文件编号：UTRY-CMMI-RD-T-06

**Java代码规范说明书**

版本：V1.0

发布时间：2022-11-14

**浙江远传信息技术股份有限公司**

**变更记录**

\*A - 增加 M - 修订 D - 删除

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 变更  版本号 | 日期 | 变更类型 | 修改人 | 变更摘要 | 备注 |
| V1.0 | 2022-11-14 | A-新增 | 时李樊 | 新增Java代码规范说明书 |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

目录

[1. 前言 6](#_Toc119487373)

[2. 适用范围 6](#_Toc119487374)

[3. 专有名词解释 6](#_Toc119487375)

[4. 命名规范\* 6](#_Toc119487376)

[4.1 Package 的命名 6](#_Toc119487377)

[4.2 Class 的命名 6](#_Toc119487378)

[4.3 Class 成员的命名 7](#_Toc119487379)

[4.4 Static Final 变量的命名 7](#_Toc119487380)

[4.5 数组的命名 7](#_Toc119487381)

[4.6 缩写 7](#_Toc119487382)

[4.7 标识符命名中应注意的问题 7](#_Toc119487383)

[**4.7.1** 不用数字定义名字 7](#_Toc119487384)

[**4.7.2** 用正确的反义词组命名 8](#_Toc119487385)

[5. 样式 8](#_Toc119487386)

[5.1 Java 文件样式 8](#_Toc119487387)

[**5.1.1** 版权信息 8](#_Toc119487388)

[**5.1.2** Package/Imports 9](#_Toc119487389)

[5.2 Class的样式 9](#_Toc119487390)

[**5.2.1** Class的定义 9](#_Toc119487391)

[**5.2.2** Class Field类的属性 9](#_Toc119487392)

[**5.2.3** 构造函数 9](#_Toc119487393)

[**5.2.4** 类成员变量和方法的编写顺序 10](#_Toc119487394)

[**5.2.5** main 方法 10](#_Toc119487395)

[5.3 mapper文件的样式\* 10](#_Toc119487396)

[**5.3.1** mapper文件命名 10](#_Toc119487397)

[**5.3.2** mapper文件内容 10](#_Toc119487398)

[6. 注释\* 10](#_Toc119487399)

[6.1 类和接口注释说明 11](#_Toc119487400)

[6.2 方法注释说明 11](#_Toc119487401)

[7. 书写格式规范 11](#_Toc119487402)

[7.1 代码编写规范 11](#_Toc119487403)

[**7.1.1** 缩进 11](#_Toc119487404)

[**7.1.2** 换行 11](#_Toc119487405)

[**7.1.3** 空行 11](#_Toc119487406)

[**7.1.4** 空格的使用 12](#_Toc119487407)

[**7.1.5** if等语句写法 12](#_Toc119487408)

[**7.1.6** switch语句必须有default分支 12](#_Toc119487409)

[**7.1.7** 循环、判断等语句 12](#_Toc119487410)

[**7.1.8** 一行只写一条语句 13](#_Toc119487411)

[7.2 变量编写规范 13](#_Toc119487412)

[**7.2.1** 公共变量 13](#_Toc119487413)

[**7.2.2** 局部变量 13](#_Toc119487414)

[7.3 程序编写规范 13](#_Toc119487415)

[**7.3.1** exit（） 13](#_Toc119487416)

[**7.3.2** 异常 14](#_Toc119487417)

[**7.3.3** 垃圾收集 14](#_Toc119487418)

[**7.3.4** final 类 14](#_Toc119487419)

[**7.3.5** 访问类的成员变量 14](#_Toc119487420)

[7.4 编程技巧 14](#_Toc119487421)

[**7.4.1** 让调用点显得易懂、容易理解。 14](#_Toc119487422)

[**7.4.2** 减少数据类型转换 14](#_Toc119487423)

[**7.4.3** 防止程序中的垃圾代码。 14](#_Toc119487424)

[**7.4.4** 减少递归调用 14](#_Toc119487425)

[7.5 开发过程中的技巧 15](#_Toc119487426)

[**7.5.1** byte 数组转换到 characters 15](#_Toc119487427)

[**7.5.2** 初始化数组 15](#_Toc119487428)

[**7.5.3** 不必要的对象构造 15](#_Toc119487429)

[**7.5.4** synchronized 关键字 15](#_Toc119487430)

[7.6 程序效率 15](#_Toc119487431)

[**7.6.1** 注意代码的效率 15](#_Toc119487432)

[**7.6.2** 提高代码效率 15](#_Toc119487433)

[**7.6.3** 局部效率应为全局效率服务 15](#_Toc119487434)

[**7.6.4** 循环体内工作量最小化。 15](#_Toc119487435)

[**7.6.5** 改进输入方式 16](#_Toc119487436)

[**7.6.6** 提高调用不频繁的代码效率要慎重 16](#_Toc119487437)

[**7.6.7** 提高空间效率 16](#_Toc119487438)

[**7.6.8** 循环的位置 16](#_Toc119487439)

[**7.6.9** 尽量减少循环嵌套层次。 17](#_Toc119487440)

[**7.6.10** 不要一味追求紧凑的代码。 18](#_Toc119487441)

[8. 代码编辑、编译、审查 18](#_Toc119487442)

[8.1 统一编译环境 18](#_Toc119487443)

[8.2 代码走读及评审 18](#_Toc119487444)

[8.3 软件系统目录 18](#_Toc119487445)

[9. 报销系统\* 19](#_Toc119487446)

[9.1 项目目录说明 19](#_Toc119487447)

[9.2 方法命名说明 21](#_Toc119487448)

# 前言

为确保系统源程序可读性，从而增强系统可维护性，java编程人员应具有基本类似的编程风格，兹制定下述Java编程规范，以规范系统Java部分编程。系统继承的其它资源中的源程序也应按此规范作相应修改。

# 适用范围

本文档将作为java编程人员软件开发的编程格式规范。在项目Java部分的编码、测试及维护过程中，要求严格遵守。

# 专有名词解释

1. POJO（Plain Ordinary Java Object）: 在本规约中，POJO 专指只有 setter/getter/toString 的 简单类，包括 DO/DTO/BO/VO 等。

2.. DTO（Data Transfer Object）：数据传输对象，Service 或 Manager 向外传输的对象。

3. BO（Business Object）：业务对象，可以由 Service 层输出的封装业务逻辑的对象。

4. Query：数据查询对象，各层接收上层的查询请求。注意超过 2 个参数的查询封装，禁止使用 Map 类来传输。 参数个数>=1时，建议使用对象封装。

5. VO（View Object）：显示层对象，通常是 Web 向模板渲染引擎层传输的对象。

# 命名规范\*

定义这个规范的目的是让项目中所有的文档都看起来像一个人写的，增加可读性，减少项目组中因为换人而带来的损失。代码中的命名均不能以下划线或美元符号开始，也不能以下划线或美元符号结束。

## Package 的命名

Package 的名字应该都是由一个小写单词组成。包名统一使用 单数形式，但是类名如果有复数含义，类名可以使用复数形式。

示例：u.utry.cn.admin.moudle

## Class 的命名

Class 的名字采用大驼峰命名法。要尽量准确描述类的功能。具体情况如下表所示：

|  |  |  |
| --- | --- | --- |
| 类型 | 命名格式 | 示例 |
| DTO | 传入Controller的参数使用 DTO 结尾 | 查询：XxxQueryDTO  增：XxxAddDTO  改：XxxUpdateDTO  其他：XxxDto |
| Controller | 使用 Controller结尾 | XxxController |
| Service | 使用 Service结尾 | XxxService |
| Dao | 使用 Dao结尾 | XxxDao |
| Mapper | 使用 Mapper结尾,与Dao文件名前半部分保持一致 | XxxMapper |
| Entity | 实体类，使用 Entity结尾 | XxxEntity |
| 抽象类 | 使用 Abstract 或 Base 开头 | AbstractXxx 或 BaseXxx |
| 异常类 | 使用 Exception 结尾 | ExceptionXxx |
| 测试类 | 以它要测试的类的名称开始，以 Test 结尾 | XxxTest |

## Class 成员的命名

方法名、参数名、成员变量、局部变量：采用小驼峰命名法。**不能使用美元符号或者下划线开头和结尾。**方法名尽量使用“动词+名词”的格式。

示例： projectName、projectNum、addBill

## Static Final 变量的命名

Static Final常量：大写单词组成，单词之间使用“\_”连接，力求语义表达完整清楚，不要嫌名字长。例如：MAX\_INDEX。

应避免使用\_MAX\_NUM\_之类以下划线开始和结尾的定义。

## 数组的命名

类型与中括号紧挨相连来表示数组。数组应该总是用下面的方式来命名：int[] age;而不是：int age[];

## 缩写

某些通用的缩写可以使用，如：temp 可缩写为 tmp ;message 可缩写为 msg ;

## 标识符命名中应注意的问题

### 不用数字定义名字

除非必要，不要用数字或较奇怪的字符来定义标识符。

示例：如下命名，使人产生疑惑。void set\_age00( int age );

应改为有意义的单词命名：void setAge( int age );

### 用正确的反义词组命名

用正确的反义词组命名具有互斥意义的变量或相反动作的函数等。

说明：下面是一些在软件中常用的反义词组。

begin / end create / destroy insert / delete

first / last get / set increment / decrement lock / unlock open / close min / max old / new start / stop next / previous source / target show / hide send / receive cut / paste up / down

示例：

int minSum; int maxSum;

int addAge( int age); int deleteUser( BYTE \*userName );

# 样式

## Java 文件样式

所有的 Java(\*.java) 文件都必须遵守如下的样式规则

### 版权信息

版权信息必须包含作者、版本、公司、权限、文件新建日期、“自从”哪个版本开始引入的。

示例：

/\*\*  
 \* [ 日报首页-我的审批 ]  
 \*  
 \* @author shilifan  
 \* @version 1.0  
 \* @company 远传(Utry)  
 \* @copyright (c) 远传(Utry)Inc. All rights reserved.  
 \* @date 2022-05-26 10:28:13  
 \* @since JDK1.8  
 \*/

### Package/Imports

package 行要在 import 行之前，import 中标准的包名要在本地的包名之前，而且按照字母顺序排列。如果 import 行中包含了同一个包中的不同子目录，则建议用 \* 来处理。

package com.nantian;

import java.io.\*;

import java.util.Observable;

import translator;

这里 java.io.\* 使用来代替InputStream and OutputStream 的。

## Class的样式

### Class的定义

包含了在不同的行的 extends 和 implements.

实例：public class DogServiceImpl extends Ainmal implements DogService

### Class Field类的属性

类的成员变量：

/\*\*

\* 年龄

\*/

protected int age;

public 的成员变量一定要有注释而且必须生成文档（JavaDoc）。

proceted、private和 package 定义的成员变量如果名字含义明确的话，可以没有注释。

### 构造函数

构造函数，它应该用递增的方式写（比如：参数多的写在后面）。示例：

public CounterSet(int size)

{

this.size = size;

}

public CounterSet(int size,String name)

{

this.size = size;

this.name = name;

}

### 类成员变量和方法的编写顺序

建议编写顺序为：

public protected private

final static transient

### main 方法

如果main(String[]) 方法已经定义了, 那么它应该写在类的底部。

## **mapper文件的样式\***

### mapper文件命名

mapper文件名与dao文件名保持一致。例如：ProjectDao->ProjectMapper.xml;BillDao->BillDao.

### mapper文件内容

1.mapper标签的namespace=“对应dao类的绝对路径”.mapper中的id、参数名、参数类型、返回值类型应与dao中保持一致。

2.<mapper></mapper>、<resultMap></resultMap>和增删改查标签成对出现。

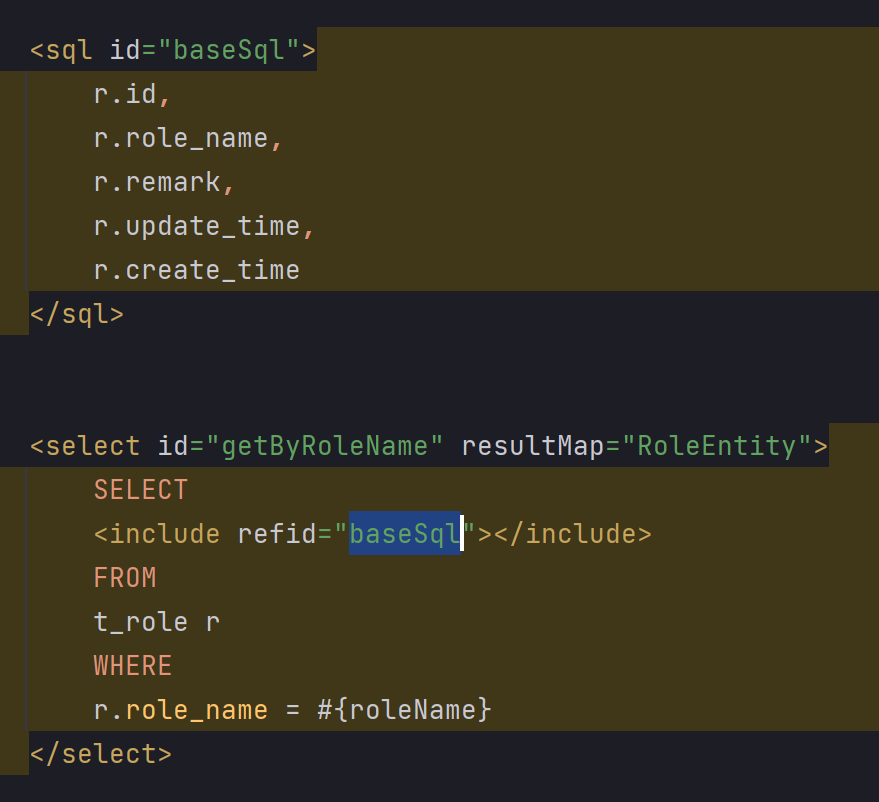
3.resultMap标签必须统一放在增删改查标签前面，也可当作增删改查标签的属性，放在里面。

4.同一个mapper文件内所有标签的id具有唯一性。

5.查询字段时尽量少使用t.\*，影响代码性能，查询那个字段，写那个字段。

6.查询时，除了查询结果是list外，都应该用limit 1限制查询条数。

**7.公共字段、sql的引入**



# 注释\*

一般情况下：源程序有效注释量必须在20％以上。

说明：注释的原则是有助于对程序的阅读理解，在该加的地方都加了，注释不宜太多也不能太少，注释语言必须准确、易懂、简洁。注释起始为“/\*\*…\*/”，注释文档的第一条为总结性语句.

## 类和接口注释说明

类注释必须置于任何一个import语句后面，同时位于class定义的前面。

@author 名字：建立一个“作者”条目。

## 方法注释说明

紧靠在每条方法的前面，必须有一个它所描述的那个方法的签名。

@param 变量描述：给“parameters”（参数）小节增添一个条目。

@return 描述：增添一个“returns”（返回值）小节。

@throws 类描述：为方法的“throws”（产生违例）小节增添一个条目。

如：

/\*\*

将一个双精度数格式化成一个字串

@param x 要格式化的数字

@return 格式化成的字串

@throws 如参数错误，产生IllegalArgumentException（非法参数违例）

\*/.

# 书写格式规范

## 代码编写规范

**使用idea快捷键：Ctrl+Alt+L，代码会自动格式化。**

### 缩进

采用 4 个空格缩进，禁止使用 Tab 字符.。

### 换行

IDE 的 text file encoding 设置为 UTF-8; IDE 中文件的换行符使用 Unix 格式，不要 使用 Windows 格式. 如果需要换行的话，尽量用 println 来代替在字符串中使用"\n"。

正例：System.out.print("Hello,world!\n");

反例：System.out.println("Hello,world!");

### 空行

相对独立的程序块之间、变量说明之后必须加空行。

### 空格的使用

任何二目、三目运算符的左右两边都需要加一个空格。

正例：c= a + b;

反例：c=a+b;

### if等语句写法

if、for、do、while、case、switch、default、try、catch等语句自占一行，“{”必须回行编写，且if、for、do、while等语句的执行语句部分必须用{}括起回行编写，若有执行语句只有一条，{}也不可缺省。

示例：如下例子不符合规范。

if (pUserCR == NULL) return;

应如下书写：

if (pUserCR == NULL)

{

return;

}

### switch语句必须有default分支

### 循环、判断等语句

循环、判断等语句中若有较长的表达式或语句，则要进行适当的划分，长表达式要在低优先级操作符处划分新行，操作符放在新行之首。

示例：

if ((taskNo < maxActTaskNumber) && (timeNo < minActTimeNumber)

&& (n7StatItemValid(statItem)))

{

... // program code

}

for (i = 0, j = 0; (i < bufferKeyword[WordIndex].word\_length)

&& (j < newKeyword.word\_length); i++, j++)

{

... // program code

}

for (i = 0, j = 0;

(i < firstWordLength) && (j < secondWordLength);

i++, j++)

{

... // program code

}

### 一行只写一条语句

不允许把多个短语句写在一行中，即一行只写一条语句。

示例：如下例子不符合规范。

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

应如下书写

rect.length = 0;

rect.width = 0;

## 变量编写规范

### 公共变量

#### 去掉没必要的公共变量。

说明：公共变量是增大模块间耦合的原因之一，故应减少没必要的公共变量以降低模块间的耦合度。

#### 仔细定义并明确公共变量的含义、作用、取值范围及公共变量间的关系。

说明：在对变量声明的同时，应对其含义、作用及取值范围进行注释说明，同时若有必要还应说明与其它变量的关系。

#### 当向公共变量传递数据时，要十分小心，防止赋与不合理的值或越界等现象发生。

说明：对公共变量赋值时，若有必要应进行合法性检查，以提高代码的可靠性、稳定性。

### 局部变量

#### 防止局部变量与公共变量同名。

说明：若使用了较好的命名规则，那么此问题可自动消除。

## 程序编写规范

### exit（）

exit 除了在 main 中可以被调用外，其他的地方不应该调用。因为这样做不给任何代码机会来截获退出。一个类似后台服务的程序不应该因为某一个库模块决定了要退出就退出。

### 异常

申明的错误应该抛出一个RuntimeException或者派生的异常。申明的错误主要指前提条件违例、处理流程违例的情况。对于功能性的分支建议采用返回值的方式。

异常建议根据模块结构，采用逐级处理的方式，并打印（或者记录在日志中）。同一模块层次的异常，按功能可由某一模块集中处理。

### 垃圾收集

JAVA使用成熟的后台垃圾收集技术来代替引用计数。但是这样会导致一个问题：你必须在使用完对象的实例以后进行清场工作（将对象置为NULL后，引用计数自动－1）。

### final 类

绝对不要因为性能的原因将类定义为 final 的（除非程序的框架要求）

如果一个类还没有准备好被继承，最好在注释中注明，而不要将它定义为 final 的。这是因为没有人可以保证会不会由于什么原因需要继承。

### 访问类的成员变量

大部分的类成员变量应该定义为 private 的来防止继承类使用他们。

## 编程技巧

一般性原则：检查所有参数输入的有效性。

检查参数输入：检查所有非参数输入的有效性，如数据文件、公共变量等。

说明：输入主要有两种：一种是参数输入；另一种是全局变量、数据文件的输入，即非参数输入。在使用输入之前，应进行必要的检查。

### 让调用点显得易懂、容易理解。

### 减少数据类型转换

在填写参数时，应尽量减少没有必要的默认数据类型转换或强制数据类型转换。

说明：因为数据类型转换或多或少存在危险。

### 防止程序中的垃圾代码。

说明：程序中的垃圾代码不仅占用额外的空间，而且还常常影响程序的功能与性能，很可能给程序的测试、维护等造成不必要的麻烦。

### 减少递归调用

说明：递归调用影响程序的可理解性；递归调用一般都占用较多的系统资源（如栈空间）；递归调用对程序的测试有一定影响。故除非为某些算法或功能的实现方便，应减少没必要的递归调用。

## 开发过程中的技巧

### byte 数组转换到 characters

为了将 byte 数组转换到 characters，你可以这么做："Hello world!".getBytes();

### 初始化数组

下面的代码是一种很好的初始化数组的方法：

objectArguments = new Object[] { arguments };

### 不必要的对象构造

不要在循环中构造和释放对象

### synchronized 关键字

避免太多的使用 synchronized 关键字

避免不必要的使用关键字synchronized，应该在必要的时候再使用它，这是一个避免死锁的好方法。

Borland Jbulider 不喜欢 synchronized 这个关键字，如果你的断点设在这些关键字的作用域内的话，调试的时候你会发现的断点会到处乱跳，让你不知所措。除非必须，尽量不要使用。

## 程序效率

### 注意代码的效率

编程时要经常注意代码的效率。

说明：代码效率分为全局效率、局部效率、时间效率及空间效率。全局效率是站在整个系统的角度上的系统效率；局部效率是站在模块或函数角度上的效率；时间效率是程序处理输入任务所需的时间长短；空间效率是程序所需内存空间，如机器代码空间大小、数据空间大小、栈空间大小等。

### 提高代码效率

在保证系统的正确性、稳定性、可读性及可测性的前提下，提高代码效率。

说明：不能一味地追求代码效率，而对软件的正确性、稳定性、可读性及可测性造成影响。

### 局部效率应为全局效率服务

局部效率应为全局效率服务，不能因为提高局部效率而对全局效率造成影响。

### 循环体内工作量最小化。

说明：应仔细考虑循环体内的语句是否可以放在循环体之外，使循环体内工作量最小，从而提高程序的时间效率。

示例：如下代码效率不高。

for (i = 0; i < MAX\_ADD\_NUMBER; i++)

{

sum += i;

BackSum = sum; /\* backup sum \*/

}

语句“BackSum = sum;”完全可以放在for语句之后，如下。

for (ind = 0; ind < MAX\_ADD\_NUMBER; ind++)

{

sum += i;

}

BackSum = sum; /\* backup sum \*/

仔细分析有关算法，并进行优化。

### 改进输入方式

仔细考查、分析系统及模块处理输入（如事务、消息等）的方式，并加以改进。

### 提高调用不频繁的代码效率要慎重

不应花过多的时间拼命地提高调用不很频繁的代码效率。

说明：对代码优化可提高效率，但若考虑不周很有可能引起严重后果。

### 提高空间效率

在保证程序质量的前提下，通过压缩代码量、去掉不必要代码以及减少不必要的局部和全局变量，来提高空间效率。

说明：这种方式对提高空间效率可起到一定作用，但往往不能解决根本问题。

### 循环的位置

在多重循环中，应将最忙的循环放在最内层。

说明：减少CPU切入循环层的次数。

示例：如下代码效率不高。

for (row = 0; row < 100; row++)

{

for (col = 0; col < 5; col++)

{

sum += a[row][col];

}

}

可以改为如下方式，以提高效率。

for (col = 0; col < 5; col++)

{

for (row = 0; row < 100; row++)

{

sum += a[row][col];

}

}

### 尽量减少循环嵌套层次。

避免循环体内含判断语句，应将循环语句置于判断语句的代码块之中。

说明：目的是减少判断次数。循环体中的判断语句是否可以移到循环体外，要视程序的具体情况而言，一般情况，与循环变量无关的判断语句可以移到循环体外，而有关的则不可以。

示例：如下代码效率稍低。

for (i = 0; i < MAX\_RECT\_NUMBER; i++)

{

if (DataType == RECT\_AREA)

{

AreaSum += RectArea[i];

}

else

{

RectLengthSum += Rect[i].length;

RectWidthSum += Rect[i].width;

}

}

因为判断语句与循环变量无关，故可如下改进，以减少判断次数。

if (DataType == RECT\_AREA)

{

for (i = 0; i < MAX\_RECT\_NUMBER; i++)

{

AreaSum += RectArea[i];

}

}

else

{

for (i = 0; i < MAX\_RECT\_NUMBER; i++)

{

RectLengthSum += Rect[i].length;

RectWidthSum += rect[i].width;

}

}

### 不要一味追求紧凑的代码。

说明：因为紧凑的代码并不代表高效的机器码

# 代码编辑、编译、审查

## 统一编译环境

在产品软件（项目组）中，要统一编译环境，相同的JDK版本

## 代码走读及评审

通过代码走读及审查方式对代码进行检查。

（1）代码走读主要是对程序的编程风格如注释、命名等以及编程时易出错的内容进行检查，可由开发人员自己或开发人员交叉的方式进行。

（2）代码审查主要是对程序实现的功能及程序的稳定性、安全性、可靠性等进行检查及评审，可通过自审、交叉审核或指定部门抽查等方式进行。

## 软件系统目录

合理地设计软件系统目录，方便开发人员使用。

说明：方便、合理的软件系统目录，可提高工作效率。目录构造的原则是方便有关源程序的存储、查询、编译、链接等工作，同时目录中还应具有工作目录----所有的编译、链接等工作应在此目录中进行，工具目录----有关文件编辑器、文件查找等工具可存放在此目录中。

# 报销系统\*

## 项目目录说明

红色字体是报销系统相关的文件夹。

new-utmp-admin // 父项目名

├── utmp-admin-api // 项目名

│ ├── doc // 存放文档

│ ├──src // 代码

│ │ ├──main

│ │ │ ├──java

│ │ │ │ ├── u.utry.cn.admain

│ │ │ │ │ ├── common // 公共模块

│ │ │ │ │ │ ├──anno // 切面

│ │ │ │ │ │ ├──constant // 常量

│ │ │ │ │ │ │ ├──activi // 流程引擎相关常量

│ │ │ │ │ │ │ ├──dailyReport // 日报相关常量

│ │ │ │ │ │ │ ├──easytrack // 项目管理工具相关常量

│ │ │ │ │ │ │ ├──finance // 业财相关常量

│ │ │ │ │ │ │ ├──sys // 系统共用相关常量

│ │ │ │ │ │ │ ├──zentao // 禅道相关常量

│ │ │ │ │ │ │ ├──reimbursementSystem // 报销系统相关常量

│ │ │ │ │ │ ├──exception // 流程引擎

│ │ │ │ │ │ ├──heartbeat // 服务心跳

│ │ │ │ │ │ ├──json //json序列化

│ │ │ │ │ │ ├──kaptcha // 验证码

│ │ │ │ │ │ ├──reload // 调度器

│ │ │ │ │ │ ├──salary // 薪资相关

│ │ │ │ │ │ ├──sendEmail // 发送邮件

│ │ │ │ │ │ ├──swagger // swagger接口相关

│ │ │ │ │ │ ├──vaildator // 系统校验

│ │ │ │ │ │ ├──xxx // 发票验真

│ │ │ │ │ │ ├──xxx // 差旅商场

│ │ │ │ │ ├──config //系统配置

│ │ │ │ │ ├──constant // 系统常量

│ │ │ │ │ ├──handler //

│ │ │ │ │ ├──interceptor //拦截器

│ │ │ │ │ ├──listenter //监听器

│ │ │ │ │ ├──module

│ │ │ │ │ │ ├──activit // 流程引擎

│ │ │ │ │ │ ├──board // 首页看板

│ │ │ │ │ │ ├──business //系统业务相关代码

│ │ │ │ │ │ │ ├──apidb // 阿里相关接口

│ │ │ │ │ │ │ ├──project // 项目管理模块

│ │ │ │ │ │ │ ├──… //其他模块

│ │ │ │ │ │ │ ├──reimbursementSystem // 报销系统相关业务代码

│ │ │ │ │ │ │ │ ├──billManage // 单据管理（大模块名）

│ │ │ │ │ │ │ │ │ ├──reimbursementManage

// 报销单管理（小模块名）

│ │ │ │ │ │ │ │ │ │ ├──controller //

│ │ │ │ │ │ │ │ │ │ ├──dao //

│ │ │ │ │ │ │ │ │ │ ├──domain //

│ │ │ │ │ │ │ │ │ │ │ ├──dto //参数

│ │ │ │ │ │ │ │ │ │ │ ├──entity //实体，与数据库字段一致，增删改，与vo保持一致。

│ │ │ │ │ │ │ │ │ │ │ ├──vo //返回参数

│ │ │ │ │ │ │ │ │ │ ├──service // 业务类

│ │ │ │ │ │ ├──hr // 人事相关的模块

│ │ │ │ │ │ ├──support // 运维管理

│ │ │ │ │ │ ├──system // 系统设置相关的模块

│ │ │ │ │ ├── third // 第三方工具类

│ │ │ │ │ ├── util // 工具类

│ │ │ │ │ ├── UTMPAdminApplication //项目启动类

│ │ │ ├── resources

│ │ │ │ ├──dev //开发环节相关配置

│ │ │ │ ├──mapper // sql文件

│ │ │ │ │ ├── activit // 流程引擎

│ │ │ │ │ ├── board //首页看板

│ │ │ │ │ ├── business//系统业务

│ │ │ │ │ ├── easytrack //项目管理工具

│ │ │ │ │ ├── hr //人事

│ │ │ │ │ ├── support // 运维管理

│ │ │ │ │ ├── system //系统设置

│ │ │ │ │ ├── reimbursementSystem //报销系统相关mapper文件

│ │ │ │ │ │ ├──billManage // 单据管理文件夹名称

│ │ │ │ │ │ │ ├──reimbursementManageMapper.xml

// 报销单管理dao对应的mapper文件

│ │ │ │ ├──pre // 预上线环境相关配置，一般不会用

│ │ │ │ ├──processes //流程定义相关文件

│ │ │ │ ├──prod // 正式环境相关配置，尽量不要动

│ │ │ │ ├──sit // 系统集成测试

│ │ │ │ ├──sql // 系统业务相关sq文件

│ │ │ │ │ ├── reimbursementSystem //报销系统相关表sql

│ │ │ │ ├──其他系统相关文档 //

│ │ ├──test //测试单元

│ │ │ ├── java

│ │ │ │ ├──u.utry.cn.admain

│ │ │ │ │ ├──module.support

│ │ │ │ │ │ ├──codegenerator // 自动生成代码的测试文件

│ │ │ │ │ │ ├──mengtest // 小孟测试文件

│ │ │ │ │ │ ├──p9915test // 彭强测试文件

│ │ │ │ │ │ ├──shitest // 小时测试文件

│ │ │ │ │ │ ├──lyptest // 云鹏测试文件

│ ├──target // 已编译的类

│ ├──pom.xml // 依赖

├── pom.xml //父项目依赖

目前做的UTMP系统，里面包含报销系统等小系统，目录结构须按照模板目录来。

比如，在moudle里建一个文件夹reimbursementSystem，里面放报销系统相关业务文件。（如果有新增的文件类型参照模板目录创建）

例如：

├──business //系统业务相关代码

│ ├──apidb // 阿里相关接口

│ ├──project // 项目管理模块

│ ├──… //其他模块

├──reimbursementSystem // 报销系统相关业务代码

│ ├──billManage // 单据管理（大模块名）

│ │ ├──reimbursementManage // 报销单管理（小模块名）

│ │ │ ├──controller //

│ │ │ ├──dao //

│ │ │ ├──domain //

│ │ │ │ ├──dto //

│ │ │ │ ├──entity //

│ │ │ │ ├──vo //

│ │ │ ├──service // 业务类

mapper 同理

├──mapper // sql文件

│ ├── activit // 流程引擎

│ ├── board //首页看板

│ ├── business//系统业务

│ ├── easytrack //项目管理工具

│ ├── hr //人事

│ ├── support // 运维管理

│ ├── system //系统设置

│ ├── reimbursementSystem //报销系统相关mapper文件

│ │ ├──billManage // 单据管理文件夹名称

│ │ │ ├──reimbursementManageMapper.xml // 报销单管理dao对应的mapper文件

## 方法命名说明

以报销系统中”我的单据”模块为例：

|  |  |  |
| --- | --- | --- |
| 操作类型 | url（Controller中有） | 方法命名 |
| 增 | /reimbursementSystem/myBill/insert/bill | insertBill() |
| 删 | /reimbursementSystem/myBill/delete/bill | deleteBill()、  batchDeleteBill() |
| 改 | /reimbursementSystem/myBill/update/bill | updateBill()、beatchUpdateBill() |
| 查 | /reimbursementSystem/myBill/select/billList | **queryBillList()** |

删、改、查批量操作时，方法名后加s.