**Java编码规范**

[1. 说明 3](#_Toc13645)

[1.1. 为什么要有编码规范 3](#_Toc10936)

[2. 标识符命名规范 3](#_Toc5299)

[2.1. 概述 3](#_Toc19321)

[2.1.1. 统一 3](#_Toc3765)

[2.1.2. 达意 3](#_Toc32068)

[2.1.3. 简洁 4](#_Toc31744)

[2.1.4. 骆驼法则 4](#_Toc1799)

[2.1.5. 英文和拼音 4](#_Toc23790)

[2.2. 包名 4](#_Toc4146)

[2.3. 类名 4](#_Toc18866)

[2.3.1. 首字母大写 4](#_Toc3262)

[2.3.2. 后缀 4](#_Toc30882)

[2.4. 方法名 5](#_Toc19150)

[2.5. 域名（field） 5](#_Toc13302)

[2.5.1. 静态常量 6](#_Toc13799)

[2.5.2. 枚举 6](#_Toc8836)

[2.5.3. 其它 6](#_Toc17745)

[3. 代码格式 6](#_Toc9010)

[3.1. 源文件编码 6](#_Toc30244)

[3.2. 行宽 6](#_Toc14656)

[3.3. 缩进 6](#_Toc30470)

[3.4. 包的导入 6](#_Toc14230)

[3.5. 域格式 7](#_Toc6832)

[3.6. 代码块格式 7](#_Toc2993)

[3.6.1. 缩进风格 7](#_Toc4012)

[3.6.2. 空格的使用 7](#_Toc19700)

[3.6.3. 空行的使用 8](#_Toc3816)

[4. 注释 8](#_Toc6997)

[4.1. 注释和代码 8](#_Toc28)

[4.2. 注释类型 8](#_Toc28881)

[4.3. 一般注释 9](#_Toc18955)

[4.4. Java Doc 9](#_Toc25343)

[4.4.1. Java Doc的位置 9](#_Toc30251)

[5. 实践和禁忌 10](#_Toc32444)

[5.1. 每次保存的时候，都让你的代码是符合规范的 10](#_Toc19006)

[5.2. 使用log替代System.out.println() 10](#_Toc30156)

[5.3. 每个if while for等语句，都必须加大括号{} 10](#_Toc20434)

[5.4. 善用TODO: 11](#_Toc22238)

[5.5. 在需要留空的地方放一个注释，告述读者，你是故意的 11](#_Toc16595)

[5.6. 不再对boolean值做true false判断 11](#_Toc2648)

[5.7. 减少代码嵌套层次 11](#_Toc18828)

[5.8. 程序职责单一 13](#_Toc25299)

[5.9. 变量的声明，初始化和被使用尽量放到一起 13](#_Toc21450)

[5.10. 循环惯例 13](#_Toc17970)

[5.11. 字符串操作 14](#_Toc27648)

[6. eclipse统一格式化 14](#_Toc23736)

[6.1. 程序职责单一 14](#_Toc6296)

[6.2. eclipse中显示空白字符 15](#_Toc23949)

# 说明

## 为什么要有编码规范

编码规范对于程序员而言尤为重要，有以下几个原因：

* 一个软件的生命周期中，80%的花费在于维护。
* 几乎没有任何一个软件，在其整个生命同期中，均由最初的开发人员来维护。
* 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新代码。
* 如果你将源码作为产品发布，就需要确认它是否被很好的打包并且清晰无误，一如你已构建的其它任何产品。

为了执行规范，每个软件开发人员必须一致遵守编码规范。

# 标识符命名规范

## 概述

标识符的命名必须做到统一、达意、简洁三要素。

### 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：地区用area和region表示，电话用phone和telephone表示，选择比较常用的其中一个统一使用。

### 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：用户列表用userList命名，用户服务接口用userService命名。list1，service2等则是不好的命名方式。准确有两成含义，一是正确，二是丰富。如果给一个代表用户的变量起名是 order，显然没有正确表达。同样的，list1, 远没有userList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。

比如一个方法名子叫做findAllAvailableMenuByRoleId 你可以明确的知道他的意思是根据角色ID查找所有可用的菜单。但是如果简写为findAllAvaMenu就不能通过名子明确代码的真实涵意。

### 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: userName, shopId，而不是 user\_name, shop\_id。

### 英文和拼音

尽量使用通俗易懂的英文单词，如果不会可以查词典或向队友求助。尽量避免使用拼音，更不能作用拼音简写以免带来歧义。

## 包名

使用小写字母并且没有特殊符号

如：com.richinfo.admin.common.service 是符合规范的。

com.Richinfo.admin.common.service、Com.richinfo.admin.common\_2.service是不符合规范的。

## 类名

### 首字母大写

类名首字母大写，比如UserService, SystemController是符合规范的；

userService, systemController是不合规范的。

### 后缀

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 |
| Controller | 直接处理页面请求，管理页面逻辑的类。 | OrderController |
| View | SpringMVC中的视图。 | JSONView |
| Util | 可重用功能工具类。 | HTTPUtil |
| Filter | 过滤器，做请求统一处理的类。 | SecurityFilter |
| Thread | 线程类，用于处理多线程业务。 | ClientClearThread |
| Service | 服务层接口。 | MenuService |
| ServiceImpl | 服务层接口的实现类。 | MenuServiceImpl |
| Dao | 数据访问层接口。 | MenuDao |
| DaoImpl | 数据访问层接口实现类。 | MenuDaoImpl |
| Exception | 自定义异常类。 | TMallException |
| ExceptionHandler | 异常处理类。 | TMallExceptionHandler |
| Constants | 常量接口。 | Constants |
| Task | 某种按时间运行的任务。 | OrderCancelTask |
| Listener | 响应某种事件的监听类。 | PaymentListener |
| Factory | 生成某种对象工厂的类。 | SessionFactory |

## 方法名

首字母小写遵循动宾结构，如createOrder()。

动词前缀往往表达特定的含义，如下表：

|  |  |  |
| --- | --- | --- |
| 前缀名 | 意义 | 举例 |
| add | 添加。 | addUser |
| delete | 删除。 | deleteUser |
| remove | 删除。 | removeUser |
| destroy | 销毁，用于资源释放。 | destroyPool |
| open | 打开。 | openConnection |
| close | 关闭。 | closeConnection |
| get | 获得。 | getName |
| set | 设置值。 | setName |
| find | 查找。 | findArticle |
| modify | 修改。 | modifyAmount |
| start | 开始。 | startFileProcessing |
| stop | 停止。 | stopFileProcessing |
| send | 发送消息。 | sendMessage |
| receive | 接收消息。 | receiveMessage |

## 域名（field）

A field is an attribute. A field may be a class's variable, an object's variable, an object's method's variable, or a parameter of a function.

field，域是一种属性，可以是一个类变量，一个对象变量，一个对象方法变量或者是一个函数的参数。

### 静态常量

全大写用下划线分割，如：

public static final String ORDER\_PAID\_EVENT = "ORDER\_PAID\_EVENT";

### 枚举

全大写，用下划线分割，如：

public enum AccountType {

USERNAME(1, "用户名"), EMAIL(2, "邮箱"), MOBILE\_NO(3, "手机号");

}

### 其它

首字母小写，骆驼法则，如：

public String userName;

# 代码格式

## 源文件编码

源文件作用utf-8编码，结尾用回车换行。

## 行宽

每行宽度限制在130以内。

## 缩进

使用空格字符代替TAB键缩进源代码，1个TAB等于4个空格。

## 包的导入

精确导入，不以整个包为单位，删除不用的导入。

如：import java.util.HashMap; 符合规范。

import java.util.\*; 不符合规范。

## 域格式

每行只能声明一个域。域的声明用空行隔开。

## 代码块格式

### 缩进风格

大括号的开始在代码块开始的行尾，闭合在和代码块同一缩进的行首，例如：

public User getUserById(Long userId){

User user = userDao.getUserById(userId);

return user;

}

for(int i = 0; i < 10; i++){

System.out.println(i);

}

### 空格的使用

#### 表示分割时用一个空格

以下是不合规范的：

if( a > b){

// do something here

}

#### 二元三元运算符两边用一个空格隔开

规范的写法：

a + b = c;

b - d = e;

return a == b ? 1 : 0;

不规范的写法：

a+b=c;

b-d=e;

return a==b?1:0;

#### 逗号语句后如不还行，紧跟一个空格

规范的写法：

call(a, b, c);

不规范的写法：

call(a,b,c);

### 空行的使用

空行可以表达代码在语义上的分割，注释的作用范围，等等。将类似操作，或一组操作放在一起不用空行隔开，而用空行隔开不同组的代码， 如下：

//插入message表

Message messaage = new Message();

message.setContent(content);

messageDao.insert(message);

//将消息以短信形式发送

SmsUtil.sendMessage(content);

连续两行的空行代表更大的语义分割。

方法之间用空行分割。

域之间用空行分割。

# 注释

## 注释和代码

* 注释宜少且精，不宜多而滥，更不能误导。
* 命名达意，结构清晰，类和方法等责任明确，往往不需要，或者只需要很少注释，就可以让人读懂；相反，代码混乱，再多的注释都不能弥补。所以，应当先在代码本身下功夫。
* 不能正确表达代码意义的注释，只会损害代码的可读性。
* 过于详细的注释，对显而易见的代码添加的注释，罗嗦的注释，还不如不写。

## 注释类型

* // 注释一行 写在行尾
* /\* ...... \*/ 注释多行 写在行前
* /\*\* ...... \*/ 注释多行并写入javadoc文档 写在类，方法，属性前

## 一般注释

一般注释是给代码的维护者看的，着重告述读者为什么这样写，如何修改，注意什么问题等。

复杂逻辑前写注释说明，如：

// 取得数据是Map，对Map进行按首字母排序

// do something

条件判断后面写注释说明，如：

If(success){ // 如果成功，则XXX

// do something

}else{ // 如果不成功，则XXX

//do something

}

## Java Doc

表明类、域和方法等的意义和用法等的注释，要以javadoc的方式来写。Java Doc是给类的使用者来看的，主要介绍是什么，怎么用等信息。凡是类的使用者需要知道，都要用Java Doc 来写。

### Java Doc的位置

类前写注释，如：

/\*\*

\* 有关用户的业务服务接口

\*

\* @author 张三

\*

\*/

public interface UserService {

}

属性前写注释，如：

/\*\*

\* 错误码

\*/

private String code;

/\*\*

\* 错误说明

\*/

private String msg;

方法前写注释，如：

/\*\*

\* 修改密码

\* @param userId 用户id

\* @param oldPassword 旧密码

\* @param newPassword 新密码

\* @param reNewPassword 重复新密码

\* @return 封装的结果对象

\*/

Result<String> updatePassword(Long userId, String oldPassword, String newPassword, String reNewPassword);

# 实践和禁忌

## 每次保存的时候，都让你的代码是符合规范的

程序员都是懒惰的，不要想着等我完成了功能，再来优化代码的格式和结构，等真的把功能完成，很少有人会再愿意回头调整代码。

## 使用log替代System.out.println()

log可以设定级别，可以控制输出到哪里，容易区分是在代码的什么地方打印的，而System.out.print则不行。而且，System.out.print的速度很慢。所以，除非是有意的，否则，都要用log。至少在提交到svn之前把System.out.print换成log。

## 每个if while for等语句，都必须加大括号{}

如：

if (a > b)

a++; 是不符合规范的。

if (a > b) {

a++;

} 是符合规范的。结构清晰，维护不容易出错。

## 善用TODO:

在代码中加入 //TODO: ，大部分的ide都会帮你提示，让你知道你还有什么事没有做。比如：

if (order.isPaid()) {

//TODO: 更新订单

}

## 在需要留空的地方放一个注释，告述读者，你是故意的

比如：

if (!exists(order)) { // 如果订单不存在，则什么也不做

//nothing to do

}

## 不再对boolean值做true false判断

比如：

if (order.isPaid() == true) {

// Do something here

}

不如写成：

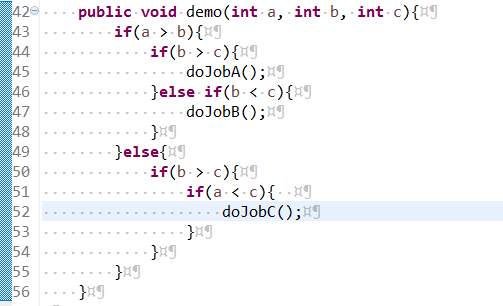
if (order.isPaid()) {

//Do something here

}

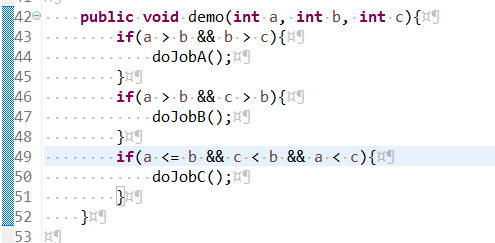
## 减少代码嵌套层次

代码嵌套层次达3层以上时，一般人理解起来都会困难。下面的代码是一个简单的例子：

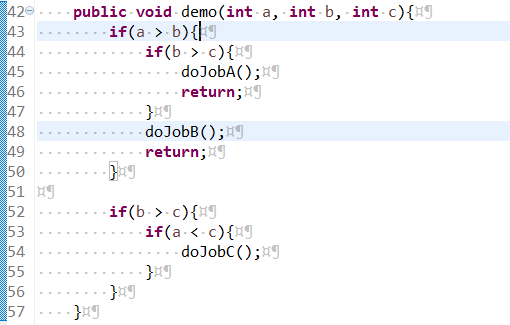


减少嵌套的方法有很多：

* 合并条件



* 利用 return 以省略后面的else



* 利用子方法

利用子方法，就是将嵌套的程序提取出来放到另外的方法里。

## 程序职责单一

关注点分离是软件开发的真理。人类自所以能够完成复杂的工作，就是因为人类能够将工作分解到较小级别的任务上，在做每个任务时关注更少的东西。让程序单元的职责单一，可以使你在编写这段程序时关注更少的东西，从而降低难度，减少出错。

## 变量的声明，初始化和被使用尽量放到一起

比如下面代码：

int orderNum= getOrderNum();

//do something withou orderNum here

call(orderNum);

上例中的注释处代表了一段和orderNum不相关的代码。orderNum的声明和初始化离被使用的地方相隔了很多行的代码，这样做不好，不如这样：

//do something withou orderNum here

int orderNum= getOrderNum();

call(orderNum);

## 循环惯例

* 变量声明在循环外进行

wrong：

while(true) {

String abc=blah();

}

right:

String abc;

while(true) {

abc=blah();

}

* for 循环检查条件应可能采用效率更高的语句

wrong：

for(int i=0;i<list.size();i++) {

}

right

int ubound=list.size();

for(int i=0;i<ubound;i++) {

}

注意：应正确分辨各种操作的开销情况。例如：

for(int i=0;i<arr.length;i++) {

}

这里的.length操作为成员数据访问而非方法调用，没有额外开销。

## 字符串操作

应尽量避免字符串连加。如果要拼接字符串，应尽量采用StringBuffer：

StringBuffer sb=new StringBuffer();

for(int i=0;i<ubound;i++) {

sb.append(i).append(“:”).append(getVal(i)).append(“\r\n”);

}

return sb.toString();

同理适用于多次输出字符串的操作：

wrong：

for(int i=0;i<ubound;i++) {

out.print(i);

out.print(“:”);

out.println(getVal(i));

}

right：

StringBuffer sb=new StringBuffer();

for(int i=0;i<ubound;i++) {

sb.append(i).append(“:”).append(getVal(i)).append(“\r\n”);

}

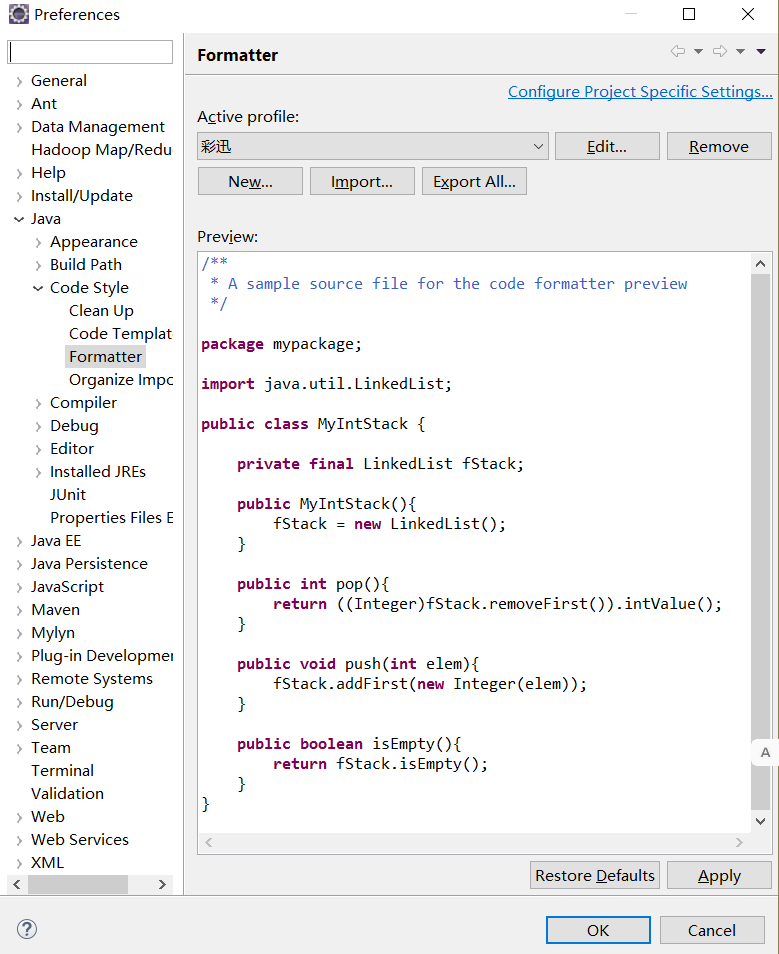
out.print(sb.toString());

# eclipse统一格式化

## 导入格式化文件

Eclipse 导入 richinfo\_java\_formatter.xml ,使用Ctrl+Shift+F格式化JAVA代码。操作步骤：

Window -> Preferences 找到Java -> Code Style -> Formatter 如下图所示

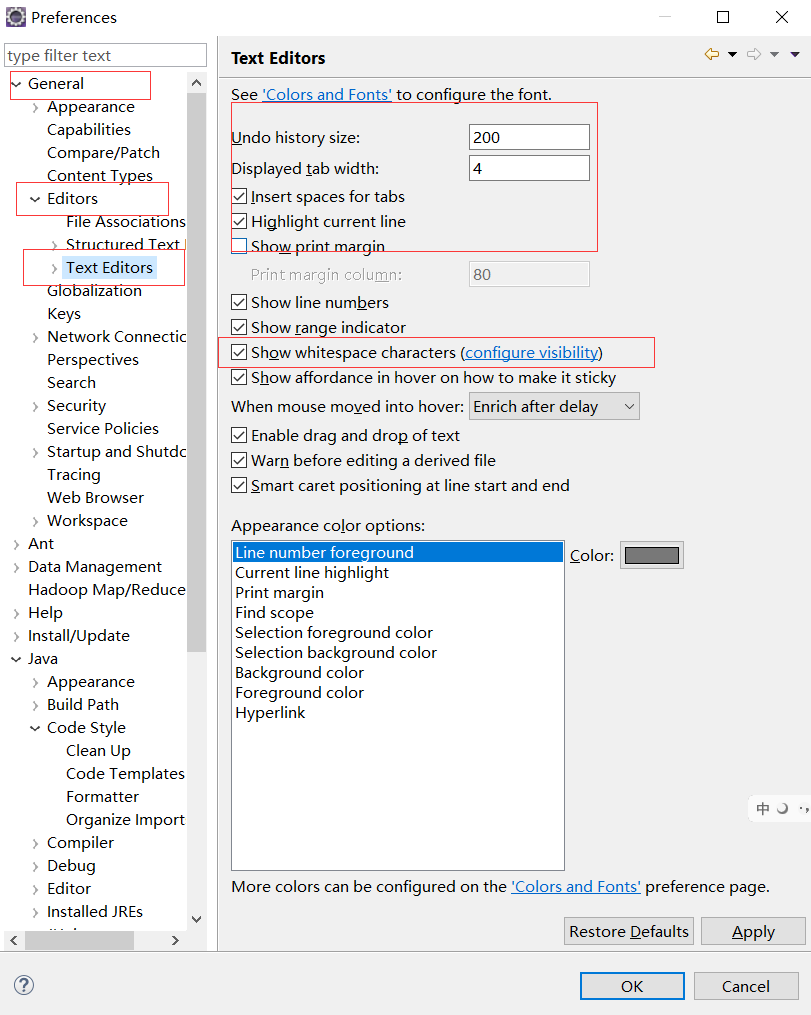


点击Import 选择 richinfo\_java\_formatter.xml 并导入，点击OK确定。

## eclipse中显示空白字符

为了避免空白字符的相互干扰，在eclipse设置空白字符如空格、制表符、换行符等以可见方式显示，步骤如下。

Window -> Preferences 按下图所示设置并点击OK应用



## Xml统一格式化

Window -> Preferences 如下图进行配置

