**JAVA程序编码规范**

版 本

|  |  |  |  |
| --- | --- | --- | --- |
| 版本 | 时间 | 修改内容 | 附注 |
| V1.0 | 2008/09/16 | 新版 |  |
| V1.1 | 2008/10/17 | 增加Hibernate,WebWork应用时应注意的一些细节 |  |
| V2.0 | 2015/04/04 | 增加Spring MVC开发注意事项 |  |
| V3.0 | 2016/08/01 | 更新格式化风格 |  |

目 录

[1 目的 4](#_Toc17920)

[2 适用范围 4](#_Toc21573)

[2.1 内部开发人员 4](#_Toc4805)

[2.2 外包厂商开发人员 4](#_Toc30728)

[3 执行细则 4](#_Toc4073)

[3.1 集成开发环境 4](#_Toc13175)

[3.2 自动生成的代码注释 4](#_Toc173)

[3.3 代码格式化风格 5](#_Toc17687)

[3.3.1 JAVA代码格式化 5](#_Toc14500)

[3.3.2 Javascript代码格式化 6](#_Toc3277)

[3.3.3 Ant代码格式化 6](#_Toc28755)

[3.3.4 CSS代码格式化 7](#_Toc8307)

[3.3.5 HTML代码格式化 7](#_Toc20456)

[3.3.6 XML代码格式化 8](#_Toc21047)

[3.4 程序目录结构 8](#_Toc26803)

[3.5 新建接口与类时，应遵循命名规范 9](#_Toc7862)

[3.5.1 一般服务类不实用接口方式 9](#_Toc23354)

[3.6 新建接口、类、变量与方法时，应添加一些必要的注释 9](#_Toc26932)

[3.6.1 Javadoc标签用法 10](#_Toc5925)

[3.7 单元测试 13](#_Toc24166)

# 目的

　　规范公司内部开发人员的编码习惯，使开发人员具有一致的编码风格，从而使得代码更具有可读性与可维护性。也使得新的开发人员能够比较快速地进入开发状态，以及开发人员能够轻松地在不同项目间进行切换。

# 适用范围

## 内部开发人员

　　专案开发及维护人员，应遵循此文件规范准则。

## 外包厂商开发人员

　　专案委外开发或采人力外包之专案及需求/维护等应用程式开发设计工作，外包厂商开发人员应遵循此文件规范准则。

# 执行细则

## 集成开发环境

　　我们采用Eclipse或MyEclipse 这个Java IDE。同时，也可以配合使用其他一些插件，比如反编译插件等。

## 自动生成的代码注释

　　新建类或者函数时，Eclipse会自动产生一些代码注释。请导入配置文件“comwave-code-templates.xml”，并进行一些设置，以统一代码注释风格（*关键是去掉了一些不必要的代码注释块，使代码更简洁一些*）。

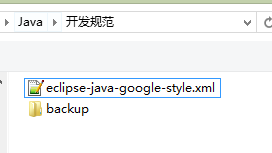
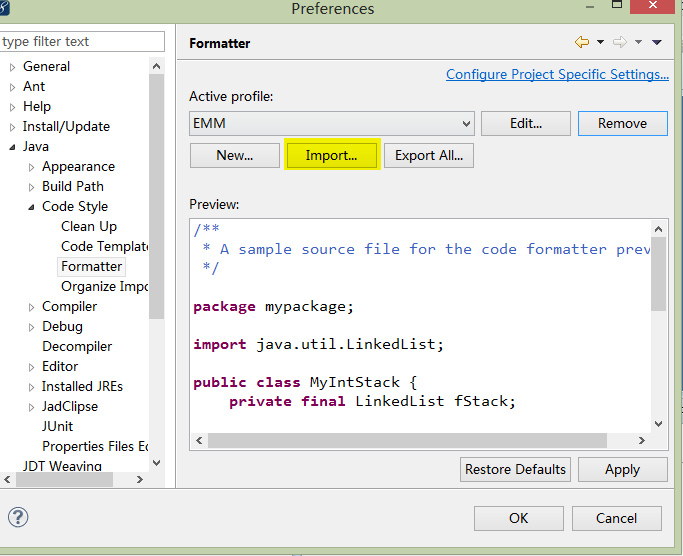
　　打开Eclipse菜单Window-Preferences/Java/Code Style/Code Templates，点击右边面板的Import按钮，选择前面这个配置文件，将配置导入。点击打开中间面板的Comments结点，选中Files结点，点击右边面板的Edit按钮，将“Author: Eric Yang”该句中的名字修改成各自的英文名和姓氏。保存配置，退出该对话框。

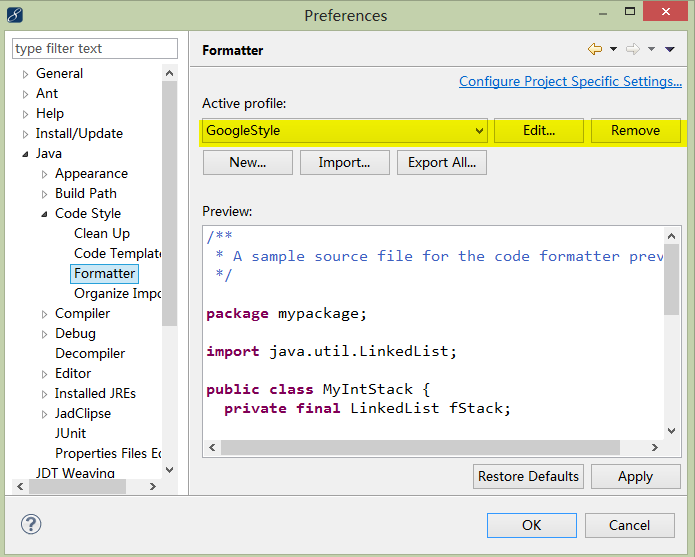
## 代码格式化风格

### JAVA代码格式化

Eclipse提供了代码格式化的功能，有利于提高代码的可读性与可维护性。请导入google Java代码风格配置文件“eclipse-java-google-style.xml”，以统一格式化风格。

打开Eclipse菜单Window-Preferences/Java/Code Style/Formatter，点击右边面板的Import按钮，选择前面这个配置文件。接下来出现Load Profile的对话框。保存配置，退出该对话框。



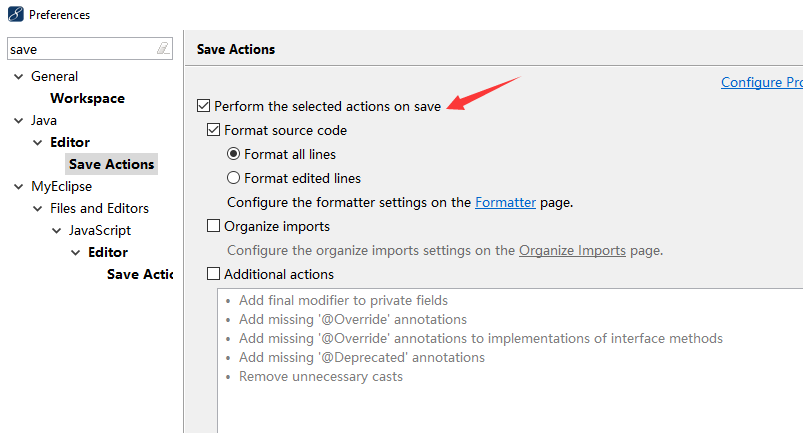


编辑任何文件后，必须利用Eclipse格式化代码的功能将代码进行格式化。如果是修改旧系统的代码，请先不作任何修改，先将代码格式化后并上传至CVS/Git保存旧版本，再进行原先要做的代码修改动作。

*【注】Google Java代码风格配置原始文件下载地址如下，因为现代浏览器窗口较大，为了在一个屏幕下浏览更多代码，主要对其格式化每行长度改为200.注释每行长度改为100.*

[*https://raw.githubusercontent.com/google/styleguide/gh-pages/eclipse-java-google-style.xml*](https://raw.githubusercontent.com/google/styleguide/gh-pages/eclipse-java-google-style.xml)

配置保存的时候自动格式化java代码：



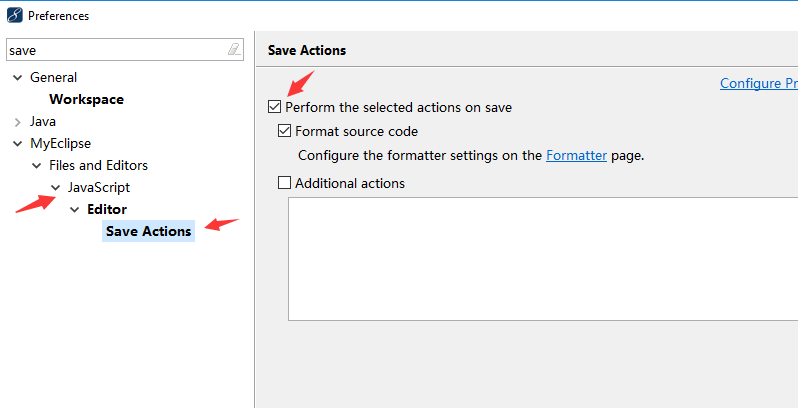
### Javascript代码格式化

格式化文件采用google Javascript格式化文件。打开Eclipse菜单Window-Preferences/MyEclipse/File and Editors/Javascript/Code Style/Formatter，点击右边面板的Import按钮，选择前面这个配置文件。接下来出现Load Profile的对话框。保存配置，退出该对话框。

*【注】Google Javascript格式化原始文件下载地址如下，因为现代浏览器窗口较大，为了在一个屏幕下浏览更多代码，主要对其格式化每行长度改为200.注释每行长度改为100.*

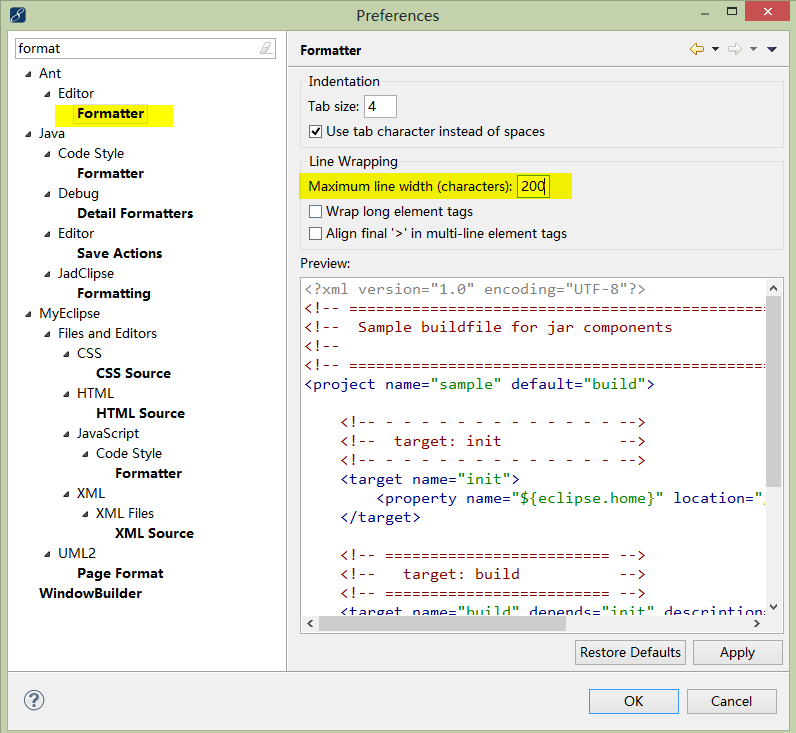
[*https://raw.githubusercontent.com/jokeyrhyme/eclipse-formatter-profiles/master/google-style-guide-javascript-eclipse.xml*](https://raw.githubusercontent.com/jokeyrhyme/eclipse-formatter-profiles/master/google-style-guide-javascript-eclipse.xml)

配置保存的时候自动格式化javascript代码：



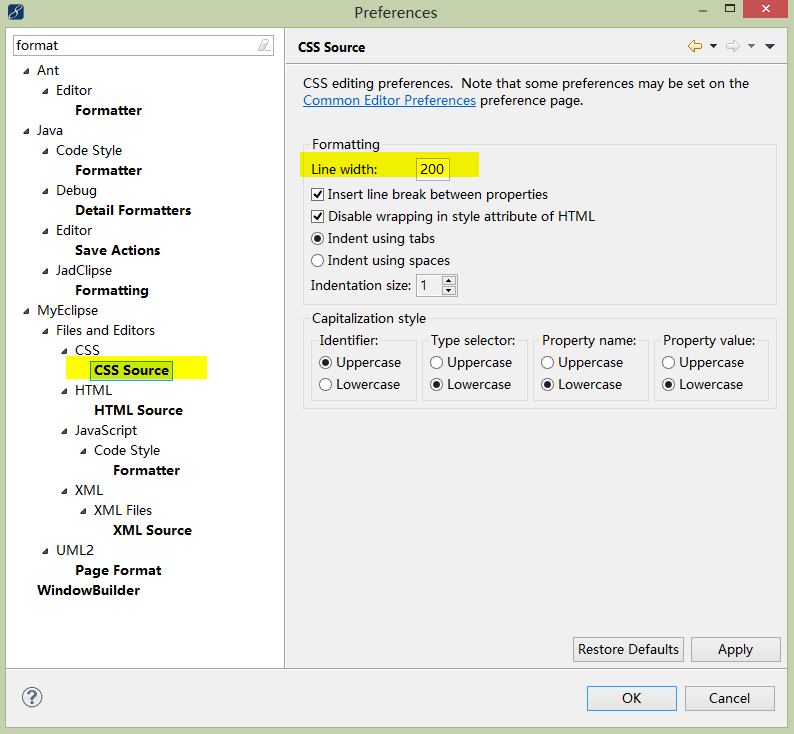
### Ant代码格式化

将Ant代码最大行宽度改为200：



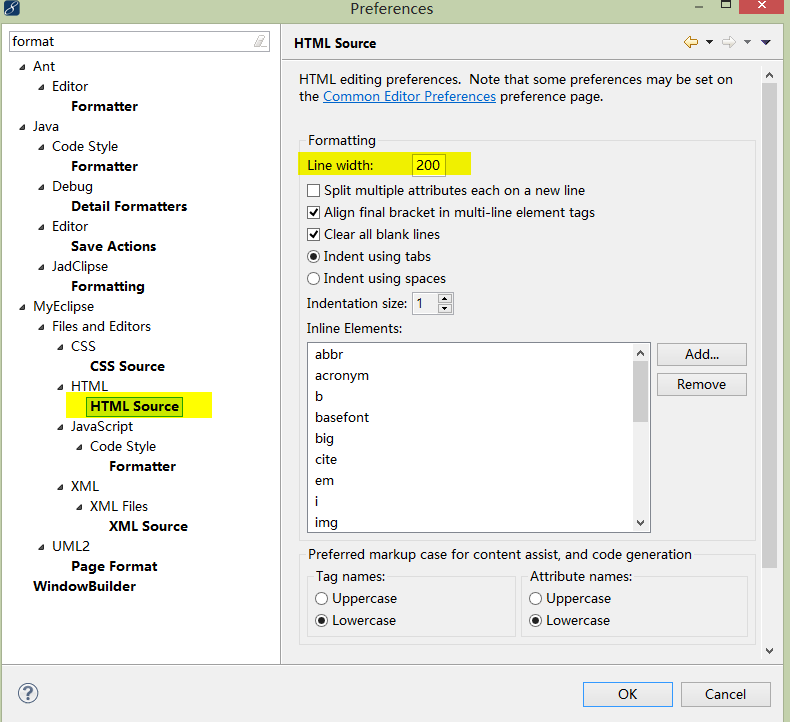
### CSS代码格式化

将CSS代码最大行宽度改为200：



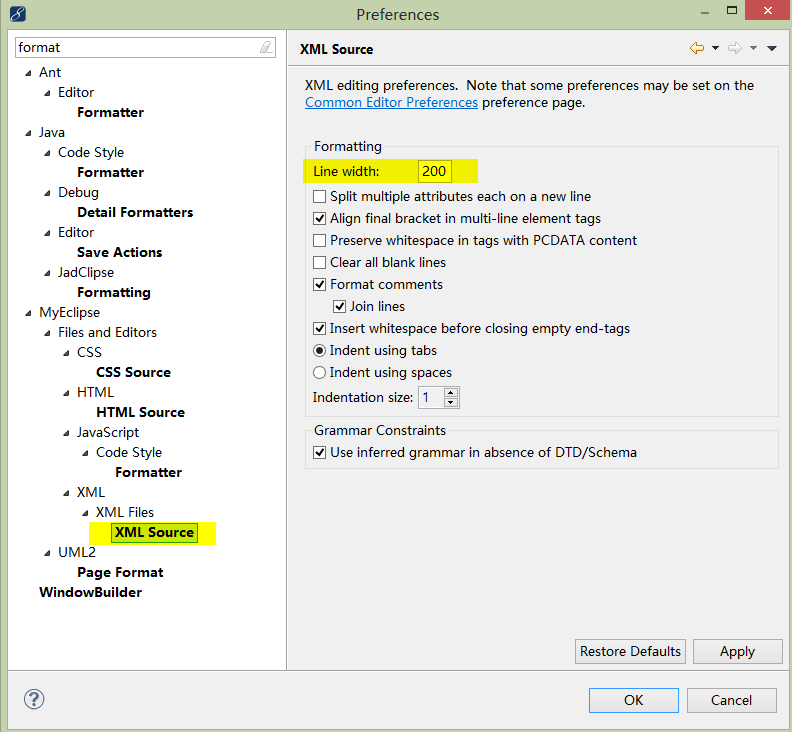
### HTML代码格式化

将HTML代码最大行宽度改为200：



### XML代码格式化

将XML代码最大行宽度改为200：



## 程序目录结构

　　项目代码基于maven进行构建，采用maven标准代码结构，代码存放位置严格按照maven标准。

　　各大模块的程序划分以package名进行区分，比如com.comwave.swiftnet.fund.rdp与com.comwave.swiftnet.fund.fxdm等。

## 新建接口与类时，应遵循命名规范

　　新建接口与类时，应遵循以下命名规范：

　　接口名称，以大写字母“I”开头，以已相应逻辑名称结尾，名称中含有的各单词的第一个字母应该大写，如：IOrderBatchManager。

　　接口实现类的名称，由接口名称去掉“I”的剩余部分和 “Impl”字符串组成，如：OrderBatchManagerImpl。

普通类名称，应取个有业务意义的名称，以便从类名称上就能看出它所代表的业务意义，如：OrderBatch。

其他特殊意义类，遵照相应规则定义。如Repository类名称，由Domain名称 + Repository组成；如DAO类名称，由Domain名称 + Dao组成；如Service类名称，由Domain名称 + Service组成；

* + - 接口一般放于service包下，如：com.comwave.swiftnet.fund.service。repository一般放于repository包下，如：com.comwave.swiftnet.fund.repository。
    - DAO一般放于dao包下，如：com.comwave.swiftnet.fund.dao。
    - 接口实现类一般放于service.impl包下，如：com.comwave.swiftnet.fund.service.impl。
    - 业务对象一般放于domain包下，如：com.comwave.swiftnet.fund.domain。

### 一般服务类不实用接口方式

一般服务类，只会有一个实现类，不定义接口，直接实现服务类。

比如 OrderService.java, MerchantService.java

只有在有多个实例类的情况下才需要定义接口，接口中定义实例类共有实现方法。

## 新建接口、类、变量与方法时，应添加一些必要的注释

　　新建接口与类时，必须在接口名与类名之上添加一些注释，阐明该接口或类所提供的功能。请用/\*\*...\*/注释符，注释分成多行。如：

/\*\*

\* Used for RedemptionXMLCreator and SubscriptionXMLCreator, it's a template xml file creator, provided special features

\* from {@link IXMLCreator}.

\* <p>

\* Call {@link #create(List, Template, Template, Template, IXMLCreator)} method to get a list of {@link OrderMessage}

\* objects.<br>

\*

\* **@author** eric

\*/

**public** **class** SwiftNetXMLCreator {

...

｝

　　在类中新建实例变量或者类变量时，必须添加一些必要的注释，且是以中文为主的业务意义。请用/\*\*...\*/注释符，但保持注释在同一行。如：

/\*\* message level: pro(正本), copy(副本) and copy2(副本2) \*/

**public** **static** **final** String *MSGLEVEL\_PRO* = "1";

/\*\* 基金公司批号 \*/

**private** String batchNo;

/\*\* 批号流水号 \*/

**private** **int** batchNoSeq;

　　新建的接口函数或者类中的重要函数时，必须添加一些必要的注释。请用/\*\*...\*/注释符，注释分成多行。如：

/\*\*

\* Create a list of <code>SwitchOrderDetail</code> xml files in the special directory and with special file names.

\* Return a list of {@link OrderMessage} objects, containing the document name and path.

\*

\* **@param** switchOrderDetails

\* a list of <code>SwitchOrderDetail</code> objects

\* **@return** a list of <code>OrderMessage</code> objects

\* **@throws** IOException

\* **@throws** TemplateException

\*/

**public** List<OrderMessage> create(List<SwitchOrderDetail> switchOrderDetails) **throws** SwiftnetRuntimeException;

　　在函数中某些具有逻辑判断的地方，必要时也需要添加一些注释，请用//注释符。如：

// create OrderMessage

OrderMessage msg = **new** OrderMessage();

**if** (OrderBatch.*ORDERMETHOD\_SWIFT*.equals(orderBatch.getOrderMethod()))

result.add(msg); // 正本消息

### Javadoc标签用法

|  |  |  |
| --- | --- | --- |
| 标签 | 参数 | 范围 |
| @author | name-text | Class and Method |
| @deprecated | deprecated-text |  |
| @exception | class-name description | Method |
| {@link} | name label |  |
| @param | parameter-name description | Method and Constructor |
| @return | description | Method |
| @see | reference |  |
| @throws | class-name description | Method |
| @version | version-text | Classe and Interface |

#### 3.6.1.1 @author

@author name-text：作者声名标签。可以用在Class中，说明谁开发此Class。亦可用在Method中，说明谁开发此Method，或是谁修改了此Method。每一个标签中只允许一次新增或修正人员。

#### 3.6.1.2 @deprecated

@deprecated deprecated-text：已不被使用或不能使用声名标签。在deprecated-text叙述会改用斜体字，并且在最前会有警告：”Deprecated”。

deprecated-text分为两个部份，第一部份是宣告最后使用的版本，第二部份是被取代的link。第二部份也可以说明为何被取代，必须包含{@link}标签，这样才可以指到取代的API

/\*\*

\* @deprecated JDK 1.2, replaced by {@link \*#setBounds(int, int)}

\*/

#### 3.6.1.3 @exception

@exception class-name description：异常声名标签。class-name是异常产生时所抛出的class。如果Method宣告中有用到throws，就可以利用此标签。@exception标签是与@throws相同，只用在Method之中。

/\*\*

\* @exception IOException if an I/O error occurs when \*creating socket.

\*/

#### 3.6.1.4 @throws

@throws class-name description：@throws和@exception标签是一样的。

/\*\*

\* @throws IOException if an I/O error occurs when creating \*the socket.

\*/

#### 3.6.1.5 @link

{@link name label}：插入一个内部连结到一个特定的名称。这个标签接受正确的名称或标记(label)如同@see标签。

{@link}没有限制在一个叙述上的数量。你可以使用这个标签在注解文件中的叙述部份或者是在标签中的文字，例如@deprcated, @return or @param。

下面的范例是联结到getComponentAt(int, int) method：

Use the {@link #getComponentAt(int, int) getComponentAt} method.

在javadoc所产生出来的会是：

Use the <a href="Component.html#getComponentAt(int, int)">getComponentAt</a> method

#### 3.6.1.6 @param

@param parameter-name description：Method参数声名标签。当Method传多个参数时，就必须有多个@param宣告，每一个声名必须分开，每一个叙述可以写超过一行以上，以能够让人了解参数为原则，可包含型态、合理值范围。

#### 3.6.1.7 @return

@return description：返回值声名标签。当Method有返回值时，就必须写此标签，叙述可以写超过一行以上，以能够让人了解参数为原则，包含回传型态、可能的回传值范围。

#### 3.6.1.8 @version

@version version-text：版本控制标签。通常是用在软体的版本控制上。

#### 3.6.1.9 @see

@see *reference*：相联结标签。一般使用在Class或Method注解中，是将有相关性的文件做一联结，或一串文字说明。

文件注解中可以包含许多的@see标签。

## 单元测试

　　必须为类中的每一个重要函数编写对应的单元测试(JUnit)。重要的处理类，在具备了足够的单元测试后，还应该编写小规模的集成测试，测试流程的处理过程是否达到预期的效果。

　　测试程序应该放在与该模块相应的/src/test/java源文件夹下，包名与被测试类的包名相同。测试程序的命名为在被测试类的名称前加上“Test”前缀，如：TestOrderBatchManagerImpl。测试程序使用到的Spring配置文件或DataSet数据文件，以“springContext-”，“dataset-”作为前缀，如：springContext-orderBatchManager，dataset-orderBatchManager。测试程序使用到的配置文件皆应放于/src/test/resources与测试程序相同的包名目录下，不可与正式的配置文件同目录层次。

　　在编写DAO service实现类的测试时，应结合Spring与DBUnit进行单元测试。

　　在编写其他单元测试时，如果引用到其他接口提供的方法，这时需要进行解依赖的处理，请一率使用继承已有实现类或者Dimple工具包的方法提供接口的Stub，以便解除对其他接口实现类的依赖。