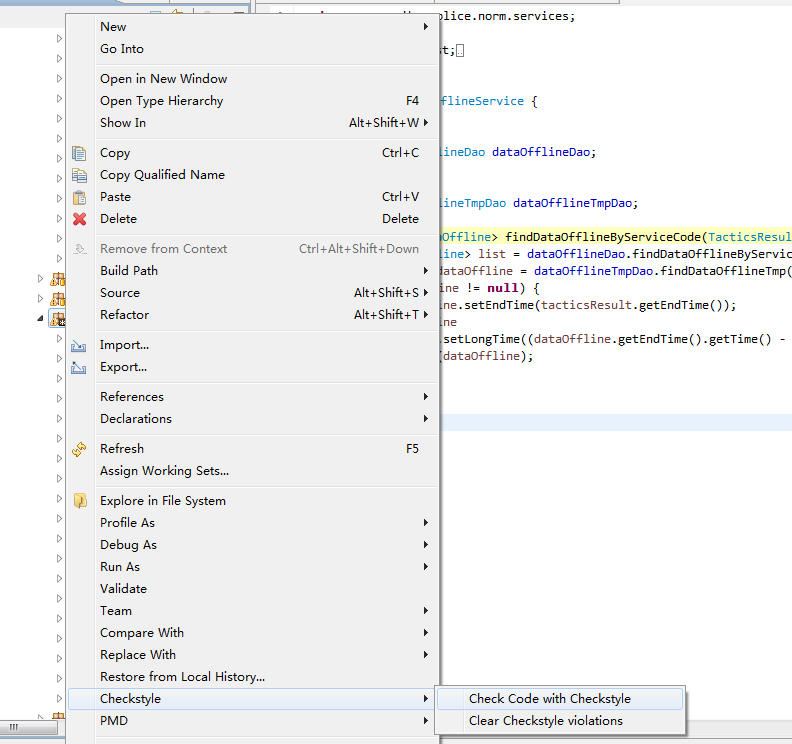




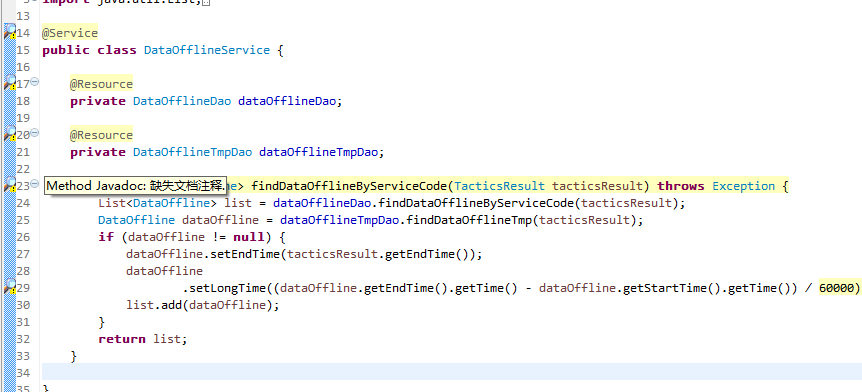
选中导入文件并设置为默认的



检查



检查结果



# Maven插件安装和使用

首先，修改要检查代码库top级的pom.xml文件，在build部分配置CheckStyle的Maven插件，以便于下载安装对应版本的插件（Maven会自动从其镜像库中下载），方法如下：

Java代码  [收藏代码](javascript:void())

1. <project>
2. ...
3. <build>
4. <plugins>
5. <plugin>
6. <groupId>org.apache.maven.plugins</groupId>
7. <artifactId>maven-checkstyle-plugin</artifactId>
8. <version>2.3</version>
9. </plugin>
10. </plugins>
11. </build>
12. ...
13. </project>

maven-checkstyle-plugin的最新版本为2.5，其对应的CheckStyle核心版本为5.0；maven-checkstyle-plugin 2.3对应的CheckStyle核心版本为4.4。查看插件的pom文件，可看到如下内容，其中的版本号就为对应的CheckStyle的版本号。

Java代码  [收藏代码](javascript:void())

1. <dependency>
2. <groupId>checkstyle</groupId>
3. <artifactId>checkstyle</artifactId>
4. <version>4.4</version>
5. </dependency>

接下来，将自定义的规则配置文件拷贝到top级目录，在reporting部分的CheckStyle插件配置中引用配置。

Java代码  [收藏代码](javascript:void())

1. <reporting>
2. <plugins>
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-checkstyle-plugin</artifactId>
6. <configuration>
7. <configLocation>my\_checks.xml</configLocation>
8. </configuration>
9. </plugin>
10. </plugins>
11. </reporting>

也可以将配置文件放在子文件夹下，配置中带上相对路径即可。

Java代码  [收藏代码](javascript:void())

1. <reporting>
2. <plugins>
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-checkstyle-plugin</artifactId>
6. <configuration>
7. <configLocation>build-tools/src/main/resources/xx/my\_checks.xml</configLocation>
8. </configuration>
9. </plugin>
10. </plugins>
11. </reporting>

如果使用插件自带的规则文件，可以作如下配置。maven-checkstyle-plugin插件自带的规则有sun\_checks.xml、maven\_checks.xml等，可查看插件包。

Java代码  [收藏代码](javascript:void())

1. <reporting>
2. <plugins>
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-checkstyle-plugin</artifactId>
6. <configuration>
7. <configLocation>config/maven\_checks.xml</configLocation>
8. </configuration>
9. <version>2.3</version>
10. </plugin>
11. </plugins>
12. </reporting>

在reporting部分增加jxr插件，生成代码报告，这样在CheckStyle报告中点击问题对应的链接就可以直接看到出错的代码。 

Java代码  [收藏代码](javascript:void())

1. <reporting>
2. <plugins>
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-checkstyle-plugin</artifactId>
6. <configuration>
7. <configLocation>my\_checks.xml</configLocation>
8. </configuration>
9. <version>2.3</version>
10. </plugin>
11. <plugin>
12. <groupId>org.apache.maven.plugins</groupId>
13. <artifactId>maven-jxr-plugin</artifactId>
14. </plugin>
15. </plugins>
16. </reporting>

在build和reporting部分增加javadoc插件，如果pom文件中已经配置，则只需作相应修改。charset、encoding、docencoding配置用于解决生成的javadoc文件中文乱码问题；aggregate配置为true则javadoc报告会集中显示所有子模块的javadoc。 

Java代码  [收藏代码](javascript:void())

1. <reporting>
2. <plugins>
3. <plugin>
4. <groupId>org.apache.maven.plugins</groupId>
5. <artifactId>maven-javadoc-plugin</artifactId>
6. <version>2.4</version>
7. <configuration>
8. <aggregate>**true**</aggregate>
9. <charset>UTF-8</charset>
10. <encoding>UTF-8</encoding>
11. <docencoding>UTF-8</docencoding>
12. </configuration>
13. </plugin>
14. <plugin>
15. <groupId>org.apache.maven.plugins</groupId>
16. <artifactId>maven-checkstyle-plugin</artifactId>
17. <configuration>
18. <configLocation>my\_checks.xml</configLocation>
19. </configuration>
20. <version>2.3</version>
21. </plugin>
22. <plugin>
23. <groupId>org.apache.maven.plugins</groupId>
24. <artifactId>maven-jxr-plugin</artifactId>
25. </plugin>
26. </plugins>
27. </reporting>

在maven插件中使用 install命令将pom文件中配置的插件下载安装到本地，然后使用checkstyle:checkstyle命令进行检查并生成报告，运行完毕，各项目目录下会生成target目录，target\site\checkstyle.html即为该项目的问题报告。   
需要注意的是checkstyle:checkstyle仅生成CheckStyle相关报告，因此不能从报告中直接链接到错误代码；需要同时生成jxr源代码，使用site。   
如果运行checkstyle:checkstyle或site过程中出现如下错误，则应该修改CheckStyle规则配置文件，去除其中的中文字符。

Java代码  [收藏代码](javascript:void())

1. “[ERROR] BUILD ERROR
2. [INFO] ------------------------------------------------------------------------
3. [INFO] An error has occurred in Checkstyle report generation.
5. Embedded error: Failed during checkstyle configuration
6. Invalid **byte** 1 of 1-**byte** UTF-8 sequence.
7. ”

最佳实践   
自定义的checkstyle配置文件   
　　以下代码是自定义的checkstyle配置文件内容，相关说明都已经用注释形式写在文件中。代码如下：

Java代码  [收藏代码](javascript:void())

1. <!DOCTYPE module PUBLIC
2. "-//Puppy Crawl//DTD Check Configuration 1.2//EN"
3. "http://www.puppycrawl.com/dtds/configuration\_1\_2.dtd">
4. <module name="Checker">
6. <!--
7. 重复代码的检查，超过8行就认为重复，UTF-8格式 本检查一定要放在"TreeWalker"节点前，否则在
8. Checkclipse中会无法使用。(在ant下可以)
9. -->
10. <module name="StrictDuplicateCode">
11. <property name="min" value="8" />
12. <property name="charset" value="UTF-8" />
13. </module>
15. <module name="TreeWalker">
17. <!-- javadoc的检查 -->
18. <!-- 检查所有的**interface**和**class** -->
19. <module name="JavadocType" />
21. <!-- 命名方面的检查，它们都使用了Sun官方定的规则。 -->
22. <!-- 局部的**final**变量，包括**catch**中的参数的检查 -->
23. <module name="LocalFinalVariableName" />
24. <!-- 局部的非**final**型的变量，包括**catch**中的参数的检查 -->
25. <module name="LocalVariableName" />
26. <!-- 包名的检查（只允许小写字母） -->
27. <module name="PackageName">
28. <property name="format" value="^[a-z]+(\.[a-z][a-z0-9]\*)\*$" />
29. </module>
30. <!-- 仅仅是**static**型的变量（不包括**static** **final**型）的检查 -->
31. <module name="StaticVariableName" />
32. <!-- 类型(Class或Interface)名的检查 -->
33. <module name="TypeName" />
34. <!-- 非**static**型变量的检查 -->
35. <module name="MemberName" />
36. <!-- 方法名的检查 -->
37. <module name="MethodName" />
38. <!-- 方法的参数名 -->
39. <module name="ParameterName " />
40. <!-- 常量名的检查 -->
41. <module name="ConstantName" />
43. <!-- **import**方面的检查 -->
44. <!-- **import**中避免星号"\*" -->
45. <module name="AvoidStarImport" />
46. <!--
47. 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**
48. 与该类在同一个**package**的
49. -->
50. <module name="UnusedImports" />

53. <!-- 长度方面的检查 -->
54. <!-- 文件长度不超过1500行 -->
55. <module name="FileLength">
56. <property name="max" value="1500" />
57. </module>
58. <!-- 每行不超过120个字-->
59. <module name="LineLength">
60. <property name="max" value="120" />
61. </module>
62. <!-- 方法不超过30行 -->
63. <module name="MethodLength">
64. <property name="tokens" value="METHOD\_DEF" />
65. <property name="max" value="30" />
66. </module>
67. <!-- 方法的参数个数不超过3个。 并且不对构造方法进行检查-->
68. <module name="ParameterNumber">
69. <property name="max" value="3" />
70. <property name="tokens" value="METHOD\_DEF" />
71. </module>
73. <!-- 空格检查  -->
74. <!-- 允许方法名后紧跟左边圆括号"(" -->
75. <module name="MethodParamPad" />
76. <!-- 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 -->
77. <module name="TypecastParenPad" />
78. <!-- 不允许使用"tab"键  -->
79. <module name="TabCharacter" />
81. <!-- 关键字 -->
82. <!--
83. 每个关键字都有正确的出现顺序。比如 **public** **static** **final** XXX 是对一个常量的声明。如果使用 **static**
84. **public** **final** 就是错误的
85. -->
86. <module name="ModifierOrder" />
87. <!-- 多余的关键字 -->
88. <module name="RedundantModifier" />
90. <!-- 对区域的检查 -->
91. <!-- 不能出现空白区域 -->
92. <module name="EmptyBlock" />
93. <!-- 所有区域都要使用大括号。 -->
94. <module name="NeedBraces" />
95. <!-- 多余的括号 -->
96. <module name="AvoidNestedBlocks">
97. <property name="allowInSwitchCase" value="true" />
98. </module>
100. <!-- 编码方面的检查 -->
101. <!-- 不许出现空语句 -->
102. <module name="EmptyStatement" />
103. <!-- 每个类都实现了equals()和hashCode() -->
104. <module name="EqualsHashCode" />
105. <!-- 不许使用**switch**,"a++"这样可读性很差的代码 -->
106. <module name="IllegalToken" />
107. <!-- 不许内部赋值 -->
108. <module name="InnerAssignment" />
109. <!-- 绝对不能容忍魔法数 -->
110. <module name="MagicNumber">
111. <property name="tokens" value="NUM\_DOUBLE, NUM\_INT" />
112. </module>
113. <!-- 循环控制变量不能被修改 -->
114. <module name="ModifiedControlVariable" />
115. <!-- 多余的**throw** -->
116. <module name="RedundantThrows" />
117. <!-- 不许使用未被简化的条件表达式 -->
118. <module name="SimplifyBooleanExpression" />
119. <!-- 不许使用未被简化的布尔返回值 -->
120. <module name="SimplifyBooleanReturn" />
121. <!-- String的比较不能用!= 和 == -->
122. <module name="StringLiteralEquality" />
123. <!-- **if**最多嵌套3层 -->
124. <module name="NestedIfDepth">
125. <property name="max" value="3" />
126. </module>
127. <!-- **try**最多被嵌套2层 -->
128. <module name="NestedTryDepth">
129. <property name="max" value="2" />
130. </module>
131. <!-- clone方法必须调用了**super**.clone() -->
132. <module name="SuperClone" />
133. <!-- finalize 必须调用了**super**.finalize() -->
134. <module name="SuperFinalize" />
135. <!-- 不能**catch** java.lang.Exception -->
136. <module name="IllegalCatch">
137. <property name="illegalClassNames" value="java.lang.Exception" />
138. </module>
139. <!-- 确保一个类有**package**声明 -->
140. <module name="PackageDeclaration" />
141. <!-- 一个方法中最多有3个**return** -->
142. <module name="ReturnCount">
143. <property name="max" value="3" />
144. <property name="format" value="^$" />
145. </module>
146. <!--
147. 根据 Sun 编码规范， **class** 或 **interface** 中的顺序如下： 1.**class** 声明。首先是 **public**,
148. 然后是**protected** , 然后是 **package** level （不包括access modifier ） 最后是**private** .
149. (多个**class**放在一个java文件中的情况） 2.变量声明。 首先是 **public**, 然后是**protected**然后是 **package**
150. level （不包括access modifier ） 最后是**private** . (多个**class**放在一个java文件中的情况）
151. 3.构造函数 4.方法
152. -->
153. <module name="DeclarationOrder" />
154. <!-- 不许对方法的参数赋值 -->
155. <module name="ParameterAssignment" />
156. <!-- 确保某个**class** 在被使用时都已经被初始化成默认值(对象是**null**,数字和字符是0,**boolean** 变量是**false**.) -->
157. <module name="ExplicitInitialization" />
158. <!-- 不许有同样内容的String -->
159. <module name="MultipleStringLiterals" />
160. <!-- 同一行不能有多个声明 -->
161. <module name="MultipleVariableDeclarations" />
162. <!-- 不必要的圆括号 -->
163. <module name="UnnecessaryParentheses" />
165. <!-- 各种量度 -->
166. <!-- 布尔表达式的复杂度，不超过3 -->
167. <module name="BooleanExpressionComplexity" />
168. <!-- 类数据的抽象耦合，不超过7 -->
169. <module name="ClassDataAbstractionCoupling" />
170. <!-- 类的分散复杂度，不超过20 -->
171. <module name="ClassFanOutComplexity" />
172. <!-- 函数的分支复杂度，不超过10 -->
173. <module name="CyclomaticComplexity" />
174. <!-- NPath复杂度，不超过200 -->
175. <module name="NPathComplexity" />
177. <!-- 杂项 -->
178. <!-- 禁止使用System.out.println -->
179. <module name="GenericIllegalRegexp">
180. <property name="format" value="System\.out\.println" />
181. <property name="ignoreComments" value="true" />
182. </module>
184. <!-- 不许使用main方法 -->
185. <module name="UncommentedMain" />
186. <!-- 检查并确保所有的常量中的L都是大写的。因为小写的字母l跟数字1太象了 -->
187. <module name="UpperEll" />
188. <!-- 检查数组类型的定义是String[] args，而不是String args[] -->
189. <module name="ArrayTypeStyle" />
190. <!--
191. 检查java代码的缩进 默认配置：基本缩进 4个空格，新行的大括号：0。新行的**case** 4个空格。
192. -->
193. <module name="Indentation" />
194. </module>
196. <!-- 检查翻译文件     -->
197. <module name="Translation" />
198. </module>

CheckStyle应用的最佳实践   
　　采用CheckStyle以后，编码规范的检查就变得及其简单，可以作为一项切实可行的实践加以执行。   
　　一般情况下，在项目小组中引入CheckStyle可以按照下面的步骤进行：   
　　1．强调Code Review与Code Conventions的重要作用；   
　　2．介绍CheckStyle；   
    3．初步应用CheckStyle：参照CheckStyle附带的配置文件，酌情加以剪裁，在项目的Maven配置文件中，添加CheckStyle任务，可以单独执行；   
　　4．修改、定型CheckStyle的配置文件：按照基本配置文件执行一段时间（2～3周），听取开发人员的反馈意见，修改配置信息；   
　 5．作为开发过程的日常实践，强制执行CheckStyle：稳定CheckStyle的配置信息，同时将CheckStyle任务作为Build的依赖任务或者配置SCM（目前，CheckStyle可以与SVN有效集成），使得代码在加入系统之前必须通过检查。   
　　同时需要指出的是，CheckStyle的有效执行需要依赖的条件：   
    •IDE Format Code的强大功能：由于CheckStyle本身并没有提供很强大的Code Format等功能，因此，需要借助IDE的帮助，从而使得在发生错误的时候，可以很容易的进行修复。