**客户名字和logo**

餐厅订餐系统\_开发编码规范

[项目ID]

Zhi Luo

日期：2015-01-17

文档版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修订日期 | 修订人 | 审核人 | 变更内容 |
| 0.1 | 2015-01-08 | LUO ZHI |  | 初始化 |
| 0.2 | 2015-01-15 | LUO ZHI | CHEN PENG | 完善章节5，6，7 |
| 0.3 | 2015-01-17 | LUO ZHI |  | 增加日志输出等级使用场景说明 |

目录

[1 文档介绍 1](#_Toc409267927)

[1.1 目的 1](#_Toc409267928)

[1.2 范围 1](#_Toc409267929)

[1.3 缩写词列表 1](#_Toc409267930)

[1.4 参考内容 1](#_Toc409267931)

[2 命名规范 2](#_Toc409267932)

[2.1 驼峰命名 2](#_Toc409267933)

[2.2 源文件命名及编码 2](#_Toc409267934)

[2.3 包命名规范 2](#_Toc409267935)

[2.4 类命名规范 2](#_Toc409267936)

[2.5 方法命名规范 2](#_Toc409267937)

[2.6 常量命名规范 3](#_Toc409267938)

[2.7 变量命名规范 3](#_Toc409267939)

[2.8 参数命名规范 3](#_Toc409267940)

[3 源文件结构 3](#_Toc409267941)

[3.1 包引入规范 3](#_Toc409267942)

[3.2 类声明规范 3](#_Toc409267943)

[3.3 注释规范 4](#_Toc409267944)

[3.3.1 源文件注释 4](#_Toc409267945)

[3.3.2 类注释 4](#_Toc409267946)

[3.3.3 变更注释 4](#_Toc409267947)

[3.3.4 方法注释 4](#_Toc409267948)

[3.3.5 代码内注释 5](#_Toc409267949)

[4 代码排版 5](#_Toc409267950)

[4.1 括号 5](#_Toc409267951)

[4.2 缩进 5](#_Toc409267952)

[4.3 每行字符限制 5](#_Toc409267953)

[4.4 单行语句 5](#_Toc409267954)

[4.5 代码行限制 5](#_Toc409267955)

[5 代码书写规范 5](#_Toc409267956)

[5.1 简单语句 5](#_Toc409267957)

[5.2 复杂语句 6](#_Toc409267958)

[5.3 返回语句 6](#_Toc409267959)

[5.4 条件语句 6](#_Toc409267960)

[5.5 分支语句 6](#_Toc409267961)

[5.6 循环语句 7](#_Toc409267962)

[5.7 异常控制 7](#_Toc409267963)

[6 日志规范 7](#_Toc409267964)

[6.1 禁止使用system.out.println 7](#_Toc409267965)

[6.2 log4j介绍 7](#_Toc409267966)

[6.3 日志等级 8](#_Toc409267967)

[6.4 日志分类 8](#_Toc409267968)

[6.5 日志格式 8](#_Toc409267969)

[6.6 日志配置说明 8](#_Toc409267970)

[7 编程惯例 9](#_Toc409267971)

[7.1 常量 9](#_Toc409267972)

[7.2 异常按层次抛出 10](#_Toc409267973)

[7.3 国际化 10](#_Toc409267974)

[7.4 字符串生成 10](#_Toc409267975)

[7.5 Model类实现 10](#_Toc409267976)

[7.6 面向接口编程 10](#_Toc409267977)

[7.7 避免使用自带同步机制的集合类 10](#_Toc409267978)

[7.8 变量初始化 10](#_Toc409267979)

# 

# 文档介绍

## 目的

本文档描述用于餐厅订餐系统实现代码的编码规范。

## 范围

本文档描述内容包括地餐厅订餐系统实现代码的代码命名，排版，注释，语句以及日志的编写规范。本文档的不包括任何餐厅订餐系统的系统框架，设计和实现的任何部分。本文仅供餐厅订餐系统开发成员使用。

## 缩写词列表

n/a

## 参考内容

|  |  |
| --- | --- |
| 文献名称 | 文献资源 |
| 谷歌编码规范 | http://google-styleguide.googlecode.com/svn/trunk/javaguide.html |
| SUN JAVA编码规范 | http://www.oracle.com/technetwork/articles/javase/codeconvtoc-136057.html |
|  |  |

# 命名规范

## 驼峰命名

餐厅订餐系统Java实现代码均使用驼峰命名规则。驼峰命名规则用户类，函数名，变量名定义时，如果使用多个英文单词连接，将单词首字母大写，形势骆驼的驼峰而得名。例如：eOrderService。

在形式上驼峰命名又分为小驼峰和大驼峰，区别在于第一个单词的首字母是否大写。如果大写，则是大驼峰命名；反之，则是小驼峰命名。

本文约定餐厅订餐系统中，源文件，类，接口定义均使用大驼峰命名；而方法，变量使用小驼峰。

本文唯一例子是常量的命名采用全大写希腊式命名。希腊式命名使用下划线连接单词，例如：USR\_CUSTOMER\_UID

## 源文件命名及编码

源文件使用大驼峰命名，使用UTF-8编码。

## 包命名规范

包名使用com.<group\_name>.<proj\_name>.<module>的命名方式。针对餐厅订餐系统，group\_name值为innovaee，proj\_name为eorder

|  |  |
| --- | --- |
| 顶级包名 | com.innovaee.eorder |
| 工具包 | com.innovaee.eorder.util |
| 配置包 | com.innovaee.eorder.conf |
| Model包 | com.innovaee.eorder.model |
| Service接口包 | com.innovaee.eorder.service |
| Service 实现包 | com.innovaee.eorder.service.imp |
| Controller包 | com.innovaee.eorder.controller |
| 异常包 | com.innovaee.eorder.exception |

## 类命名规范

类命名使用大驼峰命名，且必须采用名词。例如：UserInfo。接口定义最好采用名词，也可以采用形容词，例如：Readable。单元测试类采用相应类名加Test命名规则，例如：LoginServiceTest。

## 方法命名规范

方法名采用小驼峰命名，多采用动词和动词短语。例如：payBill()

## 常量命名规范

常量采用全大写希腊式命名，必须使用名词。例如：USR\_CUSTOMER\_UID

## 变量命名规范

变量命名使用小驼峰命名，使用名词。例如：currentDish

## 参数命名规范

参数命名采用小驼峰命名，使用名词。

# 源文件结构

## 包引入规范

import 语句不允许出现通配符，以避免引入未使用的类/接口。包引入根据提供方不同进行分组，且应遵循如下顺序

* static imports
* com.innovaee.eorder 命名空间下的引入
* 第三方包引入，例如Spring，Hibernate
* Java包引入
* Javax包引入

## 类声明规范

一个源文件只允许有一个类定义，如果因为编码需要引入内部类(inner-class)，需将内部类单独列出。对于匿名内部类不做要求。

## 注释规范

### 源文件注释

每个源文件在开始处应有关于文件版权声明，源文件名，创建日期等信息

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Filename : User.java

\* Copyright : Copyright (c) 2014

\* Company : Innovaee

\* Created : 11/27/2014

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

### 类注释

每个类名定义开始位置应有关于作者，类简单说明等信息

/\*\*

\* @Title: User

\* @Description: 用户实体

\*

\* @version V1.0

\*/

### 变更注释

接着类注释下面应该有完整的变更历史，每条变更历史包括变更人，时间，引起变更的需求描述。

/\*\*

\* CR Responsibility Date Description

\* -----------------------------------------------------------------------

\* #0123 Aaron 1/1/2015 Add comments

\*/

### 方法注释

每个方法应有方法注释，包括方法作用，输入参数，输出参数等信息

/\*\*

\* 构造函数

\*

\* @param username用户名称

\* @param password密码

\* @param cellphone手机号码

\* @param levelId等级ID

\* @param userStatus用户状态

\*/

### 代码内注释

代码内注释用于解释比较复杂的业务逻辑，核心算法，使用单行注释

//判断用户是否属于管理员

# 代码排版

## 括号

括号与if, else, for ,do, while等连接代码块的语句一起使用，规则如下，

* 左括号{之前没有换行符
* 左括号之后紧跟换行符
* 右括号之前换行

例如下代码：

while (conditionIsTrue) {

}

## 缩进

需要缩进的语句采用4字符缩进。

## 每行字符限制

单行代码字符不要超过80字符，如果超出应合理换行

## 单行语句

一个代码行只有一个语句

## 代码行限制

对于逻辑复杂的代码块或者方法，代码行除开括号应控制在60行以内。

# 代码书写规范

## 简单语句

简单语句必须满足一行至多只有一句，例如：

int amountOfDish;

int tableNumber;

下面书写方式是不合适的，

int amountOfDish, tableNumber;

## 复杂语句

复杂语句只一组需要使用花括号{}来组织在一起的语句，例如下面会介绍的条件语句，分支语句等。

## 返回语句

返回语句return 一般不建议使用括号()除非有方法调用或者二元语句，从编码最佳实践来看，应该尽量避免，例如：

return dishList.size();

## 条件语句

条件语句包括if，if-else和if-else if-else，无论是那种形式，都必须使用花{}来组织代码块，例如：

if (condition) {

statements;

}

if (condition) {

statements:

} else {

statements;

}

if (condition) {

statements;

} else if (condition) {

statements;

} else {

statements;

}

## 分支语句

分支语句switch使用如下形式。在使用分支语句时一般建议给每个case分支添加break语句，同时提供一个default分支处理以上case分支没有覆盖到的情况。

switch (condition) {

case ABC:

statements;

break;

case DEF:

statements;

break;

default:

statements;

}

## 循环语句

循环语句有for, while和do-while三种形式，而在Java5以后版本，对for循环语句添加了一种新的形式。对于循环语句的几种形式中，需要对do-while特别注意的就是无论判断条件condition是否为真(true)，都会执行一次循环体中的语句。

for (initialization; condition; update) {

statements;

}

for(iterator : collection) {

statements;

}

while (condition) {

statements;

}

do {

statements;

} while (condition);

## 异常控制

异常捕获使用try-catch-finally语句，finally语句不是必须的，但是对于文件，数据库连接等IO资源操作的时候，finally是必须添加用来释放资源，以免因为异常抛出而导致资源没有被释放，

try {

statements;

} catch (ExceptionClass e) {

statements;

} finally {

statement;

}

# 日志规范

## 禁止使用system.out.println

禁止在代码中使用system.out.println ()对调试信息进行输出

## log4j介绍

log4j是一款开源的Java应用日志输出组件，其配置灵活，简单，学习曲线低而被大量组织和个人采用。官方网站<http://logging.apache.org/log4j/1.2/>

## 日志等级

日志等级根据log4j的定义从优先级高到低可以分为FATAL、ERROR、WARN、INFO和DEBUG。通常情况下，使用除FATAL和WARN之外的三种等级。优先级高于设置日志等级的日志信息会被输出，例如如果日志等级设置为INFO，则系统运行期间的ERROR和WARN信息会被输出，而DEBUG信息则不会被输出

* ERROR：在系统捕获异常需要记录异常信息的时候时候ERROR级别
* INFO：用于记录系统运行期的操作流程日志，比如用户登录，用户查询等
* DEBUG：用于系统开发阶段，输出调试信息

## 日志分类

log4j的日志比较常用的种类有控制台日志、循环文件日志和每日文件日志。

* **控制台日志：** 将日志信息输出到控制台，用来取代system.out.println()，方便开发期间的调试工作，通常不用于生产环境
* **循环文件日志：**将日志信息输出添加到某个特定文件最后一行，但文件大小达到设定的值，则文件大小不再增加，而是将最早的日志进行删除，从而保证日志文件大小稳定，不会占用硬盘空间资源，但过往日志丢失可能会影响故障排查
* **每日文件日志：**每日文件日志会为某个特定文件自动每天产生一个带有日期信息后缀的备份文件，例如，一个日志文件名为dish\_ops.log，那么除开当前正在使用的这个日志文件，系统会按天为过往日志生产名如dish\_ops.log.yyyymmdd的日志备份，这种日志类型确保单个日志文件不会过大，又对过往日志有存档备份，建议在生产环境中采用这种类型的日志

## 日志格式

日志信息本体使用key-value键值对的类JSON方式输出，例如：

logger.info(“[功能=用户登录,用户名=huihui,登录结果=false,失败原因=密码错误]”);

## 日志配置说明

log4j的配置文件有XML和properties文件两种格式，properties文件简单易懂，建议使用properties文件，下面是一个典型的log4j配置文件样例，

#console输出用appender

log4j.appender.console=org.apache.log4j.ConsoleAppender

log4j.appender.console.layout=org.apache.log4j.PatternLayout

log4j.appender.console.layout.ConversionPattern=[%d{yyyy-MM-dd HH:mm:ss}][%-5p] [%c] - %m%n

#debug输出用appender

log4j.appender.debug\_log=org.apache.log4j.RollingFileAppender

log4j.appender.debug\_log.File=debug.log

log4j.appender.debug\_log.MaxFileSize=500KB

log4j.appender.debug\_log.MaxBackupIndex=1

log4j.appender.debug\_log.layout=org.apache.log4j.PatternLayout

log4j.appender.debug\_log.layout.ConversionPattern=[%d{yyyy-MM-dd HH:mm:ss}][%-5p] [%c] - %m%n

#info输出用appender

log4j.appender.sysinfo\_log=org.apache.log4j.DailyRollingFileAppender

log4j.appender.sysinfo\_log.File=sysinfo.log

log4j.appender.sysinfo\_log.DatePattern='.'yyyy\_MM\_dd

log4j.appender.sysinfo\_log.layout=org.apache.log4j.PatternLayout

log4j.appender.sysinfo\_log.layout.ConversionPattern=[%d{yyyy-MM-dd HH:mm:ss}][%-5p] [%c] - %m%n

#error输出用appender

log4j.appender.error\_log=org.apache.log4j.DailyRollingFileAppender

log4j.appender.error\_log.File=error.log

log4j.appender.error\_log.DatePattern='.'yyyy\_MM\_dd

log4j.appender.error\_log.layout=org.apache.log4j.PatternLayout

log4j.appender.error\_log.layout.ConversionPattern=[%d{yyyy-MM-dd HH:mm:ss}][%-5p] [%c] - %m%n

log4j.rootLogger=debug, console

log4j.logger.debug=debug,debug\_log

log4j.additivity.debug=false

log4j.logger.sysinfo=info,sysinfo\_log

log4j.additivity.sysinfo=false

log4j.logger.error=error,error\_log

log4j.additivity.error=false

log4j中的layout样式可以输出一些辅助信息方便调试定位

%r   --  程序启动到输出点的毫秒数

%d  --  日期(date)    %d{yyyy-MM-dd HH:mm:ss}

%t   --   线程名(thread)

%p  --   输出级别(priority)

%c   --   类名包括完成的包名(class)，只想看到clazz的名字，%c{1}

%F  --    java类文件名(filename)

%L  --    log输出调用在文件中的行号(line number)

%M --   调用logger输出的方法名(name of method)

# 编程惯例

## 常量

建议在将代码中所有使用到的常量集中定义在一个类名为Constants.java的静态类中，将常量的值定义为有可读性的词组和短语，例如，一个方法需要返回1来表示成功，而0表示失败，建议在Constants类里面定义一个名为RETRUN\_SUCCESS的整型常量，值为1；定义一个名为RETURN\_FALURE的整型常量，值为0。这样做可以大大增加代码的可阅读性。

## 异常按层次抛出

可捕获异常在一个分层架构的系统中，可以在任意层抛出。异常的抛出往往意味着运行的程序可能被中断，从而带来不友好的用户体验，比如一个错误码500的错误页面。但是不建议在所有层都进行异常处理，建议将异常由系统架构的最底层，通常是数据访问层依次往上一层抛出，一直到处理用户交互逻辑的控制层，再进行统一的异常处理。

## 国际化

国际化应该在系统架构之初就被考虑到，对系统中的字符串资源进行管理。开源的MVC的框架都有比较好的国际化支持实现方法，这里不做过多描述。

## 字符串生成

字符串String类对象在Java里面是不可变对象，这就意味着大量的字符串生成和操作会在内存里产生大量的String对象实例，占用内存，可能导致虚拟机垃圾回收机制频繁被触发。建议使用StringBuilder替代直接构建String对象。

## Model类实现

通常一个Model的值对象是一个POJO，直接继承Object对象。为了方便调试，建议重载toString()，hashCode()和equals()方法；并实现serializable接口。

## 面向接口编程

面向接口编程是面向对象编程的升华，在系统架构之初，应充分考虑系统功能和实现类的分离，将系统功能抽取为接口，使用Spring的IOC依赖注入功能，灵活地为接口匹配不同的实现类。举例来说，一个用户服务类对象需要使用到数据库访问对象实例(DAO)，如果采用面向接口编程的理念，则应该将DAO层需要提供的服务抽象为接口，那么在用户服务类里面直接使用接口来操作DAO实例，而不用考虑DAO的实现是基于关系型数据库还是非关系型数据库。而且代码里面也不用因为DAO的实现改变而去改动代码，需要做的只是在Spring的配置文件里面为这个接口配置上相应的实现类即可。

## 避免使用自带同步机制的集合类

在Java1.4之前，只有HashTable，Vector这样的具有同步机制的集合类，这些类的性能较HashMap，ArrayList低，应当在编码过程中避免使用

## 变量初始化

所有变量在定义的时候就应该进行初始化，给出默认值