客户名字和logo

餐厅订餐系统—系统详细设计文档

Xu LIN

日期：2015-03-07

文档版本历史

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 版本号 | 修订日期 | 修订人 | 审核人 | 变更内容 |
| 1.0 | 2014-11-23 | Xu Lin |  | 初始化文档结构 |
|  | 2014-12-5 | Xu Lin |  | 新增用户管理模块和数据库 |
|  | 2014-12-8 | Xu Lin |  | 用户管理模块的用户需求更新和新增类说明 |
|  | 2014-12-26 | LUO ZHI |  | 1. 增加章节2 开发环境配置大纲 2. 增加章节3项目目录结构说明大纲 3. 增加章节4用户管理模块大纲 |
|  | 2014/12/30 | Xu Lin |  | 增加章节4的内容 |
|  | 2014/12/31 | Xu Lin |  | 更新章节4 的内容 |
|  | 2015-01-07 | LUO ZHI |  | 增加章节2 |
|  | 2015-01-12 | LUO ZHI |  | 修改章节4，去掉类说明，加入类图，加入IPO图说明功能输入输出 |
|  | 2015-01-27 | LUOZHI |  | 修改网络协议URL定义 |
|  | 2015-02-02 | LUO ZHI |  | 补充章节4，5 |
|  | 2015-03-05 | LUO ZHI |  | 添加章节7.6,7.7 |
|  | 2015-03-07 | LUO ZHI |  | 补充7.6节调用返回部分，调整输入参数格式 |

**目录**

[1 文档介绍 1](#_Toc413363545)

[1.1 文档目的 1](#_Toc413363546)

[1.2 文档范围 1](#_Toc413363547)

[1.3 缩写词列表 1](#_Toc413363548)

[1.4 参考内容 1](#_Toc413363549)

[2 开发环境配置 2](#_Toc413363550)

[2.1 下载及安装JDK 2](#_Toc413363551)

[2.2 下载及安装Eclipse 4](#_Toc413363552)

[2.2.1 设置网络代理 5](#_Toc413363553)

[2.2.2 使用空格缩进 5](#_Toc413363554)

[2.2.3 打开代码行号和80字符标尺 7](#_Toc413363555)

[2.2.4 关闭自动编译 9](#_Toc413363556)

[2.2.5 修改默认工作区编码 9](#_Toc413363557)

[2.3 下载及安装Maven 10](#_Toc413363558)

[2.4 下载及安装MySQL 12](#_Toc413363559)

[2.5 下载及安装Tomcat 18](#_Toc413363560)

[2.6 下载及安装Git 19](#_Toc413363561)

[3 项目目录结构说明 21](#_Toc413363562)

[3.1 后台管理项目目录结构 21](#_Toc413363566)

[3.2 移动点菜服务项目目录结构 21](#_Toc413363567)

[3.3 导入项目 22](#_Toc413363568)

[3.4 编译项目 23](#_Toc413363569)

[4 移动点菜模块服务端（eorder-ws） 24](#_Toc413363570)

[4.1 浏览菜单 24](#_Toc413363571)

[4.1.1 功能IPO图 24](#_Toc413363572)

[4.1.2 时序图 24](#_Toc413363573)

[4.1.3 代码构成 24](#_Toc413363574)

[4.1.4 类图 25](#_Toc413363575)

[4.1.5 配置文件说明 25](#_Toc413363576)

[4.1.6 数据库脚本说明 26](#_Toc413363577)

[4.2 浏览菜品 27](#_Toc413363578)

[4.2.1 功能IPO图 27](#_Toc413363579)

[4.2.2 时序图 27](#_Toc413363580)

[4.2.3 代码构成 28](#_Toc413363581)

[4.2.4 类图 29](#_Toc413363582)

[4.2.5 配置文件 29](#_Toc413363583)

[4.2.6 数据库脚本 30](#_Toc413363584)

[4.3 获取会员信息 30](#_Toc413363585)

[4.3.1 功能IPO图 30](#_Toc413363586)

[4.3.2 时序图 31](#_Toc413363587)

[4.3.3 代码构成 31](#_Toc413363588)

[4.3.4 类图 32](#_Toc413363589)

[4.3.5 配置文件 32](#_Toc413363590)

[4.3.6 数据库脚本 33](#_Toc413363591)

[4.4 浏览会员订单历史 34](#_Toc413363592)

[4.4.1 功能IPO图 34](#_Toc413363593)

[4.4.2 时序图 34](#_Toc413363594)

[4.4.3 代码构成 35](#_Toc413363595)

[4.4.4 类图 36](#_Toc413363596)

[4.4.5 配置文件 36](#_Toc413363597)

[4.4.6 数据库脚本 37](#_Toc413363598)

[5 权限管理模块 (eorder) 39](#_Toc413363599)

[5.1 用户登录 39](#_Toc413363600)

[5.1.1 功能IPO图 39](#_Toc413363601)

[5.1.2 时序图 39](#_Toc413363602)

[5.1.3 代码构成 39](#_Toc413363603)

[5.1.4 类图 41](#_Toc413363604)

[5.1.5 配置文件 41](#_Toc413363605)

[5.1.6 数据库脚本 43](#_Toc413363606)

[5.2 创建/编辑用户 43](#_Toc413363607)

[5.2.1 功能IPO图 44](#_Toc413363608)

[5.2.2 时序图 44](#_Toc413363609)

[5.2.3 类图 45](#_Toc413363610)

[5.2.4 代码构成 45](#_Toc413363611)

[5.2.5 配置文件 45](#_Toc413363612)

[5.3 用户角色分配 46](#_Toc413363613)

[5.3.1 功能IPO图 47](#_Toc413363614)

[5.3.2 时序图 47](#_Toc413363615)

[5.3.3 类图 48](#_Toc413363616)

[5.3.4 代码构成 48](#_Toc413363617)

[5.3.5 配置文件 49](#_Toc413363618)

[5.3.6 数据库脚本 49](#_Toc413363619)

[5.4 创建/编辑角色 49](#_Toc413363620)

[5.4.1 功能IPO图 50](#_Toc413363621)

[5.4.2 时序图 50](#_Toc413363622)

[5.4.3 类图 51](#_Toc413363623)

[5.4.4 代码构成 51](#_Toc413363624)

[5.4.5 配置文件 52](#_Toc413363625)

[5.4.6 数据库脚本 53](#_Toc413363626)

[5.5 创建/编辑权限 53](#_Toc413363627)

[5.5.1 功能IPO图 53](#_Toc413363628)

[5.5.2 时序图 54](#_Toc413363629)

[5.5.3 类图 54](#_Toc413363630)

[5.5.4 代码构成 55](#_Toc413363631)

[5.5.5 配置文件 55](#_Toc413363632)

[5.5.6 数据库脚本 56](#_Toc413363633)

[5.6 角色权限分配 57](#_Toc413363634)

[5.6.1 功能IPO图 57](#_Toc413363635)

[5.6.2 时序图 57](#_Toc413363636)

[5.6.3 类图 58](#_Toc413363637)

[5.6.4 代码构成 58](#_Toc413363638)

[5.6.5 配置文件 58](#_Toc413363639)

[5.6.6 数据库脚本 59](#_Toc413363640)

[6 数据库设计 59](#_Toc413363641)

[6.1 用户表 59](#_Toc413363642)

[6.2 角色表 59](#_Toc413363643)

[6.3 功能表 59](#_Toc413363644)

[6.4 用户角色关联表 60](#_Toc413363645)

[6.5 角色功能关联表 60](#_Toc413363646)

[6.6 菜品分类表 60](#_Toc413363647)

[6.7 菜品表 61](#_Toc413363648)

[6.8 订单表 61](#_Toc413363649)

[6.9 订单条目表 61](#_Toc413363650)

[6.10 会员等级表 62](#_Toc413363651)

[7 Web-Service接口协议 62](#_Toc413363652)

[7.1 获取用户信息 62](#_Toc413363653)

[7.2 获取目前所有分类列表 63](#_Toc413363654)

[7.3 获取某一个分类下面所有菜品列表 63](#_Toc413363655)

[7.4 查询某个会员的历史订餐记录 64](#_Toc413363656)

[7.5 查询某个订单详情 64](#_Toc413363657)

[7.6 订单下单 65](#_Toc413363658)

[7.7 接口调用异常信息返回 66](#_Toc413363659)

[8 附录 67](#_Toc413363660)

# 文档介绍

## 文档目的

本文档用于对餐厅订餐系统的详细设计思路，包括类定义，逻辑流程和算法进行描述，指导开发团队进行系统开发实现。

## 文档范围

本文档仅对项目开发环境配置，系统模块详细描述做论述，对于技术选型，实现框架搭建均不属于本文范围内。

## 缩写词列表

|  |  |
| --- | --- |
| 缩写词 | 解释 |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

## 参考内容

# 开发环境配置

餐厅订餐系统使用Java/Android技术开发前后端应用，所以下面依次介绍在项目说使用到的开发环境和工具的安装配置。

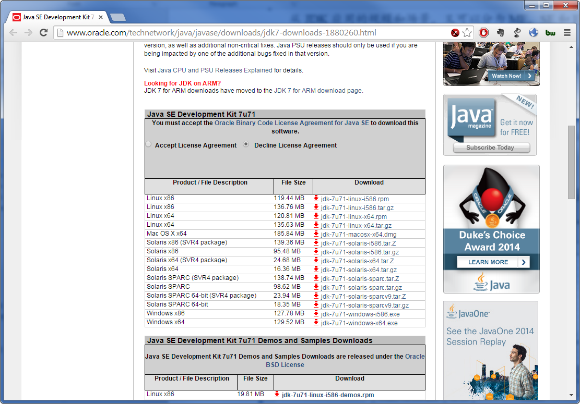
## 下载及安装JDK

JDK是Java Development Kit （Java开发工具箱）的简写，是使用Java进行开发的必备工具。JDK不仅提供了Java核心类库还提供了编译，调试，打包等命令行工具。

很多公司都有JDK产品，标准的JDK是由Sun Microsystem公司拥有，但现在Sun已经被Oracle公司收购；Oracle公司有自有JDK产品的JRocket；另外还有OpenJDK。

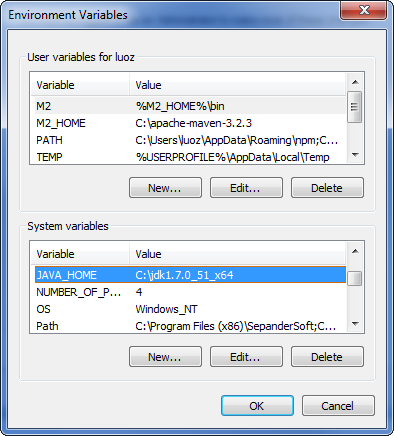
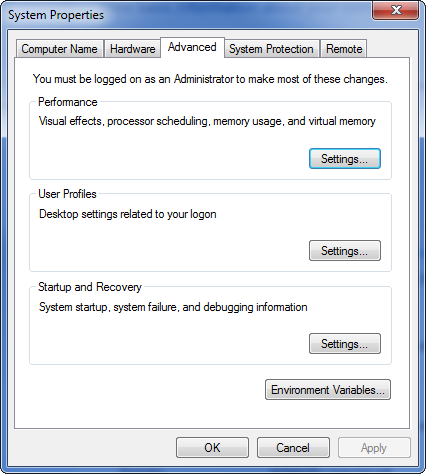
从JDK应用的规模和场景，又可以分为ME，SE和EE三个版本，依次对应于移动程序，标准程序和企业级程序

本项目采用标准JDK SE 7 64位版本，可以从Oracle公司的网站下载(<http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html>) 。在下载之前必须选择**Accept License Agreement**选项，才能获取下载链接，然后点击**jdk-7u71-windows-x64.exe**下载

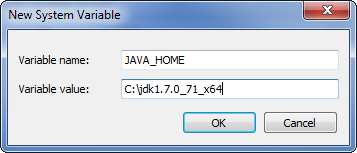


下载完成后以管理员身份运行安装包进行安装，在选择安装路径的时候建议选择安装在某个分区的根目录，例如**C:\jdk1.7.0\_71\_x64**，方便接下来的配置。

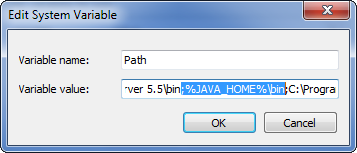
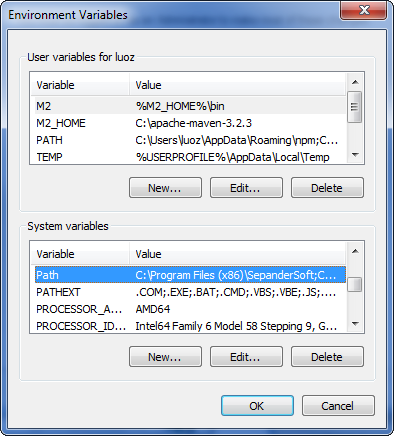
安装完成之后，需要为JDK配置环境变量，打开**控制面板->系统->高级->环境变量…**



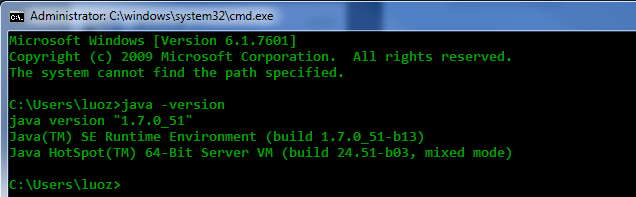
点击**新建…**按钮，在变量名中输入**JAVA\_HOME**，在变量值中填入JDK的安装路径，点击**确定**按钮



编辑系统变量中的**Path**，在变量值输入框中添加**;%JAVA\_HOME%\bin**，点击确定按钮。



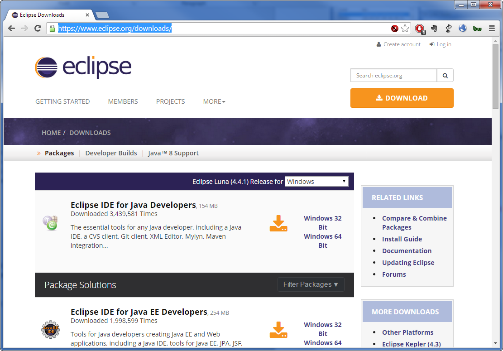
打开命令行窗口，输入**java –version**得到如图所示的文本信息，则说明JDK安装以及环境变量设定正确



注：截图中安装的JDK SE 7小版本为51，文中给出下载链接为71，这是没有关系的，用户安装和配置的时候自行将51字样改为71即可

## 下载及安装Eclipse

Eclipse是一个集成开发环境框架，在这个集成开发框架下提供了对C++、Java等众多开发语言的支持。项目需要用到Eclipse IDE for Java Developers，可以到Eclipse的官方网站下载（https://www.eclipse.org/downloads/），最新的Eclipse版本名称是Luna，下载**Windows 64 Bit** 版本



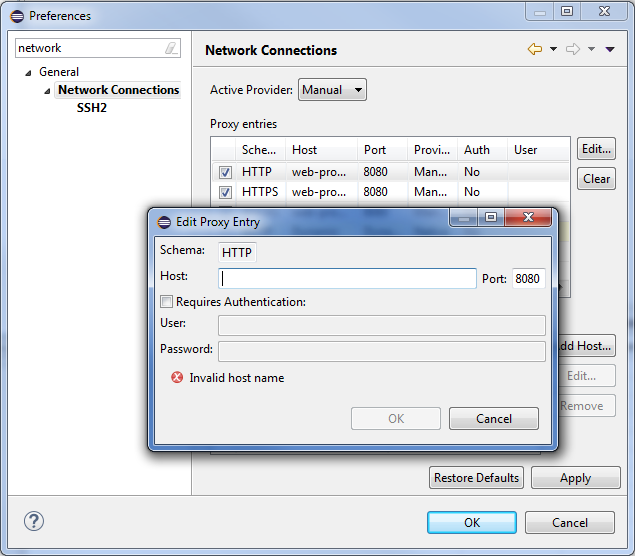
因为没有用到JavaEE的高级特性，诸如JPA，JSF等，所以下载Java Developers版本即可，这个版本已经内置了Git，Maven支持，很好的满足项目开发的需求。下载的安装包为zip格式，使用解压缩软件，比如7Zip（http://www.7-zip.org/）将压缩包解压，建议解压至某个分区的根目录，例如**C:\eclipse**，运行安装目录下的**eclipse.exe**即可打开Eclipse集成开发工具。

第一次运行eclipse会提示设定工作区（workspace），将分区上的任一目录设置为默认工作区即可，例如：**C:\MyWorkspace**

在使用eclipse进行项目开发之前，还需要对eclipse进行一些设定

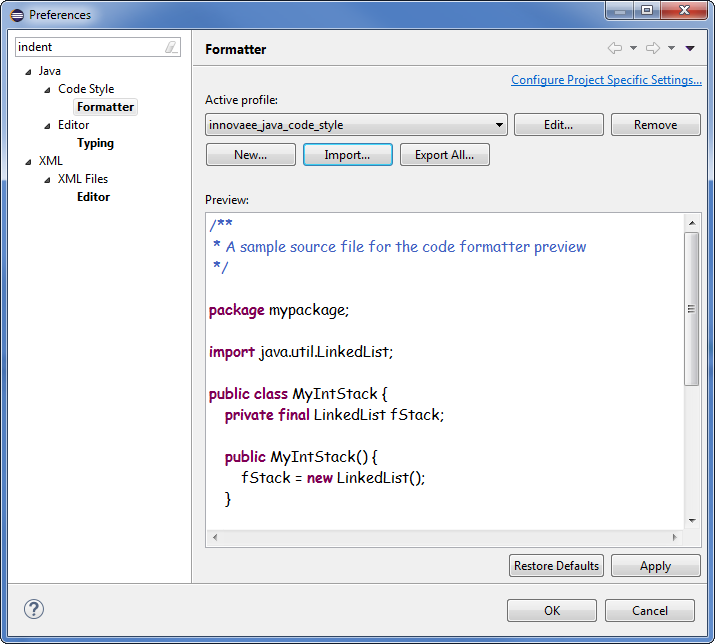
### 设置网络代理

很多时候eclipse需要访问网络来获取最新的依赖或者插件更新，但是如果需要通过代理服务器才能访问网络时，则需要进行代理设置，点击**Window->Preferences**，在左上角的输入框中输入**network**，选中**Network Connections**，在右边面板中的**Active Provider**下拉框中选择**Manual**，勾选要配置的代理类型，点击**Edit…**按钮，在弹出的对话框中输入代理服务器地址和端口，点击**OK**按钮即可

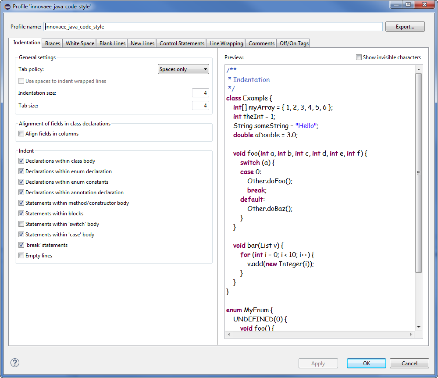
****

### 使用空格缩进

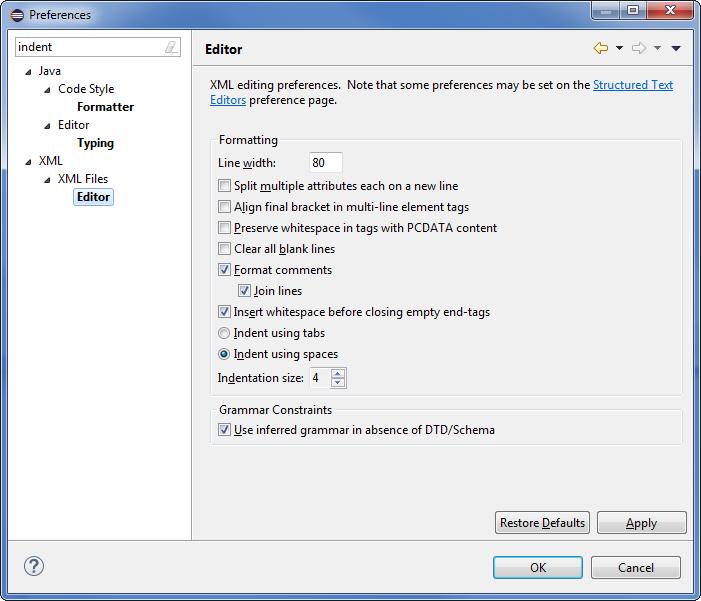
因为不同系统对缩进的处理不同，所以使用空格而不是制表符可以让代码在不同的系统上的表现形式一致。点击**Window->Preferences**，在左上角的输入框中输入**indent**，在左边菜单中选择**Code Style->Formatter**，在右边面板中选择**Import…**按钮，导入项目编码风格定义文件**innovaee\_java\_code\_style.xml**



导入完成后，在**Active profile**下拉框中选择导入的自定义编码风格，点击**Edit…**按钮，确认在**General settings**下的**Tab Policy**下拉框值为**Spaces only**，且**Indentation size**和**Tab size**的值都为4

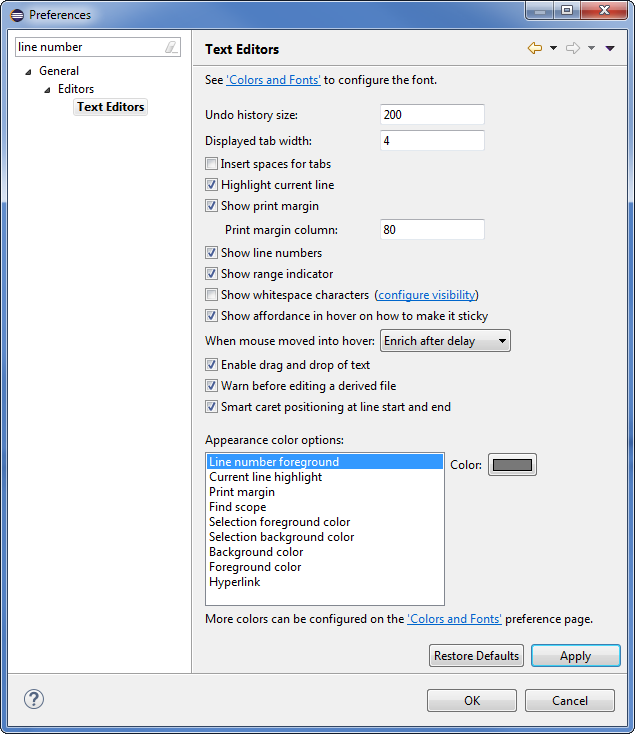


选择**XML->XML Files->Editors**，在左边面板中选择**Indent using spaces**，**Indentation size**设置为**4**

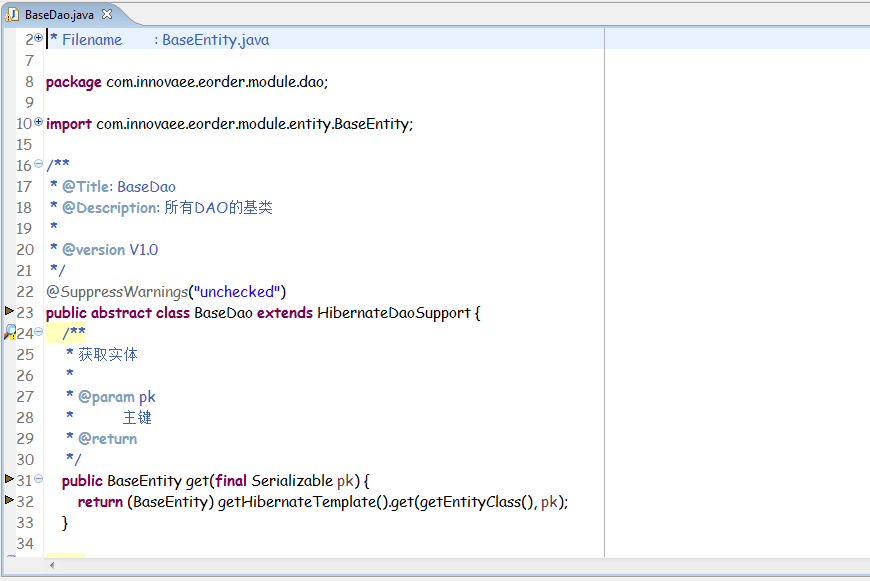


### 打开代码行号和80字符标尺

在编辑区显示行号有利于调试定位，而80字符标尺有利于在编写长语句的时候进行有效换行。点击**Window->Preferences**，在左上角的输入框中输入**line number**，点击左边菜单中的**General->Editors->Text Editors**，在右边面板中勾选**Show print margin**，在**Print margin column**输入框输入80；勾选**Show line numbers**



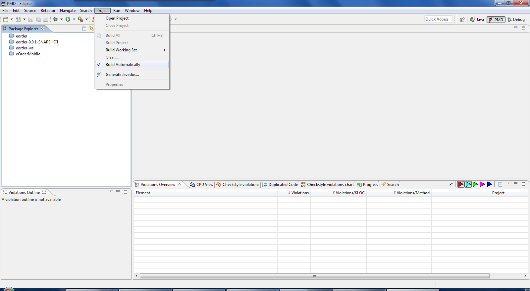
点击OK按钮后，打开Java文件会看到如下效果，



### 关闭自动编译

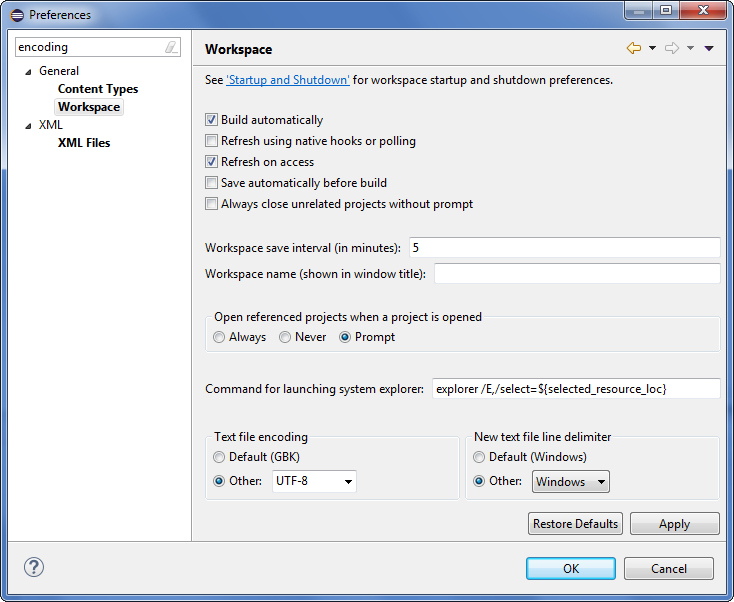
默认情况下，eclipse在代码变动被保存的时候自动重新编译工程，但是在工程代码很大或者需要网络验证XML文件合法性的时候会导致整个编译过程十分缓慢，这里建议关闭eclipse的自动编译特性

点击**Project**菜单，点击取消**Build Automatically**的选中标志



### 修改默认工作区编码

Windows操作系统默认的编码方式是GB1252，但在项目中通常会用到国际化这一特性，所以文件编码方式要修改为UTF-8，点击**Window->Preferences**，在左上角的输入框中输入**encoding**，选中**Workspace**，在右边的面板下方找到**Text file encoding**，选择**Other**，并在下拉框中选择**UTF-8**

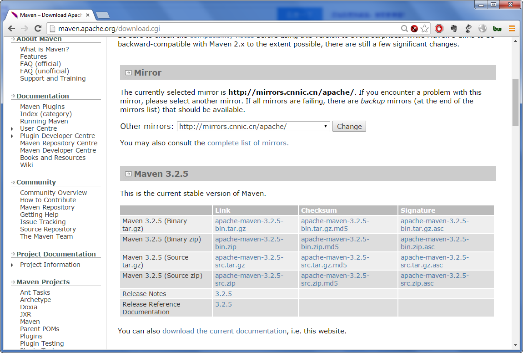


## 下载及安装Maven

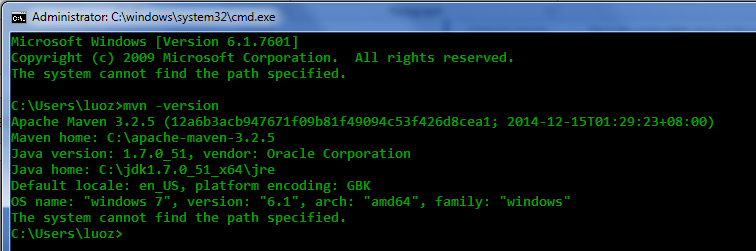
Maven是一个开源的项目构建管理工具，与耳熟能详的Ant相比，Maven提供了软件开发周期更为全面的管理。

* 从项目创建，开发测试，打包发布，覆盖软件开发整个生命周期，而Ant更多的被用作编译工具。
* Maven的另外一个特点是对Java相关项目的依赖组件Jar包的管理，通过在POM.xml文件中的依赖定义，开发团队成员能保持统一的组件库，而且无须将Jar包纳入代码的配置管理库，节约团队对Jar包的管理工作
* Maven的项目创建工件，可以新建众多类型的Java项目，例如web application，JSF application。在创建项目的同时，会为开发团队自动创建标准的目录结构以及默认的配置文件
* 围绕Maven，有众多的插件可以使用，最为常见的是Maven-compiler插件，在很多时候，Maven的默认编译参数往往不能满足项目的要求，通过Maven-compiler插件，可以定义项目需要的编译参数

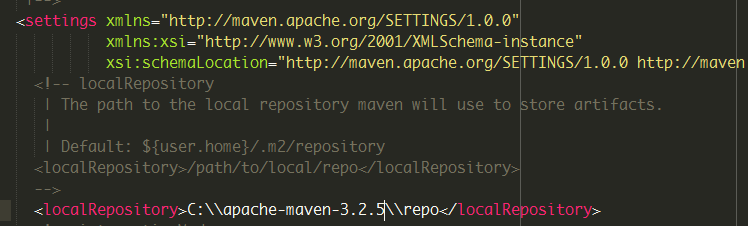
前往Maven的官方网站下载当前最稳定版本（3.2.25）<http://maven.apache.org/download.cgi> ，点击**apache-maven-3.2.5-bin.zip**进行下载，将下载后的zip文件解压缩到任一分区的任一位置，建议放入某个分区的根目录，便于接下来的配置，例如**C:\apache-maven-3.2.5**



按照[2.1](#_下载及安装JDK) 章节中设置JDK环境变量的方法，新建环境变量**M2\_HOME**，值为Maven安装的目录，例如**C:\apache-maven-3.2.5**，在系统环境变量**Path**后添加**;%M2\_HOME%\bin**, 打开命令行窗口，输入**mvn –version，**显示如下图类似文本，则Maven安装完毕



在使用Maven进行项目构建管理之前，需对Maven中的2个配置进行修改，打开**%M2\_HOME%\conf\settings.xml**，找到<localRepository>节点，添加如下图所示节点，设定Maven下载Jar的本地仓库路径



找到<proxies>节点，添加如下图所示的节点，设定Maven使用的代理服务器（如果需要使用代理服务器访问网络的情况下）

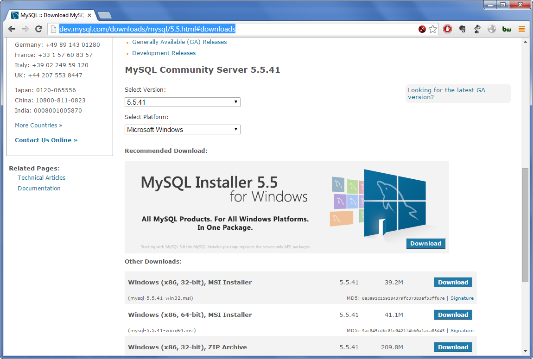


* id : 代理服务器配置标识符
* active : 是否激活此代理服务器配置
* protocol : 代理服务器的协议，例如http和https
* host : 代理服务器域名或者ip地址
* port : 代理服务器使用的端口
* username : 代理服务器使用的用户名
* password : 代理服务器使用的用户密码
* nonProxyHosts : 使用|分割不需要使用代理的网络地址或IP

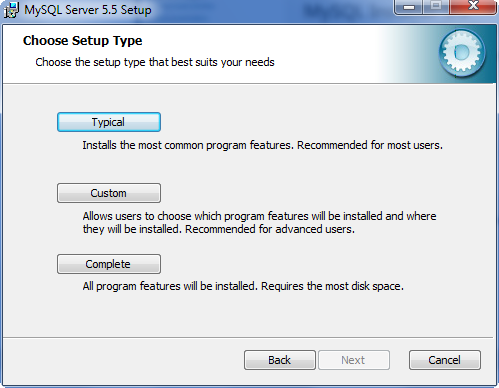
后面三个元素在上图中没有使用，如果用户的代理服务器需要用户名和密码登录，而且有些网络资源不需要代理服务器，则设定后三个元素

## 下载及安装MySQL

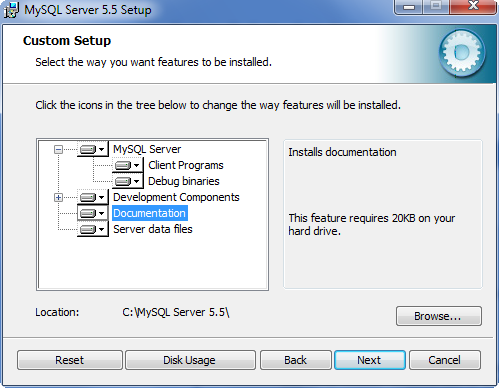
MySQL可以说是最为流行的开源关系型数据库，但目前MySQL被Oracle收购后，原来的分支独立出来成为社区版，而且与Oracle资源整合后退出的企业版在获得技术支持和高级特性的时候是需要支付费用的。项目使用社区版中比较流行的5.5作为开发数据库（<http://dev.mysql.com/downloads/mysql/5.5.html#downloads>）。



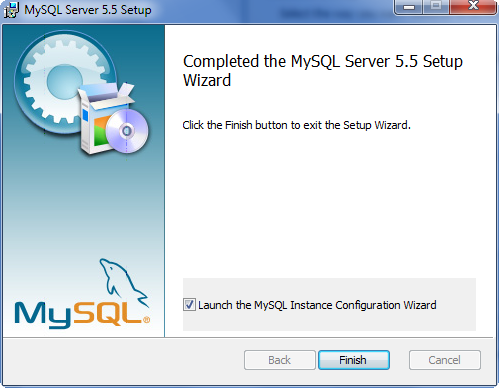
下载**Windows (x86, 64bit), MSI Installer**，下载完成后，以管理员身份运行安装，在选择安装类型的时选择**Custom…**



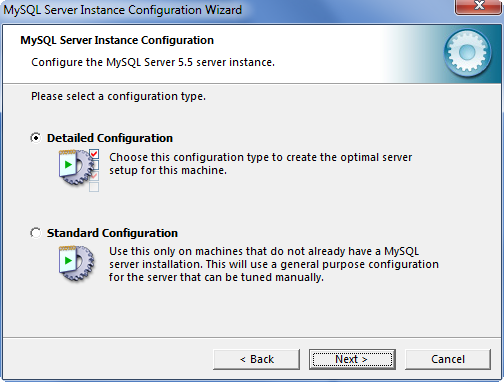
选择完全安装Documentation以便后期查看，点击**Browse…**将安装路径修改为任一分区的根目录，例如 **C:\MySQL 5.5**



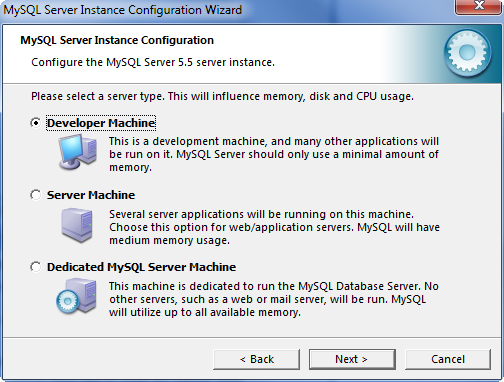
点击**Next**按钮进行安装，安装过程中有企业版的广告弹出，一路**Next**，只到安装界面如下，确保**Launch the MySQL Instance Configuration Wizard**前面的选择框是选中状态，点击**Finish**



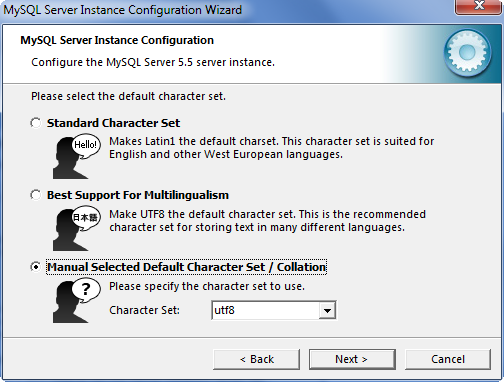
弹出的配置对话框中选择**Detailed Configuration**



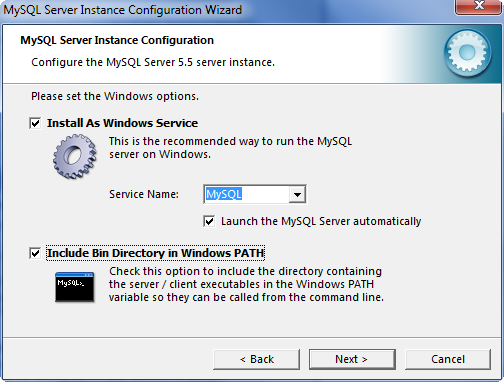
选择**Developer Machine**，以开发环境来配置MySQL



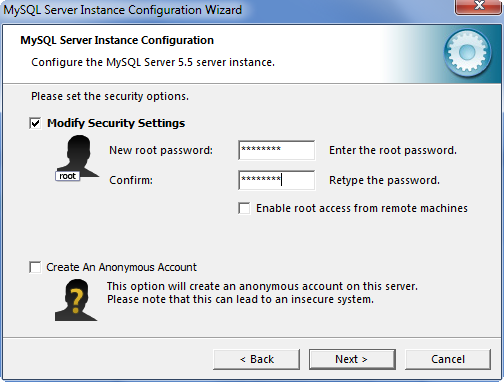
**Next**到选择字符集的界面，选择**Manual Selected Default Charset / Collation**选项，在下拉框中选择UTF-8



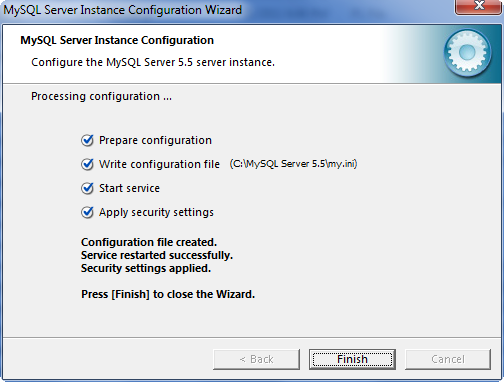
**Next**下一个界面中将**Include Bin Directory in Windows Path**选择勾选中，保证将MySQL命令行工具添加到Windows环境变量，以便在命令行窗口下直接使用



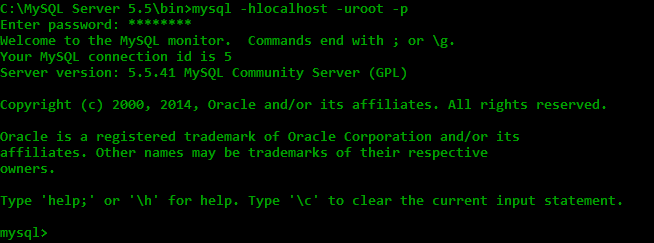
在下一个界面设定root用户密码



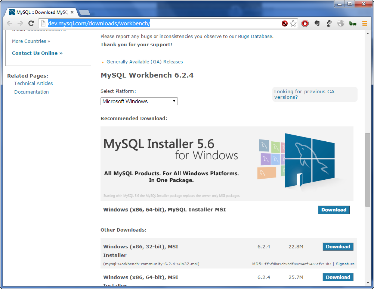
最后一步点击**Execute**按钮执行配置



打开命令行窗口，输入**mysql –hlocalhost –uroot –p，**输入设定的root用户密码，进入登录到MySQL数据库实例



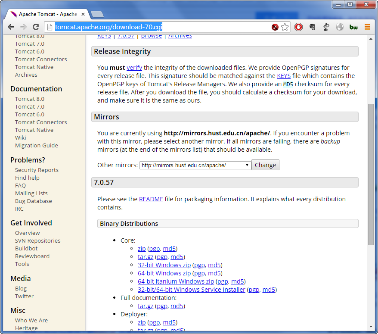
为了提高和简化开发过程，建议安装MySQL Workbench图形界面工具，下载页面为（<http://dev.mysql.com/downloads/workbench/>），选择64bit版本下载，已对应安装的MySQL 64bit 实例



注：MySQL Workbench的使用不在本文的描述范围内

## 下载及安装Tomcat

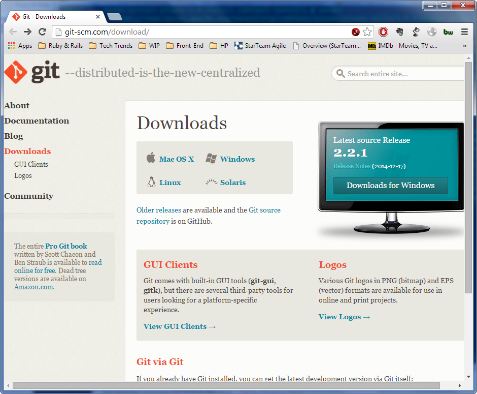
Tomcat是一个开源的WEB应用服务器，本项目采用Tomcat 7作为开发应用服务器，下载链接为<http://tomcat.apache.org/download-70.cgi>，点击Core下面的zip链接，下载zip格式的安装包



将下载的zip包解压到某一分区的根目录，例如**C:\Tomcat**，按照[2.1](#_下载及安装JDK)小节配置JDK环境变量的方法新增**CATALINA\_HOME**变量，值为**Tomcat**的安装目录，并将**;%CATALINA\_HOME%\bin**追加到**Path**变量中。

## 下载及安装Git

Git最初是Linux之父Linus为了对他的Linux内核代码做版本管理而开发的一个轻量级的版本控制工具，在开源之后迅速成为开源社区的主流版本管理工具之一。目前比较流行的Git公共仓库有GitHub，国内也有CSDN Code。Git不同于其他版本控制工具，例如CVS，SVN，之处是Git可以同时在本地和联网状态下进行代码版本管理。也就是说每个安装了Git的客户端都是一个版本仓库，无须在线仍然可以进行版本提交。Git的官方网站可以下载最新版的Git 1.8.9 （<http://git-scm.com/download/>），选择Windows版本，已管理员身份运行安装程序即可。



Git自带一个非常简陋的GUI工具，并且对中文支持不够好，这里推荐下载Tortoise Git工具，官方网站（<http://download.tortoisegit.org/tgit/1.8.12.0/>），这里对Git的使用和Git版本管理的概念不做详细描述。



# 项目目录结构说明

整个餐厅订餐系统由2个项目构成，分别是后台管理项目，对应实现系统后台管理的功能（权限管理模块，会员管理模块，菜单管理模块和后台收银模块）；第二个项目是为移动点菜客户端提供Web Service服务的项目。



## 后台管理项目目录结构

项目采用Maven标准WEB项目构建目录结构，如下图所示

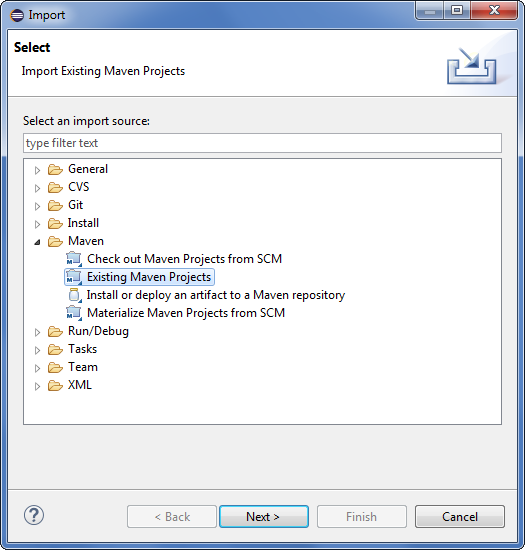
|  |  |  |
| --- | --- | --- |
|  | eorder | 项目根目录 |
| .settings | Eclipse工程配置目录 |
| src | 源代码目录 |
| main | 工程源代码目录 |
| java | Java源文件目录 |
| resources | 配置文件目录 |
| webapp | WEB端源代码目录 |
| META-INF | 原数据文件目录 |
| pages | 页面文件目录 |
| resources | 图片，JS和CSS文件目录 |
| WEB-INF | WEB配置文件目录 |
| test | 测试源代码目录 |
| java | 测是Java源文件目录 |

## 移动点菜服务项目目录结构

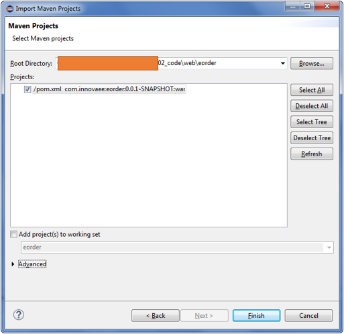
|  |  |  |
| --- | --- | --- |
|  | eorder-ws | 项目根目录 |
| src | 源代码目录 |
| main | 工程源代码目录 |
| java | Java源文件目录 |
| resources | 配置文件目录 |
| webapp | WEB端源代码目录 |
| META-INF | 原数据文件目录 |
| pages | 页面文件目录 |
| resources | 图片，JS和CSS文件目录 |
| WEB-INF | WEB配置文件目录 |
| test | 测试源代码目录 |
| java | 测是Java源文件目录 |

## 导入项目

2个项目的导入方式是一致的，这里以eorder项目为例。首先，从Git仓库迁出最新代码，打开eclipse，选择**File->Import …**，在弹出对话框中选中**Existing** **Maven Projects**，点击**Next**按钮，

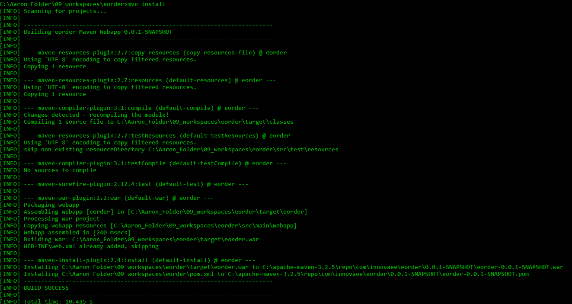


在下一步对话框中选择Root Directory指向本地从Git仓库迁出的项目目录，点击**Finish**按钮



## 编译项目

因为采用了Maven进行项目构建管理，所以项目的编译过程是十分简洁的，在命令行中输入**mvn install，**即可完成编译，



# 移动点菜模块服务端（eorder-ws）

移动点菜模块服务端使用REST风格Web-Service提供点菜相关功能给移动点菜客户端（Android）使用。整个Web-Service服务打包成一个项目结构，名字为eorder-ws，请按照[3.3导入项目](#_导入项目)所述方法导入项目进行开发。

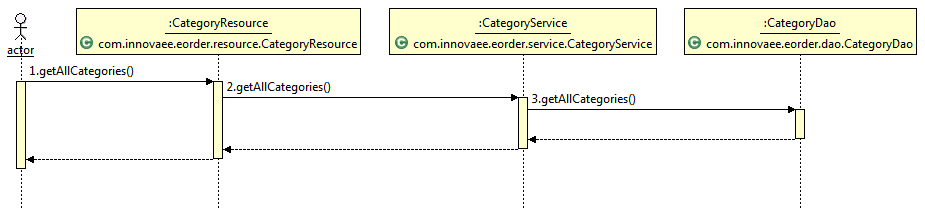
## 浏览菜单

该服务响应移动客户端发送的获取菜单列表请求，将菜单列表信息以JSON格式发还给移动客户端。

### 功能IPO图



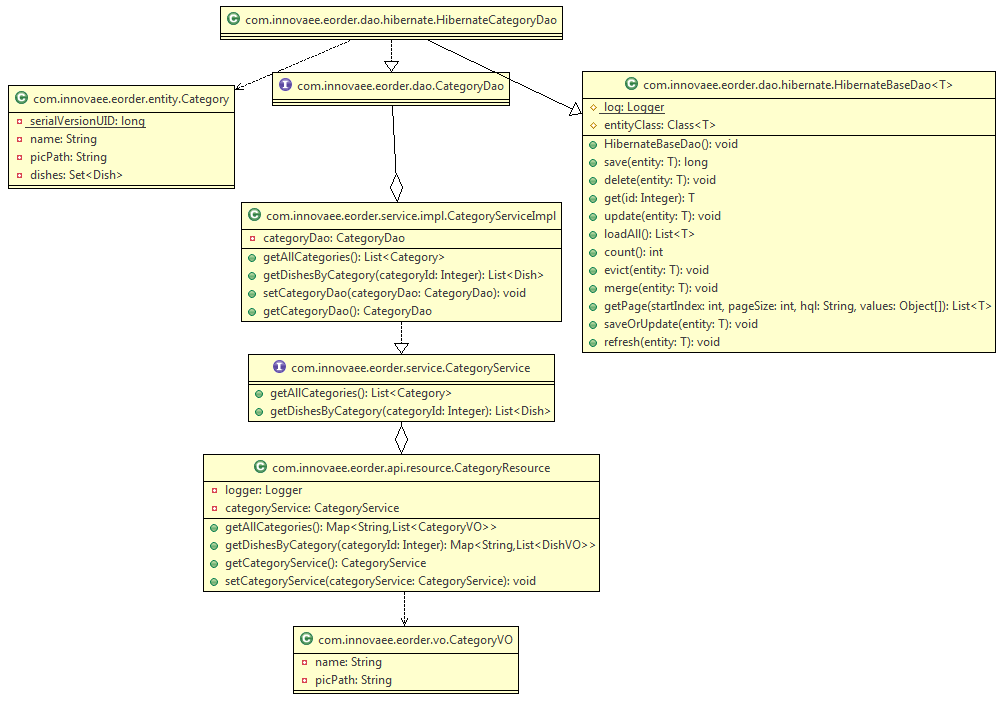
### 时序图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| CategoryVO.java | com.innovaee.eorder.vo | 菜单的值域对象，用于菜单列表信息的传递 |
| Category.java | com.innovaee.eorder.entity | 菜单的实体对象，用于建立与数据库表的映射 |
| CategoryDao.java | com.innovaee.eorder.dao | 菜单对象的数据库访问层，用于数据库CRUD以及查询的调用 |
| DishDao.java | com.innovaee.eorder.dao | 菜品对象的数据库访问层，用于数据库CRUD以及查询的调用 |
| CategoryService.java | com.innovaee.eorder.service | 对Dao层的业务封装，进行异常处理 |
| CategoryResource.java | com.innovaee.eorder.resource | 用于建立Service层与REST风格Web-Service的URI映射 |
| BaseDao | com.innovaee.eorder.dao | 所有数据访问对象的基类，定义了数据库CRUD等基本操作接口 |
| HibernateBaseDao | com.innovaee.eroder.dao.hibernate | 基于Hibernate的BaseDao实现类 |

### 类图



### 配置文件说明

* Hibernate 映射配置在Category.java实体类中

@Entity

@Table(name = "t\_category")

@XmlRootElement

public class Category extends BaseEntity

* REST URL配置在CategoryResource.java资源类中

@Path("/categories")

public class CategoryResource extends AbstractBaseResource {

@GET

@Produces({ MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON })

public Map<String, List<Category>> getAllCategories() {

}

}

### 数据库脚本说明

-- ----------------------------

-- Table structure for `t\_category`

-- ----------------------------

DROP TABLE IF EXISTS `t\_category`;

CREATE TABLE `t\_category` (

` id` int(32) NOT NULL AUTO\_INCREMENT,

`category\_name` varchar(128) DEFAULT NULL,

`category\_picture` varchar(256) DEFAULT NULL,

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=11 DEFAULT CHARSET=utf8;

## 浏览菜品

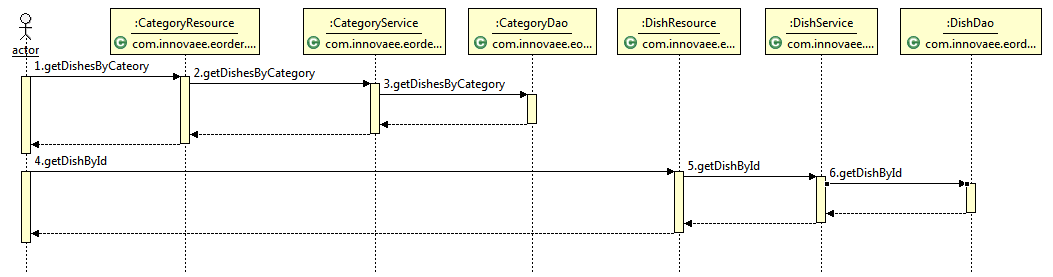
该服务响应移动客户端发送的获取菜单中菜品列表与菜品详情请求。

### 功能IPO图





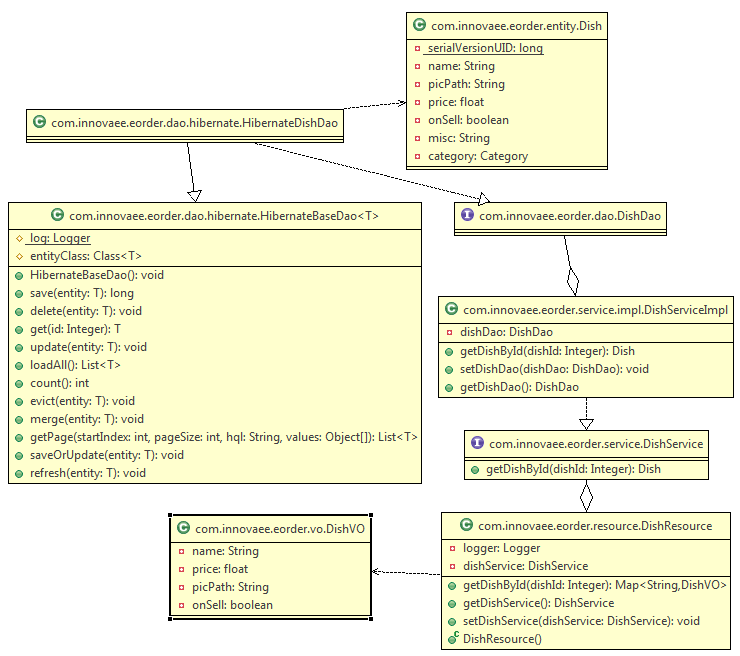
### 时序图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| DishVO.java | com.innovaee.eorder.vo | 菜品的值域对象，用于菜品列表信息的传递 |
| Dish.java | com.innovaee.eorder.entity | 菜品的实体对象，用于建立与数据库表的映射 |
| DishDao.java | com.innovaee.eorder.dao | 菜品对象的数据库访问层，用于数据库CRUD以及查询的调用 |
| DishService.java | com.innovaee.eorder.service | 对菜品Dao层的业务封装，进行异常处理 |
| DishResource.java | com.innovaee.eorder.resource | 用于建立Service层与REST风格Web-Service的URI映射 |

### 类图



### 配置文件

* Hibernate 映射配置在Dish.java实体类中

@Entity

@Table(name = "t\_dish")

@XmlRootElement

public class Dish extends BaseEntity

* REST URL配置在CategoryResource.java资源类中

@Path("/dishes")

public class DishResource extends AbstractBaseResource {

@GET

@Path("/{dishId}")

@Produces({ MediaType.APPLICATION\_XML, MediaType.APPLICATION\_JSON })

public Map<String, DishVO> getDishById(@PathParam("dishId") Integer dishId)

…

}

### 数据库脚本

-- ----------------------------

-- Table structure for `t\_dish`

-- ----------------------------

DROP TABLE IF EXISTS `t\_dish`;

CREATE TABLE `t\_dish` (

`dish\_id` int(32) NOT NULL AUTO\_INCREMENT,

`category\_id` int(32) DEFAULT NULL,

`dish\_name` varchar(128) DEFAULT NULL,

`dish\_picture` varchar(256) DEFAULT NULL,

`dish\_price` float(64,0) DEFAULT NULL,

`on\_sell` tinyint(1) DEFAULT NULL,

`misc` varchar(128) DEFAULT NULL,

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

PRIMARY KEY (`dish\_id`)

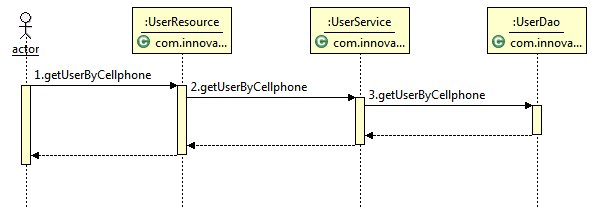
) ENGINE=InnoDB AUTO\_INCREMENT=60 DEFAULT CHARSET=utf8;

## 获取会员信息

### 功能IPO图



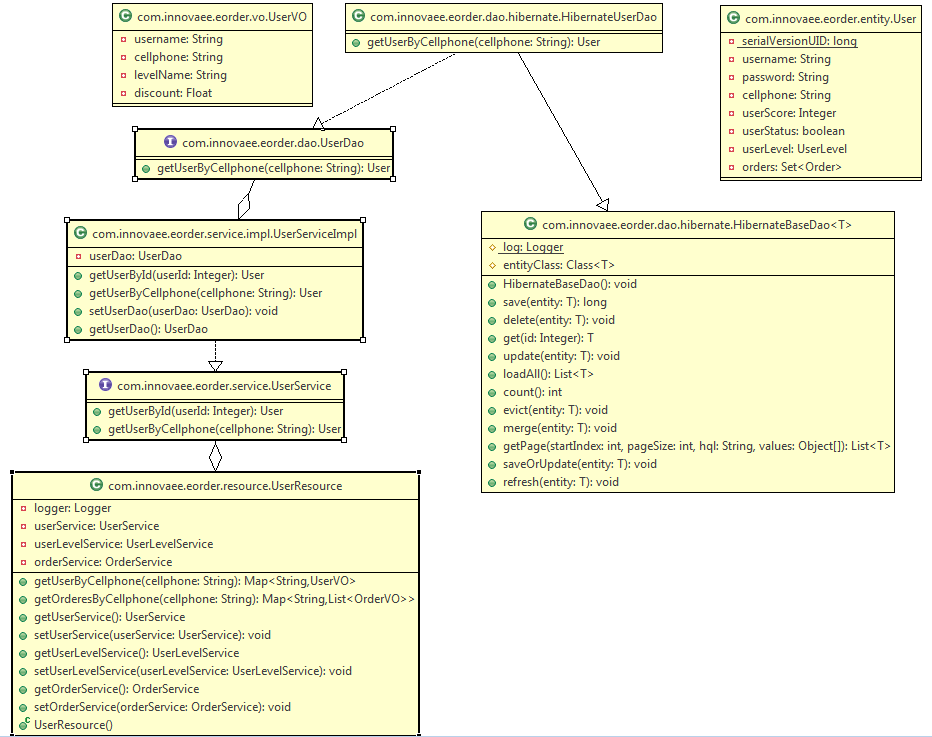
### 时序图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| UserVO.java | com.innovaee.eorder.vo | 用户的值域对象，用于用户列表信息的传递 |
| User.java | com.innovaee.eorder.entity | 用户的实体对象，用于建立与数据库表的映射 |
| UserDao.java | com.innovaee.eorder.dao | 用户对象的数据库访问层，用于数据库CRUD以及查询的调用 |
| UserService.java | com.innovaee.eorder.service | 对用户Dao层的业务封装，进行异常处理 |
| UserResource.java | com.innovaee.eorder.resource | 用于建立Service层与REST风格Web-Service的URI映射 |

### 类图



### 配置文件

* Hibernate 映射配置在User.java实体类中，详见代码

@Entity

@Table(name = "T\_USER")

public class User extends BaseEntity {

* REST URL配置在UserResource.java资源类中，详见代码

*@Path("/users")*

*public class UserResource {*

*@GET*

*@Path("/{cellphone}")*

*@Scope("request")*

*@Produces(MediaType.APPLICATION\_JSON)*

*public Map<String, UserVO> getUserByCellphone*

### 数据库脚本

-- ----------------------------

-- Table structure for `t\_user`

-- ----------------------------

DROP TABLE IF EXISTS `t\_user`;

CREATE TABLE `t\_user` (

`id` int(32) NOT NULL AUTO\_INCREMENT,

`username` varchar(20) COLLATE utf8\_bin DEFAULT NULL,

`password` varchar(256) COLLATE utf8\_bin DEFAULT NULL,

`cellphone` varchar(16) COLLATE utf8\_bin DEFAULT NULL,

`user\_score` int(32) DEFAULT NULL,

`level\_id` int(32) DEFAULT NULL,

`user\_status` tinyint(1) DEFAULT '1',

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

PRIMARY KEY (`user\_id`)

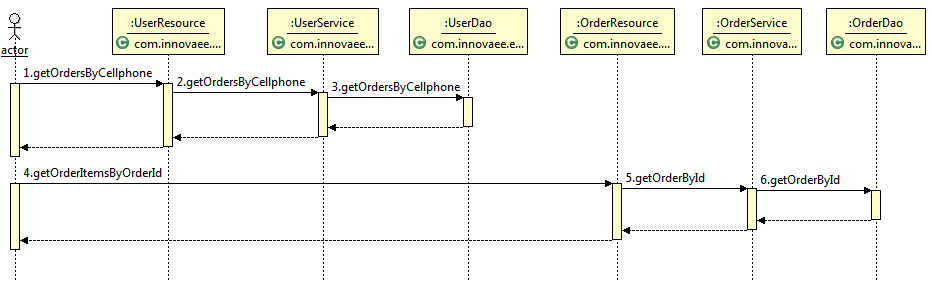
) ENGINE=InnoDB AUTO\_INCREMENT=21 DEFAULT CHARSET=utf8 COLLATE=utf8\_bin COMMENT='user''s basic information';

## 浏览会员订单历史

### 功能IPO图



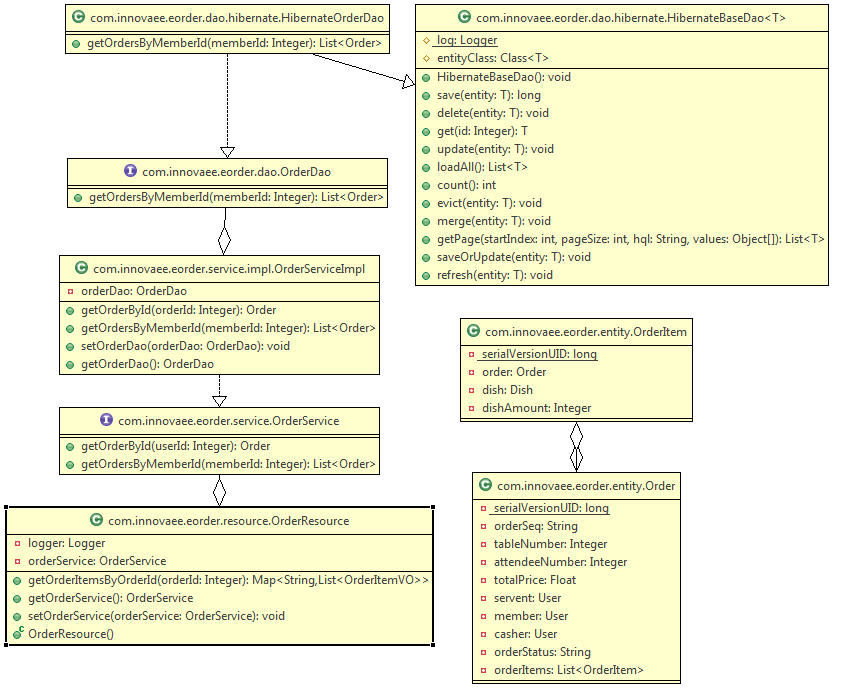
### 时序图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| OrderVO.java | com.innovaee.eorder.vo | 订单的值域对象，用于订单信息的传递 |
| Order.java | com.innovaee.eorder.entity | 订单的实体对象，用于建立与数据库表的映射 |
| OrderDao.java | com.innovaee.eorder.dao | 订单对象的数据库访问层，用于数据库CRUD以及查询的调用 |
| OrderService.java | com.innovaee.eorder.service | 对订单Dao层的业务封装，进行异常处理 |
| OrderResource.java | com.innovaee.eorder.resource | 用于建立Service层与REST风格Web-Service的URI映射 |
| OrderItemVO.java | com.innovaee.eorder.vo | 订单条目的值域对象，用于订单条目信息的传递 |
| OrderItem.java | com.innovaee.eorder.entity | 订单条目的实体对象，用于建立与数据库表的映射 |
| OrderItemDao.java | com.innovaee.eorder.dao | 订单条目对象的数据库访问层，用于数据库CRUD以及查询的调用 |

### 类图



### 配置文件

* Hibernate 映射配置在Order.java和OrderItem.java实体类中，详见代码

@Entity

@Table(name = "T\_ORDER")

public class Order extends BaseEntity {

@OneToMany(targetEntity = OrderItem.class, fetch = FetchType.EAGER, mappedBy = "order")

public List<OrderItem> getOrderItems() {

}

}

@Entity

@Table(name = "T\_ORDER\_ITEM")

public class OrderItem extends BaseEntity {

@ManyToOne(targetEntity = Order.class, fetch = FetchType.EAGER)

@JoinColumn(name = "ORDER\_ID")

public Order getOrder() {

}

}

* REST URL配置在UserResource.java和OrderResource.java资源类中，详见代码

*@Path("/users")*

*public class UserResource {*

*@GET*

*@Path("/{cellphone}/orders")*

*@Scope("request")*

*@Produces(MediaType.APPLICATION\_JSON)*

*public Map<String, List<OrderVO>> getOrderesByCellphone(*

*@PathParam("cellphone") String cellphone) {*

*}*

*}*

*@Path("/orders")*

*public class OrderResource {*

*@GET*

*@Path("/{orderId}")*

*@Scope("request")*

*@Produces(MediaType.APPLICATION\_JSON)*

*public Map<String, List<OrderItemVO>> getOrderItemsByOrderId(){*

*}*

*}*

### 数据库脚本

DROP TABLE IF EXISTS `t\_order`;

CREATE TABLE `t\_order` (

`id` int(32) NOT NULL AUTO\_INCREMENT,

`order\_seq` varchar(128) DEFAULT NULL,

`table\_number` int(8) DEFAULT NULL,

`attendee\_number` int(8) DEFAULT NULL,

`total\_price` float(64,2) DEFAULT NULL,

`servent\_id` int(32) DEFAULT NULL,

`member\_id` int(32) DEFAULT NULL,

`casher\_id` int(32) DEFAULT NULL,

`order\_status` int(32) DEFAULT NULL,

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

`cellphone` varchar(255) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=3 DEFAULT CHARSET=utf8;

DROP TABLE IF EXISTS `t\_order\_item`;

CREATE TABLE `t\_order\_item` (

`id` int(32) NOT NULL AUTO\_INCREMENT,

`order\_id` int(32) DEFAULT NULL,

`dish\_id` int(32) DEFAULT NULL,

`dish\_amount` float(32,0) DEFAULT NULL,

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

`dish\_account` int(11) DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=7 DEFAULT CHARSET=utf8;

# 权限管理模块 (eorder)

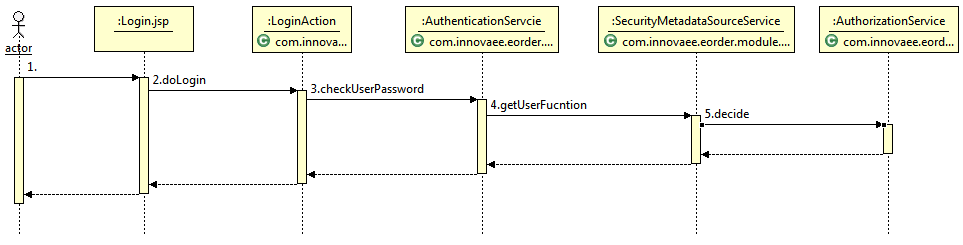
## 用户登录

用户使用系统合法的用户名和密码登录到系统。对于不正确的用户名和密码信息进行提示。详细需求说明见需求功能说明书UC\_005-用户登录

### 功能IPO图



### 时序图

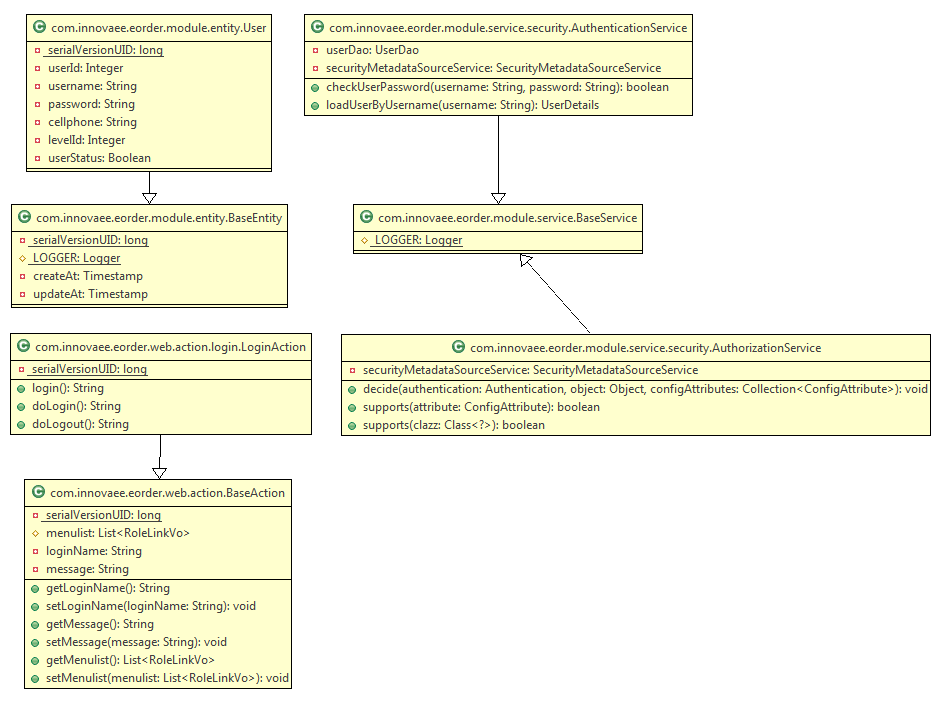


### 代码构成

实现用户登录系统的功能，需要下列文件支持，描述如下表

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| BaseVO.java | com.innovaee.eorder.module.vo | 所有VO（值对象）类的父类 |
| RoleLinkVo.java | com.innovaee.eorder.module.vo | 页面链接值对象，用户显示用户可以看到的菜单。 |
| BaseEntity.java | com.innovaee.eorder.module.entity | 所有实体类的父类 |
| User.java | com.innovaee.eorder.module.entity | 用户实体类，该类与数据库的t\_user对应，通过注解进行映射。 |
| BaseService.java | com.innovaee.eorder.module.service | 所有Service类的父类 |
| AuthenticationService  .java | com.innovaee.eorder.module.service.security | 提供验证服务，验证用户输入的用户名和密码是否匹配。 |
| AuthorizationService  .java | com.innovaee.eorder.module.service.security | 提供授权服务，根据用户角色获取相关权限信息。 |
| SecurityMetadataSource Service.java | com.innovaee.eorder.module.service.security | 安全元数据源服务类，根据用户角色获取相关权限信息，生成用户权限列表。 |
| BaseAction.java | com.innovaee.eorder.web.action | 所有Action类的父类 |
| LoginAction.java | com.innovaee.eorder.web.action.login | 登录模块的Action，响应登录页面提交的登录请求或其他页面的退出系统请求。 |
| struts.xml | com.innovaee.eorder.web.action.login | 登录模块的Action配置文件 |
| login.jsp | eorder\src\main\webapp\pages\login | 登录页面 |

### 类图



### 配置文件

* struts.xml

下面配置文件展现了如何将用户登录Action与URI路径以及对应的Java类配置起来

<struts>

<package name="com.innovaee.eorder.web.action.login" namespace="/login"

extends="struts-base">

<action name="login" class="com.innovaee.eorder.web.action.login.LoginAction"

method="login">

<result name="success">/pages/login/login.jsp</result>

</action>

<action name="doLogin" class="com.innovaee.eorder.web.action.login.LoginAction"

method="doLogin">

<result type="redirect">../user/doUser.action</result>

</action>

</package>

</struts>

* applicationContext-total.xml

在applicationContext-total.xml 中添加如下配置，使得用户登录相关Java对象成为Spring容器托管对象

<bean id="baseDAO" class="com.innovaee.eorder.module.dao.BaseDao"

abstract="true">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<bean id="roleFunctionDao" class="com.innovaee.eorder.module.dao.RoleFunctionDao"

parent="baseDAO" />

<bean id="functionDao" class="com.innovaee.eorder.module.dao.FunctionDao"

parent="baseDAO" />

<bean id="roleDao" class="com.innovaee.eorder.module.dao.RoleDao" parent="baseDAO" />

<bean id="userDao" class="com.innovaee.eorder.module.dao.UserDao" parent="baseDAO" />

<bean id="userRoleDao" class="com.innovaee.eorder.module.dao.UserRoleDao"

parent="baseDAO" />

<bean id="userLevelDao" class="com.innovaee.eorder.module.dao.UserLevelDao"

parent="baseDAO" />

<bean id="baseService" class="com.innovaee.eorder.module.service.BaseService"

abstract="true" />

<bean id="authorizationService"

class="com.innovaee.eorder.module.service.security.AuthorizationService"

parent="baseService" />

<bean id="authenticationService"

class="com.innovaee.eorder.module.service.security.AuthenticationService"

parent="baseService" />

<bean id="securityMetadataSourceService"

class="com.innovaee.eorder.module.service.security.SecurityMetadataSourceService"

parent="baseService" />

<bean id="userService" class="com.innovaee.eorder.module.service.UserService"

parent="baseService" />

* Hibernate 配置

项目使用Java 5之后的注解特性来进行Hibernate ORM配置，比之前XML文件配置更为简洁，用户实体与数据库表映射配置如下

@Entity

@Table(name = "t\_user")

public class User extends BaseEntity {

@Id

@Column(name = "USER\_ID", unique = true, nullable = false)

@GeneratedValue(strategy = GenerationType.AUTO)

private Integer userId;

@Column(name = "USERNAME")

private String username;

@Column(name = "PASSWORD")

private String password;

@Column(name = "CELLPHONE")

private String cellphone;

@Column(name = "LEVEL\_ID")

private Integer levelId;

@Column(name = "USER\_STATUS")

private Boolean userStatus;

### 数据库脚本

通过命令行或者GUI工具执行下面的SQL语句到目标数据库生成用户实体表

-- ----------------------------

-- Table structure for `t\_user`

-- ----------------------------

DROP TABLE IF EXISTS `t\_user`;

CREATE TABLE `t\_user` (

`id` int(32) NOT NULL AUTO\_INCREMENT,

`username` varchar(20) COLLATE utf8\_bin DEFAULT NULL,

`password` varchar(256) COLLATE utf8\_bin DEFAULT NULL,

`cellphone` varchar(16) COLLATE utf8\_bin DEFAULT NULL,

`user\_score` int(32) DEFAULT NULL,

`level\_id` int(32) DEFAULT NULL,

`user\_status` tinyint(1) DEFAULT '1',

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

PRIMARY KEY (`user\_id`)

) ENGINE=InnoDB AUTO\_INCREMENT=21 DEFAULT CHARSET=utf8 COLLATE=utf8\_bin COMMENT='user''s basic information';

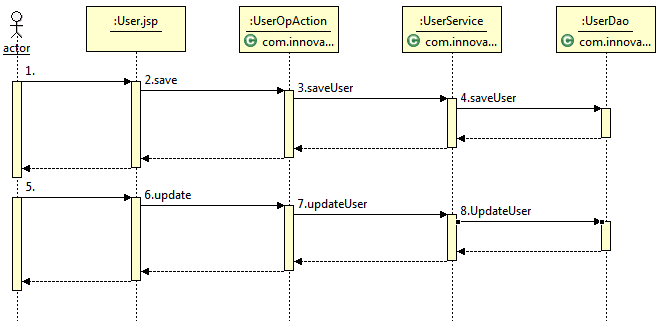
## 创建/编辑用户

拥有权限管理模块功能使用权限的系统用户，使用这个功能创建新的用户和更新现有用户信息。详细需求说明请查阅需求功能说明-UC\_005\_用户登录。

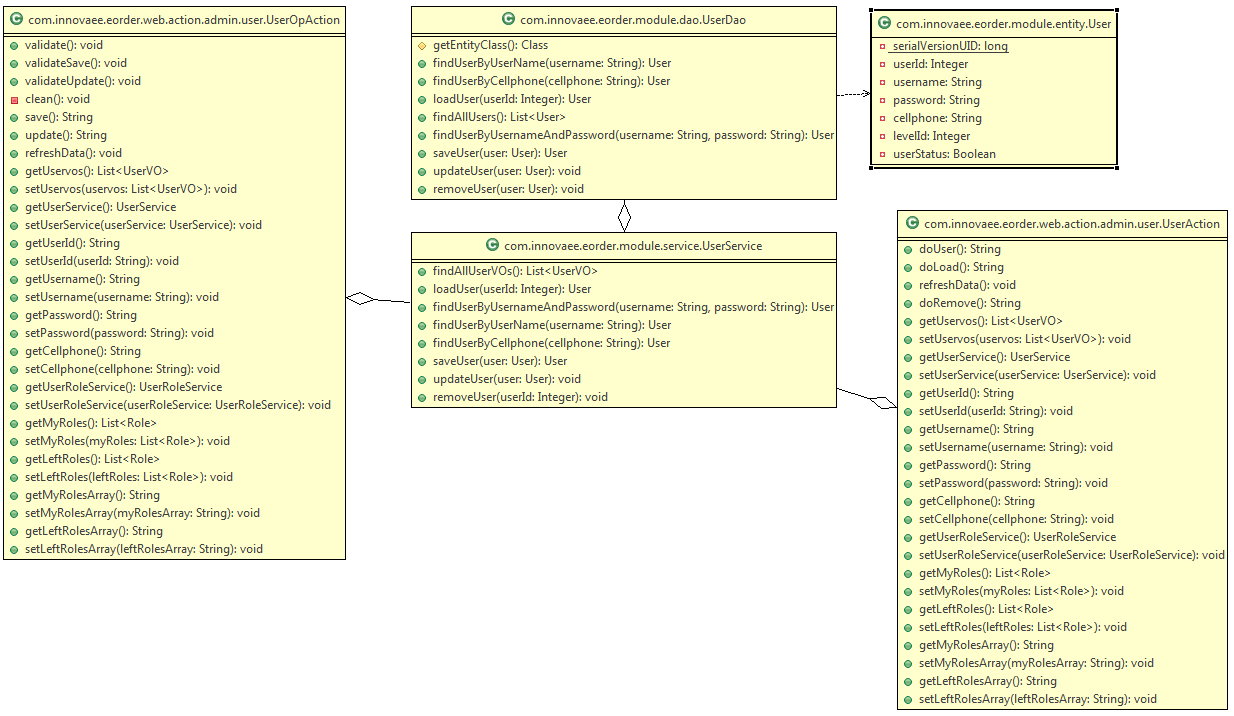
### 功能IPO图



### 时序图



### 类图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| UserService.java | com.innovaee.eorder.module.service | 提供用户操作相关的服务，比如用户创建和编辑等 |
| UserDao.java | com.innovaee.eorder.module.service.dao | 用户对象实体数据库访问类 |
| UserAction.java | com.innovaee.eorder.web.action.user | 用户管理功能负责显示用户数据功能的Action |
| UserOpAction.java | com.innovaee.eorder.web.action.user | 用户管理功能负责用户创建/编辑功能的Action |
| struts.xml | com.innovaee.eorder.web.action.user | 用户管理相关功能的Action配置文件 |
| user.jsp | eorder\src\main\webapp\pages\user | 用户管理页面 |

### 配置文件

用户创建/编辑功能的Hibernate配置和Spring配置在[5.1](#_用户登录)用户登录章节已经介绍过了。

* struts.xml

<struts>

<package name="com.innovaee.eorder.web.action.admin.user" namespace="/user"

extends="struts-base">

<action name="doLoad" class="com.innovaee.eorder.web.action.admin.user.UserAction"

method="doLoad">

<result name="input">/pages/admin/user/user.jsp</result>

<result>/pages/admin/user/user.jsp</result>

</action>

<action name="doUser" class="com.innovaee.eorder.web.action.admin.user.UserAction"

method="doUser">

<result name="input">/pages/admin/user/user.jsp</result>

<result>/pages/admin/user/user.jsp</result>

</action>

<action name="doEdit" class="com.innovaee.eorder.web.action.admin.user.UserAction"

method="doLoad">

<result name="input">/pages/admin/user/user.jsp</result>

<result>/pages/admin/user/user.jsp</result>

</action>

<action name="doRemove" class="com.innovaee.eorder.web.action.admin.user.UserAction"

method="doRemove">

<result name="input">/pages/admin/user/user.jsp</result>

<result>/pages/admin/user/user.jsp</result>

</action>

<action name="save" class="com.innovaee.eorder.web.action.admin.user.UserOpAction"

method="save">

<result name="input">/pages/admin/user/user.jsp</result>

<result>/pages/admin/user/user.jsp</result>

</action>

<action name="update" class="com.innovaee.eorder.web.action.admin.user.UserOpAction"

method="update">

<result name="input">/pages/admin/user/user.jsp</result>

<result>/pages/admin/user/user.jsp</result>

</action>

</package>

</struts>

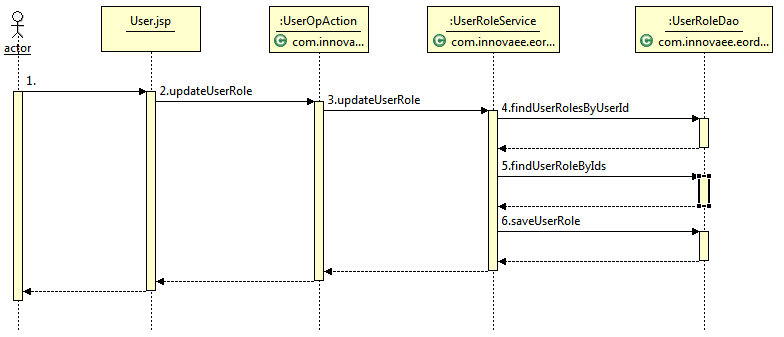
## 用户角色分配

拥有权限管理模块功能使用权限的系统用户，使用这个功能为系统用户分配相应的角色。一个用户可以拥有多个角色，默认新建用户获得会员这一角色。详细需求说明参见需求规格说明书UC\_008-用户管理-更新用户。

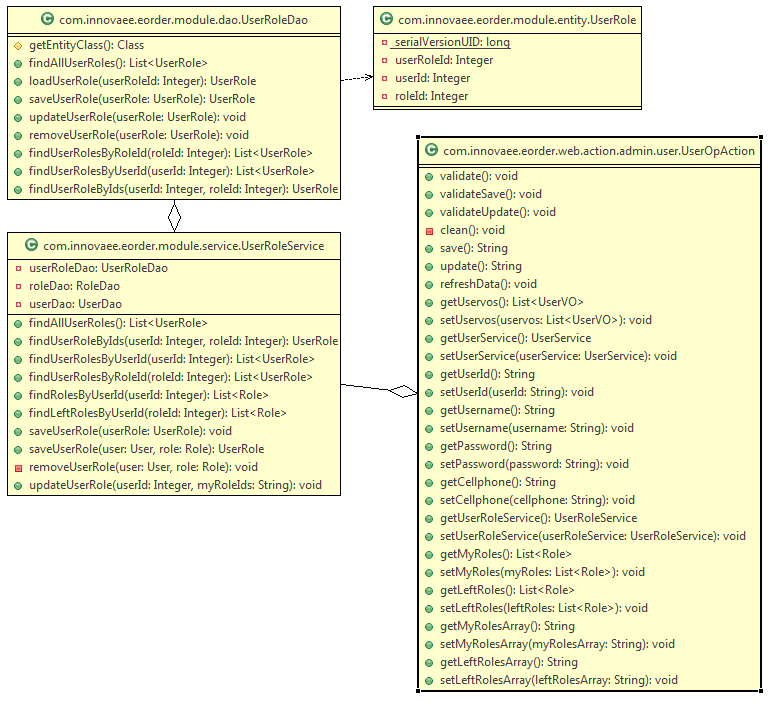
### 功能IPO图



### 时序图



### 类图



### 代码构成

UserOpAction.java代码说明在[5.2.4](#_代码构成)描述

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| UserRoleService.java | com.innovaee.eorder.module.service | 用户角色关联关系操作相关的服务，比如用户添加/删除角色 |
| UserRoleDao.java | com.innovaee.eorder.module.dao | 用户角色关联关系数据库访问对象 |
| UserRole.java | com.innovaee.eorder.module.entity | 用户角色关联关系实体对象 |

### 配置文件

* struts.xml

使用[5.2.5](#_配置文件)中说明的配置文件，不再重复描述

* applicationContext-total.xml

使用[5.1.5](#_配置文件_1)中说明的配置文件，不再重复描述

* Hibernate 配置

详见UserRole.java代码

@Entity

@Table(name = "t\_user\_role")

public class UserRole extends BaseEntity {

}

### 数据库脚本

DROP TABLE IF EXISTS `t\_user\_role`;

CREATE TABLE `t\_user\_role` (

`id` int(32) NOT NULL AUTO\_INCREMENT,

`user\_id` int(32) NOT NULL,

`role\_id` int(32) NOT NULL,

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

PRIMARY KEY (`id`),

KEY `fk\_userid` (`user\_id`),

KEY `fk\_roleid` (`role\_id`),

CONSTRAINT `fk\_roleid` FOREIGN KEY (`role\_id`) REFERENCES `t\_role` (`id`),

CONSTRAINT `fk\_userid` FOREIGN KEY (`user\_id`) REFERENCES `t\_user` (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=27 DEFAULT CHARSET=utf8;

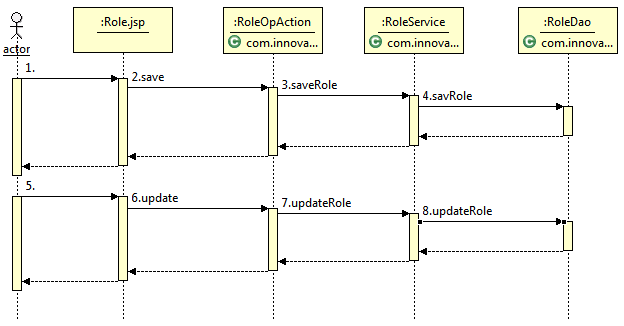
## 创建/编辑角色

拥有权限管理模块功能使用权限的系统用户，使用这个功能创建新的角色和更新现有角色信息。一个角色代表了一组相关的权限集合。详细需求说明参见需求规格说明书-UC\_009\_角色管理。

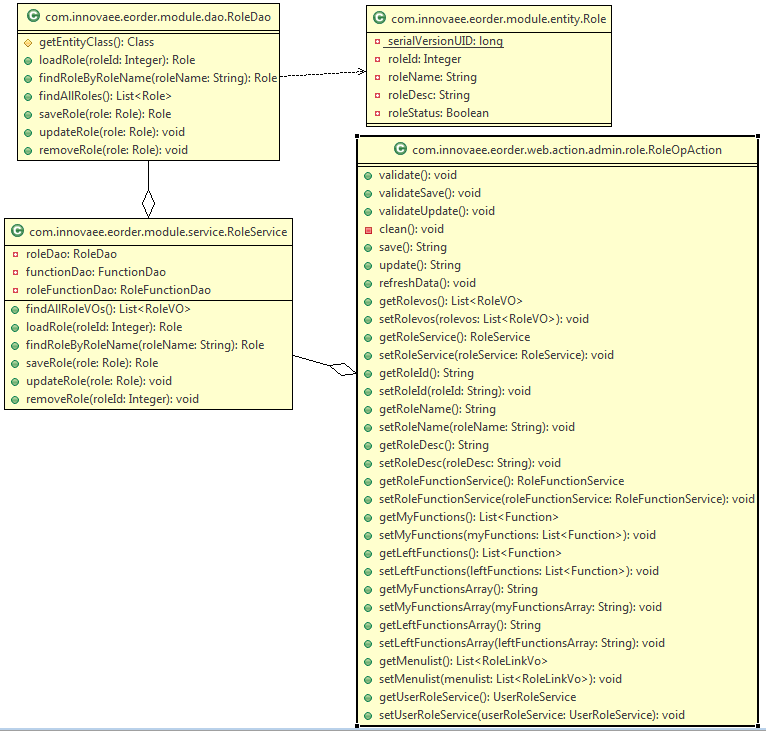
### 功能IPO图



### 时序图



### 类图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| RoleService.java | com.innovaee.eorder.module.service | 提供角色管理操作相关的服务，比如角色创建和编辑等 |
| RoleDao.java | com.innovaee.eorder.module.service.dao | 角色对象实体数据库访问类 |
| RoleAction.java | com.innovaee.eorder.web.action.user | 角色管理功能负责显示角色数据功能的Action |
| RoleOpAction.java | com.innovaee.eorder.web.action.user | 角色管理功能负责用户创建/编辑功能的Action |
| struts.xml | com.innovaee.eorder.web.action.role | 角色管理相关功能的Action配置文件 |
| role.jsp | eorder\src\main\webapp\pages\role | 角色管理页面 |

### 配置文件

* struts.xml

<struts>

<package name="com.innovaee.eorder.web.action.admin.role" namespace="/role"

extends="struts-base">

<action name="doRole" class="com.innovaee.eorder.web.action.admin.role.RoleAction"

method="doRole">

<result name="success">/pages/admin/role/role.jsp</result>

</action>

<action name="doEdit" class="com.innovaee.eorder.web.action.admin.role.RoleAction"

method="load">

<result name="input">/pages/admin/role/role.jsp</result>

<result>/pages/admin/role/role.jsp</result>

</action>

<action name="save" class="com.innovaee.eorder.web.action.admin.role.RoleOpAction"

method="save">

<result name="input">/pages/admin/role/role.jsp</result>

<result>/pages/admin/role/role.jsp</result>

</action>

<action name="update" class="com.innovaee.eorder.web.action.admin.role.RoleOpAction"

method="update">

<result name="input">/pages/admin/role/role.jsp</result>

<result>/pages/admin/role/role.jsp</result>

</action>

<action name="remove" class="com.innovaee.eorder.web.action.admin.role.RoleAction"

method="remove">

<result name="input">/pages/admin/role/role.jsp</result>

<result>/pages/admin/role/role.jsp</result>

</action>

</package>

</struts>

* applicationContext-total.xml

<bean id="roleDao" class="com.innovaee.eorder.module.dao.RoleDao" parent="baseDao" />

<bean id="roleService" class="com.innovaee.eorder.module.service.RoleService"

parent="baseService" />

* Hibernate 配置

详细配置见Role.java

@Entity

@Table(name = "t\_role")

public class Role extends BaseEntity {

}

### 数据库脚本

DROP TABLE IF EXISTS `t\_role`;

CREATE TABLE `t\_role` (

`id` int(32) NOT NULL AUTO\_INCREMENT,

`role\_name` varchar(128) COLLATE utf8\_bin DEFAULT NULL,

`role\_desc` varchar(128) COLLATE utf8\_bin DEFAULT NULL,

`role\_status` tinyint(1) DEFAULT '1',

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

PRIMARY KEY (`id`)

) ENGINE=InnoDB AUTO\_INCREMENT=28 DEFAULT CHARSET=utf8 COLLATE=utf8\_bin;

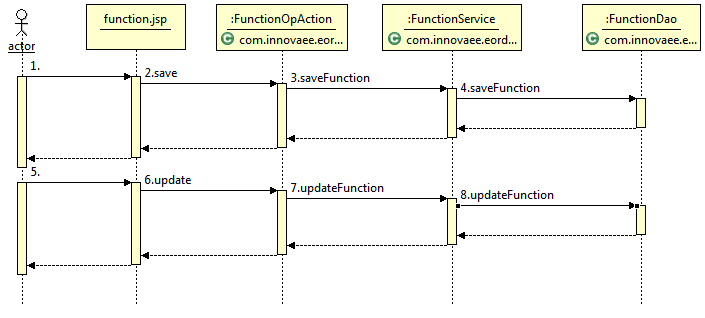
## 创建/编辑权限

拥有权限管理模块功能使用权限的系统用户，使用这个功能创建权限和对系统鲜有权限信息进行更新。详细需求说明参见需求规格说明书-UC\_012\_权限管理。

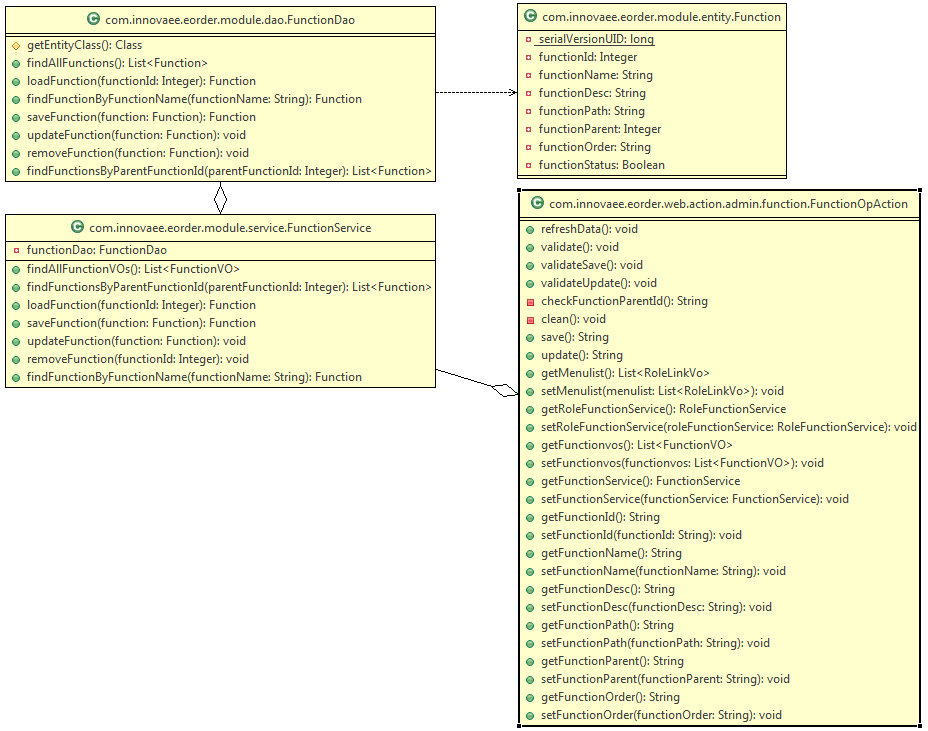
### 功能IPO图



### 时序图



### 类图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| FunctionService.java | com.innovaee.eorder.module.service | 提供权限管理操作相关的服务，比如权限创建和编辑等 |
| FunctionDao.java | com.innovaee.eorder.module.service.dao | 权限对象实体数据库访问类 |
| FunctionAction.java | com.innovaee.eorder.web.action.user | 权限管理功能负责显示角色数据功能的Action |
| FunctionOpAction.java | com.innovaee.eorder.web.action.user | 权限管理功能负责用户创建/编辑功能的Action |
| struts.xml | com.innovaee.eorder.web.action.role | 权限管理相关功能的Action配置文件 |
| function.jsp | eorder\src\main\webapp\pages\role | 权限管理页面 |

### 配置文件

* struts.xml

<?xml version="1.0" encoding="UTF-8" ?>

<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"

"http://struts.apache.org/dtds/struts-2.0.dtd">

<struts>

<package name="com.innovaee.eorder.web.action.admin.function"

namespace="/function" extends="struts-base">

<action name="doFunction"

class="com.innovaee.eorder.web.action.admin.function.FunctionAction" method="doFunction">

<result name="success">/pages/admin/function/function.jsp</result>

</action>

<action name="doEdit"

class="com.innovaee.eorder.web.action.admin.function.FunctionAction" method="doLoad">

<result name="input">/pages/admin/function/function.jsp</result>

<result>/pages/admin/function/function.jsp</result>

</action>

<action name="save"

class="com.innovaee.eorder.web.action.admin.function.FunctionOpAction"

method="save">

<result name="input">/pages/admin/function/function.jsp</result>

<result>/pages/admin/function/function.jsp</result>

</action>

<action name="update"

class="com.innovaee.eorder.web.action.admin.function.FunctionOpAction"

method="update">

<result name="input">/pages/admin/function/function.jsp</result>

<result>/pages/admin/function/function.jsp</result>

</action>

<action name="remove"

class="com.innovaee.eorder.web.action.admin.function.FunctionAction" method="remove">

<result name="input">/pages/admin/function/function.jsp</result>

<result>/pages/admin/function/function.jsp</result>

</action>

</package>

</struts>

* applicationContext-total.xml

<bean id="functionDao" class="com.innovaee.eorder.module.dao.FunctionDao"

parent="baseDao" />

<bean id="functionService" class="com.innovaee.eorder.module.service.FunctionService"

parent="baseService" />

* Hibernate 配置

详细配置见Function.java

@Entity

@Table(name = "t\_function")

public class Function extends BaseEntity {

}

### 数据库脚本

DROP TABLE IF EXISTS `t\_function`;

CREATE TABLE `t\_function` (

`id` int(32) NOT NULL AUTO\_INCREMENT,

`function\_name` varchar(128) COLLATE utf8\_bin DEFAULT NULL COMMENT 'the name of function',

`function\_desc` varchar(128) COLLATE utf8\_bin DEFAULT NULL COMMENT 'detail information',

`function\_path` varchar(128) COLLATE utf8\_bin DEFAULT NULL COMMENT 'the access path to this function',

`function\_parent` int(32) DEFAULT NULL COMMENT 'parent function',

`function\_order` varchar(16) COLLATE utf8\_bin DEFAULT NULL COMMENT 'use to organize menu in order\r\n the top function should be a integer as such 100000, 140000\r\n the first 2 bit is for top function',

`function\_status` tinyint(1) DEFAULT '1',

`create\_at` datetime DEFAULT NULL,

`update\_at` datetime DEFAULT NULL,

PRIMARY KEY (`id`)

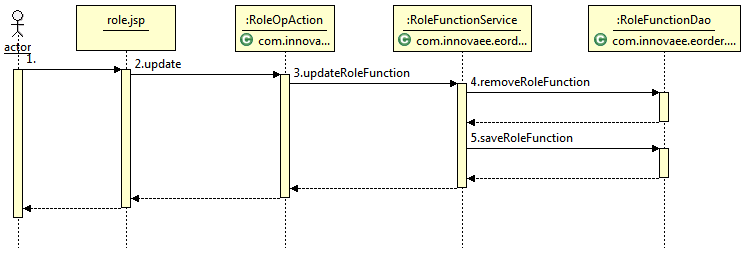
) ENGINE=InnoDB AUTO\_INCREMENT=49 DEFAULT CHARSET=utf8 COLLATE=utf8\_bin COMMENT='define system functions & path';

## 角色权限分配

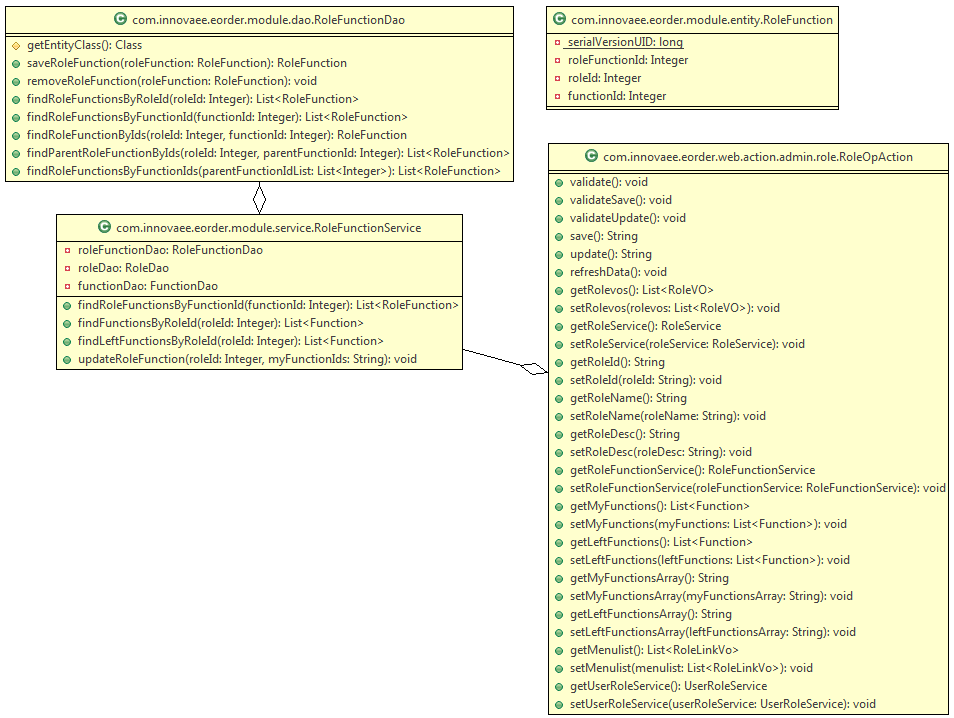
### 功能IPO图



### 时序图



### 类图



### 代码构成

|  |  |  |
| --- | --- | --- |
| 文件名 | 包路径 | 文件说明 |
| RoleFunctionService.java | com.innovaee.eorder.module.service | 角色权限关联关系操作相关的服务，比如角色添加/删除权限 |
| RoleFunctionDao.java | com.innovaee.eorder.module.dao | 角色权限关联关系数据库访问对象 |
| RoleFunction.java | com.innovaee.eorder.module.entity | 角色权限关联关系实体对象 |

### 配置文件

* struts.xml

使用[5.4.5](#_配置文件_2)中说明的配置文件，不再重复描述

* applicationContext-total.xml

<bean id="roleFunctionDao" class="com.innovaee.eorder.module.dao.RoleFunctionDao"

parent="baseDao" />

<bean id="roleFunctionService" class="com.innovaee.eorder.module.service.RoleFunctionService"

parent="baseService" />

* Hibernate 配置

详细配置见RoleFunction.java

*@Entity*

*@Table(name = "t\_role\_function")*

*public class RoleFunction extends BaseEntity {*

*}*

### 数据库脚本

# 数据库设计

## 用户表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 用户表 | | t\_user | |
| 数据项描述 | 数据项名 | 类型(长度) | 默认值 | 限制 |
| 用户ID | id | Int(32) | n/a | 主键，自动增加 |
| 用户名 | username | Varchar(128)r | n/a | 非空，英文字母和数字组成不少于6位 |
| 密码 | password | Varchar(128) | n/a | 非空，英文字母和数字组成不少于6位 |
| 手机号 | cellphone | Varchar(16) | n/a |  |
| 会员等级ID | level\_id | Int(32) | n/a |  |
| 用户状态 | User\_status | tinyint | 1 |  |
| 创建时间 | create\_at | datetime |  | 当前日期 |
| 修改时间 | update\_at | datetime |  | 当前日期 |

## 角色表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 角色表 | | t\_role | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 角色ID | id | Int(32) | n/a | 主键 |
| 角色名称 | role\_name | Varchar(128) | n/a |  |
| 角色描述 | role\_desc | Varchar(128) | n/a |  |
| 角色状态 | role\_status | tinyint | 1 |  |
| 创建时间 | create\_at | datetime |  | 当前日期 |
| 修改时间 | update\_at | datetime |  | 当前日期 |

## 功能表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 功能表 | | t\_function | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 功能ID | id | Int(32) | n/a | 主键 |
| 功能名称 | function\_name | Varchar(128) | n/a | 非空 |
| 功能描述 | function\_desc | Varchar(128) | n/a | 非空 |
| 功能路径 | function\_path | Varchar(128) | n/a | 非空 |
| 父功能ID | function\_parent | Int(32) | n/a | 可为空 |
| 功能排序 | function\_order | Varchar(128) | n/a | 非空 |
| 功能状态 | function\_status | tinyint | 1 | 非空 |
| 创建时间 | create\_at | datetime |  | 当前日期 |
| 修改时间 | update\_at | datetime |  | 当前日期 |

## 用户角色关联表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 用户角色关联表 | | t\_user\_role | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 用户角色ID | id | int(32) | n/a | 主键 |
| 用户ID | user\_id | Int(32 | n/a | 外键 |
| 角色ID | role\_id | Int(32 | n/a | 外键 |
| 创建时间 | create\_at | datetime | n/a | 当前日期 |
| 修改时间 | update\_at | datetime | n/a | 当前日期 |

## 角色功能关联表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 角色功能关联表 | | t\_role\_function | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 角色功能ID | id | int(32) | n/a | 主键 |
| 功能ID | function\_id | Int(32 | n/a | 外键 |
| 角色ID | role\_id | Int(32 | n/a | 外键 |
| 创建时间 | create\_at | datetime | n/a | 当前日期 |
| 修改时间 | update\_at | datetime | n/a | 当前日期 |

## 菜品分类表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 菜品分类表 | | t\_dish\_category | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 菜品类别ID | id | Int(32) | n/a | 主键，自动增加 |
| 类别名称 | category\_name | Varchar(r128) | n/a | 非空 |
| 类别图片 | category\_picture | Varchar(r128) | n/a | 非空 |
| 创建时间 | create\_at | datetime | n/a | 当前日期 |
| 修改时间 | update\_at | datetime | n/a | 当前日期 |

## 菜品表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 菜品表 | | t\_dish | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 菜品ID | id | Int(32) | n/a | 主键，自动增加 |
| 菜品分类ID | category\_id | Int(32) | 0 | 外键 |
| 菜品名称 | dish\_name | Varchar(128) | n/a | 非空 |
| 菜品图片 | dish\_picture | Varchar(128) | n/a | 非空 |
| 价格 | price | double(64) | 0.0 | 精确到小数点后一位 |
| 是否在售 | on\_sell | tinyint | True | 非空 |
| 其他信息 | misc | Varchar(128) | n/a | n/a |
| 创建时间 | create\_at | datetime |  | 当前日期 |
| 修改时间 | update\_at | datetime |  | 当前日期 |

## 订单表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 订单表 | | t\_order | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 订单ID | id | Int(32) | n/a | 主键，自动增加 |
| 订单流水号 | order\_seq | Varchar(128) | n/a | 格式为<yyyymmddxxxxx>  yyyy四位数年份  mm两位数月份  dd两位日期  xxxxx当天订单序号 |
| 订单状态 | order\_status | Varchar(128) | ‘new’ | 非空 |
| 台号 | table\_number | Int(32) | n/a |  |
| 人数 | attendee\_number | Int(32) | 1 |  |
| 实收金额 | total\_price | double | 0.0 | 精确小数点后一位 |
| 服务员ID | servent\_id | Int(32) | n/a | 非空 |
| 会员ID | member\_id | Int(32) | n/a | n/a |
| 收银员ID | casher\_id | Int(32) | n/a | n/a |
| 创建时间 | create\_at | datetime |  | 当前日期 |
| 修改时间 | update\_at | datetime |  | 当前日期 |

## 订单条目表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 订单条目表 | | t\_order\_item | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 订单条目ID | id | Int(32) | n/a | 主键，自动增加 |
| 订单ID | order\_id | Int(32) | 0 |  |
| 菜品ID | dish\_id | Int(32) | 0 | 非空 |
| 份数 | dish\_account | Int(32) | 1 | 最大99 |
| 创建时间 | create\_at | datetime | 当前日期 | 当前日期 |
| 修改时间 | update\_at | datetime | 当前日期 | 当前日期 |

## 会员等级表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 数据表名 | 会员等级表 | | t\_user\_level | |
| 数据项描述 | 数据项名 | 类型 | 默认值 | 限制 |
| 会员等级ID | id | Int(32) | n/a | 主键，自动增加 |
| 等级名称 | level\_name | Int(32) | n/a |  |
| 等级分数 | level\_score | Int(32) | n/a |  |
| 等级状态 | level\_status | tinyint | n/a | 非空 |
| 创建时间 | create\_at | datetime | 当前日期 | 非空 |
| 修改时间 | update\_at | datetime | 当前日期 | 非空 |

# Web-Service接口协议

## 获取用户信息

* URL：http://192.168.0.100:8080/eorder-ws/rest/users/<cellphone>
* 方法：GET
* 输入：手机号码

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数类型 | 说明 | 备注 |
| cellphone | String | 手机号码 | 例如：13988888888 |

* 输出： 会员信息

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数类型 | 说明 | 备注 |
| id | Integer | 分类id | 例如： 1 |
| username | String | 分类名称 | 例如：张三 |
| cellphone | String | 该分类的图片 | 例如：13988888888 |
| levelName | String | 用户等级名称 | 金牌会员 |
| discount | Float | 折扣信息 | 例如：钻石8折、白金8.5折、金牌9折、普通会员9.5折等等。 |

* 样列：

{

"user":

{

"id": 1,

"username": "张三",

"cellphone": “13988888888”,

"levelName": “金牌会员”,

"discount：9

}

}

## 获取目前所有分类列表

* URL：http://192.168.0.100:8080/eorder-ws/rest/categories
* 方法：GET
* 输入：无
* 输出：分类列表，包含分类名，分类id字段，分类图片URL

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 参数类型 | 说明 | 备注 |
| id | Integer | 分类id | 例如： 1 |
| name | String | 分类名称 | 例如：最新 |
| picPath | String | 该分类的图片 | 例如：http://www....33/21.jpg |

* 举例：

{

"categories": [

{

"id": 1,

"name": "最热",

"picPath": null

},

{

" id ": 2,

" name ": "最新",

" picPath ": null

},

{

" id ": 3,

" name ": "湘菜",

" picPath ": null

}

]

}

## 获取某一个分类下面所有菜品列表

* URL：http://localhost:8080/eorder-ws/rest/categories/<categoryId>/dishes
* 方法：GET
* 输入：分类id

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. categoryId | 1. int | 1. 分类id | 1. 例如： 1 |

* 输出：菜品列表，包含菜品名、菜品id字段、菜品价格和菜品图片URL

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. id | 1. Integer | 1. 菜品id | 1. 例如：30001 |
| 1. name | 1. String | 1. 菜品名称 | 1. 例如：麻婆豆腐 |
| 1. picPath | 1. String | 1. 该菜品的图片 | 1. 例如：http://www....33/31.jpg |
| 1. price | 1. Float | 1. 该菜品的价格 | 1. 例如：48.5 |

* 举例：

｛dishes:[

｛＂id＂：＂30001＂，＂name＂：＂麻婆豆腐＂，＂price＂：＂18＂，＂picPath＂：＂http://www.google.com/2332333/31.jpg＂｝

｛＂id＂：＂30002＂，＂name＂：＂豆花鱼＂，＂price＂：＂38＂，＂picPath＂：＂http://www.google.com/2332333/32.jpg＂｝

｛＂id＂：＂30003＂，＂name＂：＂夫妻肺片＂，＂price＂：＂28.9＂，＂picPath＂：＂http://www.google.com/2332333/33.jpg＂｝]

}

## 查询某个会员的历史订餐记录

* URL：http://localhost:8080/eorder-ws/rest/users/<cellphone>/orders
* 方法：GET
* 输入：会员手机号码

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. cellphone | 1. String | 1. 手机号码 | 1. 例如： 13988888888 |

* 输出：会员历史订单记录列表, 包含订单id,　订单日期，总价;

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. id | 1. Integer | 1. 订单id | 1. 例如： 1 |
| 1. createAt | 1. String | 1. 订单时间 | 1. 例如：2014-8-1 |
| 1. totalPrice | 1. Float | 1. 订单的总价 | 1. 例如：104.5 |

* 举例：

｛”orders”:

[ ｛＂id＂：＂40001＂，＂createAt＂：＂2014-8-1＂,＂totalPrice＂： "104.5"｝

｛＂id＂：＂40002＂，＂createAt＂：＂2014-9-15＂, ＂totalPrice＂："234.5"｝

｛＂id＂：＂40003＂，＂createAt＂：＂2014-10-11＂, ＂totalPrice＂："800"｝]

.......

｝

## 查询某个订单详情

* URL：http://localhost:8080/eorder-ws/rest/orders/<orderId>
* 方法：GET
* 输入：订单id

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. orderId | 1. Integer | 1. 菜品id | 1. 例如： 1 |

* 输出：订单详情，包括订单号，订单日期，订单总价，订单菜品列表，单项菜品数量;

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. dishId | 1. Integer | 1. 菜品id | 1. 例如： 1 |
| 1. dishName | 1. String | 1. 菜品名称 | 1. 例如：麻婆豆腐 |
| 1. dishPrice | 1. Float | 1. 菜品的单价 | 1. 例如：18 |
| 1. dishAmount | 1. Integer | 1. 菜品的数量 | 1. 例如：１ |
| 1. dishPicture | 1. String | 1. 菜品的图片 | 1. 例如：http://www....33/31.jpg |

* 举例：

｛ “orderitems”：[

｛＂dishId＂：＂30001＂，＂dishName＂：＂麻婆豆腐＂，＂dishPrice＂：＂18＂，　＂dishAmount＂：＂１＂，＂dishPicture＂：＂http://www.google.com/2332/31.jpg＂｝

｛＂dishId＂：＂60001＂，＂dishName＂：＂鱼香肉丝＂，＂dishPrice＂：＂32＂，＂dishAmount＂：＂１＂，＂dishPicture＂：＂http://www.google.com/2332/88.jpg＂｝

｛＂dishId＂：＂80001＂，＂dishName＂：＂紫菜蛋汤＂，＂dishPrice＂：＂28＂，＂dishAmount＂：＂１＂，＂dishPicture＂：＂http://www.google.com/2333/92.jpg＂｝

]｝

## 订单下单

* URL：http://localhost:8080/eorder-ws/rest/orders/
* 方法：POST
* 输入：参数说明见下表

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. tableNumber | 1. Integer | 1. 用餐桌号 | 1. 例如： 2 |
| 1. attendeeNumber | 1. Integer | 1. 用餐人数 | 1. 例如：4 |
| 1. serventId | 1. Integer | 服务员工号 | 1. 例如：123 |
| 1. memberId | 1. Integer | 1. 会员ID | 1. 例如：456 |
| 1. items | 1. Array | 1. 点菜明细 |  |
| 1. dishId | 1. Integer | 1. 菜品ID |  |
| 1. dishAmount | 1. Integer | 1. 菜品份数 |  |

* 举例：

{

"tableNumber": 2,

"attendeeNumber": 4,

"serventId": 123,

"memberId": 456,

"items": [

{

"dishId": 123,

"dishAmount": 2

}

]

}}

* 输出：JSON格式的调用结果

|  |  |  |  |
| --- | --- | --- | --- |
| 1. 名称 | 1. 参数类型 | 1. 说明 | 1. 备注 |
| 1. result | 1. String | 1. 调用成功或者失败 | 1. 成功返回success 2. 失败返回false |
| 1. message | 1. String | 1. 用来描述调用结果的字符串 |  |

* 举例：

{"result":"success","message":"创建订单成功"}

## 接口调用异常信息返回

接口使用过程中可能因为输入非法参数导致无法获取期望的数据结果，或者因为Web-Service端异常导致无法返回正确的结果时，接口将发还带有异常信息文本的JSON串。

* 举例：

｛ “exception”: “User Not Found – cellphone:123456789”｝

# 附录