**密级：**

**Java编码规范**

**说明书**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 文档编号 |  | | | 版本号 | V1.0 |
| 分册名称 | Java编码规范说明书 | | | 第1册/共1册 | |
| 总页数 |  | 正文 |  | 附录 |  |
| 编制 |  | 审批 |  | 生效日期 |  |

修改记录

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **版本号** | **变更控制报告编号** | **更改条款及内容** | **更改人** | **审批人** | **更改日期** |
| V1.0 |  | 创建 | 常勇 |  | 2015-12-01 |
|  |  | 增加规则导入 |  |  | 2015-12-21 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**目 录**

[1.简介 1](#_Toc479672916)

[1.1 目的 1](#_Toc479672917)

[1.2 范围 1](#_Toc479672918)

[1.3 术语 1](#_Toc479672919)

[1.4 参考文档 1](#_Toc479672920)

[2.标示符命名规范 1](#_Toc479672921)

[2.1 原则 1](#_Toc479672922)

[2.2 包名规范 2](#_Toc479672923)

[2.3 类名命名规范 2](#_Toc479672924)

[2.4 方法名命名规范 3](#_Toc479672925)

[2.5 变量命名规范 4](#_Toc479672926)

[3.排版风格 5](#_Toc479672927)

[4.注释规则 8](#_Toc479672928)

[5.规则导入 9](#_Toc479672929)

[5.1 Code Templates 10](#_Toc479672930)

[5.2 formatter 13](#_Toc479672931)

[5.3 Checkstyle 14](#_Toc479672932)

[5.4 结果 23](#_Toc479672933)

[5.5 clean up 23](#_Toc479672934)

# 1.简介

## 1.1 目的

今天人们越来越明白软件设计更多地是一种工程，而不是一种个人艺术。由于大型产品的开发通常由很多的人协同作战，如果不统一编程规范，最终合到一起的程序，其可读性将较差，这不仅给代码的理解带来障碍，增加维护阶段的工作量，同时不规范的代码隐含错误的可能性也比较大。

为了提高代码的可读性和质量，本文档针对Java编码规范详细设计说明。

## 1.2 范围

本Java代码规范以SUN的标准Java代码规范为基础，为适应我们公司的实际需要，可能会做一些修改。本文档中没有说明的地方，请参看SUN Java标准代码规范。如果两边有冲突，以SUN Java标准为准。

供开发人员进行阅读

## 1.3 术语

## 1.4 参考文档

# 2.标示符命名规范

## 2.1 原则

* **统一**

统一是指，对于同一个概念，在程序中用同一种表示方法，比如对于供应商，既可以用supplier，也可以用provider，但是我们只能选定一个使用，至少在一个Java项目中保持统一。统一是作为重要的，如果对同一概念有不同的表示方法，会使代码混乱难以理解。即使不能取得好的名称，但是只要统一，阅读起来也不会太困难，因为阅读者只要理解一次。

* **达意**

达意是指，标识符能准确的表达出它所代表的意义，比如： newSupplier, OrderPaymentGatewayService等；而 supplier1, service2，idtts等则不是好的命名方式。准确有两成含义，一是正确，而是丰富。如果给一个代表供应商的变量起名是 order，显然没有正确表达。同样的，supplier1, 远没有targetSupplier意义丰富。

* **简洁**

简洁是指，在统一和达意的前提下，用尽量少的标识符。如果不能达意，宁愿不要简洁。比如：theOrderNameOfTheTargetSupplierWhichIsTransfered 太长， transferedTargetSupplierOrderName则较好，但是transTgtSplOrdNm就不好了。省略元音的缩写方式不要使用，我们的英语往往还没有好到看得懂奇怪的缩写。

* **骆驼法则**

Java中，除了包名，静态常量等特殊情况，大部分情况下标识符使用骆驼法则，即单词之间不使用特殊符号分割，而是通过首字母大写来分割。比如: supplierName, addNewContract，而不是 supplier\_name, add\_new\_contract。

* **英文**

尽量使用通俗易懂的英文单词，如果不会可以向队友求助，实在不行则使用汉语拼音，避免拼音与英文混用。比如表示归档，用archive比较好。

## 2.2 包名规范

* **使用小写和数字组合，不要出现其他字符和符号。例如：**com.aps.alg

## 2.3 类名命名规范

* **首字母大写**
* **后缀命名**

类名往往用不同的后缀表达额外的意思，如下表：

|  |  |  |
| --- | --- | --- |
| 后缀名 | 意义 | 举例 |
| EJB | 表示这个类为EJB类 | LCIssueInfoManagerEJB |
| Service | 表明这个类是个服务类，里面包含了给其他类提同业务服务的方法 | PaymentOrderService |
| Impl | 这个类是一个实现类，而不是接口 | PaymentOrderServiceImpl |
| Inter | 这个类是一个接口 | LifeCycleInter |
| Dao | 这个类封装了数据访问方法 | PaymentOrderDao |
| Action | 直接处理页面请求，管理页面逻辑了类 | UpdateOrderListAction |
| Listener | 响应某种事件的类 | PaymentSuccessListener |
| Event | 这个类代表了某种事件 | PaymentSuccessEvent |
| Servlet | 一个Servlet | PaymentCallbackServlet |
| Factory | 生成某种对象工厂的类 | PaymentOrderFactory |
| Adapter | 用来连接某种以前不被支持的对象的类 | DatabaseLogAdapter |
| Job | 某种按时间运行的任务 | PaymentOrderCancelJob |
| Wrapper | 这是一个包装类，为了给某个类提供没有的能力 | SelectableOrderListWrapper |
| Bean | 这是一个POJO | MenuStateBean |

## 2.4 方法名命名规范

* **首字母小写，动词在前**

动词前缀往往表达特定的含义，如下表：

|  |  |  |
| --- | --- | --- |
| 前缀名 | 意义 | 举例 |
| create | 创建 | createOrder() |
| delete | 删除 | deleteOrder() |
| add | 创建，暗示新创建的对象属于某个集合 | addPaidOrder() |
| remove | 删除 | removeOrder() |
| init或则initialize | 初始化，暗示会做些诸如获取资源等特殊动作 | initializeObjectPool |
| destroy | 销毁，暗示会做些诸如释放资源的特殊动作 | destroyObjectPool |
| open | 打开 | openConnection() |
| close | 关闭 | closeConnection()< |
| read | 读取 | readUserName() |
| write | 写入 | writeUserName() |
| get | 获得 | getName() |
| set | 设置 | setName() |
| prepare | 准备 | prepareOrderList() |
| copy | 复制 | copyCustomerList() |
| modity | 修改 | modifyActualTotalAmount() |
| calculate | 数值计算 | calculateCommission() |
| do | 执行某个过程或流程 | doOrderCancelJob() |
| dispatch | 判断程序流程转向 | dispatchUserRequest() |
| start | 开始 | startOrderProcessing() |
| stop | 结束 | stopOrderProcessing() |
| send | 发送某个消息或事件 | sendOrderPaidMessage() |
| receive | 接受消息或时间 | receiveOrderPaidMessgae() |
| respond | 响应用户动作 | responseOrderListItemClicked() |
| find | 查找对象 | findNewSupplier() |
| update | 更新对象 | updateCommission() |

## 2.5 变量命名规范

* **静态常量：全大写用下划线分割**

例如：

public static find String ORDER\_PAID\_EVENT = “ORDER\_PAID\_EVENT”;

* **枚举：全大写，用下划线分割**

例如：

public enum Events {  
ORDER\_PAID,  
ORDER\_CREATED  
}

* **其他：首字母小写，骆驼法则**

例如：

public String orderName;

* **局部变量: 参数和局部变量名首字母小写，骆驼法则**

尽量不要和域冲突，尽量表达这个变量在方法中的意义。

# 3.排版风格

**规则一：程序块采用缩进风格编写，缩进为4个空格位。排版不混合使用空格和TAB键。**

**规则二：在两个以上的关键字、变量、常量进行对等操作时，它们之间的操作符之前、之后或者前后要加空格;进行非对等操作时，如果是关系密切的立即操作符，后不应加空格。**

1. 逗号、分号只在后面加空格，例如：

call(a, b, c);

1. 比较操作符, 赋值操作符"="、 "+="，算术操作符"+"、"%"，逻辑操作符"&&"、"&"，位域操作符"<<"、"^"等双目操作符的前后加空格

if(currentTime >= MAX\_TIME\_VALUE)

a = b + c;

a \*= 2;

a = b ^ 2;

1. "!"、"~"、"++"、"--"等单目操作符前后不加空格

flag = !bIsEmpty; // 非操作"!"与内容之间

i++; // "++","--"与内容之间

1. "."前后不加空格

p.id = pId; // "."前后不加空格

1. if、for、while、switch等与后面的括号间应加空格，使if等关键字更为突出、明显。

if (a >= b && c > d)

**规则三：函数体的开始，类的定义，结构的定义，if、for、do、while、switch及case语句中的程序都应采用缩进方式**

不符合规范：

for (...)

{

... // program code

}

if (...) {

... // program code

}

void example\_fun( void ){

... // program code

}

应如下书写：

for (...) {

... // program code

}

if (...) {

... // program code

}

void example\_fun( void ){

... // program code

}

**规则四：功能相对独立的程序块之间或for、if、do、while、switch等语句前后应加一空行**

不符合规范：

if (!valid\_ni(ni))

{

... // program code

}

repssn\_ind = ssn\_data[index].repssn\_index;

repssn\_ni = ssn\_data[index].ni;

应如下书写：

if (!valid\_ni(ni))

{

... // program code

}

repssn\_ind = ssn\_data[index].repssn\_index;

repssn\_ni = ssn\_data[index].ni;

**规则五：if、while、for、case、default、do等语句自占一行。**

不符合规范：

If (NULL == pUserCR) return;

应如下书写:

If (NULL == pUserCR)

{

return;

}

**规则六：若语句较长(多于80字符)，可分成多行写，划分出的新行要进行适应的缩进，使排版整齐，语句可读。**

**规则七：一行最多写一条语句。**

不符合规范：

rect.length = 0 ; rect.width = 0 ;

rect.length = width = 0;

应书写成:

rect.length = 0 ;

rect.width = 0 ;

**规则八：类先成员变量，后内部函数接口；关键字public、protected、private依次按照顺序排版。**

例如：

class A

{

public int iNum;

private int iCode;

private float fNum;

…

public void Func1(void);

public void Func2(void);

private void Func3(void);

}

**规则九：编写条件的时候采用“常量==变量”格式**

# 4.注释规则

**规则一：文件级注释**

/\*\*

\*

\*<li>模块名 : LoginController<br />

\*<li>文件名 : LoginController.java<br />

\*<li>创建时间 : 2015年12月25日<br />

\*<li>实现功能 : <br /> （说明：该位置需要换行的时候结尾加上<br />，以便生成的文档格式友好）

\*<li>作者 : changyong<br />

\*<li>版本 : v0.0.1<br />

\*<li>----------------------------------------------------------<br />

\*<li>修改记录:<br />

\*<li>日 期 版本 修改人 修改内容<br />

\*<li>2015年12月25日 v0.0.1 changyong 创建<br />（说明：该位置需要换行的时候结尾加上<br />，以便生成的文档格式友好）

\*/

注：需记录文件内容，让人看注释就可以知道文件的大致内容；修改记录重点节点记录即可。

**规则二：函数级注释**

/\*\*

\*

\*功 能： <br />

\* 注释测试使用 （说明：该位置需要换行的时候结尾加上<br />，最后一个由于下行开始加了，这里不用加，以便生成的文档格式友好）

\*<br />----------------------------------------------------------------<br />

\*修改记录 ：<br />

\*日 期 版本 修改人 修改内容<br />

\*2015年12月25日 v0.0.1 changyong 创建 （说明：该位置需要换行的时候结尾加上<br />，最后一个由于下行开始加了，这里不用加，以便生成的文档格式友好）

\*<br />----------------------------------------------------------------

\***@param** i

\***@return** String

\*/

注：重点函数进行记录说明，让人看了就知道函数功能，如何使用等信息；修改记录重点节点记录即可。

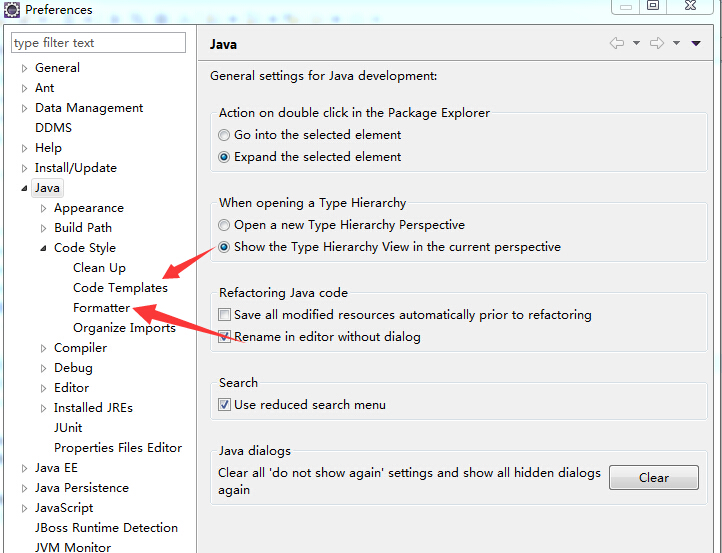
**规则三：语句及函数内部结构注释**

为了方便阅读代码尽可能的多做一些注释，特别是关键点一定要注释，但是也不要过于冗余盲目的注释，尽量达到注释即为大纲的效果。

# 5.规则导入

俗话说约定优于配置，但是在执行过程中往往发现效果不是很好（主要是指编码规范这一方面）。所以我们不得不采取一些措施来协助我们统一项目开发人员的编码风格。合理的利用工具可以做到团队统一规范，本章说明主要是有关eclipse工具的编码规范使用方式，主要包括三个方面：设置Code Templates、Eclipse formatter、Checkstyle，本篇主要介绍如何设置Code Templates，Eclipse formatter、Checkstyle具体步骤如下：

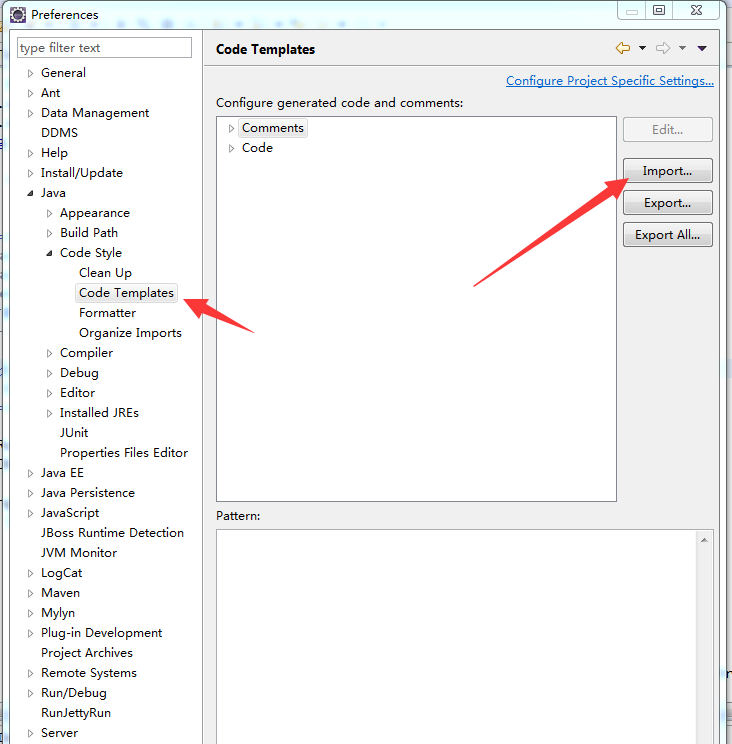
打开Window->Preferences->Java->Code Style->Code Templates/Formater



## 5.1 Code Templates

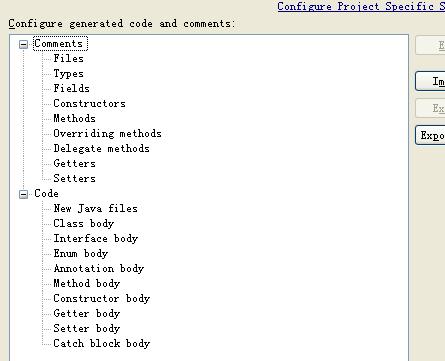
导入代码新建（主要是注释，打出（/\*\*）直接回车直接注释模版生成注释格式，有类的，方法的，如果需要其他注释可以添加）模版：Code Templates

点击【Code Templates】显示如图：



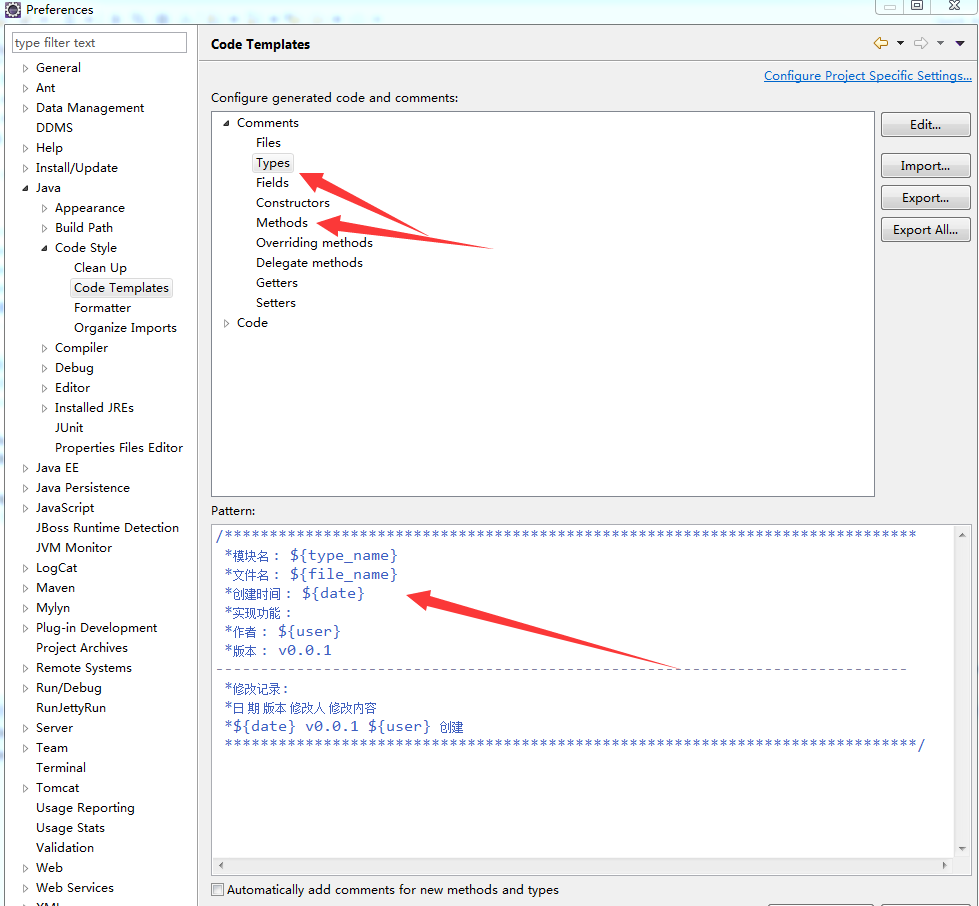
点击"Import"，导入模板codetemplates.xml文件。

codetemplates.xml内容是我们自己预先定义好的，在这里先不详细描述，我们可以看到Eclipse Code Templates界面中间Configure generated code and comments区域包含了两个菜单树：Comment、Code，如下图所示：



Comments代表注释模板，Code代表代码模板，其中每一个子菜单代表子项的模板。

我们只要点击某一个子项，就会在界面下方的Pattern区域看到该项我们所定义的模板内容和格式，如下图所示：

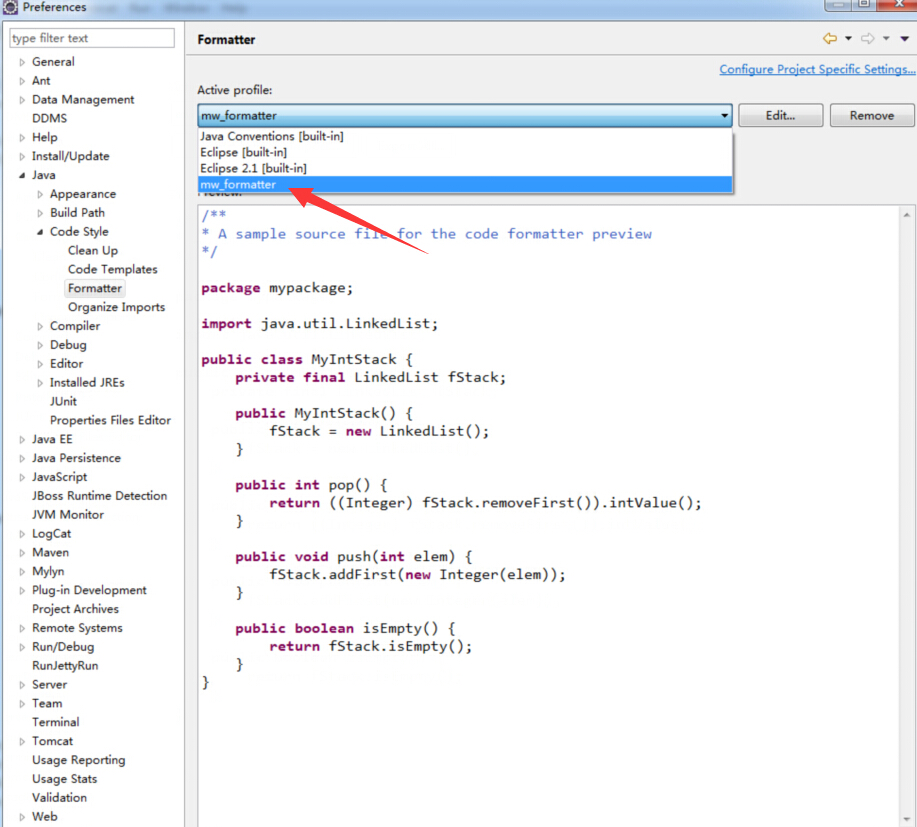


设置Code Templates的目的主要是为了统一各种注释的格式以及代码的模板，只要设定好Code Templates之后利用Eclipse就可以方便地生成我们自定义的注释，开发人员也容易接受！

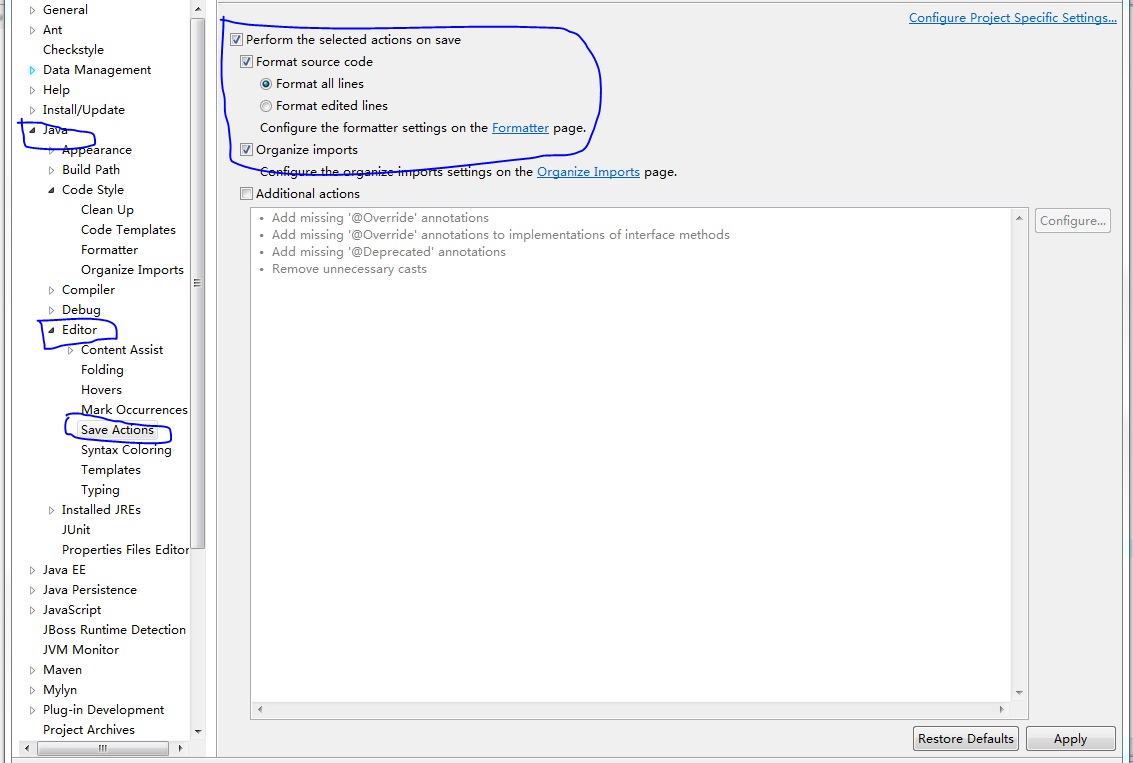
## 5.2 formatter

导入formatter模版：

同理，导入过程以上，导入名称为（mw\_formatter）选择该模版即可



导入后如果觉得 ctrl + shift + F 麻烦，可以按图勾选圈出来的选项，这样，每次保存，Eclipst会自动格式化代码

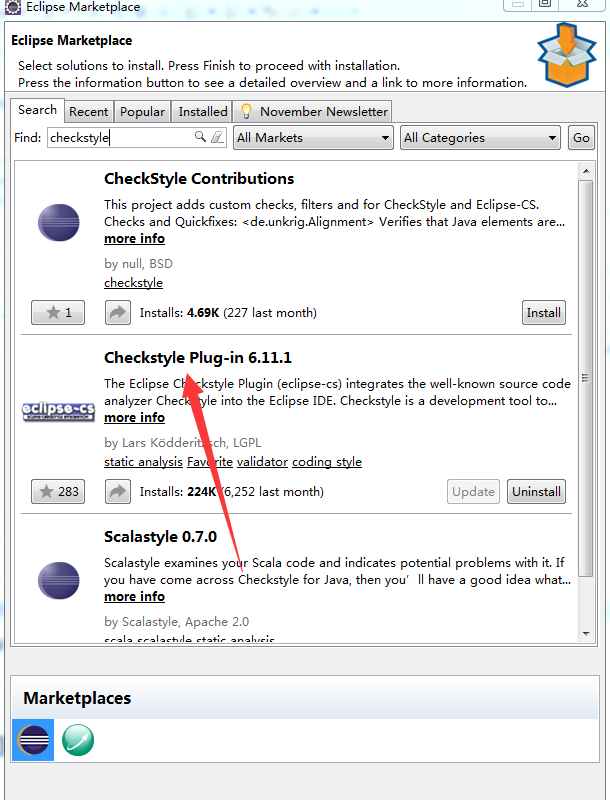


设置Formatter的目的主要是为了统一代码格式，团队代码格式统一，在svn对比合并的时候减少冲突，对代码规范起到很好的作用

## 5.3 Checkstyle

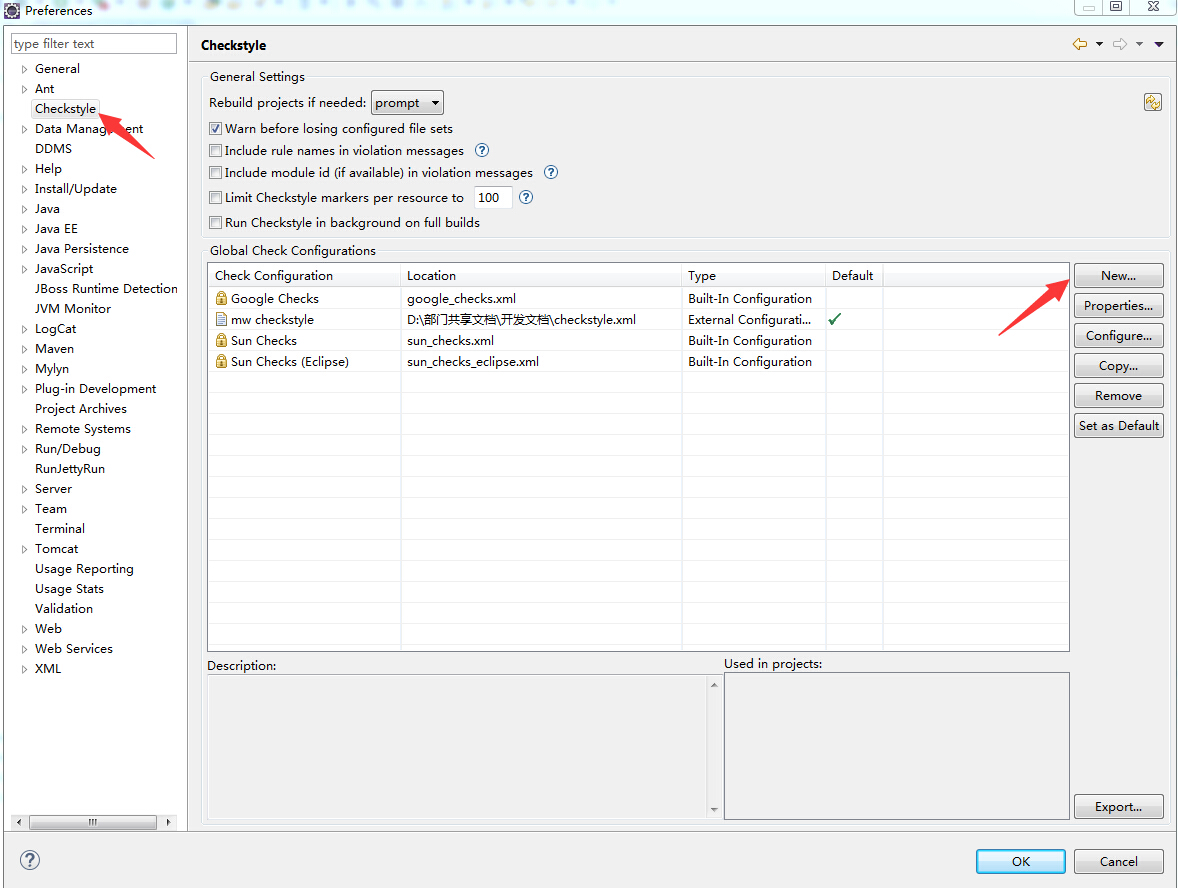
该模块需要安装Checkstyle插件

在Help->Eclipse Marketplace中搜索checkstyle，如图安装一下插件

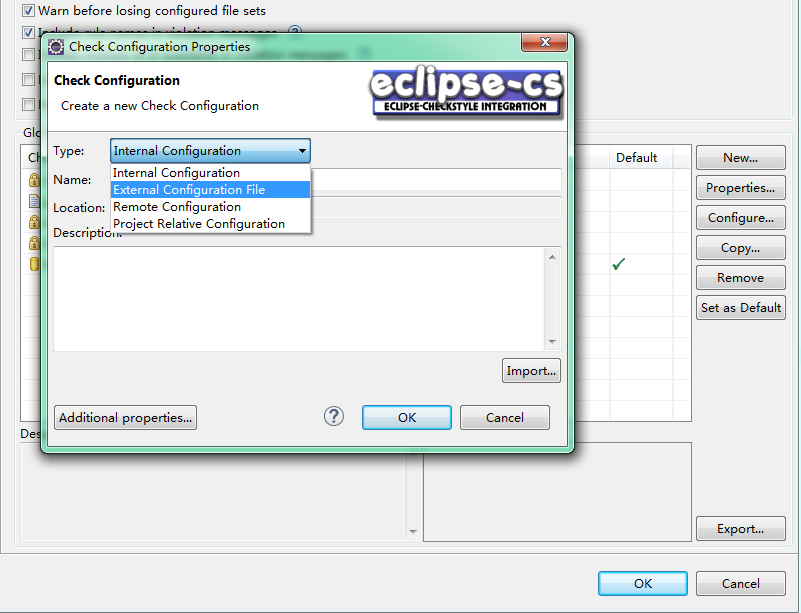


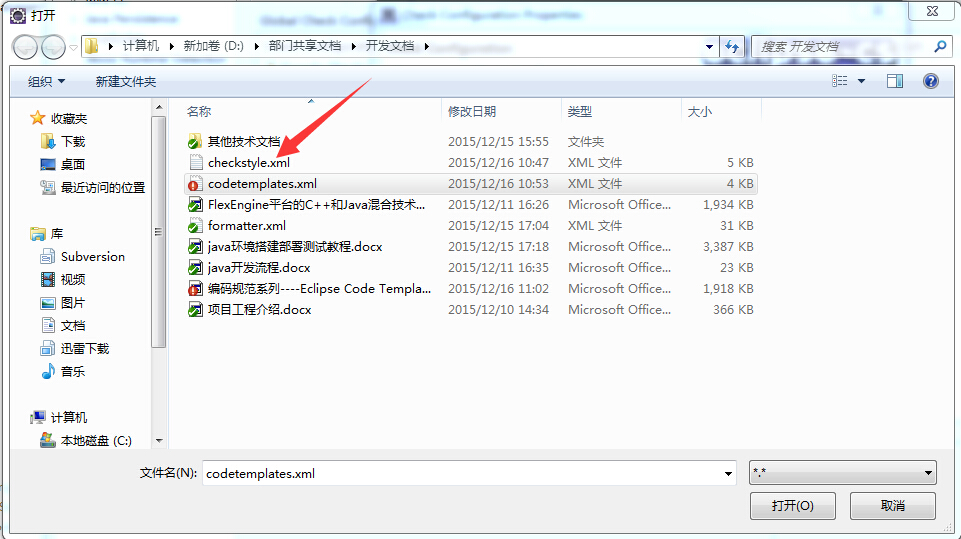
安装过程直接下一步之后完成即可等待安装完成，之后重启eclipse

之后在Window->Preferences中可以看到Checkstyle选项，如图：

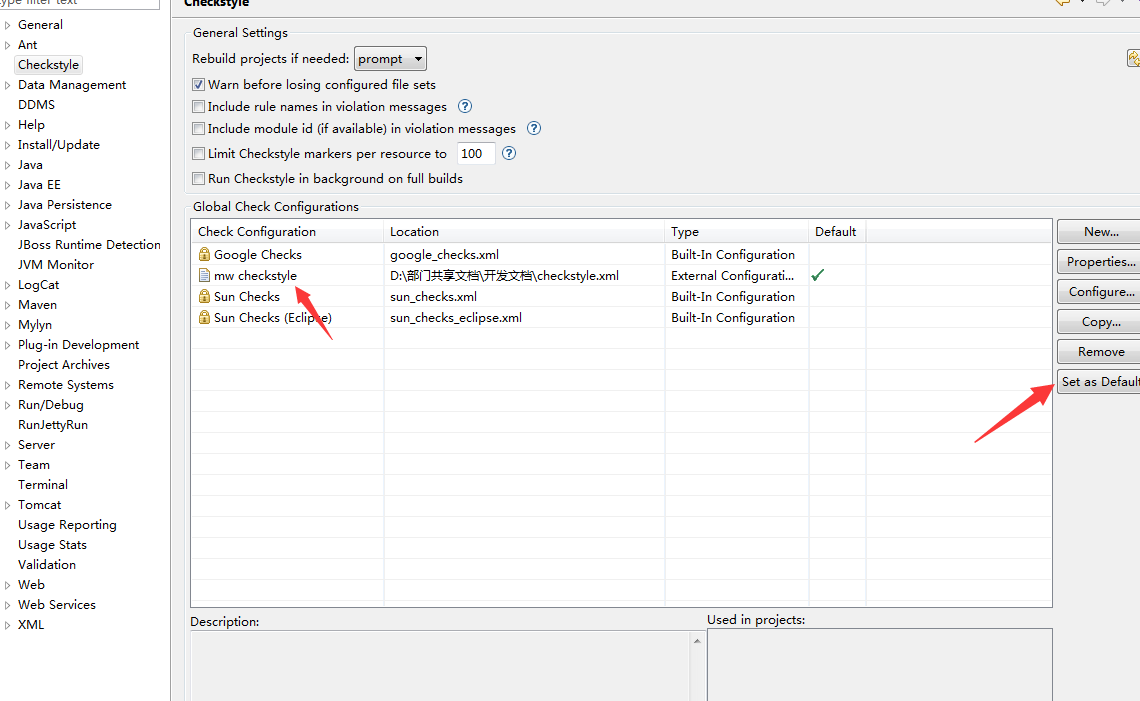


导入配置文件

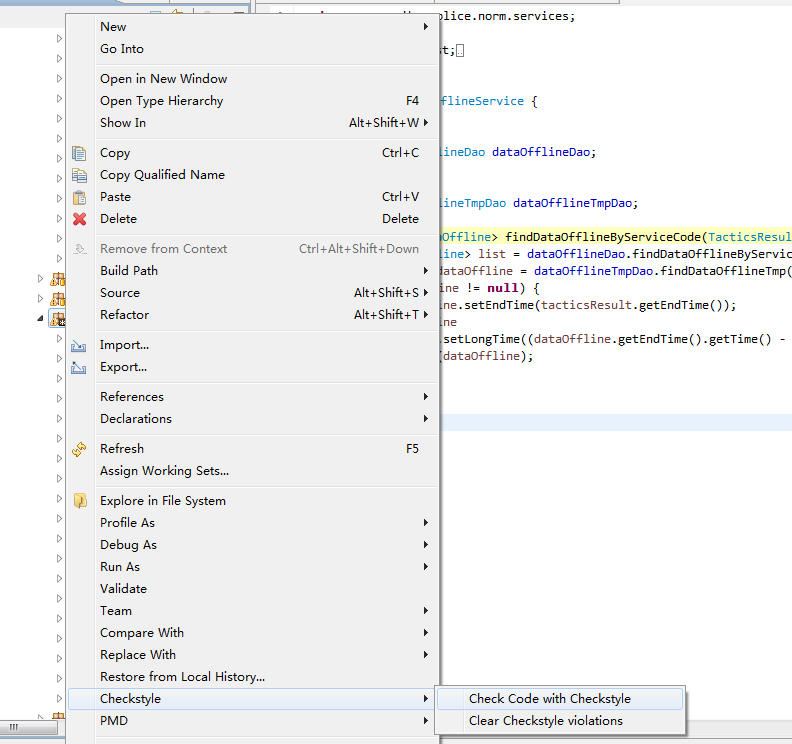




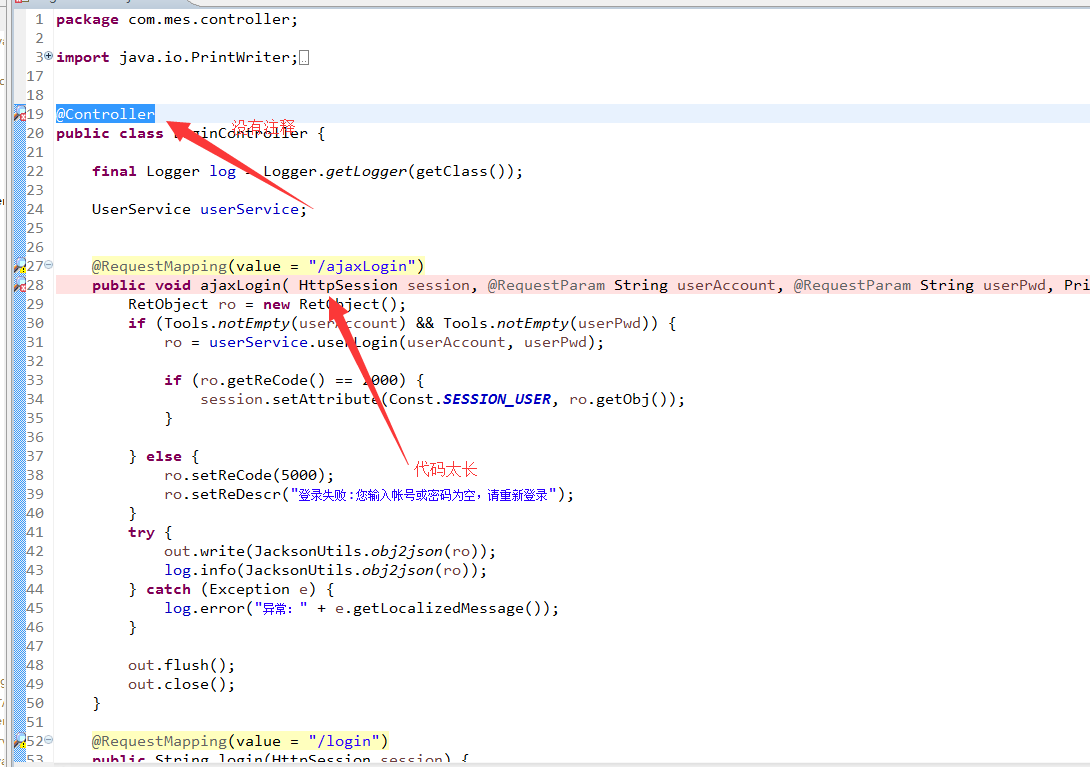
选中导入文件并设置为默认的



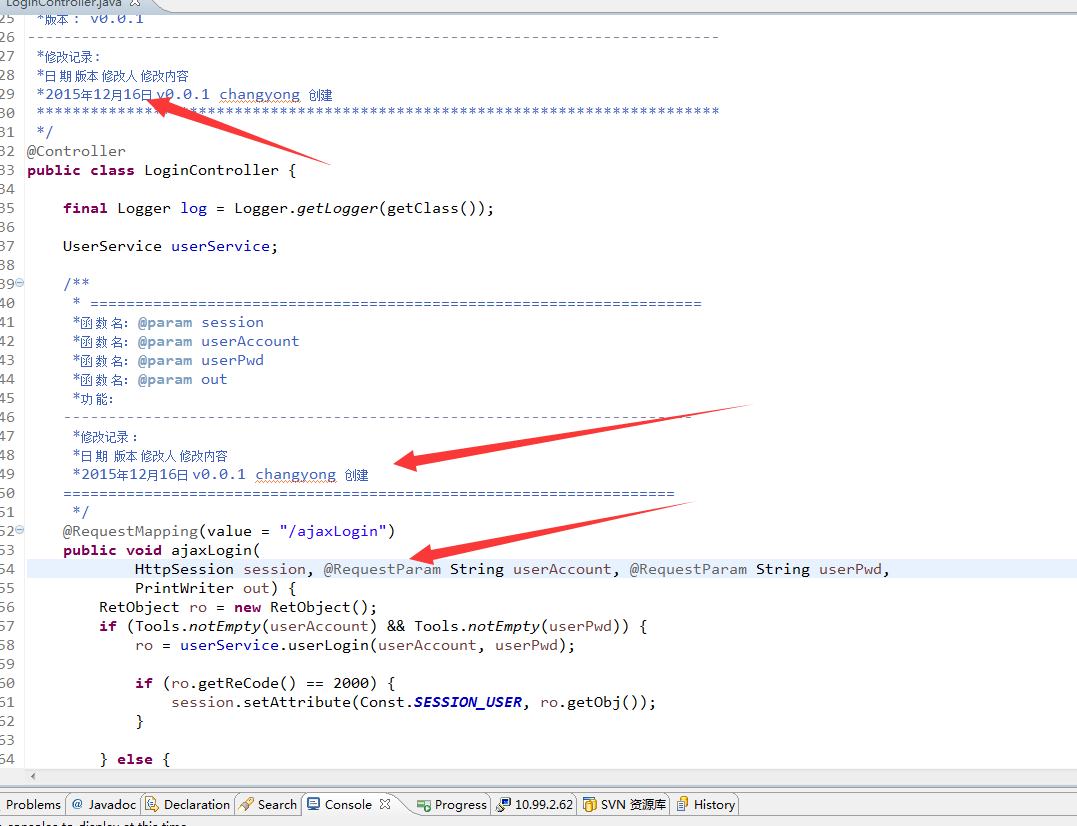
检查



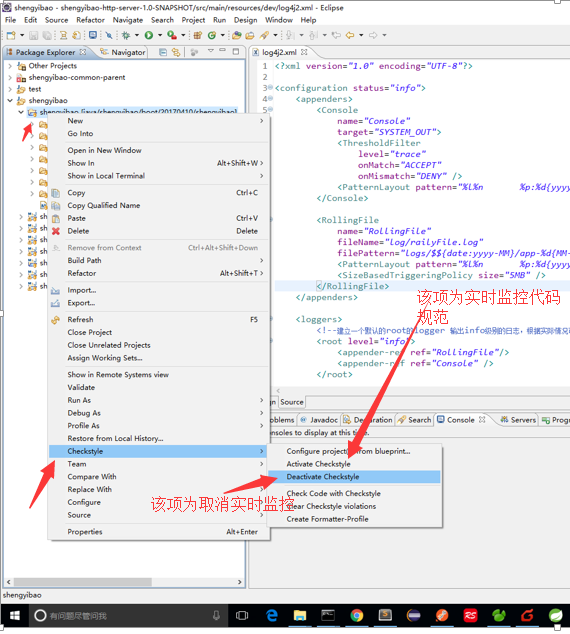
检查不通过结果



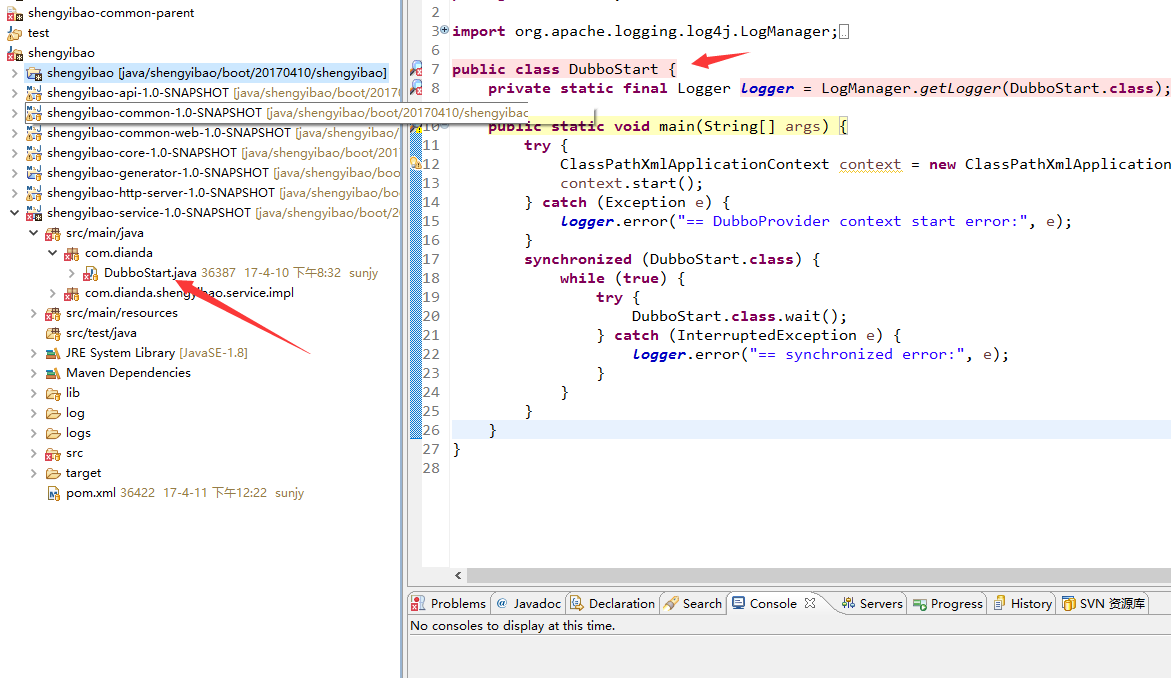
，使用前边导入的代码规范模版ctrl+shift+F，规范代码格式，还有使用代码注释完善注释，规范结果：没有提示规范报错以及警告行为



可以把监控置为活动，即实时监控工程项目的编码是否符合规范，如图，工程右键选择【Checkstyle】-》【Activate Checkstyle】即选择实施监控编码规范



如果不符合规范，工程源文件会有异常或则警告提示，如图



工程右键选择【Checkstyle】-》【Deactivate Checkstyle】即取消实施监控编码规范

## 5.4 结果

至此一套eclipse代码统一规范完成，简单，不用人工去格式化，省时省力

特别强调：由于字符串内容无法把握问题，在自动格式化中字符串长度不会自动缩进，需要自己按照实际情况换行

## 5.5 clean up

对于代码规范eclipse还有一个功能clean up，不过此功能会对代码进行改造，这里就不使用，防止对代码造成不必要的麻烦，它的功能多数为优化代码，如：去掉死代码（永远不可能运行的代码，如：if(true){}else{code}，其中code代码不可能执行，除非系统bug，定义一个int I = 0；但是后边没使用过，这个代码是没用的，还占用内存，之类的事例很多），还有一些其他优化，在此不一一介绍，有兴趣自己可以学习