# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

1. 互联网金融项目背景，概念介绍
2. 项目管理与环境搭建
3. 生产环境模拟
4. 相对路径和绝对路径
5. 项目开发规范
6. 项目模块介绍
7. Maven 项目模块划分

# 第 1 章 互联网金融

## 1.1 什么是互联网金融

1.互联网金融（ITFIN）是指传统金融机构与互联网企业利用互联网技术和信息通信技术实现资金融通、支付、投资和信息中介服务的新型金融业务模式。

2.互联网金融 ITFIN 不是互联网和金融业的简单结合，而是在实现安全、移动等网络技术水平上，被用户熟悉接受后（尤其是对电子商务的接受），自然而然为适应新的需求而产生的新模式及新业务。

3.传统金融行业与互联网技术相结合的新兴领域。

## 1.2 互联网金融的具体应用

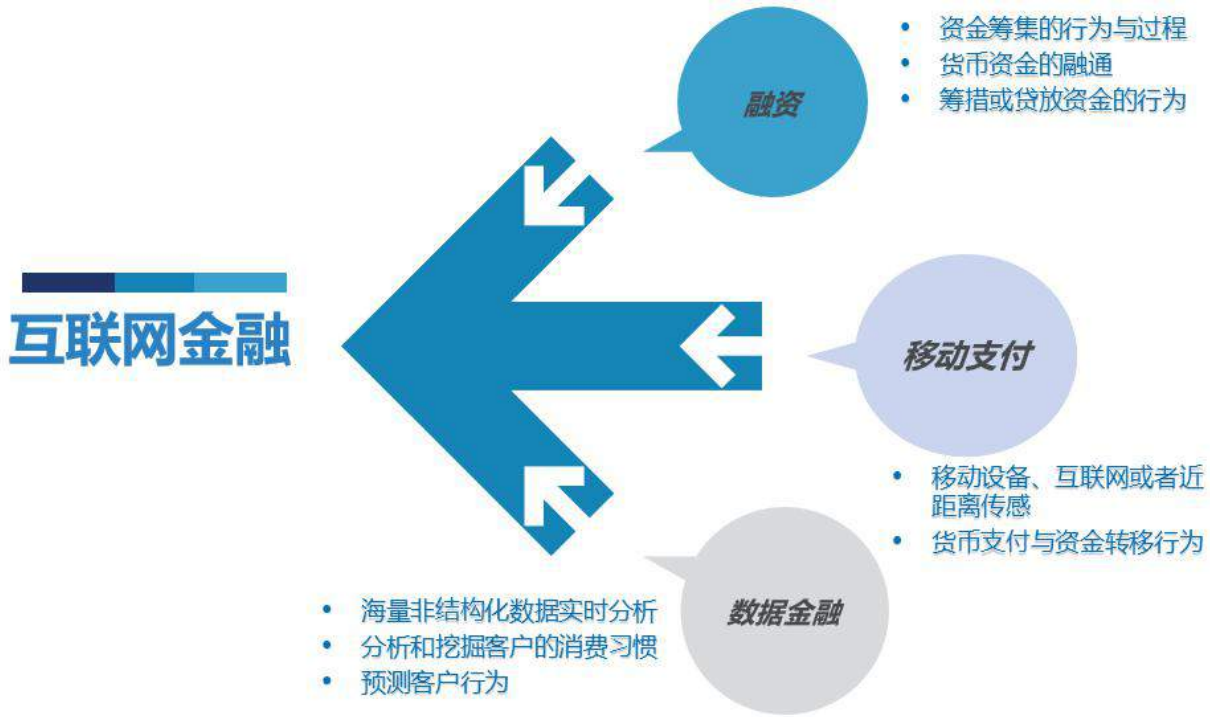
任何和金融相关的互联网应用都属于互联网金融



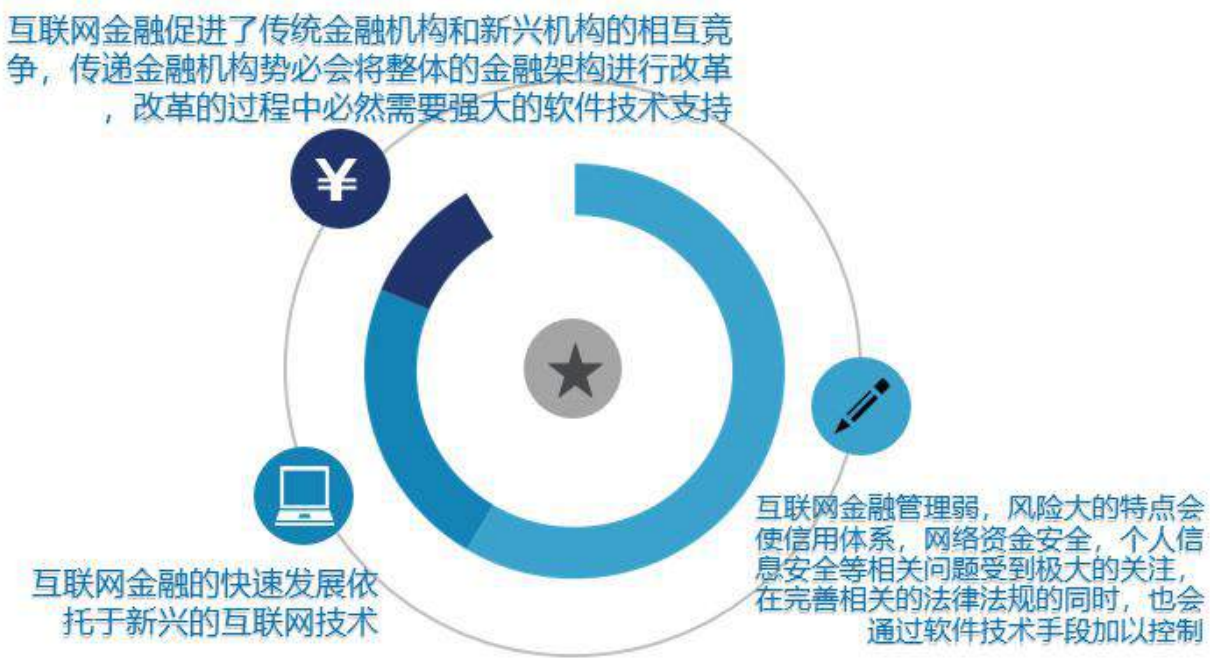

互联网金融通过互联网、移动互联网等工具，使得传统金融业务具备透明度更强、参与度更高、协作性更好、中间成本更低、操作上更便捷等一系列特征。

## 1.3 互联网金融发展的三个阶段

## 1.4 互联网金融的业务模式

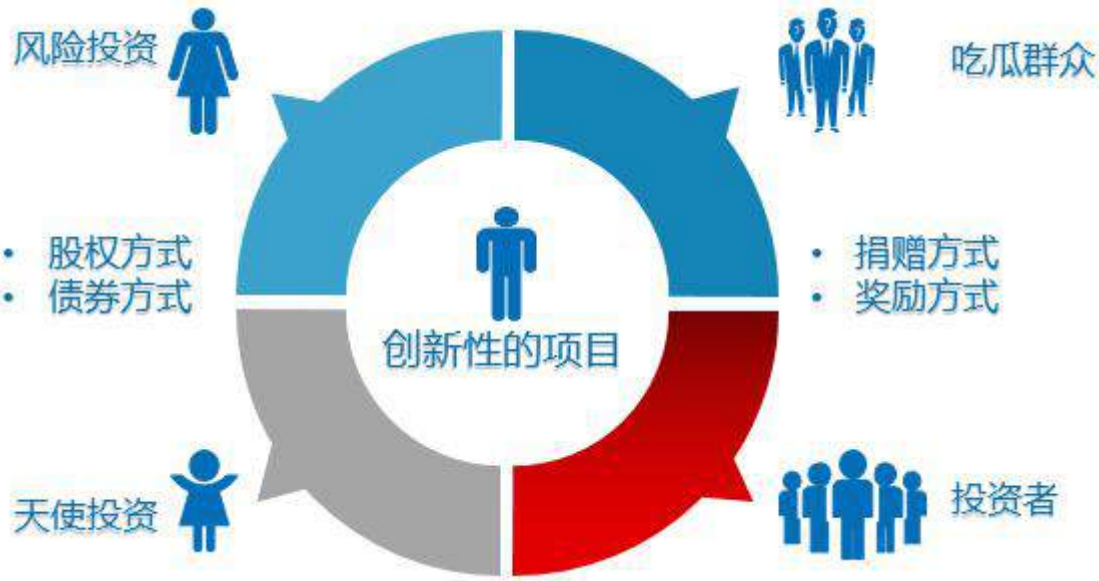



## 1.5 互联网金融与软件开发



# 第 2 章 众筹系统

众筹系统，这个在网络上不断被搜索的热门词汇，从最初的陌生到熟悉，到现在不断被更新，出现各种不同的众筹模式，不得不承认众筹系统的出现，是对传统行业的一种冲击，对传统金融模式的一种冲击，同时对于年轻的一代而言，这也是一个契机、一个机遇、一个开创自己事业的平台，正是因为这些利好，让更多的人愿意去运用众筹系统，作为其项目发展孵化的平台



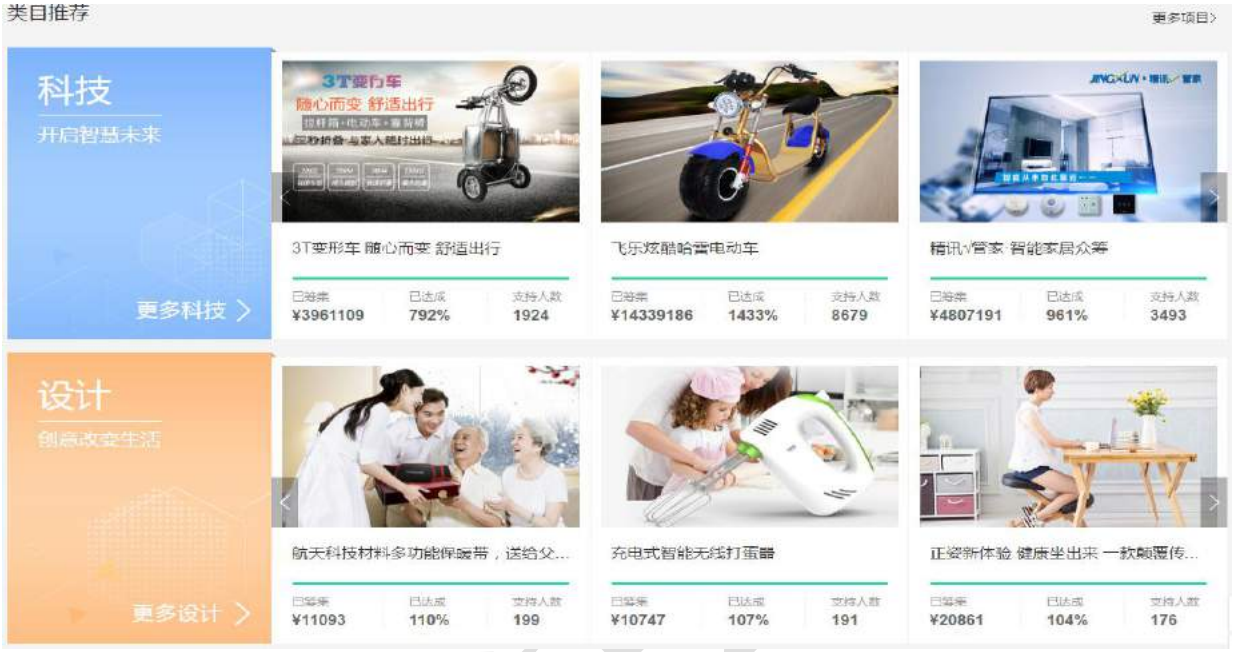

## 2.1 成功案例

美国总统选举

乐视明星代言

商学院招生

## 2.2 项目列表



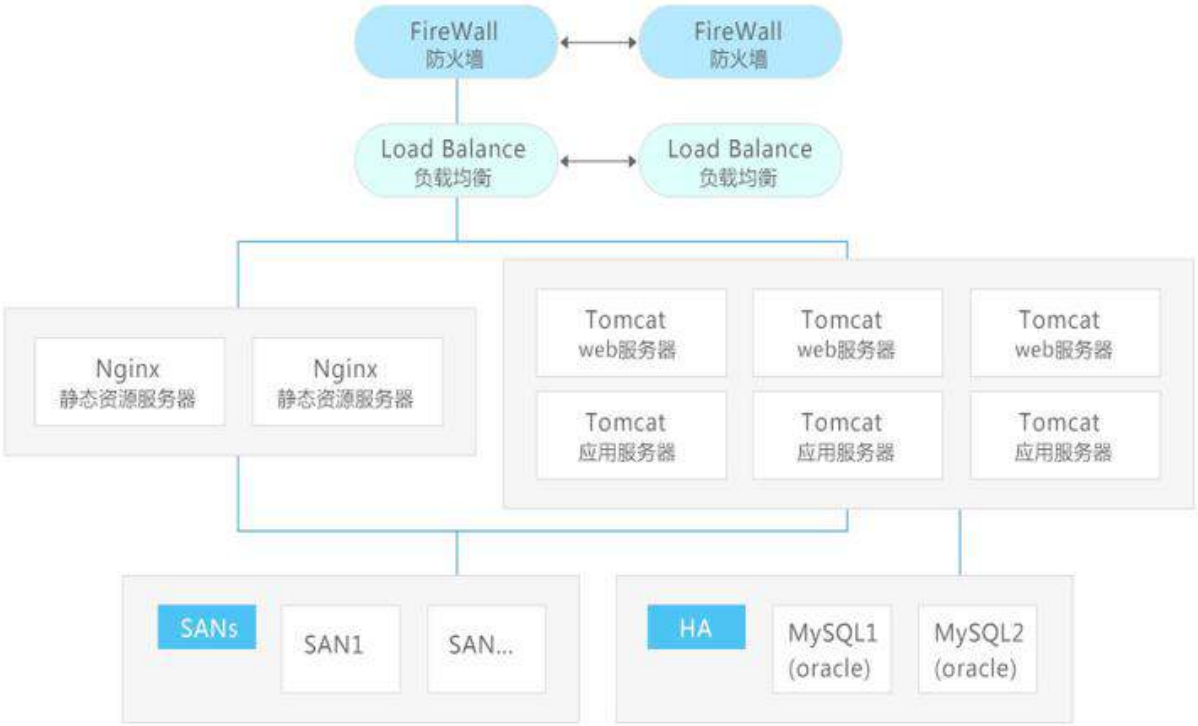

## 2.3 项目展示


确认回报内容
确认订单
付款
完成
请确认您所选择的回报项信息和购买数量
项目名称
发起人
回报内容
回报数量
支持单价
配送费用
充电式智能无线打蛋器
江门名有贸易有限公司
750元 限额1000位 感谢您的支持，您将以750元的价格获得1台 市场单价299元的充电式智能无线打蛋器。
1
¥ 750.00
¥ 25.00
总价(含运费)： ¥ 775.00
去结算
风险提示
1. 众筹并非商品交易，存在一定风险。支持者根据自己的判断选择，支持众筹项目，与发起人共同实现梦想并获得发起人承诺的回报。
2. 众筹平台仅提供平台网络空间及技术支持等中介服务，众筹仅存在于发起人和支持者之间，使用众筹平台产生的法律后果由发起人与支持者自行承担。
3. 本项目必须在2016-12-22之前达到 ¥10000.00 的目标才算成功，否则已经支持的订单将取消。订单取消或募集失败的，您支持的金额将原支付路径退回。
4. 请在支持项目后15分钟内付款，否则您的支持请求会被自动关闭。
5. 众筹成功后由发起人统一进行发货，售后服务由发起人统一提供；如果发生发起人无法发放回报、延迟发放回报、不提供回报后续服务等情况，您需要直接和发起人协商解决。
6. 如您不同意上述风险提示内容，您有权选择不支持；一旦选择支持，视为您已确认并同意以上提示内容。

## 2.4 业务模块


众筹系统总架构
访问层
前台功能
平台管理
运营推广
系统设置
PC端
移动端
（Android iOS）
微信端
平台首页
我的账户
轻众筹
账户总览
资金管理
项目管理
个人资料
消息管理
项目列表
投资人
新闻资讯
帮助中心
论坛
关于我们
发起项目
个人账户管理
企业账户管理
认证审核管理
业务管理
项目管理 订单管理 回报管理 项目进展 信息披露
评论管理 放款审核 停止放款审核 分红审核
财务管理
充值管理 提现管理 放款管理 退款管理 分红管理
财务管理 对账管理
站点管理
消息管理
基本设置
活动管理
消息模板
资料管理
后台账号管理
系统日志
APP版本管理
系统参数管理
数据库备份
数据统计
资金统计
用户统计
接口支撑层
消息通道
短信 邮件 微信
资金方
第三方支付接口 资金托管/存管
其他增值
第三方征信接入 合同保全 电子签名 实名认证

## 2.5 系统架构




## 2.6 尚筹网

**atcrowdfunding**

**互联网创意产品众筹平台**

# 第 3 章 尚筹网 项目管理与环境搭建

## 3.1 项目构成

### 3.1.1 项目范围

尚筹网众筹系统的实现(参考原型)

### 3.1.2 人员组织结构

项目经理（1） + 系统架构师（1） + 系统分析师（2） + 项目组（组长(TL)，软件工程师（SE），组员(PG)）（5-7）*N

### 3.1.3 项目周期

需求分析（20%） + 设计（30%） + 开发（20%） + 测试（30%）

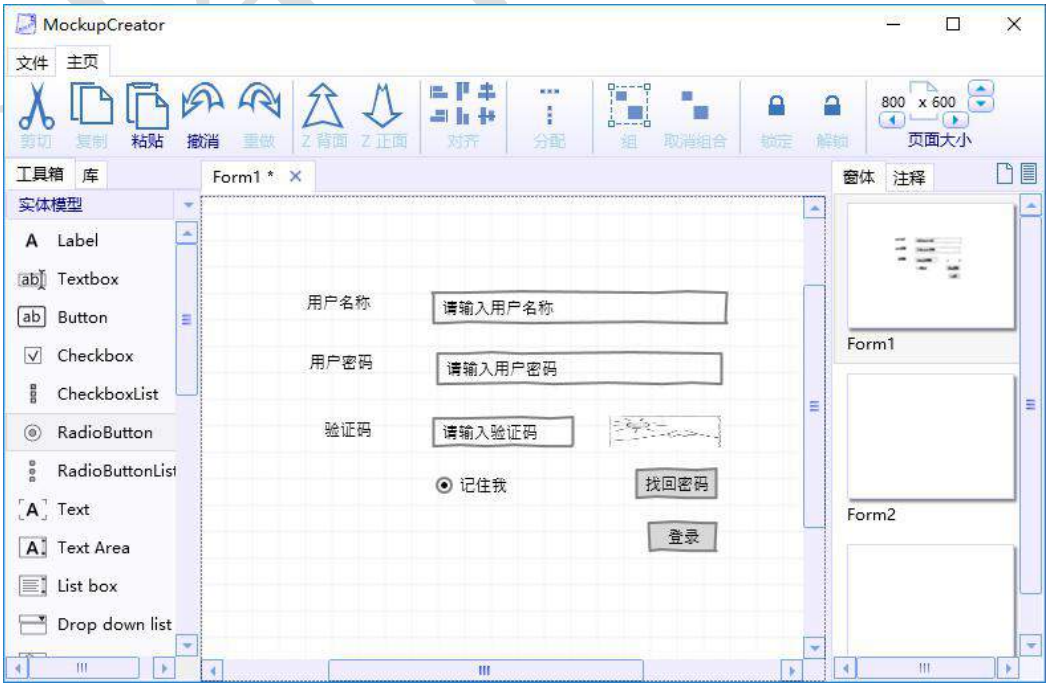
## 3.2 项目需求分析

### 3.2.1 需求采集

### 3.2.2 需求编制

### 3.2.3 需求审核

Web 项目可根据需求创建原型页面或草图设计用于确认需求



# 3.3 项目设计

## 3.3.1 页面设计

1. Frontpage
2. Dreamweaver
3. 文本编辑器

## 3.3.2 物理数据模型

1. PDM -- 数据库设计 （PowerDesigner）
2. 抽取概念模型 --> 转换对象模型 --> 转换数据模型

## 3.3.3 业务流程设计

## 3.3.4 UML (类图，时序图，用例图，页面迁移图)

*注:(Rational_Rose,操作系统为家庭版无法安装)*

UML 统一建模语言（UML，Unified Modeling Language）是面向对象开发中一种通用的图形化建模语言，它定义良好、易于表达、功能强大且普遍适用。

1. 建模工具
    1) Rational Rose
    2) Window Visio
    3) PowerDesigner
    4) 画图工具
2. 模型图
    1) 用例图
    2) 类图
    3) 行为图

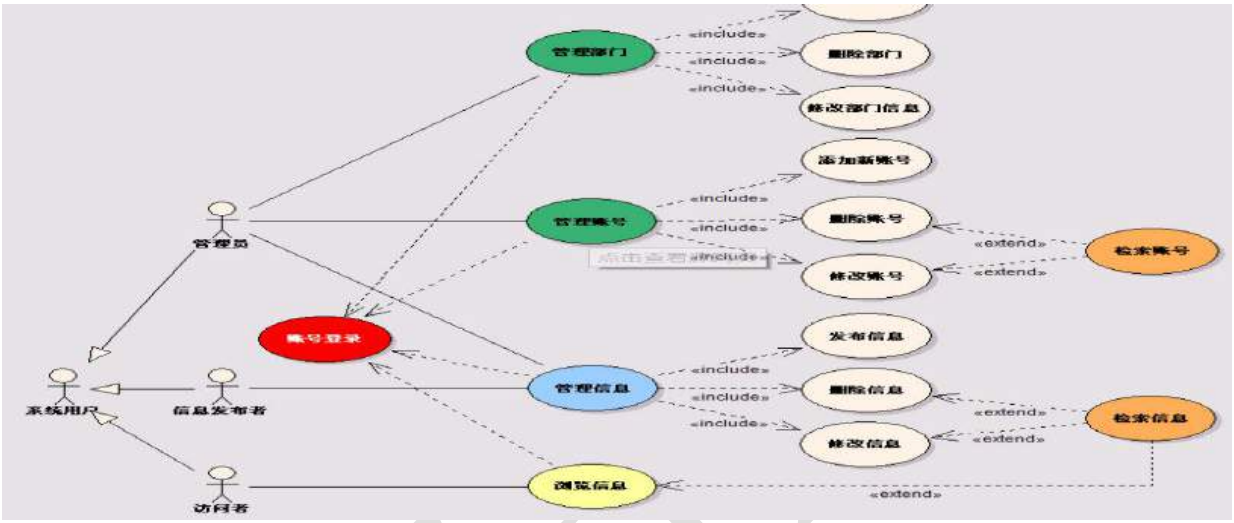
4) 状态图

5) 序列图

6) 活动图

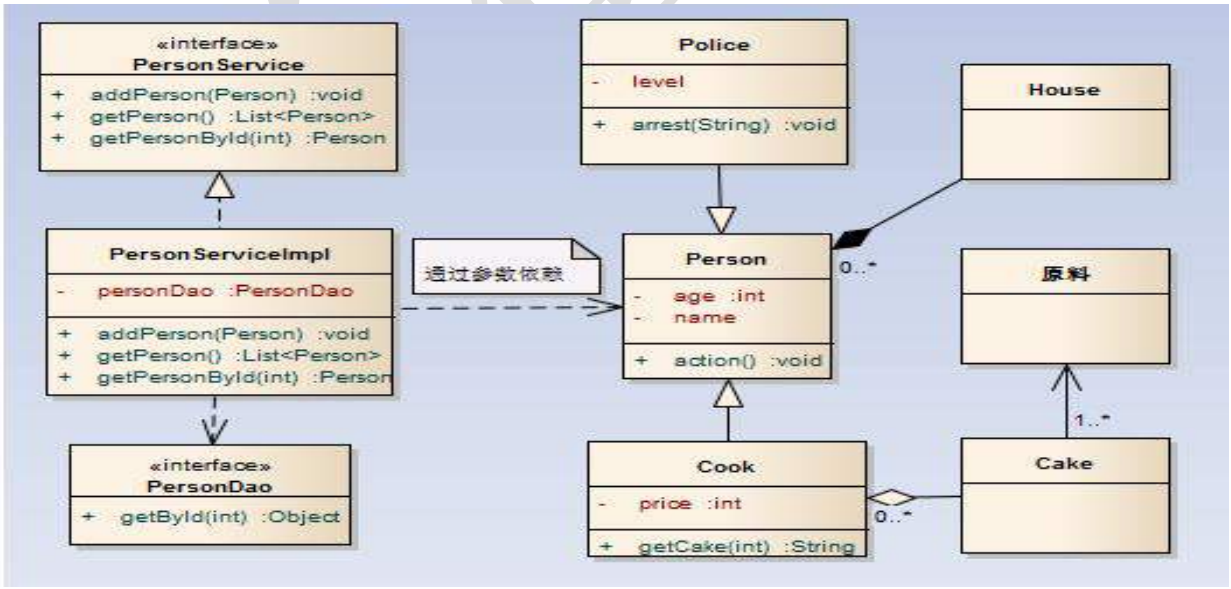
3. 用例图

描述角色及角色与用例之间的连接关系。说明的是谁要使用系统，以及他们使用该系统可以做些什么。



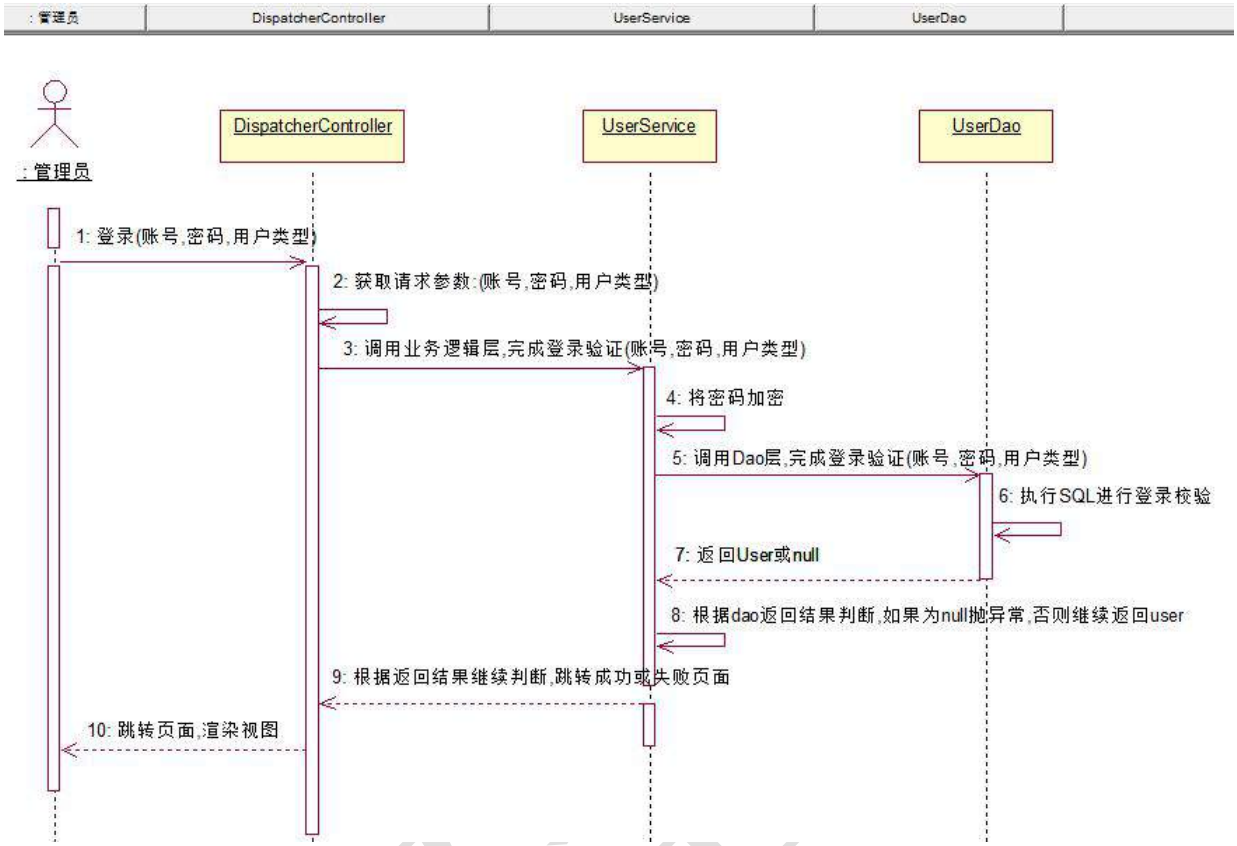

4. 类图

显示系统的静态结构，表示不同的实体是如何相关联的。




5. 时序图:

描述对象之间的交互顺序，着重体现对象间消息传递的时间顺序。




## 3.4 项目开发

### 3.4.1 项目环境搭建

1 创建 Web 项目

2 修改开发工具 Eclipse 的相关配置

1) 字符编码 UTF-8

2) JDK 版本 - 1.8

3) 去掉所有的验证，提高编译效率

3 增加基础框架的类库及相应配置文件

1) SpringMVC

2) Spring

3) Mybatis

4) DB

5) WEB

4 增加项目基础 JAVA 代码（包，接口，类）

包名起名的原则：

域名的反写(com.atguigu) + 项目名称(atcrowdfunding) + 模块名称(member) + 程序类型名称(bean)

1) Bean

2) Controller/Handler

3) Service/ServiceImpl

4) Dao/Mapper

5) Util

6) Filter

7) Listener

8) Interceptor

### 3.4.2 项目代码开发

1. 编码规范（阿里 Java 开发规范手册.pdf）
2. 模块开发

# 第 4 章 Maven 项目模块划分

atcrowdfunding-parent 父工程,聚合其他工程(pom)

atcrowdfunding-main Web 工程,存放所有页面,框架配置文件(war)

atcrowdfunding-manager-impl 后台管理系统,存放控制器类,业务层实现类(jar)

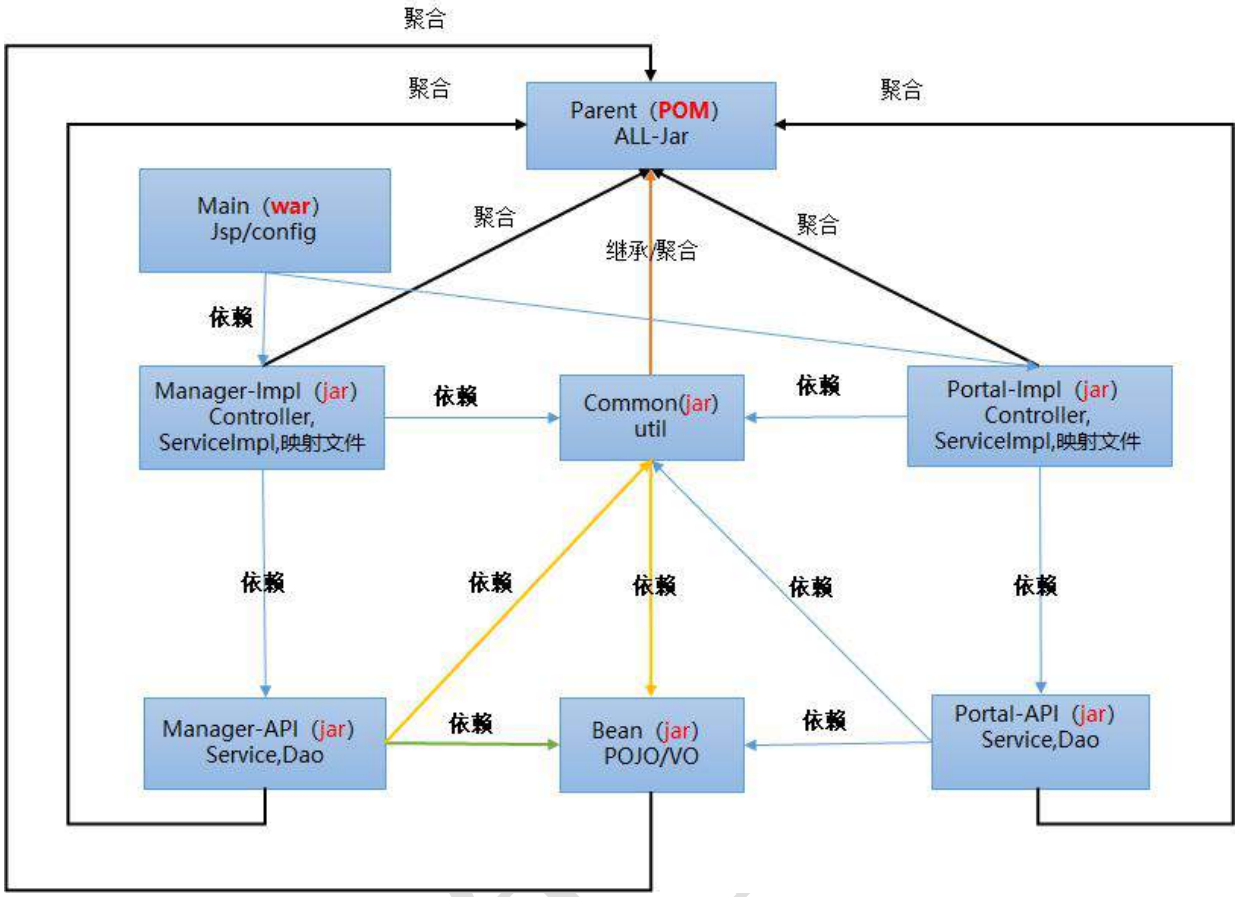
atcrowdfunding-manager-api 后台管理系统,存放业务层接口和 DAO 层接口(jar)

atcrowdfunding-portal-impl 前台系统,存放控制器类,业务层实现类(jar)

atcrowdfunding-portal-api 前台系统,存放业务层接口和 DAO 层接口(jar)

atcrowdfunding-common 存放所有模块所需要的公共类(jar)

atcrowdfunding-bean 存放所有模块的实体类(jar)




# 第 5 章 监听器

## 5.1 WEB 监听器

### 5.1.1 监听 ServletContext 对象

1. javax.servlet.ServletContextListener

监听 ServletContext 对象创建和销毁.

2. 应用场景:

- 在服务器启动时建立数据库表结构,初始化数据库
- 在服务器启动时,将数据库常量数据加载到内存,提供访问效率
- 在服务器启动时,获取项目上下文路径,存放到 application 域,给页面使用

- 存放计数器,计算在线用户数

3. javax.servlet.ServletContextAttributeListener

监听 ServletContext 对象的属性的变化:添加,覆盖,删除

### 5.1.2 监听 HttpSession 对象

4. javax.servlet.http.HttpSessionListener

监听 HttpSession 对象的创建和销毁

5. javax.servlet.http.HttpSessionAttributeListener

监听 HttpSession 对象的属性变化:添加,覆盖,删除

### 5.1.3 监听 HttpServletRequest 对象

6. javax.servlet.ServletRequestListener

监听 HttpServletRequest 对象的创建和销毁

7. javax.servlet.ServletRequestAttributeListener

监听 HttpServletRequest 对象的属性变化:添加,覆盖,删除

## 5.2 自定义监听器

1. 在服务器启动时,将应用上下文路径存放到 application 域中
2. 在 JSP 页面中使用 EL 表示式获取应用上下文路径使用它.
3. StartUpSystemListener

```
<listener>
        <listener-class>com.atguigu.atcrowdfunding.listener.StartUpSystemListener</listener-class>
</listener>
```

# 第 6 章 过滤器

## 6.1 Web 开发

1. 过滤器都应该实现 javax.servlet.Filter 接口
2. 过滤器的作用
   1) 对目标访问前或访问后进行过滤. 实现某种功能扩展
3. 应用场景
   1) 对密码进行加密
   2) 对请求参数进行字符编码设置
   3) 对请求进行权限控制
   4) 对请求参数值进行非法字符过滤
   5) 修改响应的结果数据
   6) …

## 6.2 过滤器过滤规则

### 6.2.1 默认只对请求过滤,对转发不过滤

### 6.2.2 修改规则

在<filter-mapping>中增加转发过滤设置

```
<dispatcher>REQUEST</dispatcher>
<dispatcher>FORWARD</dispatcher>
```

### 6.2.3 过滤顺序

按匹配规则过滤.

如果多个过滤器的匹配规则都匹配到这个路径,按照<filter-mapping>匹配顺序来执行过滤器

## 6.3 项目中解决乱码问题

### 6.3.1 POST 请求

字符编码过滤器只能解决 POST 请求乱码问题.不能解决 GET 请求乱码问题。

原理:request.setCharacterEncoding(this.encoding);

### 6. 3. 2 GET 请求

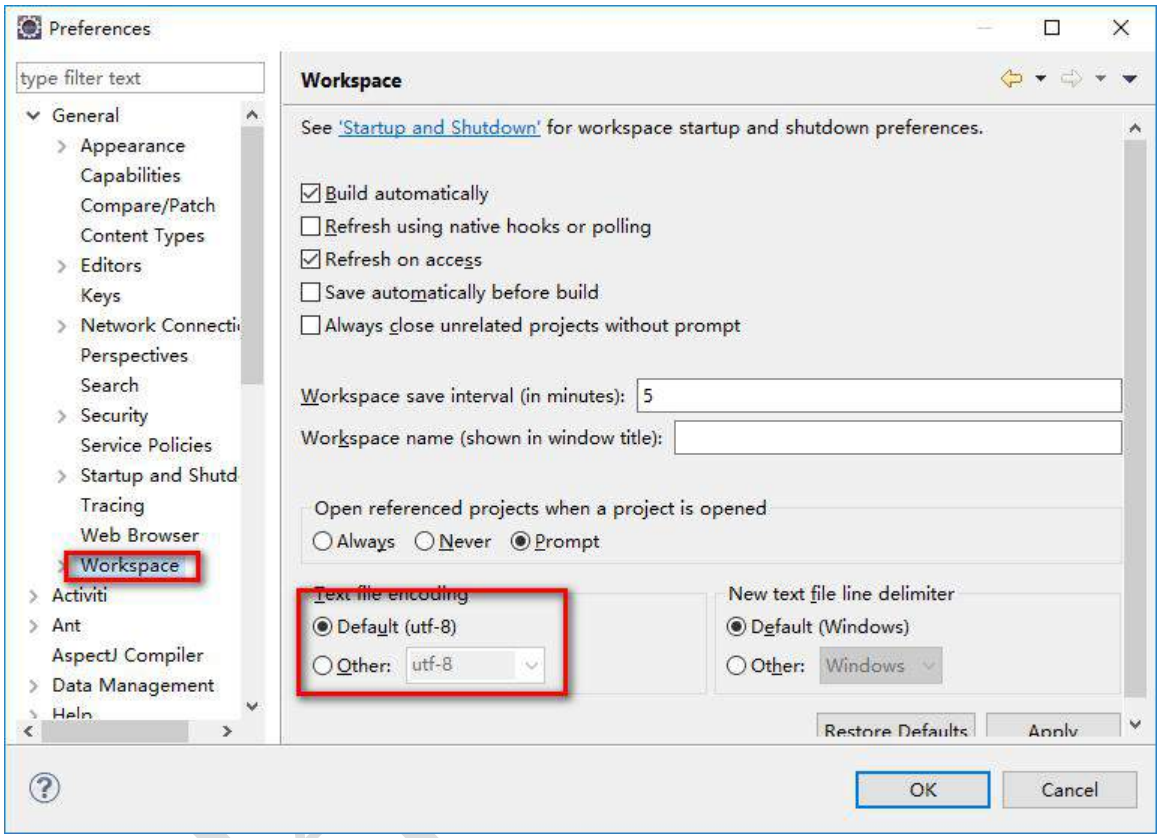
在 Tomcat/conf/server.xml 中设置编码。

<Connector URIEncoding="UTF-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1"

redirectPort="8443"/>

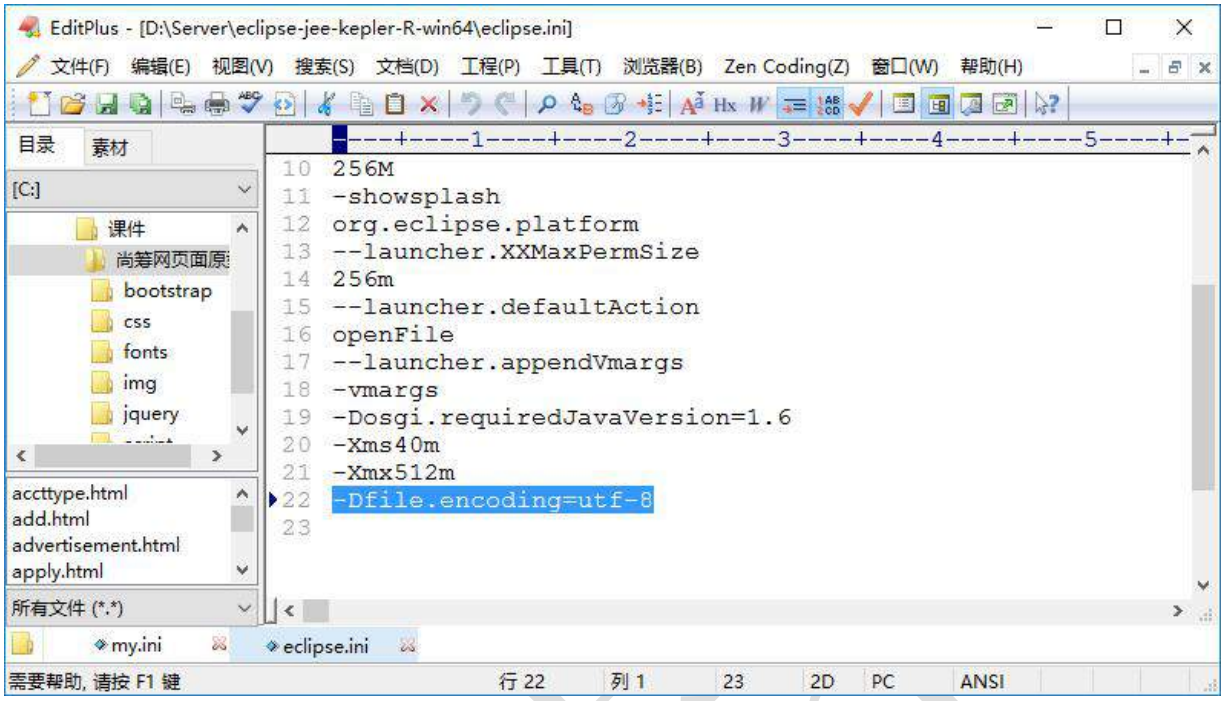
## 6.4 如何避免出现乱码

### 6.4.1 工作空间编码设置




*注:如果设置 Eclipse 初始化文件编码,所有工作空间字符编码与 Eclispe 所设置编码保持一致。*

*-Dfile.encoding=utf-8*




## 6.4.2 JSP 页面编码设置

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

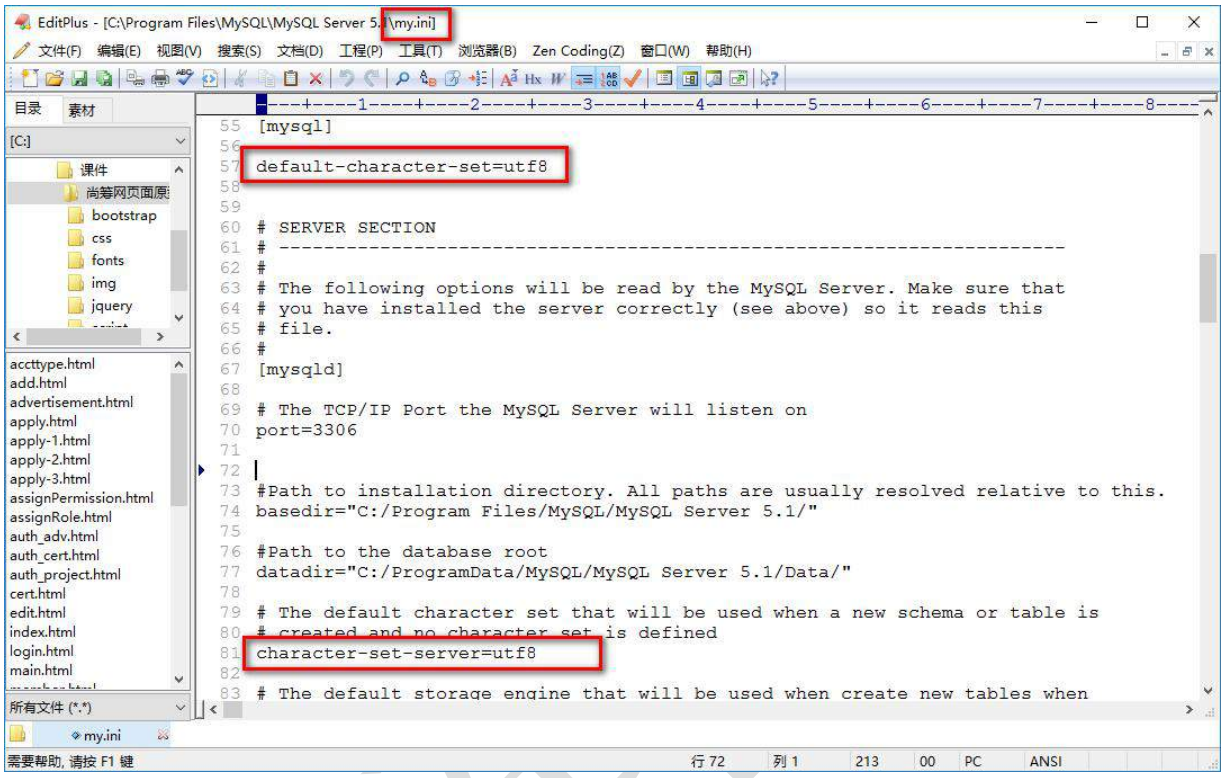
1. pageEncoding="UTF-8"

页面编码

2. contentType="text/html; charset=UTF-8"

服务器端响应编码,相当于:response.setCharacterEncoding("UTF-8");

## 6.4.3 数据库编码

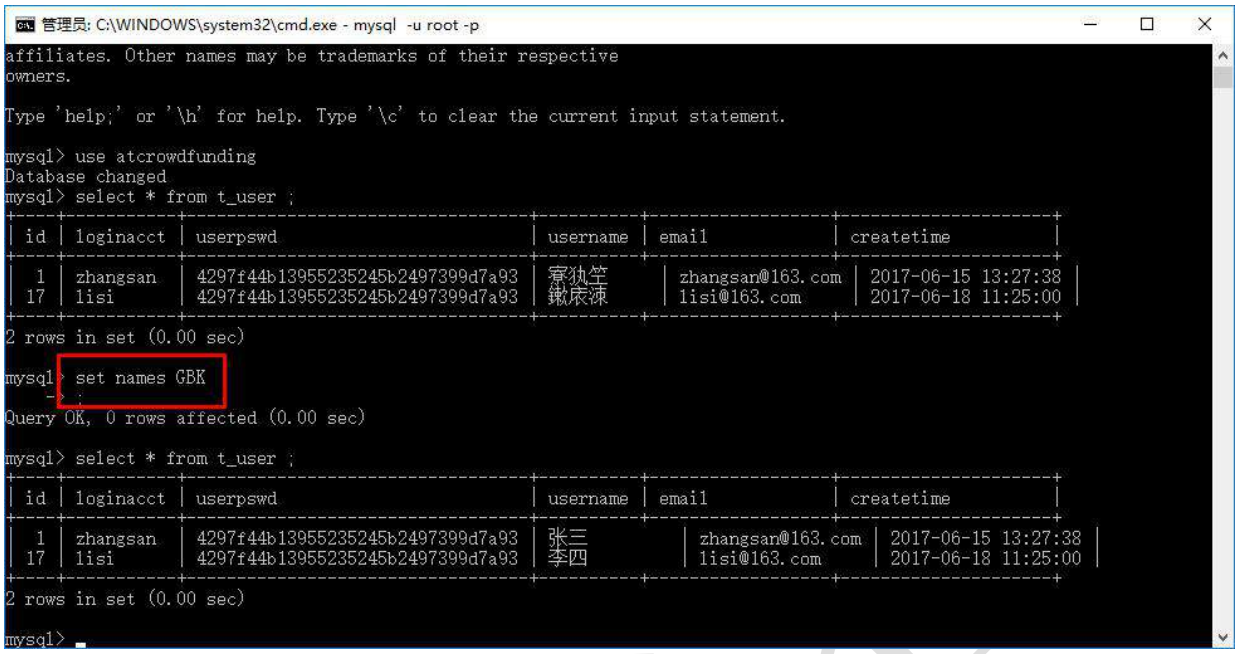



1. 修改数据库编码
   - 需重新启动服务器,否则配置不起作用
   - 新建数据库以及表采用新的编码. 以前创建的数据库还是采用以前的编码
2. 应用程序和数据库之间数据交互,是通过 url 指定中间编码进行字符编码转换的

jdbc:mysql://localhost:3306/atcrowdfunding170506?rewriteBatchedStatements=true&useUnicode=true&characterEncoding=utf8

3. 命令行查询数据显示乱码
   - 数据库采用 utf8 编码,而 DOS 窗口默认 GBK
   - 通过 set names GBK 将数据库数据转换为 GBK 编码
   - DOS 窗口显示数据就不再是乱码了

*注:这个设置是临时的,只针对于当前窗口有效*




### 6.4.4 Java 中字符串编码转换:

8. public byte[] getBytes(String charsetName)

字符串对象调用这个方法将字符串转换为二进制数组(将原来编码转换为指定的中间编码)

9. public String(byte bytes[], String charsetName)

将二进制数组数据再转换为字符串(将中间编码的数据再转换为指定的编码)

# 第 7 章 生产环境模拟

将本地开发环境和生产环境保持一致，可以提前发现用户在使用时所发生的问题。

## 7.1 80 端口的作用

用作 web 服务（https 443）的默认访问端口，Tomcat 服务器默认的端口为 8080 (安全协议端口 8443)，可以修改 server.xml 文件进行修改。

```
-->
<Connector URIEncoding="UTF-8" connectionTimeout="20000" port="80" protocol="HTTP/1.
<!-- A "Connector" using the shared thread pool-->
```

*注:80 端口可能被其他软件占用，使用时，必须将占用端口的软件关闭。*

## 7.2 Web 应用不通过 web 项目名称也可以访问？

修改.settings 目录下的配置文件 org.eclipse.wst.common.component

将项目名称更改为斜杠

```
<wb-resource deploy-path="/WEB-INF/classes" sour
<wb-resource deploy-path="/WEB-INF/classes" sour
<property name="context-root" value="/"/>
wb-module>
```

将项目清理一下,否则修改后可能不起作用.

对项目进行 maven 更新时,项目的上下文路径会恢复之前的状态

## 7.3 Web 应用通过域名访问

修改域名解析，通过指定的域名访问 web 应用，尽量模拟真实的生产环境。

修改 c:\Windows\System32\drivers\etc\hosts 文件就可以了

# 第 8 章 相对路径和绝对路径

## 8.1 绝对路径

- 不可改变的路径称之为绝对路径
- 本地路径：以盘符开始的路径称之为绝对路径, c:/test/test.html
- 网络路径：以协议，域名,端口号所组合的路径为绝对路径
- http://localhost:8080/atcrowdfunding/image/xxx.jpg
- http:// www.xxx.com/test/test.com

## 8.2 相对路径

- 可以改变的路径
- 默认的相对路径需要一个参考的位置，取值为当前资源的访问路径

## 8.3 路径以斜杠开头

路径以斜杠开头，也是相对路径，但是参考的位置和默认的相对路径不一样

- 前台路径(浏览器发起请求去加载资源的路径)（img， css， js， form）

参考的位置是服务器的根路径（ROOT）

response.sendRedirect(request.getContextPath()+"/xxx.jsp"); //前台路径

- 后台路径( java， xml )

参考的位置是 web 应用的根路径

request.getRequestDispatcher("/xxx.jsp").forward(req,resp); //后台路径

# 附录 1 导入表结构

参考数据库设计

# 作业

1. 了解需求
2. 搭建项目环境
3. 试着完成后台管理员登录功能

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

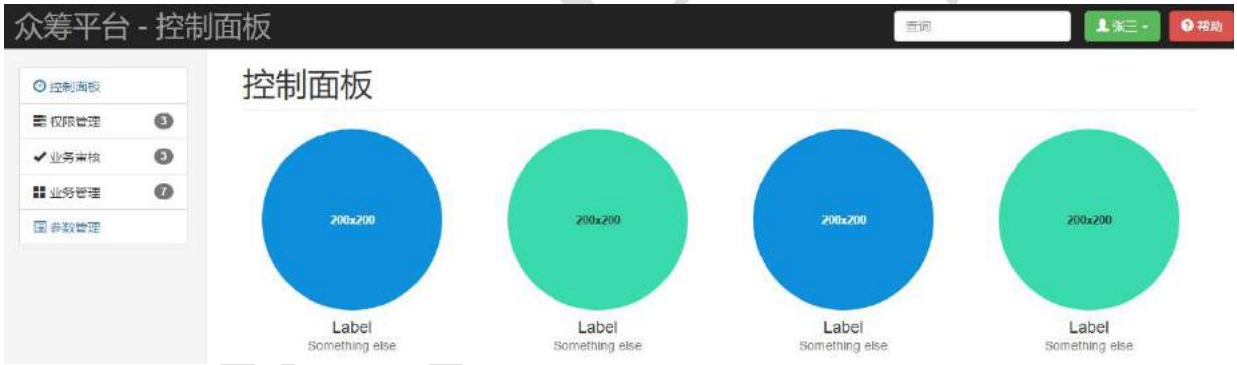
1. 完成登录功能
   - 同步请求方式
   - 异步请求方式
   - 密码加密
2. 注销功能

# 第 1 章 登录功能 分析与设计

## 1.1 首页


## 1.2 登录页面


## 1.3 管理员登录后主页面



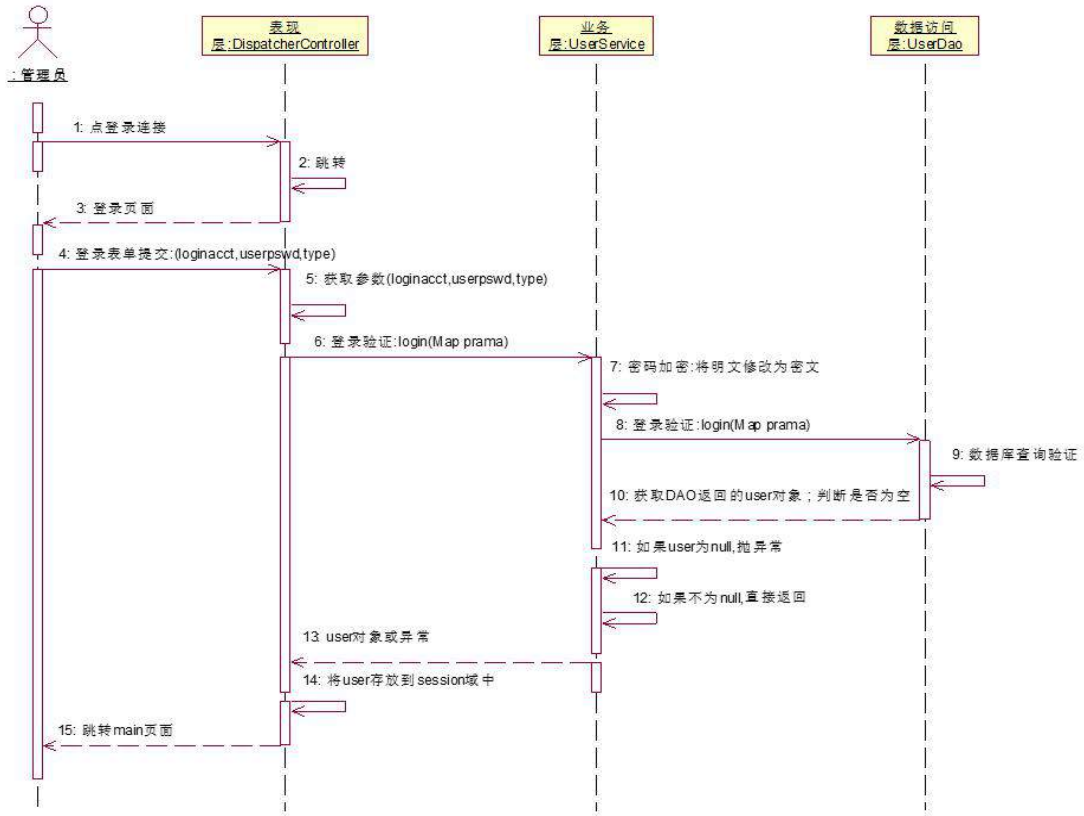

## 1.4 会员登录后主页面

## 1.5 登录涉及表

- t_user 管理员
- t_member 会员表

## 1.6 流程图

用户登录时序图



# 第 2 章 登录功能 开发

## 2.1 增加 index.jsp

## 2.2 增加跳转 login.jsp 页面控制器映射

## 2.3 修改登录页面 login.jsp 的表单,提交请求

## 2.4 处理登录请求

1. 处理器类(DispatcherController),处理请求方式,根据登录进行判断并跳转页面
2. 业务层接口(UserService)和实现类(UserServiceImpl)
3. DAO 接口(UserDao)和映射(mapper-user.xml)
4. 创建 User/Member 实体类

# 第 3 章 Bootstrap 前端框架

## 3.1 Bootstrap 介绍

1. Bootstrap 是基于 HTML5 和 CSS3 开发的前端框架，它简洁灵活，使得 Web 开发更加快捷兼容大部分 jQuery 插件
2. Bootstrap 中包含了丰富的 Web 组件，根据这些组件，可以快速搭建一个漂亮、功能完备的网站。
3. 其中包括以下组件:

下拉菜单、按钮组、按钮下拉菜单、导航、导航条、路径导航、分页、排版、缩略图、警告对话框、进度条、媒体对象等

4. Bootstrap 自带了 13 个 jQuery 插件，这些插件为 Bootstrap 中的组件赋予了“生命”。

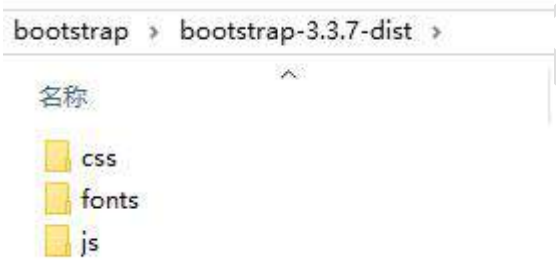
其中包括： 模式对话框、标签页、滚动条、弹出框等。

## 3.2 Bootstrap 如何使用

1. 在页面中引用 css,js 文件

```
<!-- 新 Bootstrap 核心 CSS 文件 -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap.min.css">

<!-- 可选的Bootstrap主题文件（一般不用引入） -->
<link rel="stylesheet" href="//cdn.bootcss.com/bootstrap/3.3.5/css/bootstrap-theme.min.css">

<!-- jQuery文件。务必在bootstrap.min.js 之前引入 -->
<script src="//cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>

<!-- 最新的 Bootstrap 核心 JavaScript 文件 -->
<script src="//cdn.bootcss.com/bootstrap/3.3.5/js/bootstrap.min.js"></script>
```

2. 将框架文件导入到项目中




*注:页面加载 js 存放在 body 最后位置,目的是为了展示页面时先展示 body 数据,,然后再加载 js;这样提高了页面显示效率，使得用户使用系统体验更好。*

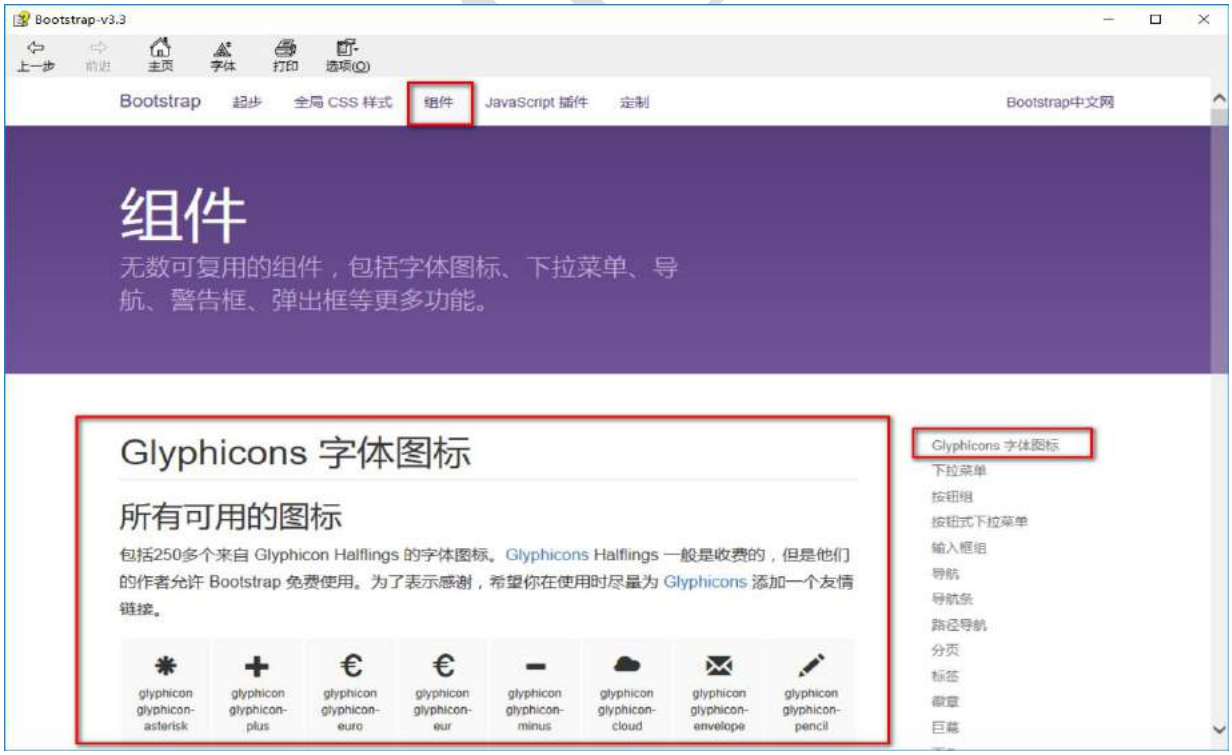
```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
    <title>Bootstrap 101 Template</title>

    <!-- Bootstrap -->
    <link href="css/bootstrap.min.css" rel="stylesheet">

    <!-- HTML5 shim and Respond.js for IE8 support of HTML5 elements and media queries -->
    <!-- WARNING: Respond.js doesn't work if you view the page via file:// -->
    <!--[if lt IE 9]>
      <script src="//cdn.bootcss.com/html5shiv/3.7.2/html5shiv.min.js"></script>
      <script src="//cdn.bootcss.com/respond.js/1.4.2/respond.min.js"></script>
    <![endif]-->
  </head>
  <body>
    <h1>你好，世界！</h1>

    <!-- jQuery (necessary for Bootstrap's JavaScript plugins) -->
    <script src="//cdn.bootcss.com/jquery/1.11.3/jquery.min.js"></script>
    <!-- Include all compiled plugins (below), or include individual files as needed -->
    <script src="js/bootstrap.min.js"></script>
  </body>
</html>
```

## 3.3 Bootstrap 字体图标

### 3.3.1 字体图标展示

## 3.3.2 操作系统字体


## 3.3.3 Bootstrap 字体图标



# 第 4 章 MD5 概述

## 4.1 加密算法

### 4.1.1 不可逆加密算法

常见算法：MD5， SHA-1 （Secure Hash Algorithm），SHA-2

Message Digest Algorithm MD5（中文名为消息摘要算法第五版）

散列函数，用以提供消息的完整性保护

MD5 算法具有以下特点：

1、压缩性：任意长度的数据，算出的 MD5 值长度都是固定的。(32 位长度字符串)

2、容易计算：从原数据计算出 MD5 值很容易。(不可逆算法)

3、抗修改性：对原数据进行任何改动，哪怕只修改 1 个字节，所得到的 MD5 值都有很大区别。(不易猜测到,不易破解)

4、强抗碰撞：已知原数据和其 MD5 值，想找到一个具有相同 MD5 值的数据（即伪造数据）是非常困难的

MD5 加密后的内容可以通过暴力破解（网站当中使用验证码，目的就是为了防止暴力破解）

### 4.1.2 可逆加密算法

常见算法：DES （Data Encryption Standard）， 3DES，AES （Advanced Encryption Standard）

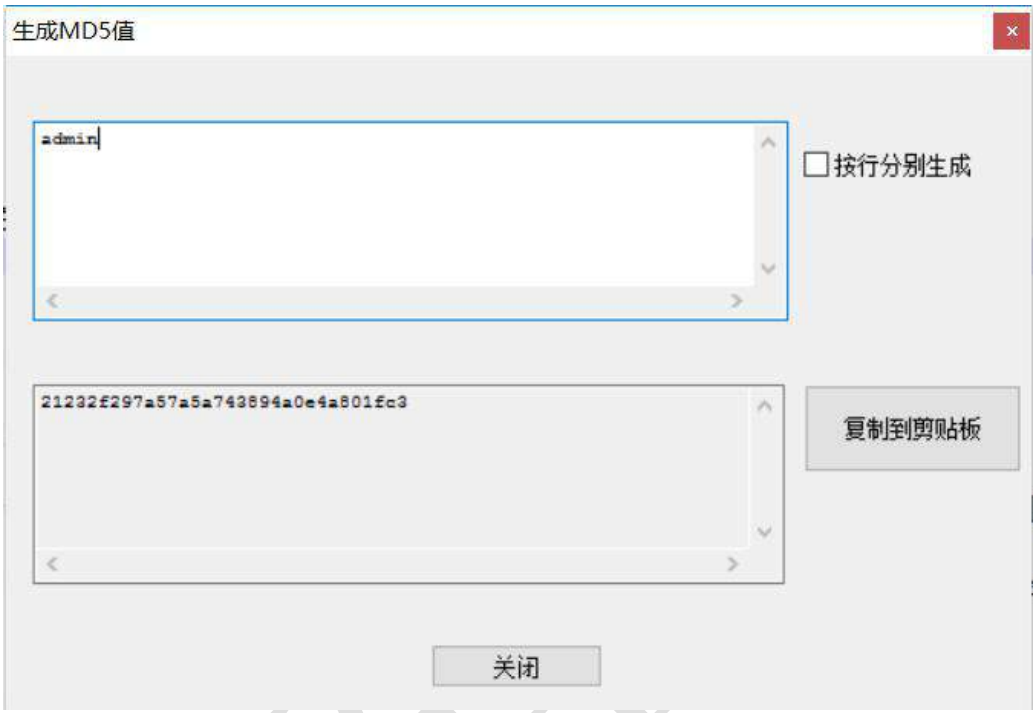
## 4.2 安全漏洞

2011 年 12 月 21 日，黑客在网上公开了知名程序员网站 CSDN 的用户数据库，高达 600 多万个明文的注册邮箱账号和密码遭到曝光和外泄，

成为中国互联网历史上一次具有深远意义的网络安全事故。

## 4.3 notepad++ 小工具

32 位长度字符串.




## 4.4 MD5 工具类

```
package com.atguigu.atcrowdfunding.util;


import java.security.MessageDigest;


/**
MD5 算法 哈希算法
MD5 算法具有以下特点：
1、压缩性：任意长度的数据，算出的 MD5 值长度都是固定的。
2、容易计算：从原数据计算出 MD5 值很容易。
3、抗修改性：对原数据进行任何改动，哪怕只修改 1 个字节，所得到的 MD5 值都有很大区别。
4、强抗碰撞：已知原数据和其 MD5 值，想找到一个具有相同 MD5 值的数据（即伪造数据）是非常
```

```
困难的。
 */
public class MD5Util {
    public static String digest16(String inStr) {
        return digest(inStr, 16);
    }
    public static String digest(String inStr) {
        return digest(inStr, 32);
    }
    private static String digest(String inStr, int rang) {
        MessageDigest md5 = null;
        if ( StringUtil.isEmpty(inStr) ) {
            return "";
        }

        try {
            md5 = MessageDigest.getInstance("MD5");    //取得算法
        } catch (Exception e) {
            e.printStackTrace();
            return "";
        }

        char[] charArray = inStr.toCharArray();
        byte[] byteArray = new byte[charArray.length];

        for (int i = 0; i < charArray.length; i++) {
            byteArray[i] = (byte) charArray[i];
```

```
        }

        byte[] md5Bytes = md5.digest(byteArray);    //加密

        StringBuilder hexValue = new StringBuilder();

        for (int i = 0; i < md5Bytes.length; i++) {
            int val = ((int) md5Bytes[i]) & 0xff;
            if (val < 16)
                hexValue.append("0");
            hexValue.append(Integer.toHexString(val));
        }
        if ( rang == 32 ) {
                return hexValue.toString();
        } else {
                return hexValue.toString().substring(8, 24);//转换为 32 位字符串
        }
    }
    public static void main(String args[]) {
        String s = new String("admin");
        System.out.println(digest(s));
    }
}
```

## 4.5 DES 工具类

```
package com.atguigu.atcrowdfunding.util;
```

```
import java.io.IOException;
import java.security.SecureRandom;

import javax.crypto.Cipher;
import javax.crypto.SecretKey;
import javax.crypto.SecretKeyFactory;
import javax.crypto.spec.DESKeySpec;

import sun.misc.BASE64Decoder;
import sun.misc.BASE64Encoder;

public class DesUtil {

private final static String DES = "DES";

    public static void main(String[] args) throws Exception {
        String data = "123 456";
        String key = "123222!@#$%";//相当于密码本
        System.err.println(encrypt(data, key));
        System.err.println(decrypt(encrypt(data, key), key));

    }

    /**
     * Description 根据键值进行加密
     * @param data
     * @param key  加密键 byte 数组
```

```
     * @return
     * @throws Exception
     */
    public static String encrypt(String data, String key) throws Exception {
            byte[] bt = encrypt(data.getBytes(), key.getBytes());
            String strs = new BASE64Encoder().encode(bt);
            return strs;
    }


    /**
     * Description 根据键值进行解密
     * @param data
     * @param key  加密键 byte 数组
     * @return
     * @throws IOException
     * @throws Exception
     */
    public static String decrypt(String data, String key) throws IOException,
                Exception {
            if (data == null)
                return null;
            BASE64Decoder decoder = new BASE64Decoder();
            byte[] buf = decoder.decodeBuffer(data);
            byte[] bt = decrypt(buf,key.getBytes());
            return new String(bt);
    }
```

```
/**
 * Description  根据键值进行加密
 * @param data
 * @param key   加密键 byte 数组
 * @return
 * @throws Exception
 */
private static byte[] encrypt(byte[] data, byte[] key) throws Exception {
    // 生成一个可信任的随机数源
    SecureRandom sr = new SecureRandom();

    // 从原始密钥数据创建 DESKeySpec 对象
    DESKeySpec dks = new DESKeySpec(key);

    // 创建一个密钥工厂，然后用它把 DESKeySpec 转换成 SecretKey 对象
    SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(DES);
    SecretKey securekey = keyFactory.generateSecret(dks);

    // Cipher 对象实际完成加密操作
    Cipher cipher = Cipher.getInstance(DES);

    // 用密钥初始化 Cipher 对象
    cipher.init(Cipher.ENCRYPT_MODE, securekey, sr);

    return cipher.doFinal(data);
}
```

```
    /**
     * Description  根据键值进行解密
     * @param data
     * @param key    加密键 byte 数组
     * @return
     * @throws Exception
     */
    private static byte[] decrypt(byte[] data, byte[] key) throws Exception {
        // 生成一个可信任的随机数源
        SecureRandom sr = new SecureRandom();

        // 从原始密钥数据创建 DESKeySpec 对象
        DESKeySpec dks = new DESKeySpec(key);

        // 创建一个密钥工厂，然后用它把 DESKeySpec 转换成 SecretKey 对象
        SecretKeyFactory keyFactory = SecretKeyFactory.getInstance(DES);
        SecretKey securekey = keyFactory.generateSecret(dks);

        // Cipher 对象实际完成解密操作
        Cipher cipher = Cipher.getInstance(DES);

        // 用密钥初始化 Cipher 对象
        cipher.init(Cipher.DECRYPT_MODE, securekey, sr);
        return cipher.doFinal(data);
    }
}
```

# 第 5 章 工具类

## 5.1 判断字符串是否为空

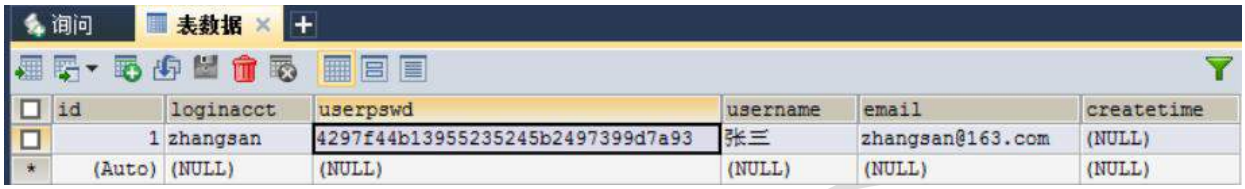
```
package com.atguigu.atcrowdfunding.util;

public class StringUtil {

    public static boolean isEmpty(String s) {

        //s == null | s.equals("");    //位与,逻辑与区别,非空字符串放置在前面,避免空指针

        return s == null || "".equals(s);

    }

    public static boolean isNotEmpty(String s) {

        return !isEmpty(s);

    }

}
```

## 5.2 常量类

```
package com.atguigu.atcrowdfunding.util;

public final class Const {

    public static final String LOGIN_LOGINACCT_ERROR = "用户名称不存在!" ;

    public static final String LOGIN_USERPSWD_ERROR = "用户密码错误!" ;

    public static final String LOGIN_USER = "loginUser";

    public static final String QUERY_DATA_ERROR = "查询列表失败!";

    public static final String SAVE_DATA_ERROR = "保存数据失败!";

    public static final String DEFAULT_USERPSWD = "123";

    public static final String UPDATE_DATA_ERROR = "修改数据失败!";

    public static final String DELETE_DATA_ERROR = "删除数据失败!";

}
```

# 第 6 章 登录功能 密码加密

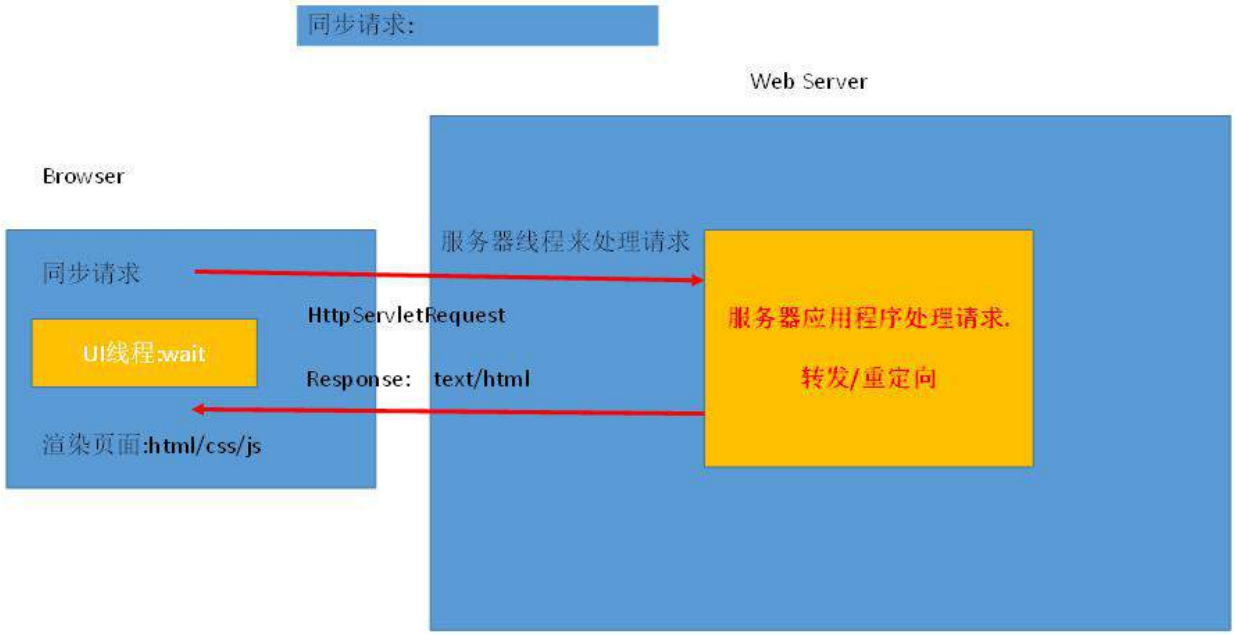
数据库存储密文,保证安全.




# 第 7 章 同步与异步开发原理

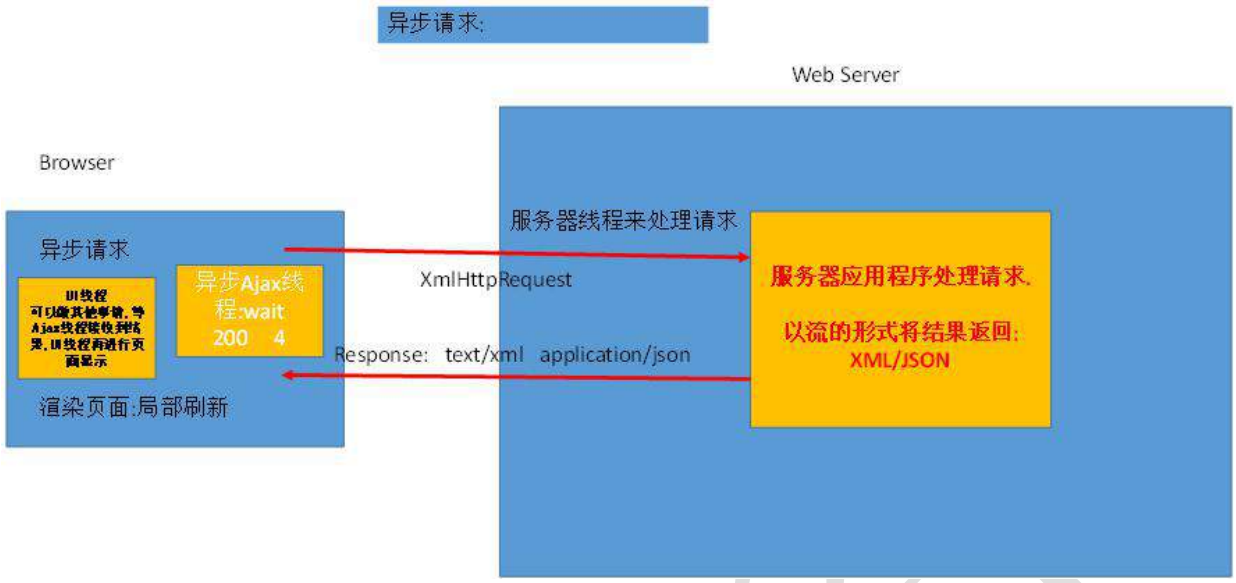
## 7.1 同步请求,页面会闪烁的原因

请求处理后页面整体刷新,所以会出现闪烁效果.




## 7.2 异步请求,解决页面闪烁问题

请求处理后,页面不需要整体刷新,而是,进行局部刷新。




# 第 8 章 登录功能 异步开发方式

## 8.1 回顾 jQuery 异步请求

1. $.post()
2. $.get()
3. $.getJson()
4. $.ajax()

jQuery 代码:

```
$.ajax({
    type: "POST",
    url: "some.php",
    data: "name=John&location=Boston",
    success: function(msg){
      alert( "Data Saved: " + msg );
    }
});
```

## 8.2 回顾 JSON 对象格式

- 表示一个对象:{key:value,key:value}

- 表示多个对象:[{key:value,key:value},{key:value,key:value}]
- JSON 格式中 key 必须采用单引号或双引号引起来.(服务器通过流的形式必须这样)
- response.getWriter().println("{'key':value,'key':value}");
- response.getWriter().println("{\"key\":value,\"key\":value}");

## 8.3 登录页面参考代码

```
<script>
    function dologin() {
        var floginacct = $("#floginacct");
        if($.trim(floginacct.val())==""){ //去掉前后两端空格
            alert("登录用户名称不允许为空!");
            floginacct.focus();
            return ;
        }
        var fuserpswd = $("#fuserpswd");
        if($.trim(fuserpswd.val())==""){
            alert("登录用户密码不允许为空!");
            fuserpswd.focus();
            return ;
        }
        $.ajax({
            type : "POST",   //$.ajax()函数中{}里的属性名称可以不用引号引起来.
            url : "${APP_PATH }/dologin.do",
            data : {
                "loginacct":floginacct.val(),
                "userpswd":fuserpswd.val(),
                "usertype":$("#fusertype").val()
```

```
            },
            beforeSend : function(){
                //一般是完成提交请求前的准备工作:例如表单数据校验.
                return true ; //继续发起 ajax 请求.
            },
             success : function(result){ //将服务器端返回的 JSON 格式字符串转换为 JSON,然后通过 JS 进行解析.
                //表示服务器端成功处理请求,并返回结果的处理
                if($("#fusertype").val()=="member"){
                    window.location.href="${APP_PATH}/index.jsp";
                }else{
                    if(result.success){
                        window.location.href="${APP_PATH}/main.htm";
                    }else{
                        alert(result.errorMessage);
                    }
                }
            },
            error : function(){
                //表示服务器端处理请求失败,执行相关操作
                alert("登录失败");
            }
        });
    }
</script>
```

## 8.4 响应内容类型和字符编码配置

```
<!-- 字符串字符编码转换 -->
<bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" >
        <property name="messageConverters">
         <list>
              <bean
class="org.springframework.http.converter.json.MappingJacksonHttpMessageConverter">
                   <property name="supportedMediaTypes">
                        <list>
                             <value>application/json;charset=UTF-8</value>
                        </list>
                   </property>
              </bean>
         </list>
    </property>
</bean>
```

## 8.5 处理异步登录请求

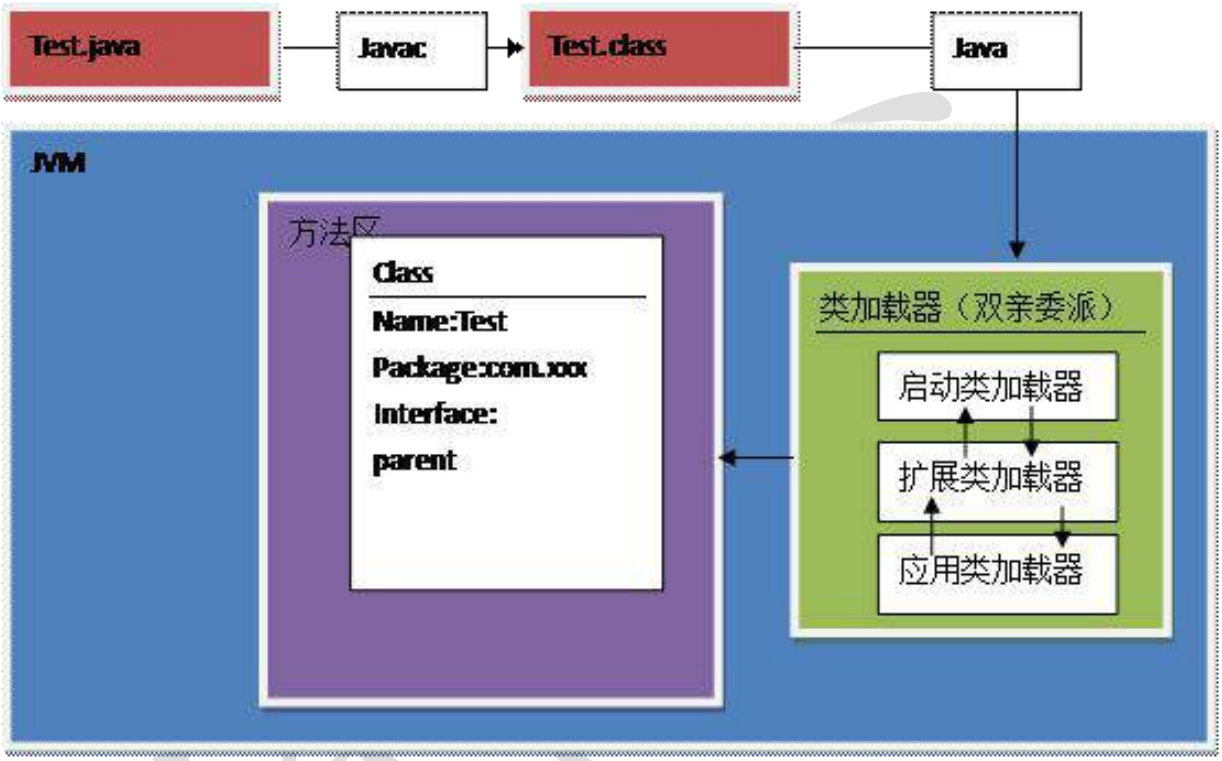
在 DispatcherController 类中增加映射方法

```
//处理异步请求登录
//@ResponseBody 表示采用框架底层的 HttpMessageConverter 进行内容类型转换
//如果引用了 Jackson 组件,返回实体对象或集合,框架就会将对象或集合转换为 JSON 格式字符串,然后以流的形式将字符串响应给客户端.
@ResponseBody
@RequestMapping("/dologin")
public Object dologin(String loginacct , String userpswd , String usertype , Map map){
```

```
    Map<String,Object> map = new HashMap<String,Object>();


    Map<String,Object> paramMap = new HashMap<String,Object>();
    paramMap.put("loginacct", loginacct);
    paramMap.put("userpswd", userpswd);


    User loginuser = userService.queryUserForLogin(paramMap);
    if(loginuser==null){
        map.put("message", "用户名称或密码不正确,登录失败!");
        map.put("success",false);
    }else{
        map.put("success",true);
    }
    return map ;
}
```

# 附录 1.JVM

## 1.1 JVM 加载类规则-双亲委派机制




## 1.2 演示类加载器,双亲委派加载机制

①定义 Test 类,main 方法打印 1111

编译后放置在启动类加载器加载位置:

在 jre 目录下建立 classes 目录,存放到 classes 目录下


②定义 Test 类,main 方法打印 2222

编译后放置在扩展类加载器加载位置:

在 jre/lib/ext 下建立 classes 目录,存放到 classes 目录下


③定义 Test 类,main 方法打印 3333

编译后放置在应用类加载器加载位置:

存放到任意目录下,例如 D:根目录下

# 附录 2.线程安全问题

## 2.1 文件冲突

采用悲观锁和乐观锁机制解决文件冲突;

例如:

VSS 微软提供的版本控制工具,采用就是悲观锁方式.

表示一个线程在访问文件时,其他线程不允许访问.所以,解决线程安全问题.

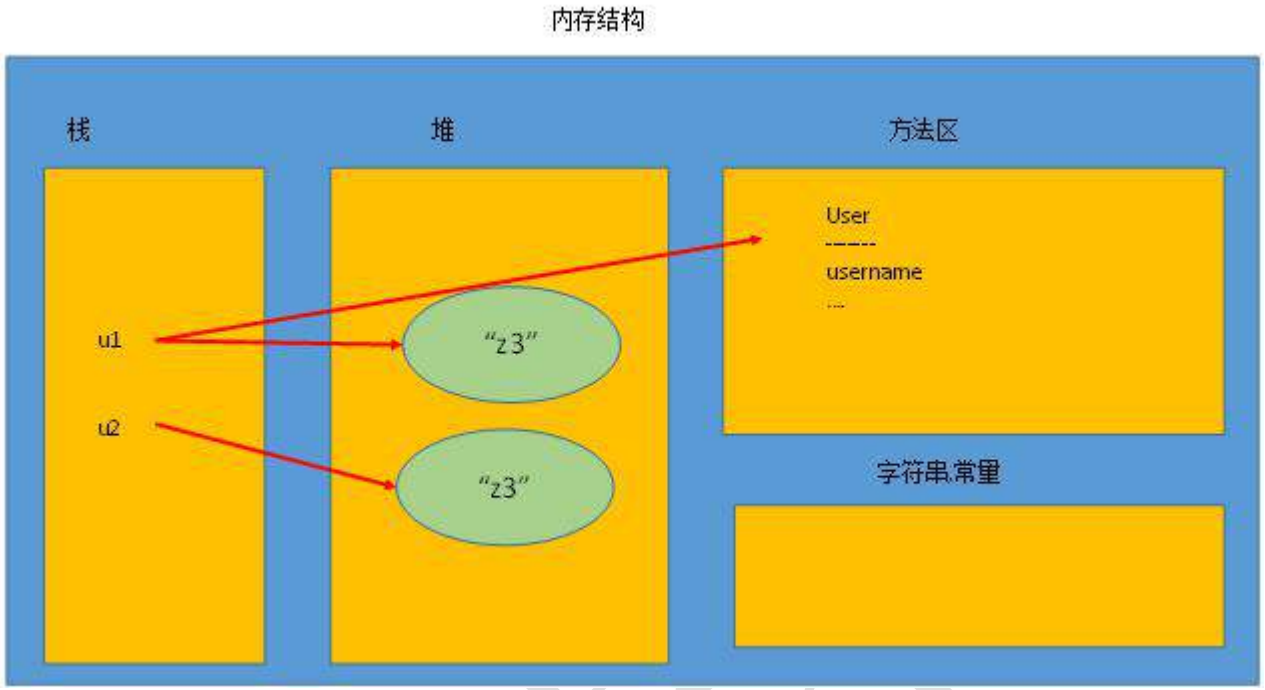
但是开发过程中,多人维护同一个文件时,工作效率低.所以,一般情况下,一个文件只分配给一个人维护.

CVS/SVN 开源的,采用乐观锁的方式维护文件的安全.

在管理文件时,给文件增加版本号,每一次修改版本号都会自动加 1.后提交修改操作的线程不允许提交.必须先更新,然后修改,然后再提交。

## 2.2 数据冲突

JVM 数据是如何存储的? 内存结构:




## 2.2.1 数据冲突原因

两个引用指向同一个内存地址对象,一个引用修改后,另一个引用的数据也修改了.

## 2.2.2 解决冲突办法

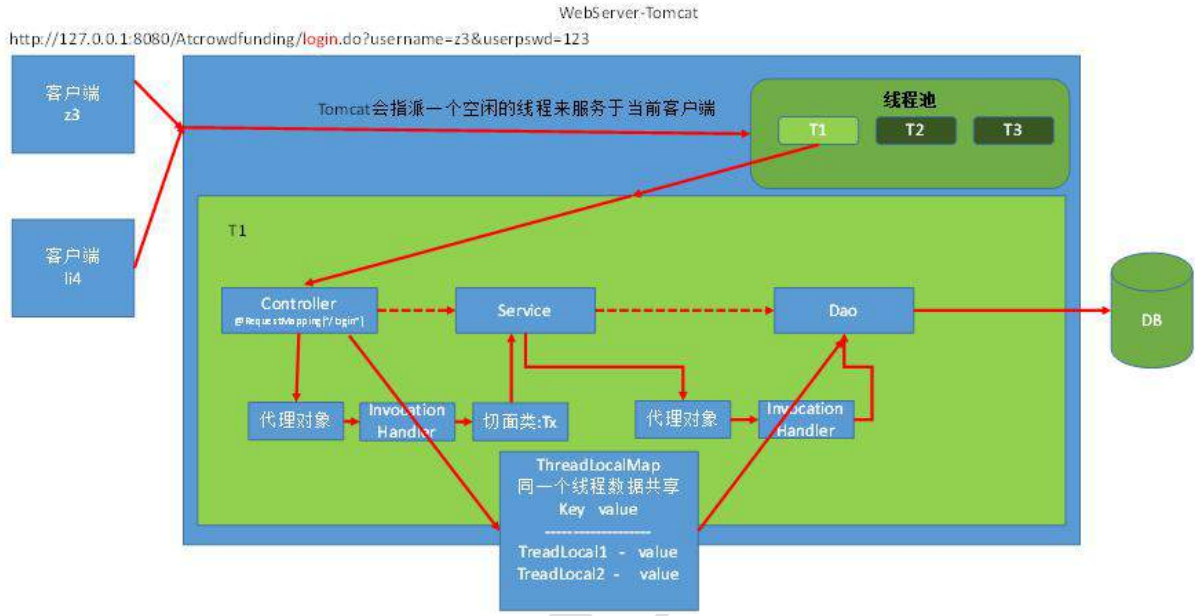
①多例创建对象

每一个线程单独操作一个内存地址对象,这样就不涉及资源共享的问题了.

②多线程并发对同一个对象进行访问,可以采用线程同步机制. (加锁)

- public synchronized V put(K key, V value) {}
- 在方法内同步代码块
  - synchronized (this){}

## 2.3 线程处理请求的流程




### 2.3.1 ThreadLocal

Thread 类定义了成员变量:ThreadLocal.ThreadLocalMap threadLocals = null; //相当于 Map 集合

| Key | Value |
| --- | --- |
| ThreadLocal1 | Connection |
| ThreadLocal2 | i |

- 一个线程对应一个 ThreadLocal.ThreadLocalMap;
- 一个 ThreadLocal 可以帮定一个数据.
  - 如何绑定数据(例如在 Controller 进行数据绑定)
    - ThreadLocal<Connection> theadLoal = new ThreadLocal<Connection>();
    - theadLoal .set(con);
  - 如何获取数据(例如:在 DAO 中进行数据获取)
    - Connection con = theadLoal .get();
  - 如何删除数据
    - threadLocal.remove();

# 附录 3.页面头信息

1.页面：login.jsp

<meta http-equiv="Refresh" content="3;URL=${APP_PATH }/index.htm">

# 附录 4.改善 handler 代码-ThreadLocal

## 4.1 封装父类代码

```
public class BaseController {
    //不能使用成员变量，因为控制器对象是单例的。会出现多线程并发问题
    // private Map<String, Object> resultMap;
    // 通过 ThreadLocal 来共享数据
    private ThreadLocal<Map<String, Object>> datas = new ThreadLocal<Map<String, Object>>();

    protected void start() {
        Map<String, Object> resultMap = new HashMap<String, Object>();
        datas.set(resultMap);
    }

    public Object end() {
        Map<String, Object> resultMap = datas.get();
        datas.remove();
        return resultMap;
    }

    public void success(boolean flg) {
```

```
        Map<String, Object> resultMap = datas.get();
        resultMap.put("success", flg);
    }

    public void data(Object val) {
        Map<String, Object> resultMap = datas.get();
        resultMap.put("data", val);
    }

    public void message(String msg) {
        Map<String, Object> resultMap = datas.get();
        resultMap.put("message", msg);
    }
}
```

## 4.2 子类使用父类封装的代码

```
@ResponseBody
@RequestMapping("/delete")
public Object delete(Integer id){
    start();
    try {
        permissionService.deletePermission(id);
        //data(list);
        success(true);
    } catch (Exception e) {
        e.printStackTrace();
        success(false);
```

```
            message(e.getMessage());
        }
        return end();
    }
```

# 作业

- 建立项目结构,项目模块,使用 Maven,SVN 进行管理
- 模拟真实的开发环境
- 完成系统登录功能(异步方式)
- 完成系统注销功能
- 完成项目中所有模块的页面流程,不需要实现业务逻辑
- 将 html 页面修改为 jsp
- 创建控制器类,业务接口,业务实现类,DAO 接口,DAO 映射配置等文件
- 分析表结构
- 分析及讨论相关业务功能

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

1.用户管理模块

- 分页查询功能
  - 同步请求方式
  - 异步请求方式
- 模糊查询
- 添加功能
- 修改功能
  - 重置功能
  - 修改后回到当前页
- 删除功能
  - 删除单条数据
  - 删除多条数据(批量删除)

# 第一章 layer 弹层组件

## 1.1 弹出框不统一

不同的浏览器显示的提示框样子不同,不统一.


## 1.2 专用弹层及使用

layer-v3.0.1.zip

### 1.2.1 提示

layer.msg(提示信息, {time:1000, icon:5, shift:6}, 回调方法); //弹出时间，图标，特效

layer.alert(提示信息, function(index){

// 回调方法

layer.close(index);

});

### 1.2.2 询问

layer.confirm("询问信息", {icon: 3, title:'提示'}, function(cindex){

```
    layer.close(cindex);
}, function(cindex){
    layer.close(cindex);
});
```

### 1.2.3 加载

```
var loadingIndex = layer.msg('处理中', {icon: 16});
...
layer.close(loadingIndex);

var index = layer.load(2, {time: 10*1000});
layer.close(index);
```

### 1.2.4 小图标

time:1000 显示时间

icon:5 图标

shift:6 抖动效果

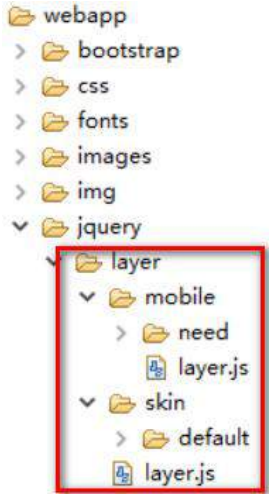
## 1.2.5 icon 图标


atcrowdfunding › docs › 插件 › layer-v3.0.1 › layer › skin › default
icon.png
loading-1.gif


http://www.atcrowdfunding.com/dologin.do
login.jsp
springmvc-context.xml
error.jsp
http://www.atcrowdfunding.com/dologin.do
尚筹网-创意产品众筹平台
用户名或密码不正确!
用户登录
请输入登录账号
用户名称不能为空!请重新输入!
管理
记住我
忘记密码
我要注册
登录

## 1.2.6 使用 layer 弹层完成提示消息

2.6.1 拷贝组件到项目中




2.6.2 在页面中引用组件/WEB-INF/jsp/login.jsp

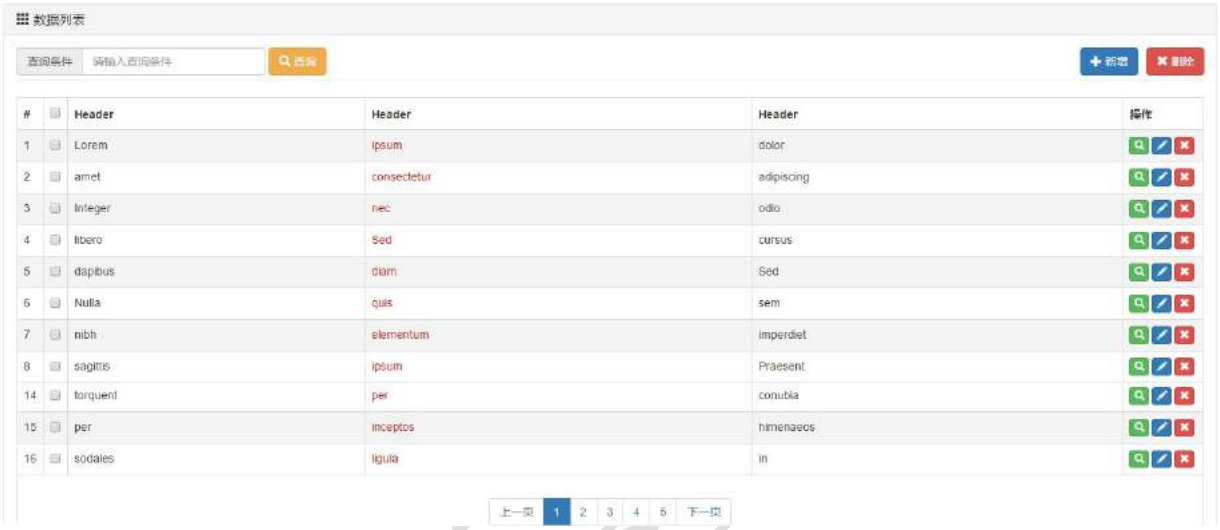
```
<script src="${APP_PATH}/jquery/layer/layer.js"></script>
```

2.6.3 代码参考

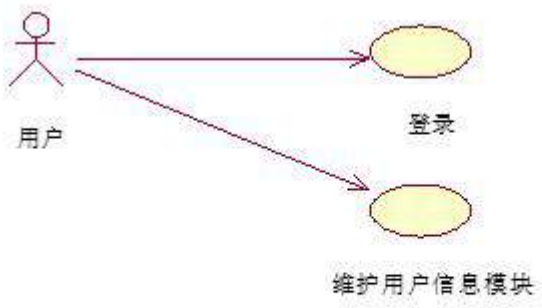
```
if(loginacct.val() == ""){
        //alert("登录账号不能为空,请重新输入!");//alert,confirm 方法会暂停 UI 线程
        layer.msg("登录账号不能为空,请重新输入!", {time:2000, icon:5, shift:6}, function(){
            loginacct.focus();
            //return ; 只是返回当前回调函数，单击事件函数并没有返回，代码继续往下执行
        }); //弹出时间，图标，特效
        return ;//单击事件函数返回
}
```

# 第二章 用户分页查询分析

## 2.1 列表




## 2.2 用例图



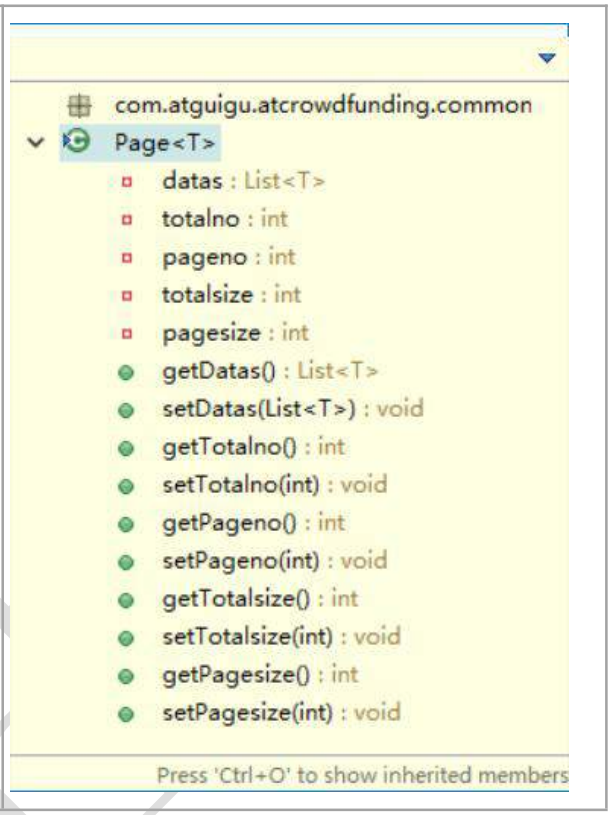

## 2.3 分页的语法

limit ?,? 两个问号:第一个问号表示开始索引,第二个问号表示查询条数.

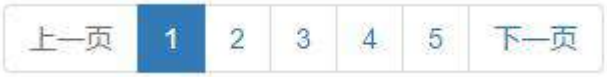
limit ? 一个问号:如果开始索引从 0 开始,可以省略开始索引;剩余的问号表示查询条数.

## 2.4 Page 类-封装分页信息

```
public class Page<T> {
    private List<T> datas;
    private int totalno;
    private int pageno;
    private int totalsize;
    private int pagesize;
}
```




## 2.5 用户分页查询-显示分页导航条




## 2.6 分析-异步请求方式

### 2.6.1 同步查询

如果数据量大的场合，用户等待的时间会变长，所以用户体验效果会非常的差

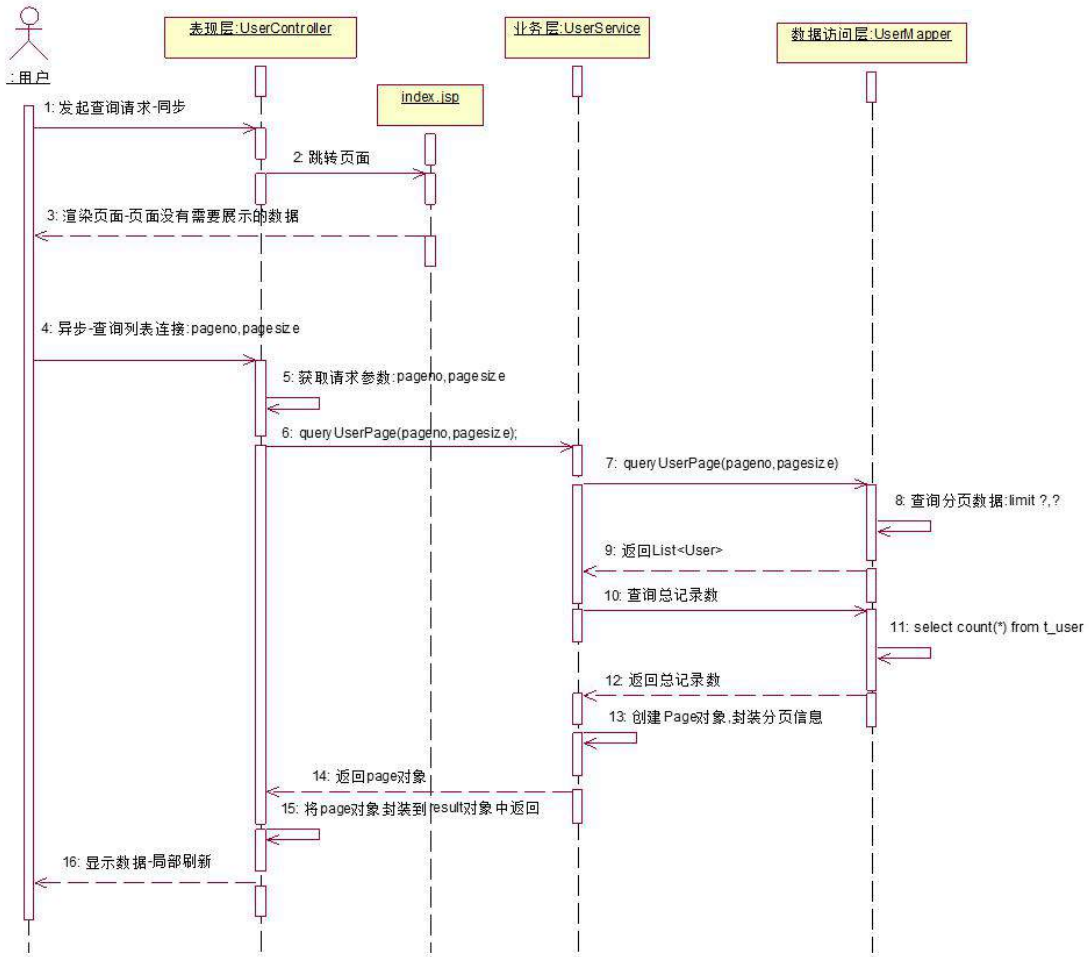
### 2.6.2 异步查询

用户希望先看到页面，然后将查询数据的加载效果提示出来

## 2.7 异步请求的流程图

①发起同步连接请求,跳转列表页面(无数据展示,只是静态页面展示)

②发起异步 ajax 请求,加载数据,局部刷新页面




## 2.8 分页方式

### 2.8.1 物理分页

根据 pageno 和 pagesize,每次都通过数据进行加载数据.

相当于延迟加载操作.

### 2.8.2 逻辑分页

表示将数据库所有数据查询出来存放到内存中,然后,根据分页信息从内存中获取每一页数据.

相当于立即加载操作.

# 第三章 用户分页异步查询

## 3.1 请求连接

## 3.2 定义控制器

## 3.3 跳转用户列表页面

不需要显示任何数据

### 3.4 异步发起请求

### 3.5 定义控制器

### 3.6 定义业务层

### 3.7 定义 DAO

### 3.8 定义 Mapper

### 3.9 页面异步刷新

# 第四章 用户分页查询-异步-条件查询

## 4.1 Mapper 动态 SQL 参考

```
<select id="queryCount" resultType="int">
    select count(*) from t_user
    <where>
        <if test="queryText!=null">
        loginacct like concat('%',#{queryText},'%')
        </if>
    </where>
</select>
<select id="pageQuery" resultType="User">
    select * from t_user
     <where>
        <if test="queryText!=null">
```

```
            loginacct like concat('%',#{queryText},'%')
            </if>
        </where>
        limit #{start}, #{size}
    </select>
```

## 4.2 模糊查询注意事项

**1.动态查询语句**

**2.SQL 中占位符不能在单引号中，否则，会以?进行查询数据**

'%#{param}%'

'%?%'

**3.SQL 中不能使用加号进行字符串拼接，加号是用来做运算的**

'%'+'D'+'%'

**4.MyBatis 进行拼串,拼串会出现 SQL 注入情况 ,例如：“or 1=1”**

'%${param}%'

**5.使用内置方法进行拼串**

concat('%',#{param},'%')

**6.查询条件值本身为%,查询出所有的数据**

concat('%',#{param},'%') => '%%%'

'%\%%' 使用转译字符再进行查询。

**7.#和\是一个意思，表示转译。使用#代替**

select * from t_user where username like '%#%%' escape '#'

select * from t_user where loginacct like '%@%%' escape '@'

SELECT * FROM t_user WHERE loginacct LIKE concat('%','@%','%') ESCAPE '@'

**8.常见的 SQL 文,在 Oracle 中，使用两个竖线用来表示字符串拼接，MySQL 中没有这样的语法。**

select * from t_user where username like '%'|| #{param} ||'%'

**9.SQL 参数问题**

```
<select id="queryCount" resultType="int">
    select count(*) from t_user
    <where>
        <if test="queryText!=null">
        loginacct like '%#{queryText}%'
        </if>
    </where>
</select>

<select id="pageQuery" resultType="User">
    select * from t_user
     <where>
        <if test="queryText!=null">
        loginacct like '%#{queryText}%'
        </if>
    </where>
    limit #{start}, #{size}
</select>
```

```
Parameter index out of range (3 > number of parameters, which is 2).
```

有 3 个参数,但是只是指定了 2 个.

Select * from t_user where loginacct like '%#{loginacct}%' limit?,?

```
select * from t_user          WHERE  loginacct like '%?%'          limit ?, ?
```

org.springframework.dao.TransientDataAccessResourceException:

### Error querying database. Cause: java.sql.SQLException: **Parameter index out of range (3 > number of parameters, which is 2).**

### The error may exist in URL

```
[jar:file:/F:/atcrowdfunding/workspace/.metadata/.plugins/org.eclipse.wst.server.core/tmp0/wtpwebapps/atcrowdfunding-main/WEB-INF/lib/atcrowdfunding-user-0.0.1-SNAPSHOT.jar!/mybatis/mapper-user.xml]
### The error may involve defaultParameterMap
### The error occurred while setting parameters
### SQL: select * from t_user                WHERE loginacct like '%?%'                limit ?, ?
### Cause: java.sql.SQLException: Parameter index out of range (3 > number of parameters, which is 2).
; SQL []; Parameter index out of range (3 > number of parameters, which is 2).; nested exception is java.sql.SQLException: Parameter index out of range (3 > number of parameters, which is 2).
```

SQL 注入问题 ：

Id = 100 OR 1=1

SELECT * FROM t_user WHERE id= ${id}

SELECT * FROM t_user WHERE id=  100 OR 1=1

在特定场合可以使用${}:

例如:

Create table ${tableName} …  //表名称位置不能使用?占位符,所以也就不能使用#{}

Order by ${fieldName} asc  //对字段进行排序,可以 传递动态字段名称.

```
<select id="queryCount" resultType="int">
    select count(*) from t_user
    <where>
        <if test="queryText!=null">
        loginacct like '%${queryText}%'
        </if>
    </where>
</select>

<select id="pageQuery" resultType="User">
```

```
        select * from t_user
          <where>
              <if test="queryText!=null">
              loginacct like '%${queryText}%'
              </if>
        </where>
        limit #{start}, #{size}
    </select>
```

**SQL 拼接问题**

不能使用加号拼接

```
3   SELECT * FROM t_user WHERE loginacct LIKE '%'+'zhangsan'+'%'
```

1 条信息

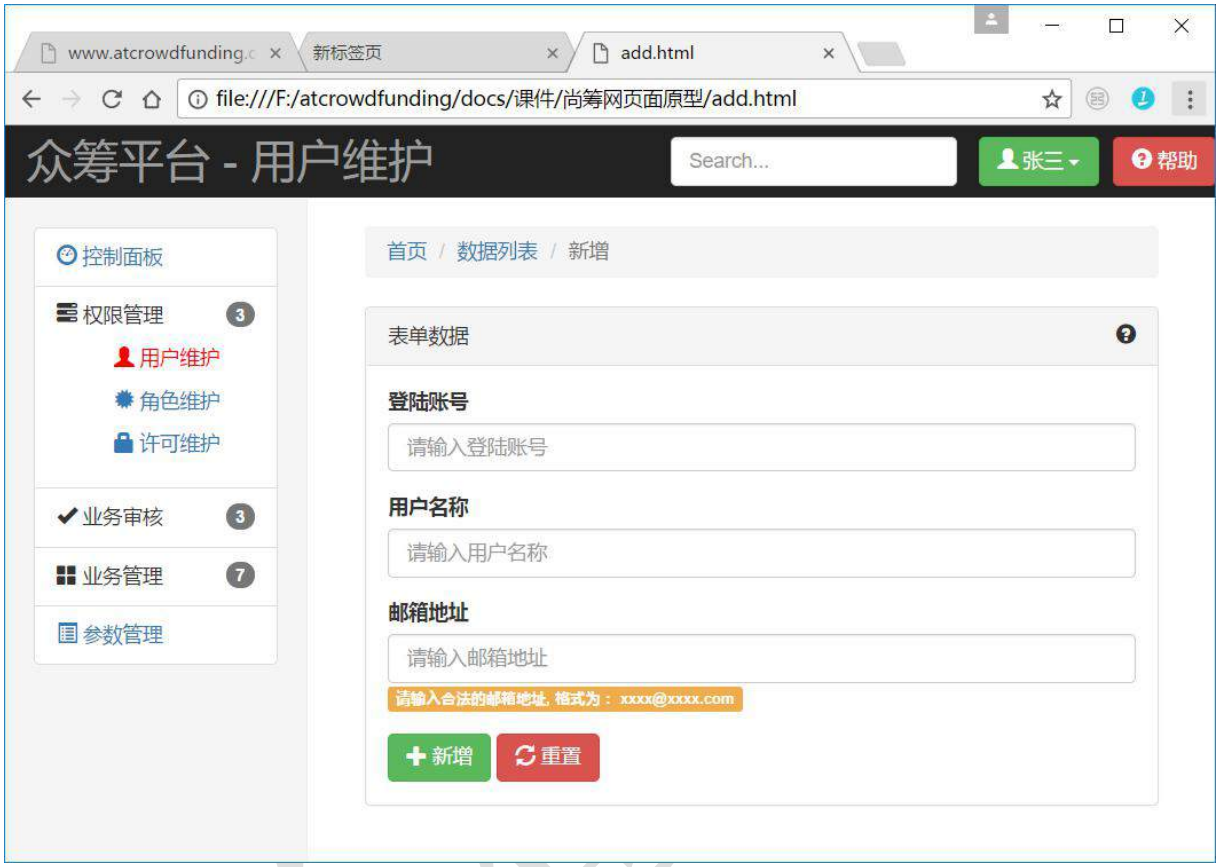
```
查询: SELECT * FROM t_user WHERE loginacct like '%'+'zhangsan'+'%' LIMIT 0, 1000错误代码: 1064
You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near
'+'zhangsan'+'%'
 LIMIT 0, 1000' at line 1

执行耗时   : 0 sec
传送时间   : 0 sec
总耗时     : 0.058 sec
```

# 第五章 用户添加功能

## 5.1 页面分析

### 5.1.1 列表页面点击添加按钮

### 5.1.2 添加用户表单




## 5.2 用户保存

### 5.2.1 修改表单页面

```
<form id="insertForm" role="form">
    <div class="form-group">
<label for="exampleInputPassword1">登录账号</label>
<input id="floginacct" type="text" class="form-control" placeholder="登录账号">
    </div>
    <div class="form-group">
<label for="exampleInputPassword1">用户名称</label>
```

```
<input id="fusername" type="text" class="form-control" placeholder="用户名称">
  </div>
  <div class="form-group">
<label for="exampleInputEmail1">邮箱地址</label>
<input id="femail" type="text" class="form-control"  placeholder="Enter email">
<p class="help-block label label-warning">请输入合法的邮箱地址，格式为： xxxx@xxxx.com</p>
  </div>
  <button id="insertBtn" type="button" class="btn btn-success"><i class="glyphicon glyphicon-plus"></i> 新增</button>
  <button type="button" class="btn btn-danger"><i class="glyphicon glyphicon-refresh"></i> 重置</button>
</form>
```

## 5.2.2 异步提交请求

```
$(function(){
    $("#insertBtn").click(function(){
        var loadingIndex = -1 ;
        $.ajax({
            url : "${APP_PATH}/user/doAdd.do",
            type : "POST",
            data : {
                loginacct : $("#loginacct").val(),
                username : $("#username").val(),
                email : $("#email").val()
            },
            beforeSend : function(){
                loadingIndex = layer.msg('数据正在保存中', {icon: 6});
```

```
                return true ;
            },
            success : function(result){
                layer.close(loadingIndex);
                if(result.success){
                    layer.msg("用户数据保存成功", {time:1000, icon:6});
                    window.location.href="${APP_PATH}/user/index.htm";
                }else{
                    layer.msg("用户数据保存失败", {time:1000, icon:5, shift:6});
                }
            },
            error : function(){
                layer.msg("用户数据保存失败", {time:2000, icon:5, shift:6});
            }
        });
    });
});
```

### 5.2.3 定义控制器

```
@RequestMapping("/add")
public String add() {
    return "user/add";
}
```

```
//保存用户
@ResponseBody
@RequestMapping(value="/doAdd")
```

```
public Object doAdd(User user){
    Map map= new HashMap ();
    try {
        int count = userService.saveUser(user);
        map.put ("success",count==1);
    } catch (Exception e) {
        e.printStackTrace();
        map.put("message","保存失败!");
        map.put ("success",false);
    }
    return result;
}
```

### 5.2.4 定义业务层

```
void saveUser(User user);
```

```
@Override
public int saveUser(User user) {
    user.setUserpswd(MD5Util.digest(Const.PASSWORD));
    user.setCreatetime(DateStringUtil.dateToString(new Date()));
    return userDao.insert(user);
}
```

### 5.2.5 定义 DAO

```
void insert(User user);
```

## 5.2.6 定义 Mapper

```
<insert id="insert" parameterType="user" useGeneratedKeys="true" keyProperty="id">
insert into t_user (
        loginacct,username,userpswd,email,createtime
    ) values (
        #{loginacct},#{username},#{userpswd},#{email},#{createtime}
    )
<!--
<selectKey keyProperty="id" resultType="int">
    select @@identity as id
</selectKey>
-->
</insert>
```

## 5.2.7 修改 Mapper 增加排序

```
<!-- 带查询条件分页
limit 在查询语句最后使用
 -->
<select id="queryPage" parameterType="map" resultType="User">
select id,loginacct,userpswd,username,email,createtime from t_user
<where>
    <if test="queryText!=null">
        loginacct like concat('%',#{queryText},'%')
    </if>
</where>
order by createtime desc
```

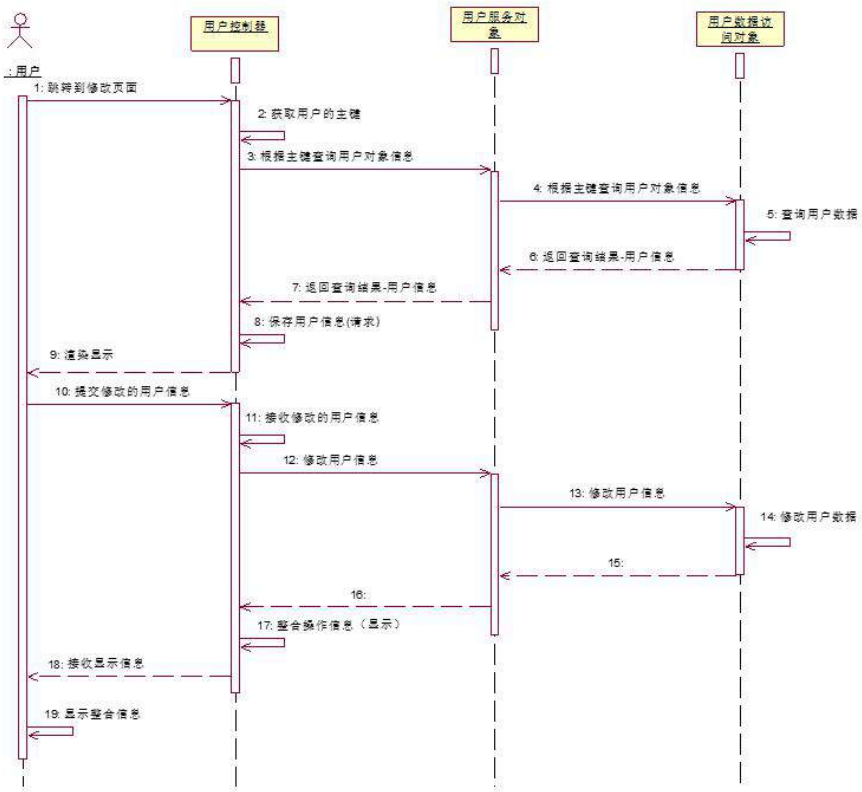
```
limit #{startIndex},#{pagesize}
</select>
```

- limit 必须在最后。
- order 必须放在查询条件之后
- 数据排序
  - 增加创建数据记录的时间戳:char(19)
  - yyyy-MM-dd HH:mm:ss

| | |
|---|---|
| 2017-02-09 12:12:12 | 固定长度 19 |
| 2017-02-09 | 固定长度 10 |

# 第六章 用户修改

## 6.1 UML 修改流程图




## 6.2 修改功能实现

①修改 list.jsp 连接

②处理请求，表单回显

③创建 edit.jsp，回显数据；修改后提交表单

④处理更新操作

## 6.3 重置功能

1.重置

```
$("#resetBtn").click(function(){
$("#updateForm")[0].reset();    //将 jQuery 对象转换为 dom 对象  ;jQuery 对象中没有 reset()函数
});
```

- jQuery 对象->DOM 对象 ： 通过索引进行转换
- DOM 对象 -> jQuery 对象 ：$(DOM 对象)

## 6.4 修改后回到当前页

1.修改连接增加 pageno 参数

```
onclick='window.location.href=\"${APP_PATH}/user/edit.htm?pageno="+pageObj.pageno+"&id="+n.id+"\"'
```

2.修改页面获取 pageno 参数

异步提交修改功能,成功后跳转主页面,增加 pageno 参数

```
window.location.href="${APP_PATH}/user/index.htm?pageno=${param.pageno}";
```

3.主页面根据 pageno 参数进行异步查询

```
$(function () {
<c:if test="${empty param.pageno}">
    queryPage(1);
</c:if>
<c:if test="${not empty param.pageno}">
    queryPage(${param.pageno});
</c:if>
});
```

# 第七章 用户删除

## 7.1 删除一条

### 7.1.1 增加删除连接

```
content+='                                <button type="button" class="btn btn-danger btn-xs"
onclick="deleteUser('+user.id+',\''+user.loginacct+'\')">
<i class=" glyphicon glyphicon-remove"></i></button>';
```

```
function deleteUser(id,loginacct){
        layer.confirm("删除["+loginacct+"]用户,确认删除?",  {icon: 3, title:'提示'}, function(cindex){
                var loadingIndex = -1 ;
                $.ajax({
                        type:"post",
                        url:"${APP_PATH}/user/doDelete.do",
                        data:{
                                "id":id
                        },
                        beforeSend:function(){
                                loadingIndex = layer.msg("正在删除数据!", {time:1000, icon:6});
                                return true ;
                        },
                        success:function(result){
                                layer.close(loadingIndex);
                                if(result.success){
                                        queryPageUser(1);
                                }else{
```

```
                                        layer.msg("删除失败!", {time:1000, icon:5, shift:6});
                                }
                        },
                        error:function(){
                                layer.msg("删除失败!", {time:1000, icon:5, shift:6});
                        }
                });
                    layer.close(cindex);
                }, function(cindex){
                    layer.close(cindex);
                });
}
```

### 7.1.2 删除操作

1. handler
2. service
3. dao

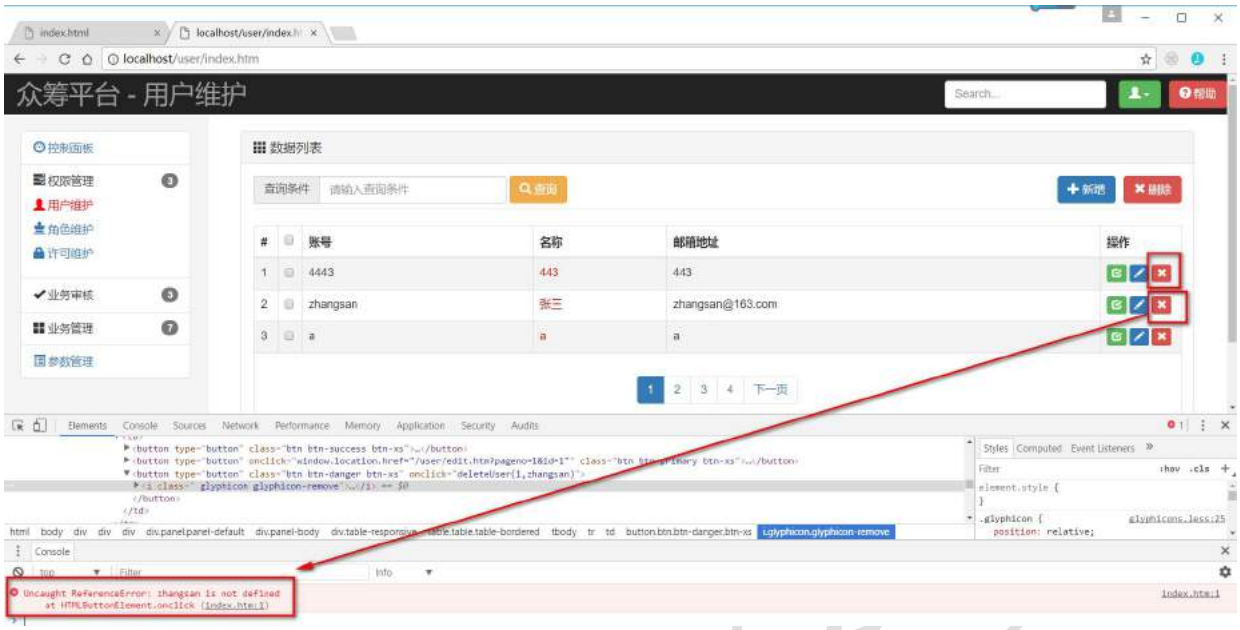
### 7.1.3 注意问题

1. 字符串拼接,外面和里面分别使用双引号和单引号,不能同时使用一种.
2. 删除执行函数传递参数

user.loginacct 应该用引号引起来,否则被当成变量使用;

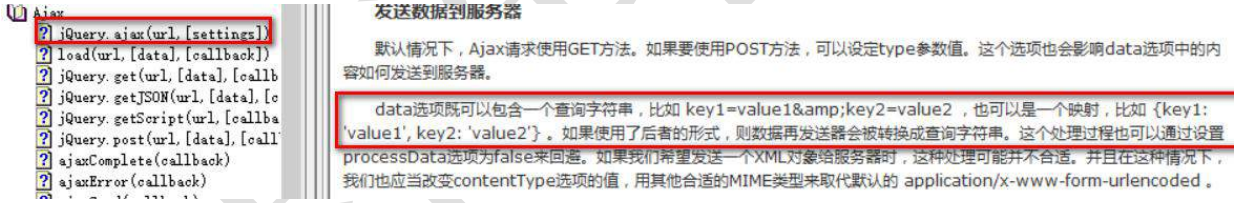
loginacct 是数值没问题;是字符串有问题.

```
content+='                                <button type="button" class="btn btn-danger btn-xs"
onclick="deleteUser('+user.id+',\''+user.loginacct+'\')">
<i class=" glyphicon glyphicon-remove"></i></button>';
```




## 7.2 批量删除

### 7.2.1 增加单击事件处理




客户端给服务器提交多条数据

1. 拼串

语法:key1=value1&key2=value2

例子:age=22&name=zhangsan&age=33&name=lisi

```
$("#checkboxAll").click(function(){
        var checkboxAllStatus = this.checked ;
        $(".checkboxClass").each(function(i,n){
                n.checked = checkboxAllStatus;
        });
        /* $.each(array,function(i,n){}); */
```

```
});
$("#deleteBatch").click(function(){
        var checkBoxChecked = $(".checkboxClass:checked");
        var length = checkBoxChecked.length;
        var paramStr = "";
        $.each(checkBoxChecked,function(i,n){
                if(i!=0){
                        paramStr += "&";
                }
                paramStr += "id="+n.value ; // id=1&id=2&id=3
        });
        if(length>0){
                layer.confirm("删除这些用户,确认删除?",  {icon: 3, title:'提示'}, function(cindex){
                    var loadingIndex = -1 ;
                $.ajax({
                        type:"post",
                        url:"${APP_PATH}/user/doDeleteBatch.do",
                        data:paramStr,   //可以是字符串,也可以是 JSON 对象
                        beforeSend:function(){
                                loadingIndex = layer.msg("正在删除数据!", {time:1000, icon:6});
                                return true ;
                        },
                        success:function(result){
                                layer.close(loadingIndex);
                                if(result.success){
                                        queryPageUser(1);
                                }else{
```

```
                                        layer.msg("删除失败!", {time:1000, icon:5, shift:6});
                                    }
                                },
                                error:function(){
                                    layer.msg("删除失败!", {time:1000, icon:5, shift:6});
                                }
                            });
                            layer.close(cindex);
                        }, function(cindex){
                            layer.close(cindex);
                        });
                    }else{
                        layer.msg("请选择要删除的用户!", {time:1000, icon:6});
                    }
});
```

## 7.2.2 处理请求

```
@ResponseBody
@RequestMapping(value="/doDeleteBatch",method=RequestMethod.POST)
public Object doDelete(Integer[] id){}
```

```
@Override
public int deleteBatchUser(Integer[] id) {
    int totalCount = userDao.deleteBatchUser(id);
    if(totalCount!=id.length){
        throw new RuntimeException("批量删除失败!");
    }
```

```
    return totalCount;
}
```

```
int deleteBatchUser(Integer[] id);
```

```
<!--
int deleteBatchUser(@Param("ids") Integer[] id);      //    collection="ids"
int deleteBatchUser(Integer[] id); //collection="array"如果参数是数组,采用默认"array"获取数组参数
int deleteBatchUser(List<Integer> id); //collection="list"如果参数是集合,采用默认"list"获取集合参数
 -->
<delete id="deleteBatchUser">
delete from t_user where id in
<foreach collection="array" open="(" separator="," close=")" item="userid">
    #{userid}
</foreach>
</delete>
```

# 作业

1.完成以下功能模块开发

①资质管理

②参数管理

③分类管理

④角色管理(必须)


控制面板
权限管理
3
用户维护
角色维护
许可维护
业务审核
3
业务管理
7
资质维护
分类管理
流程管理
广告管理
消息模板
项目分类
项目标签
参数管理

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

- 抽取菜单
- 多条数据提交
- 角色分配

# 第 1 章 表单多条数据提交

- 第 1 种方法：表单提交，以字段数组接收
- 第 2 种方法：表单提交，以 BeanListModel 接收

## 1.1 第 1 种方法：以字段数组接收

- HTML 代码如下：

```
<h1>submitUserList_1</h1>
<form action="${pageContext.request.contextPath }/customer/saveCustomer" method="post">
用户名称：<input type="text" name="custName"/><br><br>
年龄：<input type="text" name="custAge"/><br><br>

用户名称：<input type="text" name="custName"/><br><br>
年龄：<input type="text" name="custAge"/><br><br>
<input type="submit" value="保存"><br><br>
```

```
</form>
```

- Java 代码如下：

```
@RequestMapping(value="/customer/saveCustomer",method=RequestMethod.POST)
public String saveCustomer(@Param("custName") String[] custName,
                           @Param("custAge") Integer[] custAge) throws Exception{
    for (int i = 0; i < custAge.length; i++) {
        System.out.println(custAge[i]);
    }
    for (int i = 0; i < custName.length; i++) {
        System.out.println(custName[i]);
    }
    return "redirect:/customer/listCustomer";
}
```

- **字段比较少时适合使用**

## 1.2 第 2 种方法：以 BeanListModel 接收

- HTML 代码如下：

```
<h1>submitUserList_2</h1>
<form action="${pageContext.request.contextPath }/customer/saveCustomer2" method="post">
用户名称：<input type="text" name="customerList[0].custName"/><br><br>
年龄：<input type="text" name="customerList[0].custAge"/><br><br>

用户名称：<input type="text" name="customerList[1].custName"/><br><br>
年龄：<input type="text" name="customerList[1].custAge"/><br><br>
<input type="submit" value="保存"><br><br>
</form>
```

- Java 代码如下：

```
@RequestMapping(value="/customer/saveCustomer2",method=RequestMethod.POST)
public String saveCustomer2(@RequestParam("customerList") CustomerModel customerModel) throws Exception{
    System.out.println(customerModel.getCustomerList());
    return "redirect:/customer/listCustomer";
}
```

```
public class CustomerModel {//Value Object
    private List<Customer> customerList ;
    public List<Customer> getCustomerList() {
        return customerList;
    }
    public void setCustomerList(List<Customer> customerList) {
        this.customerList = customerList;
    }
}
```

- **注意索引连续问题**

# 第 2 章 用户批量删除改进

## 2.1 增加单击事件处理

```
//给删除按钮增加单击事件处理
$("#batchDelete").click(function(){
    var checkboxChecked = $(".table tbody input:checked"); //获取所有选中的复选框
    if(checkboxChecked.length>1){
        layer.confirm("请确认是否删除选中的用户?",  {icon: 3, title:'提示'}, function(cindex){
            // 多条数据的传递，可以采用对象加索引的方式
```

```
            var dataObj = {};
            $.each(checkboxChecked, function(i, n){
                dataObj["userList["+i+"].uid"] = n.value;
                dataObj["userList["+i+"].username"] = "123";
            });
            $.ajax({
                url : "${PATH}/user/deleteBatchUser.do",
                type : "POST",
                data : dataObj,
                success : function(result){
                    if(result.success){
                        layer.msg("选中的用户信息删除成功", {time:2000, icon:6}, function(){
                            queryUser(0);
                        });
                    }else{
                        layer.msg("选中的用户信息删除失败!", {time:2000, icon:5, shift:6});
                    }
                }
            });
            layer.close(cindex);
        }, function(cindex){
            layer.close(cindex);
        });
    }else{
        layer.msg("请选中要删除的用户!", {time:2000, icon:5, shift:6});
    }
});
```

## 2.2 处理请求

```
import java.util.ArrayList;
import java.util.List;

public class Data {
    private List<User> userList = new ArrayList<User>();
    public List<User> getUserList() {
        return userList;
    }
    public void setUserList(List<User> userList) {
        this.userList = userList;
    }
}
```

```
@ResponseBody
@RequestMapping("/deleteBatchUser")
public Object deleteBatchUser(Data data){
    Map map = new HashMap();
    try {
        int result = userService.deleteUserByBatchUsers(data);
        map.put("success", (result==data.getUserList().size()));
    } catch (Exception e) {
        e.printStackTrace();
        map.put("success", false);
    }
    return map;
}
```

```
@Override
public int deleteUserByBatchUsers(Data data) {
    return userDao.deleteUserByBatchUsers(data);
}
```

```
int deleteUserByBatchUsers(Data data);
```

```
<!-- 批量删除:userList 为 Data 对象的属性 -->
<delete id="deleteUserByBatchUsers">
delete from t_user where uid in
    <foreach collection="userList" item="user" open="(" separator="," close=")">
        #{user.uid}
    </foreach>
</delete>
```

## 2.3 批量删除

### 2.3.1 使用 foreach 标签将多条 SQL 语句连接在一起执行

1. 在连接数据库的 URL 地址后面附加：?allowMultiQueries=true
2. 接口方法 void batchSaveCustomer(List<Customer> customerList);
3. 配置文件

```
<insert id="batchSaveCustomer" parameterType="java.util.List">
  <foreach collection="list" item="customer" separator=";">
    INSERT INTO tbl_cust (cust_name, cust_age) VALUES (#{customer.custName},
     #{customer.custAge})
  </foreach>
</insert>
```

4. 缺陷：一起执行的 SQL 语句数量有限

### 2.3.2 使用原生 JDBC API

Connection org.apache.ibatis.session.SqlSession.getConnection()

# 第 3 章 用户角色分配

## 3.1 页面设计及代码

### 3.1.1 给用户分配角色的页面


### 3.1.2 定义用户角色分配页面参考

1 表单为横向列表-两个可以分配的下拉列选

参考官方文档



## 2 定义分配角色的表单

```
<form class="form-inline">
  <div class="form-group">
    <label for="unDispatcher">未分配角色列表</label><br>
    <select class="form-control" id="leftRoleList" multiple="multiple" size="10" style="overflow: auto;">
```

```
            <option>111</option>
            <option>222</option>
            <option>333</option>
        </select>
    </div>

    <div class="form-group">
        <ul id="roleBtns">
            <li id="leftToRightBtn"><i class="glyphicon glyphicon-chevron-right"></i></li>
            <li id="rightToLeftBtn"><i class="glyphicon glyphicon-chevron-left"></i></li>
        </ul>
    </div>

    <div class="form-group">
        <label for="dispatcher">已分配角色列表</label><br>
        <select class="form-control" id="rightRoleList" multiple="multiple" size="10" style="overflow:
auto;">
            <option>aaa</option>
            <option>bbb</option>
            <option>ccc</option>
        </select>
    </div>

</form>
```

## 3 定义按钮的事件处理

```
            $("#leftToRightBtn").click(function(){
```

```
        var items = $("#leftRoleList :selected");

        if(items.length==0){
            layer.msg("请选择要分配的角色", {time:1000, icon:6});
        }else{
            $("#rightRoleList").append(items.clone());
            items.remove();
        }
    });

    $("#rightRightBtn").click(function(){
        var items = $("#rightRoleList :selected");
        if(items.length==0){
            layer.msg("请选择要取消分配的角色", {time:1000, icon:6});
        }else{
            $("#leftRoleList").append(items.clone());
            items.remove();
        }
    });
```

## 3.2 用户角色分配-后台功能实现

### 3.2.1 Data 类

```
public class Data {
    private List<Integer> ids;
    ...
```

### 3.2.2 获取未分配角色和已分配集合

```
@RequestMapping("/toAssignRole")
public String toAssignRole(Integer userid,Map map){
    User user = userService.getUser(userid);
    map.put("user", user);

    // 查询所有角色
    List<Role> allRole = roleService.queryAll();
    //查询已经用户已经分配的角色 id
    List<Integer> roleId = userService.queryRoleIdsByUserid(userid);

    // 未分配角色
    List<Role> unassign = new ArrayList<Role>();
    // 已分配角色
    List<Role> assign = new ArrayList<Role>();
    for (Role role : allRole) {
        if(roleId.contains(role.getId())){
            assign.add(role);
        }else{
            unassign.add(role);
        }
    }
    map.put("unassign", unassign);
    map.put("assign", assign);
    return "user/assignRole";
}
```

- RoleService

```
List<Role> queryAll();
```

```
public List<Role> queryAll() {
    return roleDao.queryAll();
}
```

- UserService

```
List<Integer> queryRoleIdsByUserid(Integer id);
```

```
public List<Integer> queryRoleIdsByUserid(Integer id) {
    return userDao.queryRoleIdsByUserid(id);
}
```

- RoleDao
- mapper-role.xml

```
List<Role> queryAll();
```

```
    <select id="queryAll" resultType="role">
        select * from t_role
    </select>
```

- UserDao
- mapper-user.xml

```
List<Integer> queryRoleIdsByUserid(Integer id);
```

```
    <select id="queryRoleIdsByUserid" resultType="int">
        select roleid from t_user_role where userid = #{userid}
    </select>
```

### 3.2.3 页面显示角色列表

```
<div class="form-group">
<label for="exampleInputPassword1">未分配角色列表</label><br>
```

```
<select id="leftRoleList" class="form-control" multiple size="10" style="width:200px;overflow-y:auto;">
   <c:forEach items="${unassign }" var="role">
      <option value="${role.id }">${role.name }</option>
   </c:forEach>
</select>
   </div>
   <div class="form-group">
            <ul>
               <li id="leftToRightBtn" class="btn btn-default glyphicon glyphicon-chevron-right"></li>
               <br>
               <li id="rightRightBtn" class="btn btn-default glyphicon glyphicon-chevron-left"
style="margin-top:20px;"></li>
            </ul>
   </div>
   <div class="form-group" style="margin-left:40px;">
<label for="exampleInputPassword1">已分配角色列表</label><br>
<select id="rightRoleList" class="form-control" multiple size="10" style="width:200px;overflow-y:auto;">
   <c:forEach items="${assign }" var="role">
      <option value="${role.id }">${role.name }</option>
   </c:forEach>
</select>
</div>
```

## 3.3 用户角色分配功能实现

### 3.3.1 用户角色分配异步请求

```
            $("#leftToRightBtn").click(function(){
```

```
        var items = $("#leftRoleList :selected");


        if(items.length==0){
            layer.msg("请选择要分配的角色", {time:1000, icon:6});
        }else{
            var data = {"userid":"${user.id}"};
            $.each(items,function(i,n){
                data["ids["+i+"]"] = n.value ;
            });
            $.ajax({
                url : "${APP_PATH}/user/assign.do",
                type : "post",
                data : data ,
                success : function(result){
                    if(result.success){
                        layer.msg("分配的角色成功", {time:1000, icon:6});
                        $("#rightRoleList").append(items.clone());
                        items.remove();
                    }else{
                        layer.msg("分配的角色失败", {time:1000, icon:5});
                    }
                }
            });
        }
    });


    $("#rightRightBtn").click(function(){
```

```
            var items = $("#rightRoleList :selected");
            if(items.length==0){
                layer.msg("请选择要取消分配的角色", {time:1000, icon:6});
            }else{
                var data = {"userid":"${user.id}"};
                $.each(items,function(i,n){
                data["ids["+i+"]"] = n.value ;
                });
                $.ajax({
                    url : "${APP_PATH}/user/unassign.do",
                    type : "post",
                    data : data ,
                    success : function(result){
                        if(result.success){
                            layer.msg("取消分配角色成功", {time:1000, icon:6});
                            $("#leftRoleList").append(items.clone());
                            items.remove();
                        }else{
                            layer.msg("取消分配角色失败", {time:1000, icon:5});
                        }
                    }
                });
            }
        });
```

## 3.3.2 分配角色处理

```
@ResponseBody
```

```
@RequestMapping("/unassign")
public Object unassign(Integer userid,Data ds) {}


@ResponseBody
@RequestMapping("/assign")
public Object assign(Integer userid,Data ds) {}
```

- UserService

```
public int deleteUserRole(Integer userid, Data ds) {
    List<Integer> roleIdList = ds.getIds();
    return userDao.deleteUserRole(userid,roleIdList);
}


public int insertUserRole(Integer userid, Data ds) {
    int count = 0 ;
    List<Integer> ids = ds.getIds();
    for (Integer roleid : ids) {
        UserRole ur = new UserRole(roleid,userid);
        count += userDao.insertUserRole(ur);
    }
    return count ;
}
```

- UserDao
- mapper-user.xml

```
int deleteUserRole(@Param("userid")    Integer userid,@Param("roleIdList") List<Integer> roleIdList);
int insertUserRole(UserRole ur);
```

```
    <insert id="insertUserRole" >
        insert into t_user_role (
```

```
            userid,roleid
        ) values (
            #{userid},#{roleid}
        )
    </insert>

    <delete id="deleteUserRole">
        delete from t_user_role where userid=#{userid} and roleid in
        <foreach collection="roleIdList" item="roleid" open="(" close=")" separator=",">
            #{roleid}
        </foreach>
    </delete>
```

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

1. RBAC 权限模型
2. zTree
3. 许可树展示
4. 许可模块(增删改)

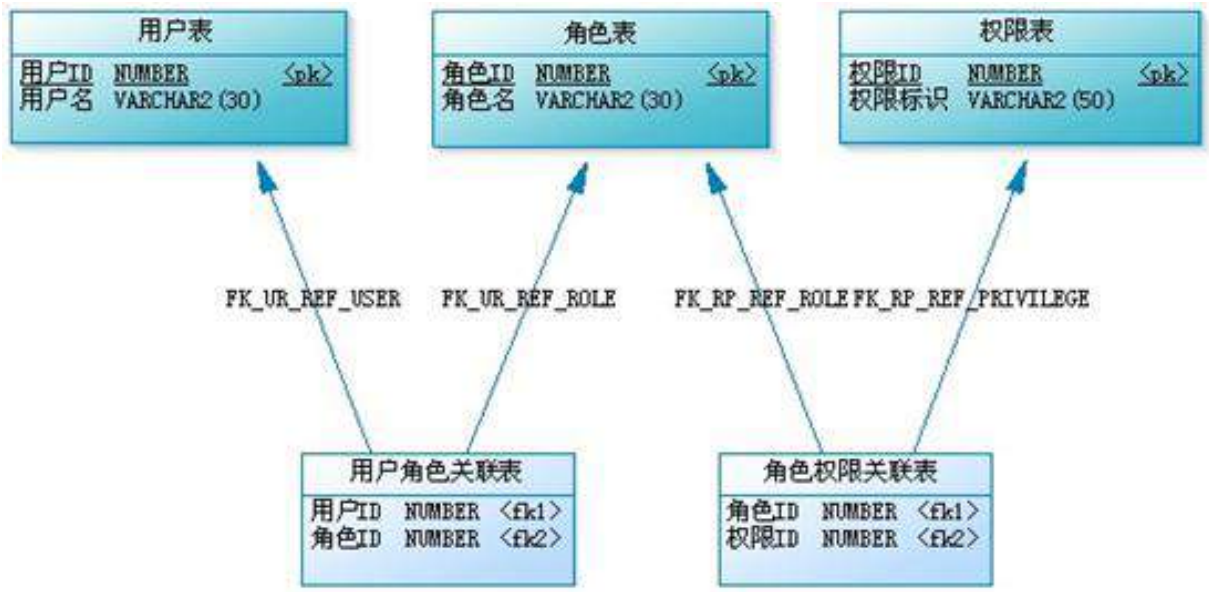
# 第 1 章 RBAC 权限模型

## 1.1 概念

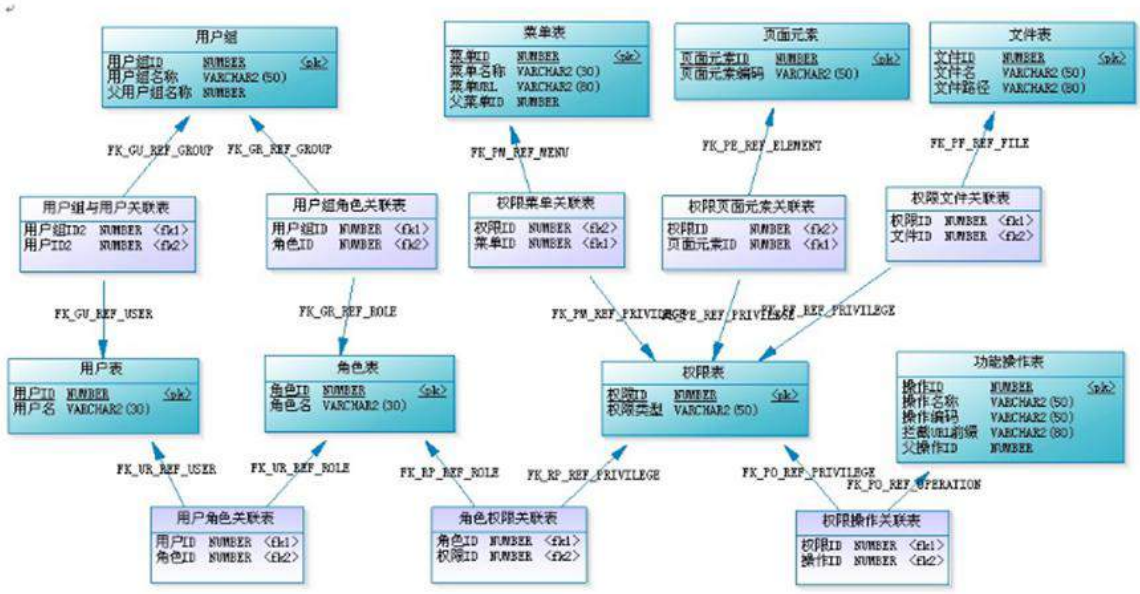
权限管理，这是每个软件系统都会涉及到的，而且权限管理的需求本质往往都是一样，不同的角色拥有不同的权限，只要你充当了某个角色，你就拥有了相对应的功能。

**RBAC（Role-Based Access Control，基于角色的访问控制）**，就是用户通过角色与权限进行关联。简单地说，一个用户拥有若干角色，每一个角色拥有若干权限。

这样，就构造成“**用户-角色-权限**”的授权模型。在这种模型中，用户与角色之间，角色与权限之间，一般都是多对多的关系

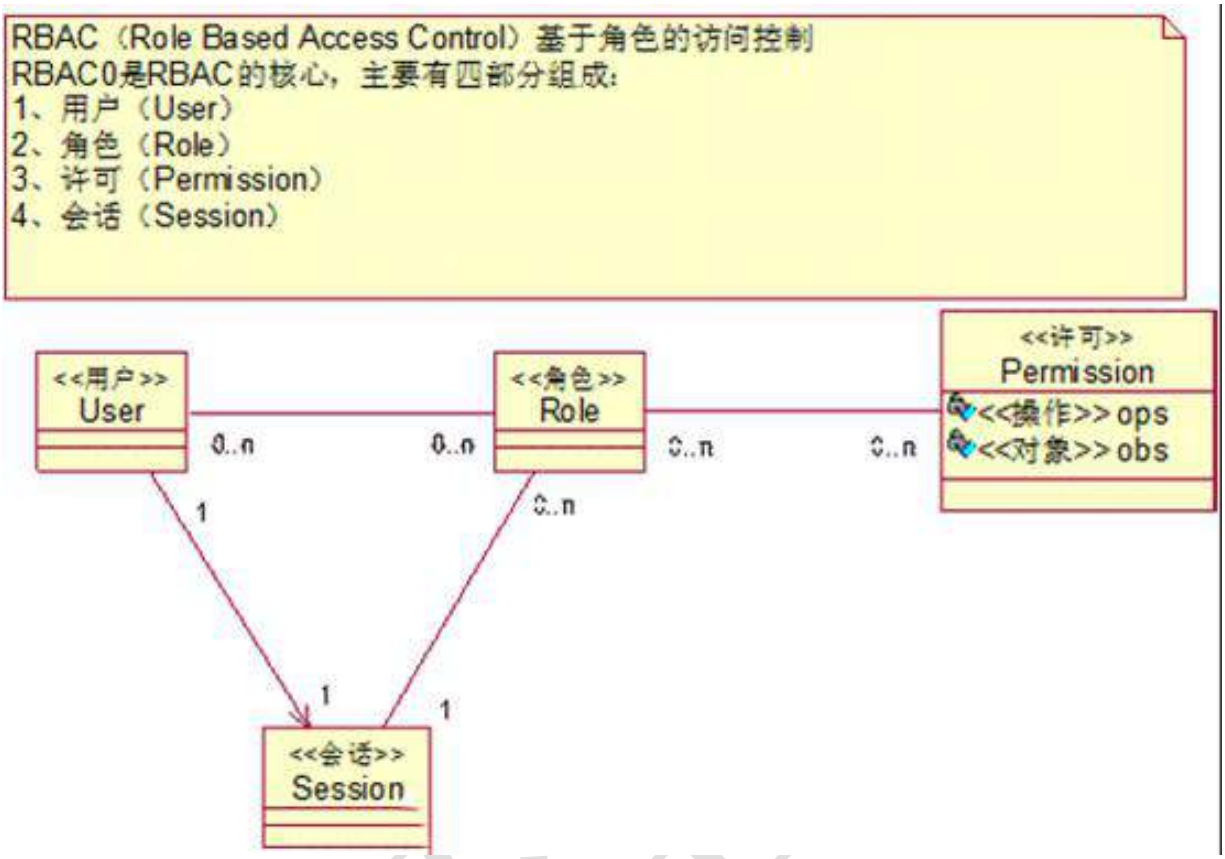



## 1.2 扩展模型




## 1.3 RBAC 级别 - RABC0

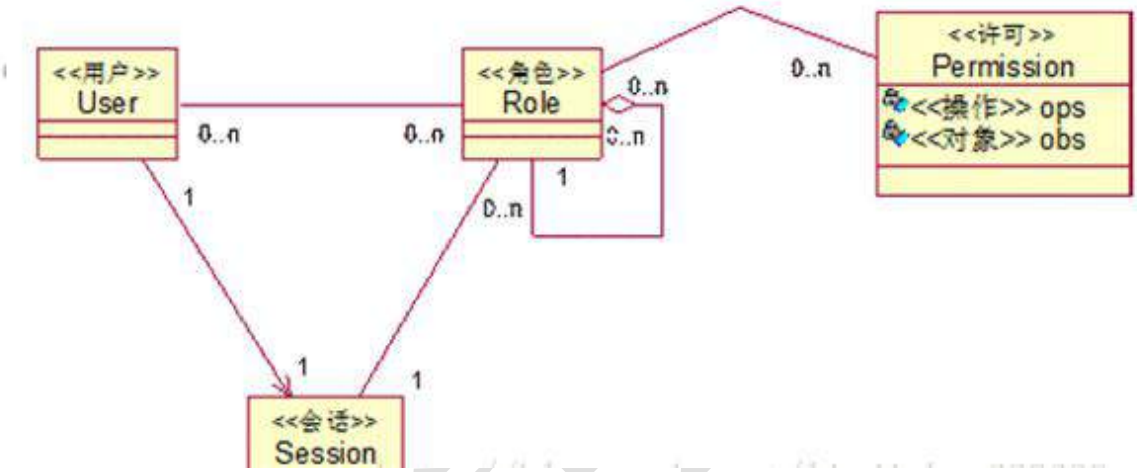
核心模型，其他的级别都是建立在该级别的基础上




## 1.4 RBAC 级别 - RABC1

基于 RBAC0 模型，进行了角色的分层，也就是说角色上有了上下级的区别

RBAC（Role Based Access Control）基于角色的访问控制
RBAC1是对RBAC0进行了扩展，是RBAC的角色分层模型，RBAC1引入了角色继承概念，有了继承就有了上下级的包含关系



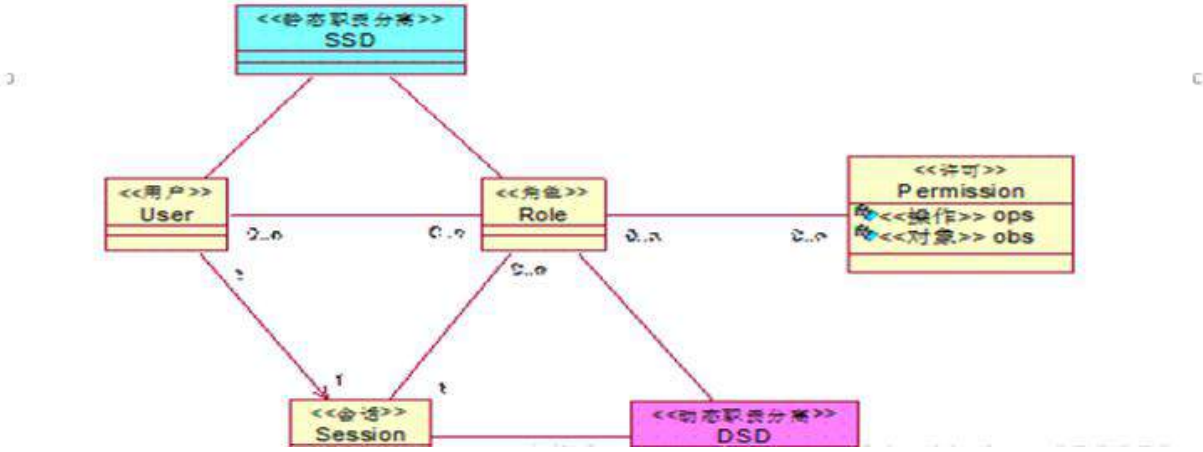

## 1.5 RBAC 级别 - RABC2

RBAC2，也是基于 RBAC0 模型的基础上，进行了角色的访问控制

RBAC（Role Based Access Control）基于角色的访问控制
RBAC2是基于RBAC0扩展的，主要引入了SSD（静态职责分离）和DSD（动态职责分离）

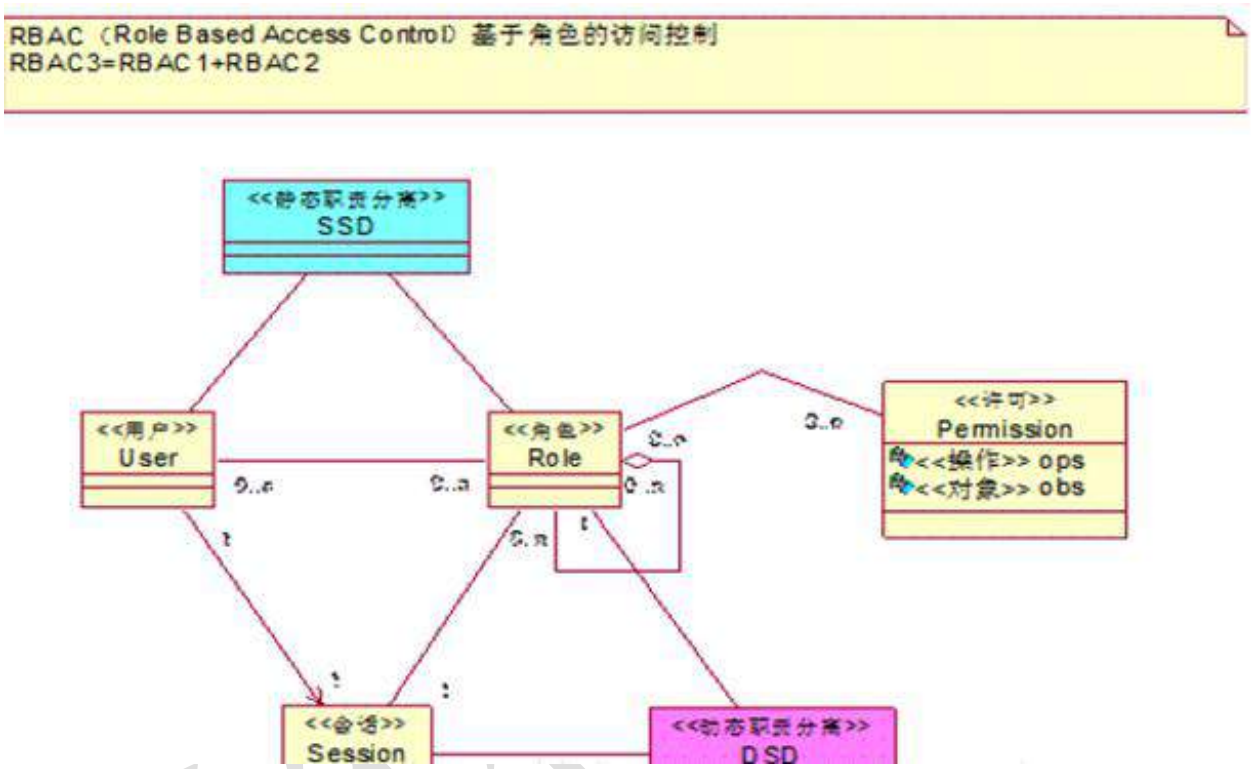
SSD主要应用在用户和角色之间（授权阶段），主要约束：
1、互斥角色，同一个用户不能授予互斥关系的角色，如：不能同时授予会计和出纳的角色
2、基数约束，一个用户拥有的角色是有限的，一个角色拥有的许可是有限的
3、先决条件约束，用户想得到高级权利，必须先拥有低级权利

DSD会话和角色之间的约束，主要动态决定怎么样计划角色，如：一个用户拥有5个角色，只激活2个



## 1.6 RBAC 级别 - RBAC3

RBAC3，也就是最全面级的权限管理，它是基于 RBAC2 的基础上，将 RBAC1 和 RBAC2 进行整合了，最全面，也最复杂的



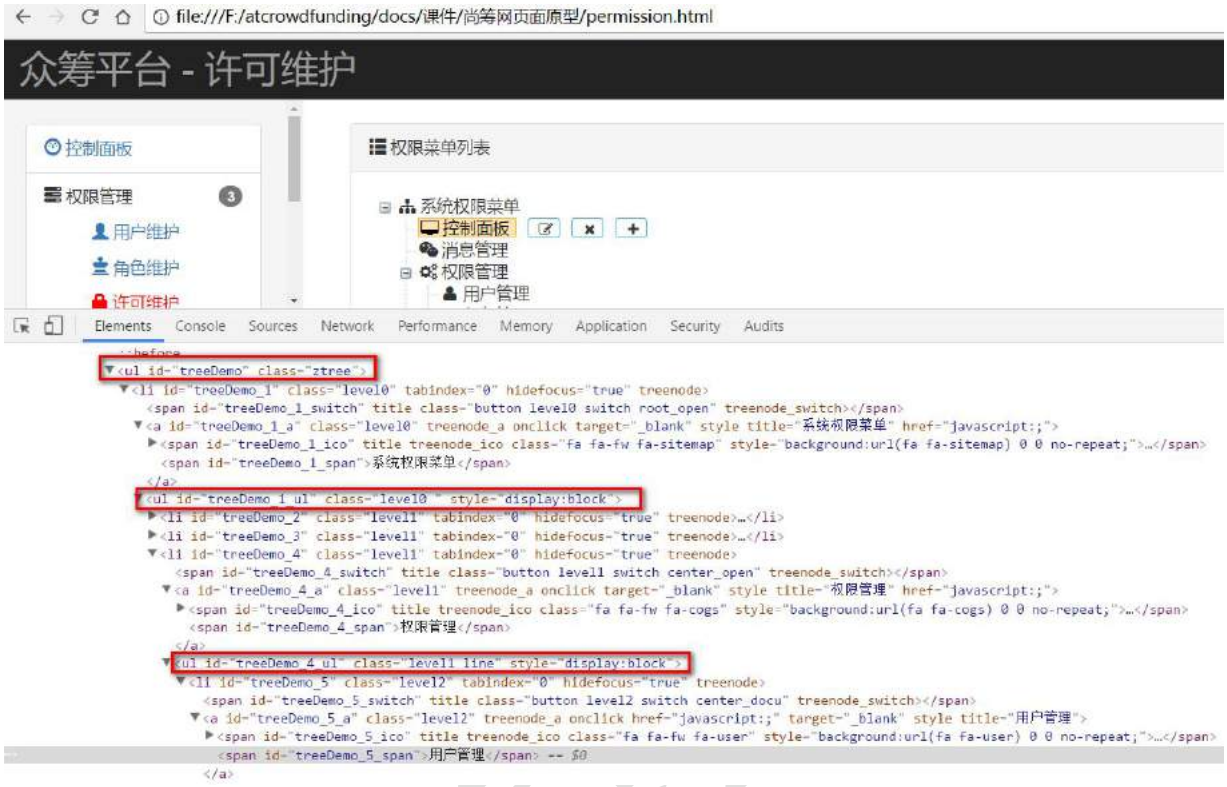

## 1.7 总结

任何系统中都会涉及到权限管理的模块，无论复杂简单，我们都可以通过以 RBAC 模型为基础，进行相关灵活运用来解决我们的问题

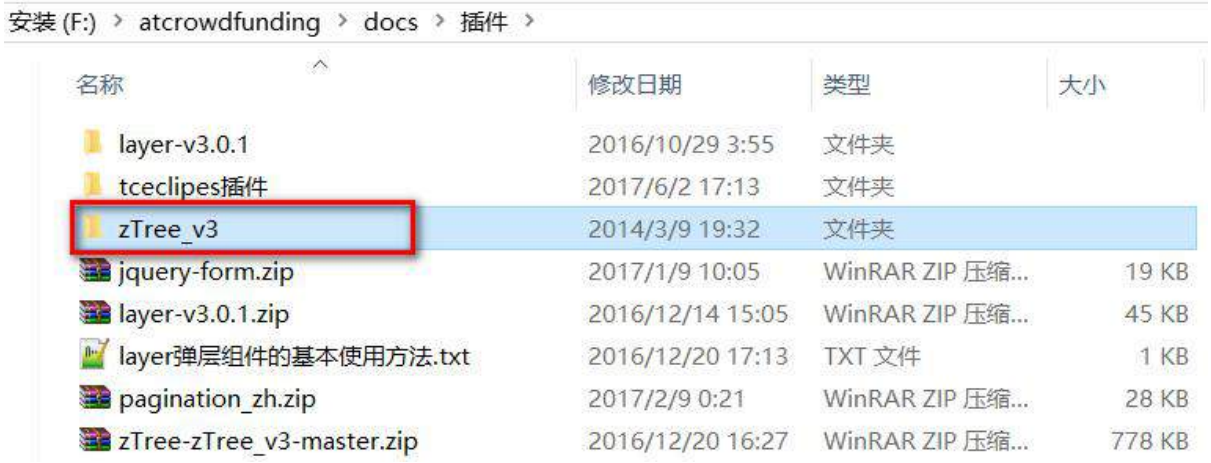
# 第 2 章 分析许可维护树型结构

## 2.1 许可树

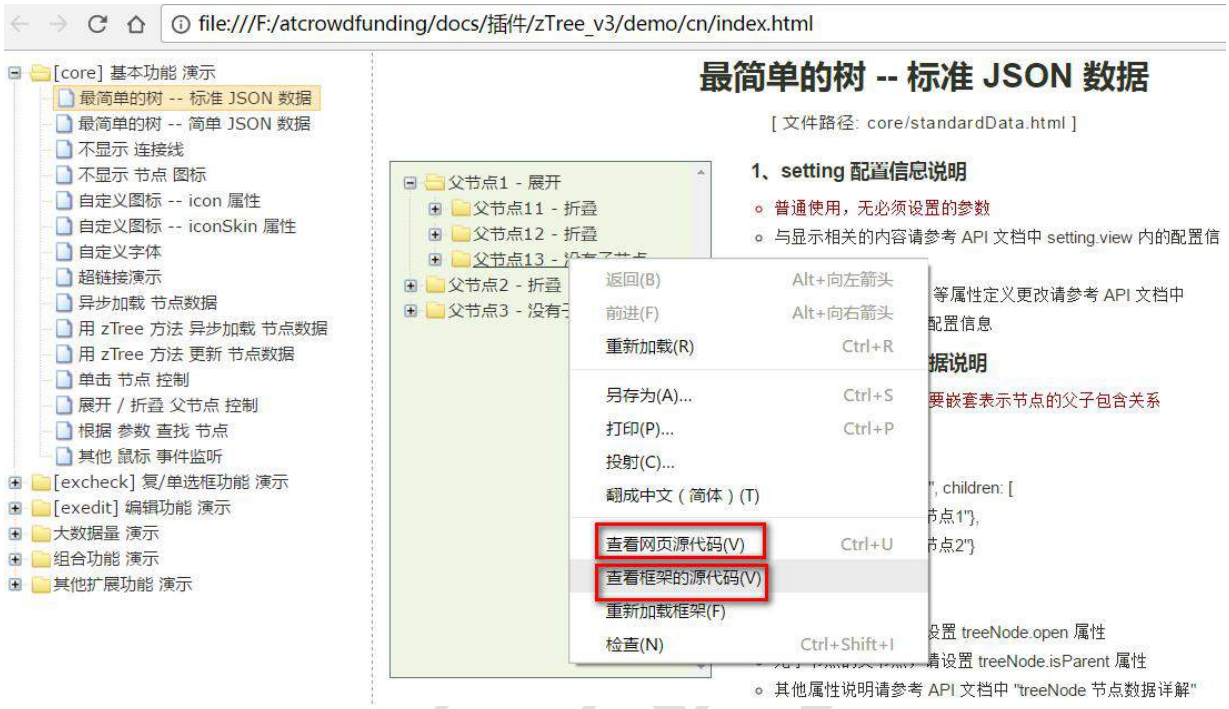
ul 嵌套 ul 来实现许可树;




## 2.2 参考 zTree-demo



## 2.3 查看框架的源代码




# 第 3 章 模拟 zTree-Demo 分析

## 3.1 zTree 树形结构介绍

## 3.2 zTree 树形结构分析许可树

### 3.2.1 连接

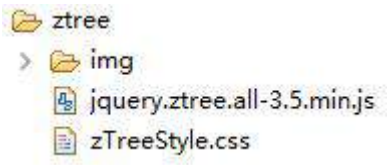
```
<a href="${APP_PATH }/permission/index.htm"><i class="glyphicon glyphicon-lock"></i>  许可维护</a>
```

### 3.2.2 许可页面

```
package com.atguigu.atcrowdfunding.manager.handler;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import com.atguigu.atcrowdfunding.manager.service.PermissionService;


@Controller
@RequestMapping("/permission")
public class PermissionHandler {


	@Autowired
	private PermissionService permissionService ;


	@RequestMapping("/index")
	public String index(){
		return "permission/index";
	}
}
```

## 3.2.3 页面展示数据分析

2.3.1 将 ztree 引入到项目中




2.3.2 引入 css 和 js，拷贝 zTree 案例代码，展示出树结构。分析树结构。

```
<link rel="stylesheet" href="${APP_PATH}/ztree/zTreeStyle.css">

<script src="${APP_PATH}/ztree/jquery.ztree.all-3.5.min.js"></script>
```

2.3.3 增加容器

```
<div class="panel-body">
<ul id="treeDemo" class="ztree"></ul>
</div>
```

2.3.4 增加模拟代码

```
var zNodes =[
                    { name:"父节点 1 - 展开", open:true,
                        children: [
                            { name:"父节点 11 - 折叠",
                                children: [
                                    { name:"叶子节点 111"},
                                    { name:"叶子节点 112"},
                                    { name:"叶子节点 113"},
                                    { name:"叶子节点 114"}
                                ]},
                            { name:"父节点 12 - 折叠",
                                children: [
```

```
                    { name:"叶子节点 121"},
                    { name:"叶子节点 122"},
                    { name:"叶子节点 123"},
                    { name:"叶子节点 124"}
                ]},
            { name:"父节点 13 - 没有子节点", isParent:true}
        ]},
    { name:"父节点 2 - 折叠",
        children: [
            { name:"父节点 21 - 展开", open:true,
                children: [
                    { name:"叶子节点 211"},
                    { name:"叶子节点 212"},
                    { name:"叶子节点 213"},
                    { name:"叶子节点 214"}
                ]},
            { name:"父节点 22 - 折叠",
                children: [
                    { name:"叶子节点 221"},
                    { name:"叶子节点 222"},
                    { name:"叶子节点 223"},
                    { name:"叶子节点 224"}
                ]},
            { name:"父节点 23 - 折叠",
                children: [
                    { name:"叶子节点 231"},
                    { name:"叶子节点 232"},
```

```
                                    { name:"叶子节点 233"},
                                    { name:"叶子节点 234"}
                                ]}
                            ]},
                        { name:"父节点 3 - 没有子节点", isParent:true}
                ];
var setting = {       };
$.fn.zTree.init($("#treeDemo"), setting, zNodes);
```

# 第 4 章 加载许可树

## 4.1 准备模拟数据

0 50 过滤

| id | name | pid | icon | url |
|---|---|---|---|---|
| 1 | 系统权限菜单 | | glyphicon glyphicon-blackboard | |
| 2 | 控制面板 | 1 | glyphicon glyphicon-dashboard | main.htm |
| 3 | 权限管理 | 1 | glyphicon glyphicon glyphicon-tasks | |
| 4 | 用户维护 | 3 | glyphicon glyphicon-user | user/index.htm |
| 5 | 角色维护 | 3 | glyphicon glyphicon-king | role/index.htm |
| 6 | 许可维护 | 3 | glyphicon glyphicon-lock | permission/index.htm |
| 7 | 业务审核 | 1 | glyphicon glyphicon-ok | |
| 8 | 实名认证审核 | 7 | glyphicon glyphicon-check | auth_cert/index.htm |
| 9 | 广告审核 | 7 | glyphicon glyphicon-check | auth_adv/index.htm |
| 10 | 项目审核 | 7 | glyphicon glyphicon-check | auth_project/index.htm |
| 11 | 业务管理 | 1 | glyphicon glyphicon-th-large | |
| 12 | 资质维护 | 11 | glyphicon glyphicon-picture | cert/index.htm |
| 13 | 分类管理 | 11 | glyphicon glyphicon-equalizer | type/index.htm |
| 14 | 流程管理 | 11 | glyphicon glyphicon-random | process/index.htm |
| 15 | 广告管理 | 11 | glyphicon glyphicon-hdd | advert/index.htm |
| 16 | 消息模板 | 11 | glyphicon glyphicon-comment | message/index.htm |
| 17 | 项目分类 | 11 | glyphicon glyphicon-list | projectType/index.htm |
| 18 | 项目标签 | 11 | glyphicon glyphicon-tags | tag/index.htm |
| 19 | 参数管理 | 1 | glyphicon glyphicon-list-alt | param/index.htm |

```
create table `t_permission` (
`id` int ,
`pid` int ,
`name` varchar ,
```

```
`icon` varchar ,
`url` varchar
);
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('1',NULL,'系统权限菜单','glyphicon glyphicon-th-list',NULL);
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('2','1','控制面板','glyphicon glyphicon-dashboard','main.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('3','1','权限管理','glyphicon glyphicon glyphicon-tasks',NULL);
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('4','3','用户维护','glyphicon glyphicon-user','user/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('5','3','角色维护','glyphicon glyphicon-king','role/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('6','3','许可维护','glyphicon glyphicon-lock','permission/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('7','1','业务审核','glyphicon glyphicon-ok',NULL);
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('8','7','实名认证审核','glyphicon glyphicon-check','auth_cert/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('9','7','广告审核','glyphicon glyphicon-check','auth_adv/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('10','7','项目审核','glyphicon glyphicon-check','auth_project/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('11','1','业务管理','glyphicon glyphicon-th-large',NULL);
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('12','11','资质维护','glyphicon glyphicon-picture','cert/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('13','11','分类管理','glyphicon glyphicon-equalizer','type/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('14','11','流程管理','glyphicon glyphicon-random','process/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('15','11','广告管理','glyphicon glyphicon-hdd','advert/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('16','11','消息模板','glyphicon glyphicon-comment','message/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('17','11','项目分类','glyphicon glyphicon-list','projectType/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('18','11','项目标签','glyphicon glyphicon-tags','tag/index.htm');
insert into `t_permission` (`id`, `pid`, `name`, `icon`, `url`) values('19','1','参数管理','glyphicon glyphicon-list-alt','param/index.htm');
```

## 4.2 异步请求加载许可树

```
var setting = {      };
$.ajax({
    url:"${APP_PATH}/permission/loadData.do",
```

```
            type:"post",
            success:function(result){
                if(result.success){
                    var zNodes = result.data ;
                    $.fn.zTree.init($("#treeDemo"), setting, zNodes);
                }
            }
        });
```

## 4.3 实体类

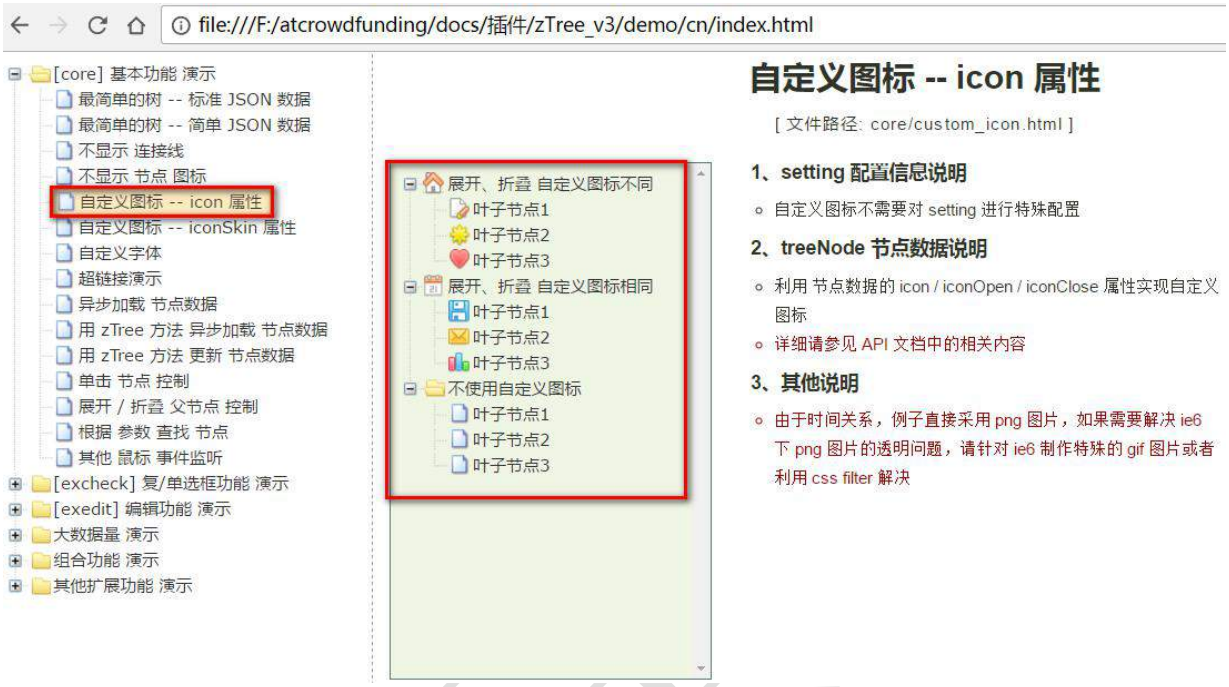
```
public class Permission {
    private Integer id;
    private String name;
    private String url;
    private Integer pid ;
    private String icon ;
    private List<Permission> children = new ArrayList<Permission>();
    private boolean open = true ;
    private boolean checked = false ;
}
```

## 4.4 准备加载许可树数据

```
@ResponseBody
@RequestMapping("/loadData")
public Object loadData(){
        start();
```

```
    try {
        List<Permission> permissons =    permissionService.queryAllPermission();
        Map<Integer,Permission> map = new HashMap<Integer,Permission>();
        for (Permission permission : permissons) {
            map.put(permission.getId(), permission);
        }

        for (Permission permission : permissons) {
            //子节点
            Permission child = permission ; //假设为子菜单
            if(child.getPid() == 0 ){
                data(permission); //根节点
            }else{
                //父节点
                Permission parent = map.get(child.getPid());
                parent.getChildren().add(permission);
            }
        }

        success(true);
    } catch (Exception e) {
        success(false);
        e.printStackTrace();
    }

    return end();
}
```

## 4.5 显示图标




参考 permission.html 源码的 js 代码

```
var setting = {
    view : {
        addDiyDom: function(treeId, treeNode){
            var icoObj = $("#" + treeNode.tId + "_ico"); // tId = permissionTree_1, $("#permissionTree_1_ico")
            if ( treeNode.icon ) {
                icoObj.removeClass("button ico_docu ico_open").addClass("fa fa-fw " + treeNode.icon).css("background","");
            }
        }
    }
};
```

# 第 5 章 许可新增

## 5.1 增加链接

permission/index.jsp

```
addHoverDom: function(treeId, treeNode){
var aObj = $("#" + treeNode.tId + "_a"); //获取连接
aObj.attr("href", "javascript:;"); //点击超级链接,什么都不做;
if (treeNode.editNameFlag || $("#btnGroup"+treeNode.tId).length>0) return;
var s = '<span id="btnGroup'+treeNode.tId+'">';
if ( treeNode.level == 0 ) { //根节点
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="window.location.href=\'${PATH}/permission/add.htm?pid='+treeNode.id+'&level='+treeNode.leve
l+'\'">  <i class="fa fa-fw fa-plus rbg "></i></a>';
} else if ( treeNode.level == 1 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"  href="#"
title="修改权限信息">  <i class="fa fa-fw fa-edit rbg "></i></a>';
if (treeNode.children.length == 0) { //当前节点没有子节点,可以存在增加的按钮.
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"
href="#" >  <i class="fa fa-fw fa-times rbg "></i></a>';
}
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="window.location.href=\'${PATH}/permission/add.htm?pid='+treeNode.id+'&level='+treeNode.leve
l+'\'">  <i class="fa fa-fw fa-plus rbg "></i></a>';
} else if ( treeNode.level == 2 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"  href="#"
title="修改权限信息">  <i class="fa fa-fw fa-edit rbg "></i></a>';
```

```
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#">  <i class="fa fa-fw fa-times rbg "></i></a>';
}

s += '</span>';
aObj.after(s); //在节点之后增加按钮.
}
```

## 5.2 跳转到添加页面

## 5.3 添加页面

## 5.4 添加表单处理

- Handler
- Service

```
void savePermission(Permission permission);
```

```
@Override
public void savePermission(Permission permission) {
    permissionDao.savePermission(permission);
}
```

- DAO

```
void savePermission(Permission permission);
```

```
<insert id="savePermission" parameterType="Permission">
insert into t_permission(name,url,pid) values(#{name},#{url},#{pid})
</insert>
```

## 5.5 添加后效果


## 5.6 注意事项

- Level 0 增加连接，url 可以为空
- Level1 增加链接，url 验证不能为空
- Level2 不能增加链接，属于叶子节点
- 增加链接需要指定链接 pid

# 第 6 章 许可修改

## 6.1 添加链接

- permission/index.jsp

```
addHoverDom: function(treeId, treeNode){
var aObj = $("#" + treeNode.tId + "_a");
```

```
aObj.attr("href", "javascript:;");
if (treeNode.editNameFlag || $("#btnGroup"+treeNode.tId).length>0) return;
var s = '<span id="btnGroup'+treeNode.tId+'">';
if ( treeNode.level == 0 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="window.location.href=\'${PATH}/permission/add.htm?pid='+treeNode.id+'&level='+treeNode.level
+\'">  <i class="fa fa-fw fa-plus rbg "></i></a>';
} else if ( treeNode.level == 1 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"  href="#"
onclick="window.location.href=\'${PATH}/permission/edit.htm?id='+treeNode.id+'&level='+treeNode.level
+\'" title="修改">  <i class="fa fa-fw fa-edit rbg "></i></a>';
if (treeNode.children.length == 0) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"
href="#" >  <i class="fa fa-fw fa-times rbg "></i></a>';
}
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="window.location.href=\'${PATH}/permission/add.htm?pid='+treeNode.id+'&level='+treeNode.level
+\'">  <i class="fa fa-fw fa-plus rbg "></i></a>';
} else if ( treeNode.level == 2 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"  href="#"
onclick="window.location.href=\'${PATH}/permission/edit.htm?id='+treeNode.id+'&level='+treeNode.level
+\'" title="修改">  <i class="fa fa-fw fa-edit rbg "></i></a>';
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"
href="#">  <i class="fa fa-fw fa-times rbg "></i></a>';
}

s += '</span>';
```

```
aObj.after(s);
}
```

## 6.2 跳转修改页面

```
@RequestMapping("/edit")
public String edit(Integer id,Map map){
    Permission permission = permissionService.getPermission(id);
    map.put("permission", permission);
    return "permission/edit";
}
```

```
Permission getPermission(Integer id);
```

```
@Override
public Permission getPermission(Integer id) {
    return permissionDao.getPermission(id);
}
```

```
Permission getPermission(Integer id);
```

```
<select id="getPermission" resultMap="permissionMap" parameterType="integer">
select * from t_permission where id=#{id}
</select>
```

## 6.3 页面表单回显

- permission/edit.jsp

```
<form >
  <div class="form-group">
<label for="permissionName">许可名称</label>
```

```
<input type="text" class="form-control" id="permissionName" value="${permission.name }" placeholder="请输入许可名称">
  </div>
  <div class="form-group">
<label for="permissionUrl">许可 URL</label>
<input type="text" class="form-control" id="permissionUrl" value="${permission.url }" placeholder="请输入许可 URL">
  </div>
  <button id="updateBtn" type="button" class="btn btn-success"><i class="glyphicon glyphicon-pencil"></i> 修改</button>
  <button type="button" class="btn btn-danger"><i class="glyphicon glyphicon-refresh"></i> 重置</button>
</form>
```

```
<script type="text/javascript">
        $("#updateBtn").click(function(){
                        var loadingIndex = -1 ;
                        $.ajax({

                                url : "${APP_PATH}/permission/doUpdate.do",
                                type : "POST",
                                data : {
                                        name : $("#permissionName").val(),
                                        url : $("#permissionUrl").val(),
                                        id : ${param.id}
                                },
                                beforeSend : function(){
```

```
                loadingIndex = layer.msg('数据正在修改中', {icon: 6});
                return true ;
            },
            success : function(result){
                layer.close(loadingIndex);
                if(result.success){
                    layer.msg("许可数据修改成功", {time:1000, icon:6});
                    window.location.href="${APP_PATH}/permission/index.htm";
                }else{
                    layer.msg("许可数据修改失败", {time:1000, icon:5, shift:6});
                }
            },
            error : function(){
                layer.msg("许可数据修改失败", {time:2000, icon:5, shift:6});
            }

        });

    });

</script>
```

## 6.4 提交表单修改数据

- Handler

```
@RequestMapping("/doUpdate")
@ResponseBody
public Object doUpdate(Permission permission){
```

```
        start();
        try {
            int count = permissionService.updatePermission(permission);
            success(count==1);
        }catch(Exception e){
            success(false);
            e.printStackTrace();
        }
        return result;
}
```

- Service

```
int updatePermission(Permission permission);
```

```
@Override
public int updatePermission(Permission permission) {
    return permissionDao.updatePermission(permission);
}
```

- Dao

```
int updatePermission(Permission permission);
```

```
<update id="updatePermission">
update t_permission set name=#{name},url=#{url} where id=#{id}
</update>
```

# 第 7 章 许可删除

## 7.1 添加链接

- permission/tree.jsp

```
addHoverDom: function(treeId, treeNode){
var aObj = $("#" + treeNode.tId + "_a");
aObj.attr("href", "javascript:;");
if (treeNode.editNameFlag || $("#btnGroup"+treeNode.tId).length>0) return;
var s = '<span id="btnGroup'+treeNode.tId+'">';
if ( treeNode.level == 0 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="window.location.href=\'${PATH}/permission/add.htm?pid='+treeNode.id+'&level='+treeNode.level
+'\'">  <i class="fa fa-fw fa-plus rbg "></i></a>';
} else if ( treeNode.level == 1 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"  href="#"
onclick="window.location.href=\'${PATH}/permission/edit.htm?id='+treeNode.id+'&level='+treeNode.level
+'\'" title="修改">  <i class="fa fa-fw fa-edit rbg "></i></a>';
if (treeNode.children.length == 0) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="deletePermission('+treeNode.id+',\''+treeNode.name+'\')">  <i  class="fa  fa-fw
fa-times rbg "></i></a>';
}
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="window.location.href=\'${PATH}/permission/add.htm?pid='+treeNode.id+'&level='+treeNode.level
+'\'">  <i class="fa fa-fw fa-plus rbg "></i></a>';
} else if ( treeNode.level == 2 ) {
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;"  href="#"
onclick="window.location.href=\'${PATH}/permission/edit.htm?id='+treeNode.id+'&level='+treeNode.level
+'\'" title="修改">  <i class="fa fa-fw fa-edit rbg "></i></a>';
s += '<a class="btn btn-info dropdown-toggle btn-xs" style="margin-left:10px;padding-top:0px;" href="#"
onclick="deletePermission('+treeNode.id+',\''+treeNode.name+'\')">  <i  class="fa  fa-fw
```

```
fa-times rbg "></i></a>';
}

s += '</span>';
aObj.after(s);
}
```

```
function deletePermission(id,name){
    layer.confirm("确定要删除["+name+"]吗?",  {icon: 3, title:'提示'}, function(cindex){

        var datas = {
            id:id
        };

        $.ajax({
            url : "${APP_PATH}/permission/deletePermission.do",
            type : "POST",
            data : datas ,
            success : function(result){
                if(result.success){
                    layer.msg("删除许可成功!", {time:1000, icon:6}, function(){
                        var treeObj = $.fn.zTree.getZTreeObj("treeDemo");
                        treeObj.reAsyncChildNodes(null, "refresh");
                });
                }else{
                    layer.msg("删除许可失败!", {time:1000, icon:5});
                }
            }
```

```
                    });
                layer.close(cindex);
            }, function(cindex){
                layer.close(cindex);
            });
}
```

## 7.2 删除处理

- Handler
- Service
- Dao

```
void deletePermission(Integer id);
```

```
<delete id="deletePermission" parameterType="integer">
delete from t_permission where id=#{id}
</delete>
```

## 7.3 同步加载数据和异步加载数据两种方式比较

### 7.3.1 同步加载数据

```
$.ajax({
        url:"${APP_PATH}/permission/loadData.do",
        type:"post",
        success:function(result){
            if(result.success){
                var zNodes = result.data ;
                $.fn.zTree.init($("#treeDemo"), setting, zNodes);
```

```
            }
        }
    });
```

## 7.3.2 异步加载数据

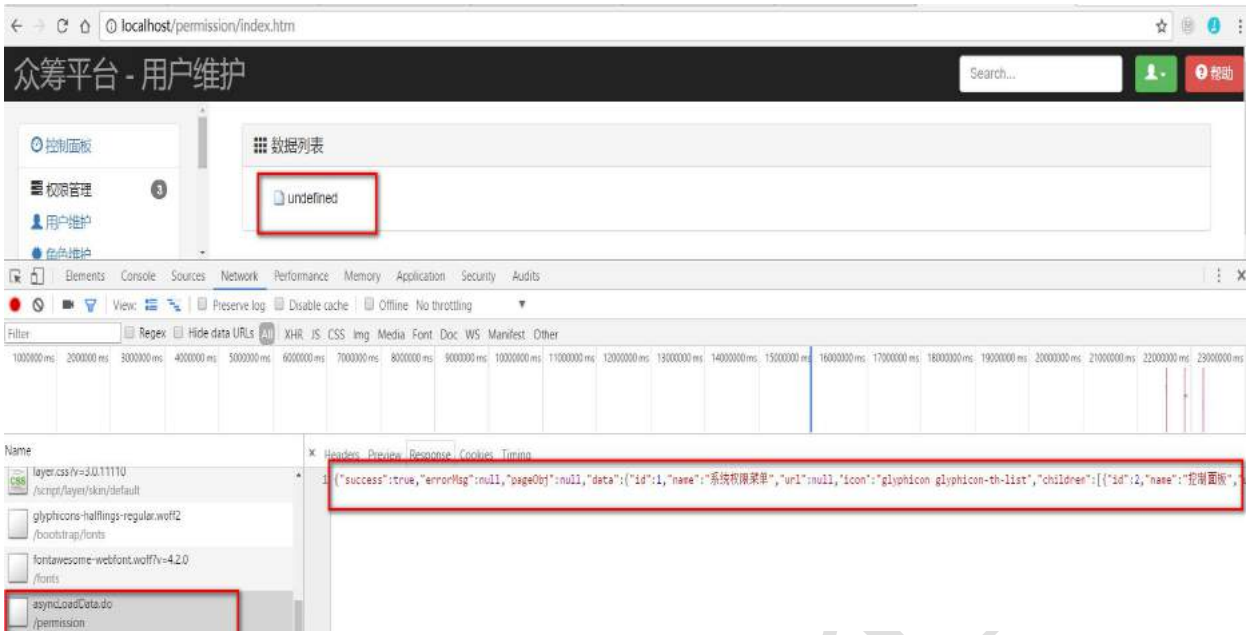
- 增加异步请求设置

```
var setting = {
                async: {
                    enable: true,
                    url:"${APP_PATH}/permission/asyncLoadData.do",
                    autoParam:["id", "name=n", "level=lv"]
                }
};
```

```
//异步加载
$.fn.zTree.init($("#treeDemo"), setting);
```

```
var treeObj = $.fn.zTree.getZTreeObj("treeDemo");
treeObj.reAsyncChildNodes(null, "refresh");
```

- 返回的结果必须是集合,不能返回 POJO 对象,否则页面无法加载许可数据;




### 7.3.3 注意事项

- 只有叶子节点可以删除。
- 也可以采用级联删除节点（递归）

# 附录 1.数据库中表与表的关系

## 1.1 一对一

例如 ： 丈夫和妻子

[1] 让其中一张表的主键,同时充当外键.

共享主键.

| husband | | Wife | |
|---|---|---|---|
| Id(PK) | name | Id(PK,FK) | name |
| 1 | zhangsan | 1 | lily |
| 2 | lisi | 2 | rose |

[2]在其中一张表中额外定义一个字段,作为外键,引用对端表的主键.

| husband | | Wife | | |
|---|---|---|---|---|
| Id(PK) | name | Id(PK) | name | hid(FK,unique) |
| 1 | zhangsan | 11 | lily | 2 |
| 2 | lisi | 21 | rose | 1 |

## 1.2 一对多,多对一

例如:部门和员工

[1]在多的一端表中增加外键,引用一的一端主键.

为什么将外键定义在多的一端,多的一端维护关系更加容易,方便.

| Dept | | Emp | | |
|---|---|---|---|---|
| Id(PK) | name | Id(PK) | name | deptid(FK) |
| 1 | DEV | 1 | zhangsan | 1 |
| 2 | TEST | 2 | lisi | 1 |

[2] 菜单(一对多和多对一关系,用一张表来进行描述-自关联)

自关联表的外键,不能设置 not null

| menu | | |
|---|---|---|
| Id | name | pid(fk) |
| 1 | 系统权限菜单 | |
| 2 | 控制面板 | 1 |
| 3 | 消息管理 | 1 |
| 4 | 权限管理 | 1 |
| 5 | 用户管理 | 4 |

| | | |
|---|---|---|
| 6 | 角色管理 | 4 |

## 1.3 多对多

在数据库中不能直接描述多对多关联关系的.

一般是借助于一张中间表,含有两个外键,分别引用多对多两个表的主键.

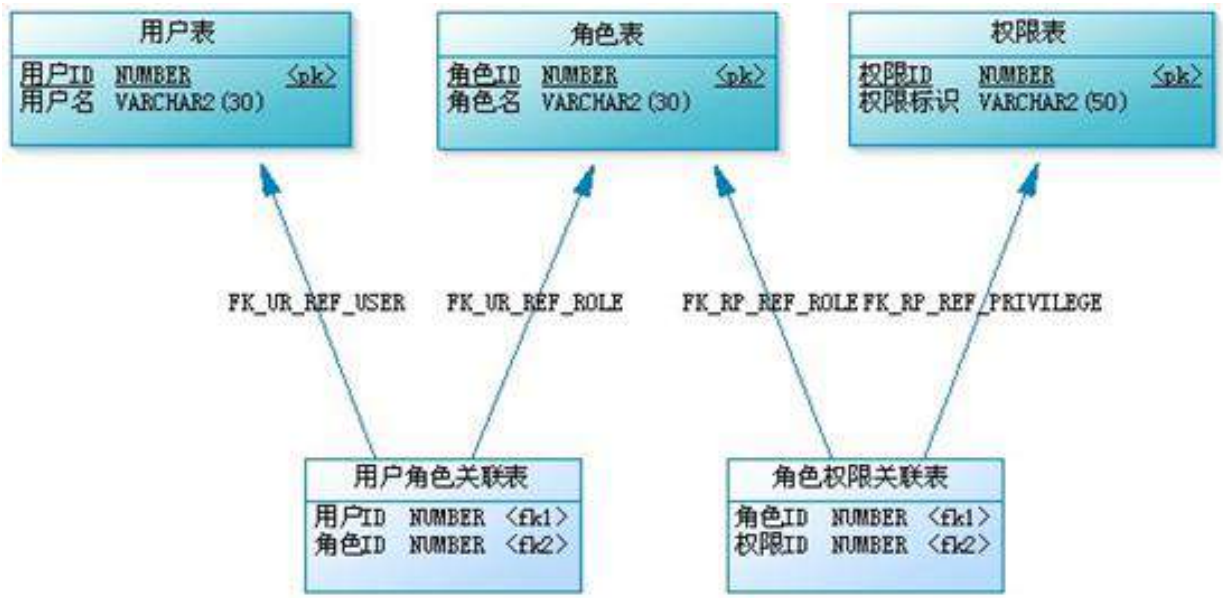
中间表可以有自己的主键,主键可以是自动增长的.

中间表也可以没有自己独立的主键,采用两个外键来充当主键,称为联合主键(表中多个主键,联合表示数据唯一性.).

中间表除了两个外键,可能含有其他的业务字段.例如:学生和课程之间多对多关系,中间表-成绩表,含有成绩信息.

| user | | role | |
|---|---|---|---|
| Id(PK) | name | Id(PK) | name |
| 1 | zhangsan | 1 | Dev |
| 2 | lisi | 2 | Test |

| User_role | | | |
|---|---|---|---|
| Id | userid(FK) | roleid(FK) | xxxx |
| 1 | 1 | 1 | |
| 2 | 1 | 2 | |
| 3 | 2 | 1 | |




## 1.4 定义表与表之间的关系

## 1.5 PK: Primary key 主键

- 代理主键:

表示由数据库来分配主键. 一般是没有业务含义的.只是用于标识一条数据,表示数据的唯一性.

例如:

MySQL : auto_increment

Oracle : sequence

- 自然主键:

表示业务来维护主键.

例如:注册用户时,采用用户名称作为主键.保存用户前要验证用户名称主键值在数据库中是否存在.

学号,身份证号,订单号

## 1.6 FK:Foreign Key 外键

# 附录 2.面向对象中类与类之间的关系

## 2.1 继承

- extends
- 单重继承.多层继承

## 2.2 依赖

- 一般就是一个方法引用另外一个类的对象.
- 体现在局部变量使用.

```
Public class     UserTest{
   public void testUser(User user){    //UserTest 类依赖 User 类
      List list = new ArrayList();    // UserTest 类依赖 List,ArrayList
   }
}
```

## 2.3 聚合/组合

- 体现整体与部分之间的关系.
- 描述方式与关联关系一样,都是用成员变量来描述的.只是从业务上看做聚合或组合.

```
Public class User{
   int id ;
   String name ;
   Address address ; //用户和地址直接可以描述为聚合或组合
}
```

```
Public class Address{
    int id ;
    String city ;
    …
}
```

## 2.4 关联关系

- 都是通过成员变量来描述的.要么是对象引用,要么是集合引用.
- [1]双向一对一

```
public classs Husband{
    private Wife wife ;
}
```

```
public class Wife{
    private Husband husband ;
}
```

- [2]单向一对多

```
public classs Dept{
    private List<Emp> empList ;
}
```

```
public class    Emp{


}
```

[3]单向多对一

```
public classs Dept{


}
```

```
public class    Emp{
   pirivate Dept dept ;
}
```

[3]双向多对多

```
public classs Student{
     private List<Course> courseList ;
}
```

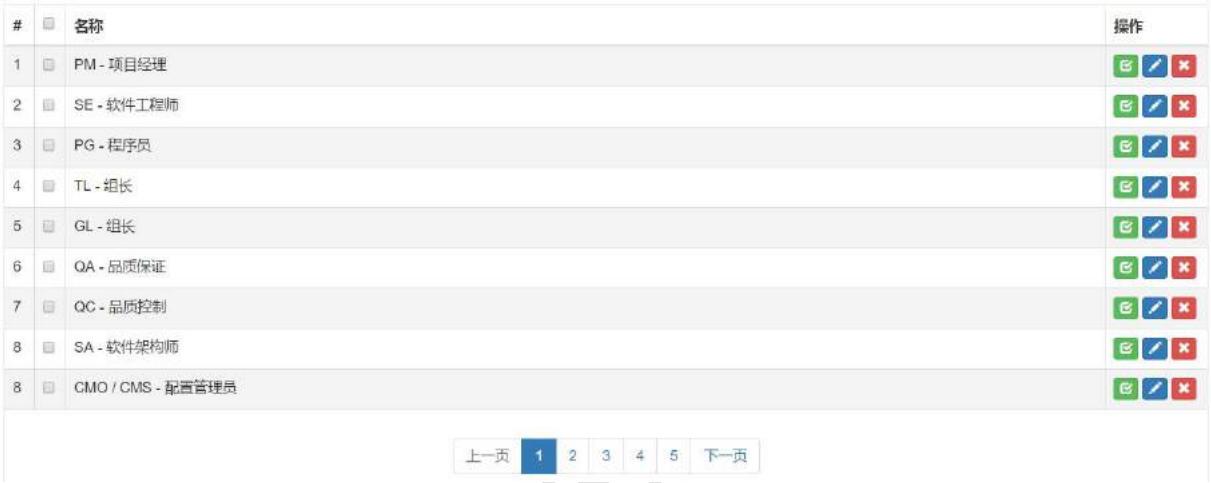
```
public class Course{
     pirivate List<Student> studentList ;
}
```

在面向对象中,描述对象与对象之间的关系,是存在单向和双向之分的.

在描述关系时,建立单向还是双向关系,是通过业务来决定的.

# 附录 3.数据展示方式

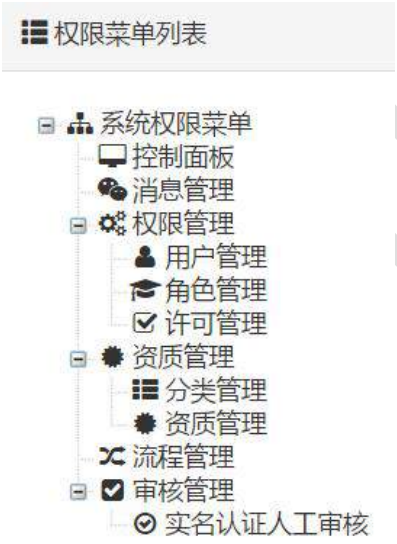
## 3.1 表格



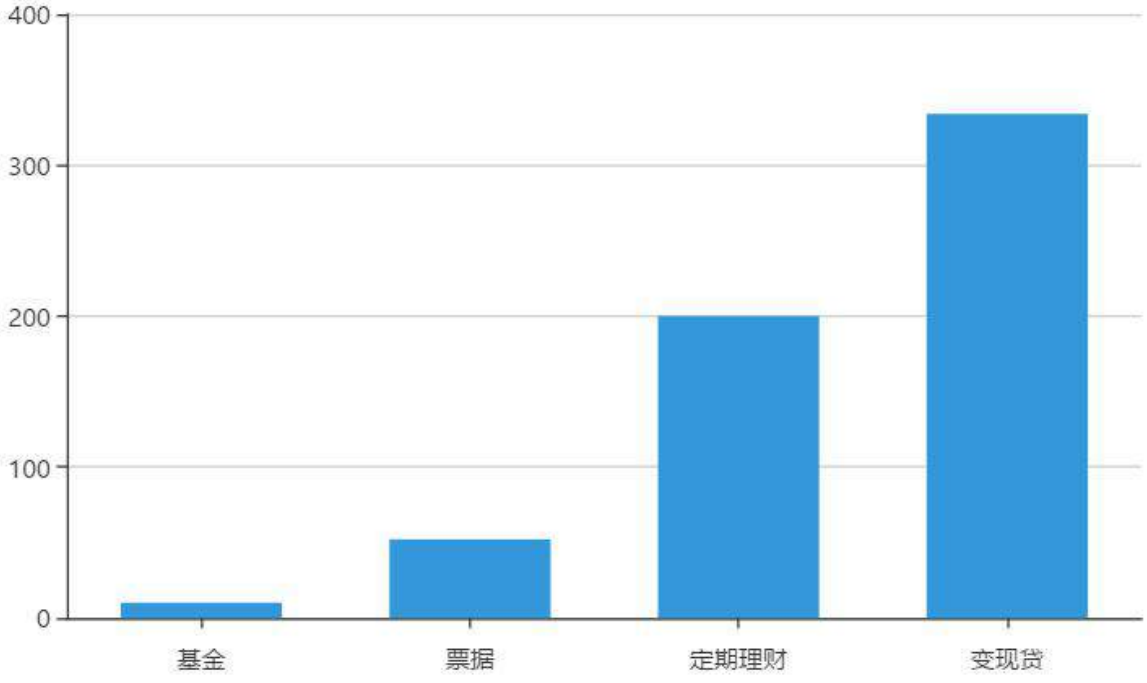
| # | | 名称 | 操作 |
|---|---|---|---|
| 1 | | PM - 项目经理 | |
| 2 | | SE - 软件工程师 | |
| 3 | | PG - 程序员 | |
| 4 | | TL - 组长 | |
| 5 | | GL - 组长 | |
| 6 | | QA - 品质保证 | |
| 7 | | QC - 品质控制 | |
| 8 | | SA - 软件架构师 | |
| 8 | | CMO / CMS - 配置管理员 | |

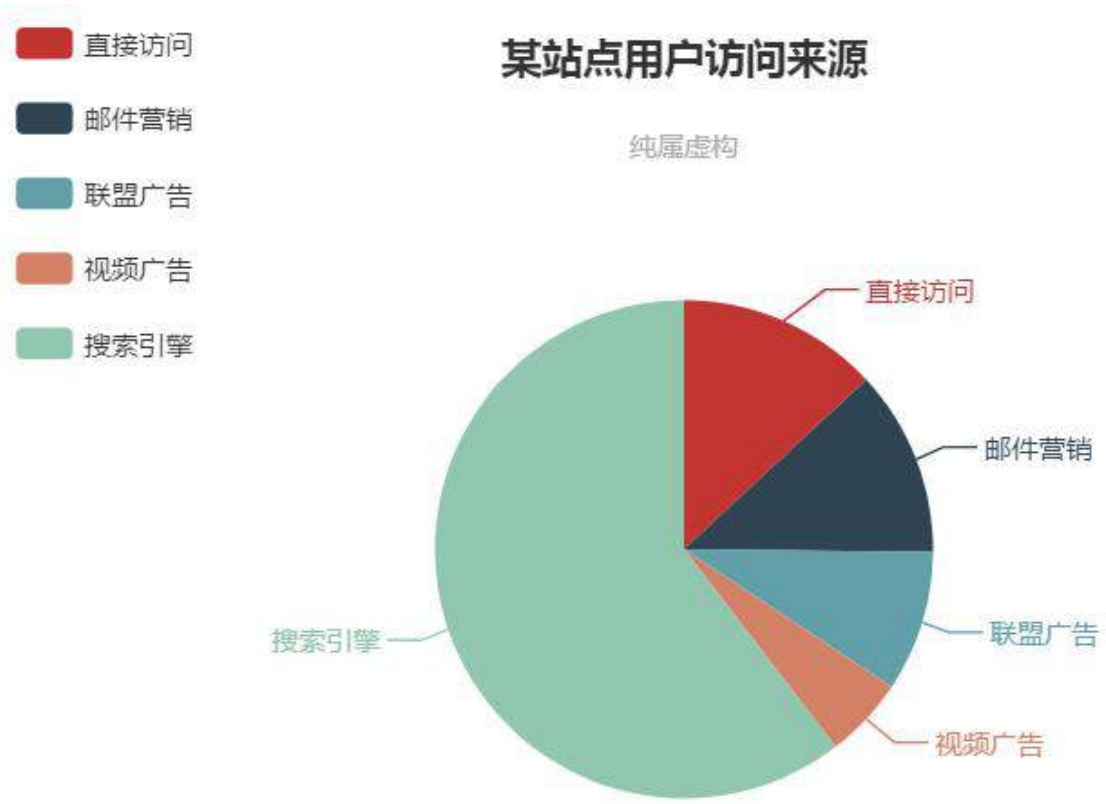
上一页 1 2 3 4 5 下一页

## 3.2 树状图



## 3.3 图表




# 附录 4.权限控制

## 4.1 权限都控制哪些内容?

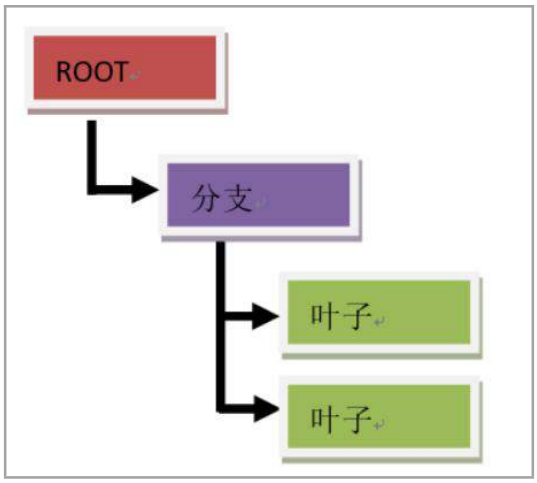
1. 控制菜单/路径是否能够访问
2. 控制页面按钮是否能够访问
3. 控制字段是否能够访问

# 附录 5.总结

## 5.1 树形结构的原理

- 关联数据形成上下级（父子）关系
- 通过查找父节点可以找到其中对应的子节点

- 树形结构的根节点只有一个，但是分支节点，叶子节点可以存在多个




## 5.2 树形结构的组件操作

### 5.2.1 ztree 插件使用环境

- css:zTreeStyle.css
- js : jquery.js; jquery.ztree.all.min.js
- html : <ul id=”**treeDemo**” class=”ztree”>

### 5.2.2 具体代码开发

```
var setting = {};
var nodes = [{name:’xxx’}, {name:’yyyy’}];
$.fn.zTree.init($(“#treeDemo”), setting, nodes);
```

## 5.3 Ztree 配置对象 setting 含义

对树形结构进行相应的配置，如事件，复选框，异步加载等

```
事件
callback: {
```

```
    onClick : function(event, treeId, json) {
    }
}
```

异步加载

```
async: {
    enable: true,
    url:"xxxxx",
    autoParam:["id", "name=n", "level=lv"]
},
```

复选框

```
check : {
    enable : true
},
```

## 5.4 ztree 节点对象 node 属性含义

```
var node = {
    name :    ‘xxxx’ , //  节点名称
    open: true, //  父节点是否展开
    icon :  ‘xxx’,//  图标图片的  url  可以是相对路径也可以是绝对路径
    checked:true,//  节点的  checkBox / radio  的  勾选状态
    children:[{}, {}] //节点的子节点数据集合
    level:1//记录节点的层级(不能由后台生成，是 ztree 组件自动生成的)
}
```

treeNode 对象的 ztree 的树形节点对象，包含了树形节点数据的所有属性字段和 ztree 自己生成的属性

- **ztree 树形数据操作**

同步读取数据

```
var setting = {
        view: {
                selectedMulti: false,
        },
        callback: {
                onClick : function(event, treeId, json) {


                }
        }
}
Var d = [{}, {}…];
$.fn.zTree.init($("#treeDemo"), setting, d);
```

异步读取数据

```
var setting = {
        view: {
                selectedMulti: false,
        },
        async: {
                enable: true,
                url:"xxxxx",
                autoParam:["id", "name=n", "level=lv"]
        },
        callback: {
                onClick : function(event, treeId, json) {
```

```
            }
        }
    }
$.fn.zTree.init($("#treeDemo"), setting);
```

## 5.5 setting 解释

```
鼠标悬停时增加按钮
var setting = {
check: {
                enable: false //  复选框
},
view: {
            selectedMulti: false, //  多选

addDiyDom: function(treeId, treeNode){
                var icoObj = $("#" + treeNode.tId + "_ico");
                if ( treeNode.icon ) {
                    icoObj.removeClass("button ico_docu ico_open").addClass("fa fa-fw " +
treeNode.icon).css("background","");
                }
            },
addHoverDom: function(treeId, treeNode){
                var aObj = $("#" + treeNode.tId + "_a"); // #treeDemo_1_a
                aObj.attr("href", "javascript:;");
                var s = '<span id="btnGroup'+treeNode.tId+'">';
                if ( treeNode.level == 0 ) {
                    s += '<a>xxxx</a>';
```

```
                }
                s += '</span>';
                aObj.after(s);
},
removeHoverDom: function(treeId, treeNode){
                $("#btnGroup"+treeNode.tId).remove();
            }
        }
};
```

## 5.6 ztree 树形数据操作

```
读取当前树对象
var treeObj = $.fn.zTree.getZTreeObj("treeDemo");
刷新当前树对象的数据（异步）：树形结构数据必须异步加载
treeObj.reAsyncChildNodes(null, "refresh");
查询节点
var nodes = treeObj.getSelectedNodes(); // 获取选择的节点
var node = treeObj.getNodeByParam("id", 1, null);// 根据参数获取节点
var nodes = treeObj.getCheckedNodes(true); // 获取被选中的节点
增加新节点
var newNodes = [{name:"newNode1"}, {name:"newNode2"}, {name:"newNode3"}];
newNodes = treeObj.addNodes(node, newNodes);
修改节点
var treeObj = $.fn.zTree.getZTreeObj("tree");
var nodes = treeObj.getNodes();
if (nodes.length>0) {
nodes[0].name = "test";
```

```
treeObj.updateNode(nodes[0]);
}
删除节点
treeObj.removeChildNodes(parentNode);
treeObj.removeNode(currentNode);
```

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

1. 分配权限
   - 树形结构中，数据节点前增加复选框，根据选择的复选框的内容，对权限进行分配
2. 业务中复选框的 3 种状态
   - 未选中状态，选中状态，未完全选中状态
3. 分配权限的时候，采用先删除，再增加的方式完成相应功能
4. 权限拦截器
5. 文件上传原理
6. 广告模块-文件上传
7. jquery-form 插件
8. pagination 插件

# 第 1 章 给角色分配权限

## 1.1 显示分配许可树

### 1.1.1 在角色列表页面增加链接事件


### 1.1.2 处理事件，跳转页面：role/assign.jsp

### 1.1.3 页面异步加载许可树,并带有复选框

```
var setting = {
        check : {
            enable : true
        },
        async: {
            enable: true,
            url:"${APP_PATH}/permission/asyncLoadData.do?roleid="+${role.id},
            autoParam:["id", "name=n", "level=lv"]
        }
```

```
};
$.fn.zTree.init($("#treeDemo"), setting);
```

## 1.2 分配许可权限

## 1.3 给分配许可按钮增加事件处理


```
<button onclick="assignPermission()" class="btn btn-success">分配许可</button>
```

```
 //分配许可权限
function assignPermission(){
    var treeObj = $.fn.zTree.getZTreeObj("treeDemo");
    var nodes = treeObj.getCheckedNodes(true); //  获取被选中的节点

    if(nodes.length==0){
```

```
            layer.msg("请选择许可进行分配", {time:1000, icon:5, shift:6});
    }else{
            var data = {"roleid":"${role.id}"};
            $.each(nodes,function(i,n){
                    data["ids["+i+"]"] = n.id ;
            });
            $.ajax({
                    url : "${APP_PATH}/role/assignPermission.do",
                    type : "post",
                    data : data ,
                    success : function(result){
                        if(result.success){ //分配成功不需要刷新页面
                            layer.msg("给[${role.name}]分配许可成功", {time:1000, icon:6});
                        }else{
                        ayer.msg("给[${role.name}]分配许可失败", {time:1000, icon:5, shift:6});
                        }
                    }
            });
    }
}
```

## 1.4 处理请求

### 1.4.1 RoleHandler

```
@ResponseBody
@RequestMapping("/assignPermission")
public Object assignPermission(Data data,Integer roleid){
```

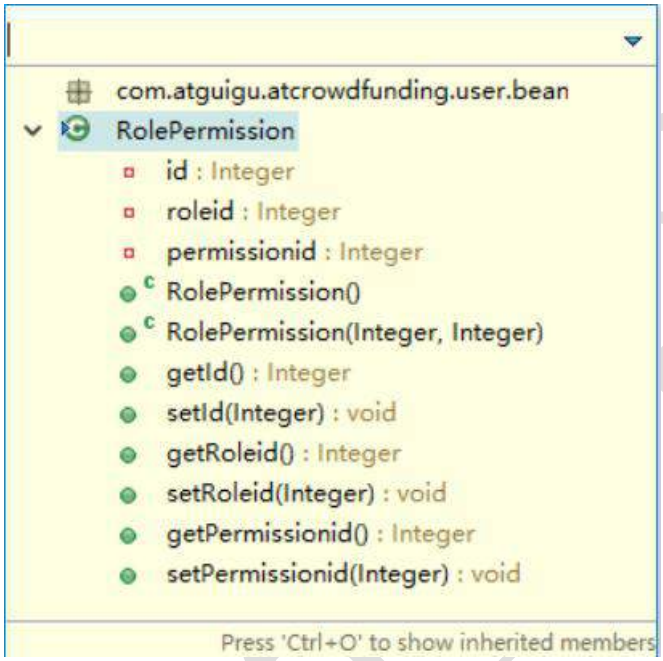
```
    start();
    try {
        int count = roleService.saveAssignPermission(roleid,data);
        if(count==datas.getIds().size()){
            success(true);
        }else{
            success(false);
        }
    } catch (Exception e) {
        e.printStackTrace();
        success(false);
    }
    return end();
}
```

## 1.4.2 RoleService

```
int saveAssignPermission(Integer roleid, Data data);
```

```
@Override
public int saveAssignPermission(Integer roleid, Data data) {
    roleDao.deleteRolePermissionByRoleid(roleid); //删除旧的分配
    int count = 0;
    List<Integer> ids = data.getIds();
    for (Integer permissionid : ids) {
        RolePermission rolePermission = new RolePermission();
        rolePermission.setRoleid(roleid);
        rolePermission.setPermissionid(permissionid);
```

```
            count += roleDao.insertRolePermission(rolePermission) ;
        }
        return count;
}
```




## 1.4.3 RoleDao

```
int insertRolePermission(RolePermission rolePermission);
void deleteRolePermissionByRoleid(Integer roleid);
```

```
<!-- 给角色分配许可 -->
<insert id="insertRolePermission" >
            insert into t_role_permission(roleid,permissionid) values(#{roleid},#{permissionid})
</insert>
<delete id="deleteRolePermissionByRoleid">
             delete from t_role_permission where roleid=#{roleid}
</delete>
```

## 1.5 回显功能

### 1.5.1 角色许可（已分配的角色许可，应该被选中）

- 原理

```
var zNodes =[
        { id:1, pId:0, name:"随意勾选  1", open:true},
        { id:11, pId:1, name:"随意勾选  1-1", open:true},
        { id:2, pId:0, name:"随意勾选  2", checked:true, open:true},
        { id:21, pId:2, name:"随意勾选  2-1"}
 ];
```

### 1.5.2 查看角色的许可权限

2.1 role/assign.jsp

```
async: {
        enable: true,
        url:"${APP_PATH}/permission/asyncLoadAssignData.do?roleid=${role.id}",
        autoParam:["id", "name=n", "level=lv"]
}
```

2.2 RoleHandler

```
//回显分配的角色许可
@ResponseBody
@RequestMapping("/asyncLoadAssignData")
public Object asyncLoadAssignData(Integer roleid){
        List<Integer> permissionids = permissionService.queryPermissionidByRoleid(roleid);
        //Demo 5  一次查询所有数据,使用 Map 存放数据进行判断,减少循环次数
        List<Permission> rootPermissions = new ArrayList<Permission>();
```

```
    List<Permission> permissons =    permissionService.queryAllPermission();
    Map<Integer,Permission> map = new HashMap<Integer,Permission>();
    for (Permission permission : permissons) {
        map.put(permission.getId(), permission);
        if(permissionids.contains(permission.getId())){
            permission.setChecked(true);
        }
    }

    for (Permission permission : permissons) {
        //子节点
        Permission child = permission ; //假设为子菜单
        if(child.getPid() == 0 ){
            rootPermissions.add(permission); //根节点
        }else{
            //父节点
            Permission parent = map.get(child.getPid());
            parent.getChildren().add(permission);
        }
    }
    return rootPermissions;
}
```

2.3 Service

```
List<Integer> queryPermissionidByRoleid(Integer roleid);
```

```
@Override
public List<Integer> queryPermissionidByRoleid(Integer roleid) {
    return permissionDao.queryPermissionidByRoleid(roleid);
```

```
}
```

2.4 Dao

```
List<Integer> queryPermissionidByRoleid(Integer roleid);
```

```
<select id="queryPermissionidByRoleid" resultType="int">
    select permissionid from t_role_permission where roleid=#{roleid}
</select>
```

# 第 2 章 登录用户的权限许可菜单

## 2.1 DispatcherController

```
/**
 * 执行登录验证,将操作结果转换为 JSON 字符串传递给浏览器
 */
@ResponseBody
@RequestMapping("/dologin")
public Object dologin(HttpSession session, String loginacct, String userpswd, String usertype) {
    // JAVA(Object) ==> NET(json string) ==> IE(JSON Object)
    start();

    Map<String, Object> paramMap = new HashMap<String, Object>();
    paramMap.put("loginacct", loginacct);
    paramMap.put("userpswd", MD5Util.digest(userpswd));
    // 返回查询结果
    User user = userService.queryUser(paramMap);

    if ( user == null ) {
```

```
        message("用户账号或密码不正确");
        success(false);
    } else {
        // 保存登录用户信息
        session.setAttribute(Const.LOGIN_USER, user);
        success(true);

        List<Permission> loginPermissions = userService.queryPermissionByUserid(user.getId());

        Permission rootPermission = null;
        Map<Integer,Permission> map = new HashMap<Integer,Permission>();
        for (Permission permission : loginPermissions) {
            map.put(permission.getId(), permission);
        }

        for (Permission permission : loginPermissions) {
            if(permission.getPid()==0){
                rootPermission = permission ;
            }else{
                Permission parent = map.get(permission.getPid());
                parent.setOpen(true);
                parent.getChildren().add(permission);
            }
        }

        session.setAttribute(Const.LOGIN_ROOT_PERMISSION, rootPermission);
    }
```

```
    // 在网络中传递的 json 字符串中的属性一定要使用双引号包含。
    //resp.getWriter().print("{age:20}");
    return end();
}
```

## 2.2 Service

```
public List<Permission> queryPermissionByUserid(Integer userid);
```

```
@Override
public List<Permission> queryPermissionByUserid(Integer userid) {
    return userDao.queryPermissionByUserid(userid);
}
```

## 2.3 Dao

```
List<Permission> queryPermissionByUserid(Integer userid);
```

```
<select id="queryPermissionByUserid" resultType="Permission">
select distinct t_permission.* from
t_user ,t_user_role,t_role ,t_role_permission,t_permission
where
t_user.id=t_user_role.userid
and t_user_role.roleid=t_role.id
and t_role.id= t_role_permission.roleid
and t_role_permission.permissionid= t_permission.id
and t_user.id=#{userid}
</select>
```

## 2.4 其他 SQL 实现方式

参考:

```
<select id="queryPermissionByUserid" resultType="Permission">
select
        *
    from t_permission
    where id in (
        select
            pid
        from t_role_permission
        where rid in (
            select
                rid
            from t_user_role
            where uid = #{userid}
        )
    )
</select>
```

## 2.5 显示用户的许可树菜单:menu.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<%@ taglib prefix="fn" uri="http://java.sun.com/jsp/jstl/functions"  %>
<ul style="padding-left:0px;" class="list-group">
<c:forEach items="${sessionScope.rootPermission.children }" var="permission">
<c:if test="${empty permission.children }">
```

```
<li class="list-group-item tree-closed">
<a href="${APP_PATH }/${permission.url}"><i class="${permission.icon }"></i>
${permission.name }</a>
</li>
</c:if>
<c:if test="${not empty permission.children }">
<li class="list-group-item">
<span><i class="${permission.icon }"></i> ${permission.name } <span class="badge"
style="float:right">${fn:length(permission.children) }</span></span>
<ul style="margin-top:10px;">
<c:forEach items="${permission.children }" var="childPermission">
<li style="height:30px;">
<a href="${APP_PATH }/${childPermission.url}"><i class="${childPermission.icon }"></i>
${childPermission.name }</a>
</li>
</c:forEach>
</ul>
</li>
</c:if>
</c:forEach>
</ul>
```

## 2.6 定义/WEB-INF/jsp/common/menu.jsp 页面

- 其他页面都 包含/menu.jsp

```
<%@ include file="/WEB-INF/jsp/common/menu.jsp" %>
```

# 第 3 章 登录权限拦截器

## 3.1 演示

- 没有给 zhangsan 分配用户管理权限许可菜单,登录后菜单无法看到,但是通过地址栏还是可以进行访问-系统不安全
- 没有登录系统,依然可以通过地址栏访问系统路径
- 分析:
  - 哪些路径允许访问
    - 帮助
    - 主页
  - 哪些路径不能访问
    - 菜单所有路径

## 3.2 登录权限拦截器

### 3.2.1 定义登录拦截器

```
//登录拦截器,未登录系统,禁止访问相关资源
public class LoginInterceptor extends HandlerInterceptorAdapter {
    @Override
    public boolean preHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler) throws Exception {
        System.out.println("LoginInterceptor - preHandle");
        String requestURI = request.getRequestURI();
        //StringBuffer requestURL = request.getRequestURL();
        System.out.println("requestURI="+requestURI);
        //System.out.println("requestURL="+requestURL);
```

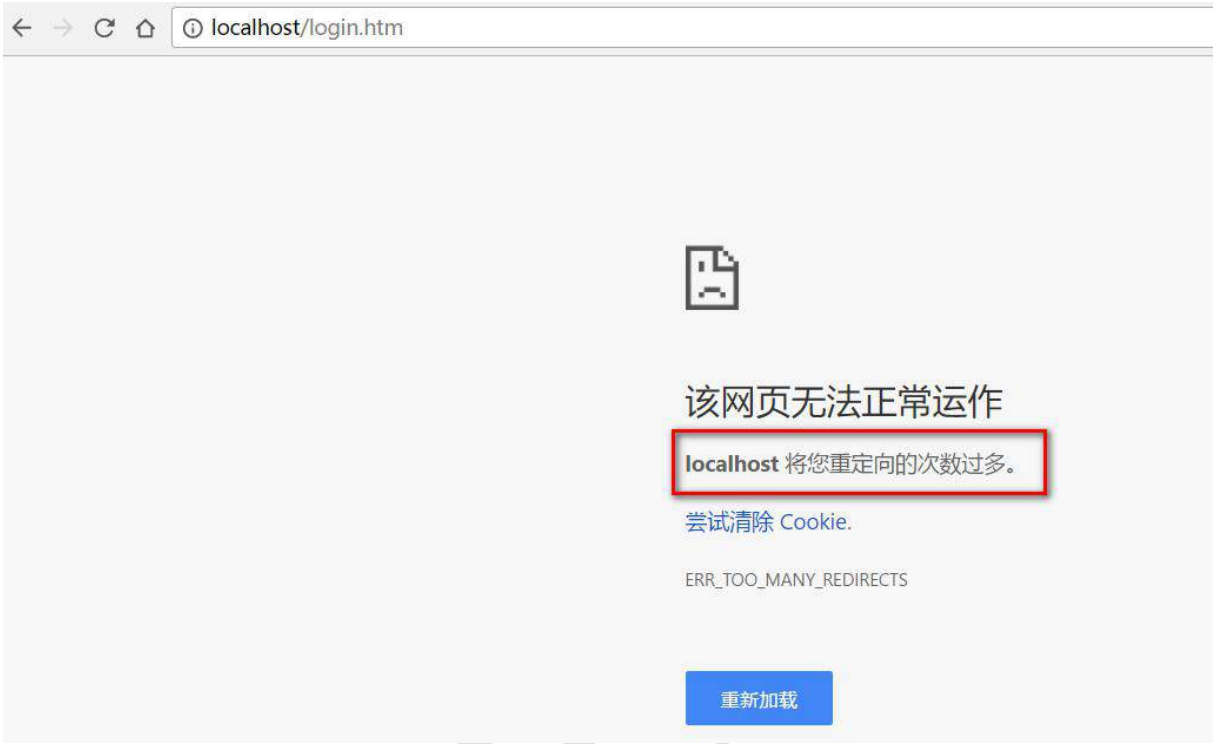
```
        //白名单
        Set<String> uris = new HashSet<String>();
        uris.add("/dologin.do");
        uris.add("/login.htm"); //必须增加到白名单,否则,死循环
        uris.add("/index.htm");
        if(uris.contains(requestURI)){
            return true ;
        }else{
            HttpSession session = request.getSession();
            User user    = (User)session.getAttribute(Const.LOGIN_USER);
            if(user==null){
                response.sendRedirect(request.getContextPath()+"/login.htm");
                return false ;
            }else{
                return true ;
            }
        }
    }
}
```

### 3.2.2 声明拦截器

```
<mvc:interceptors>
    <bean id="loginInterceptor" class="com.atguigu.atcrowdfunding.interceptor.LoginInterceptor"></bean>
</mvc:interceptors>
```

### 3.2.3 问题

登录路径需要增加到白名单,否则出现死循环.




# 第 4 章 访问权限拦截器

## 4.1 访问权限拦截器

### 4.1.1 定义访问权限拦截器

```
package com.atguigu.atcrowdfunding.interceptor;


import java.util.HashSet;
import java.util.List;
import java.util.Set;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.servlet.handler.HandlerInterceptorAdapter;

import com.atguigu.atcrowdfunding.common.Const;
import com.atguigu.atcrowdfunding.user.bean.Permission;
import com.atguigu.atcrowdfunding.user.service.PermissionService;
import com.atguigu.atcrowdfunding.util.StringUtil;

//权限拦截器,未授权禁止访问系统相关资源
public class AuthInterceptor extends HandlerInterceptorAdapter {
    @Autowired
    private PermissionService permissionService;

    @Override
    public boolean preHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler) throws Exception {
        System.out.println("AuthInterceptor - preHandle");
        String requestURI = request.getRequestURI();
        //获取系统所有许可权限菜单,请求路径必须包含于许可菜单中,否则拒绝访问
        List<Permission> allPermission = permissionService.queryAllPermission();
        Set<String> allPermissionUrls = new HashSet<String>();
        for (Permission permission : allPermission) {
            if(StringUtil.isNotEmpty(permission.getUrl())){
```

```
                allPermissionUrls.add("/"+permission.getUrl());
            }
        }
        if(allPermissionUrls.contains(requestURI)){
            //获取当前用户角色的许可菜单,请求路径必须包含于许可菜单中,否则拒绝访问
            HttpSession session = request.getSession();
            Set<String> authUris =
 (Set<String>)session.getAttribute(Const.LOGIN_AUTH_PERMISSION_URI);
            if(authUris.contains(requestURI)){
                return true ;
            }else{
                response.sendRedirect(request.getContextPath()+"/error.htm");
                return false ;
            }
        }else{ //不需要权限控制的,直接放行
            return true ;
        }
    }
}
```

### 4.1.2 声明拦截器

```
<mvc:interceptors>
  <bean id="loginInterceptor" class="com.atguigu.atcrowdfunding.interceptor.LoginInterceptor"></bean>
  <bean id="authInterceptor" class="com.atguigu.atcrowdfunding.interceptor.AuthInterceptor"></bean>
</mvc:interceptors>
```

### 4.1.3 修改登录方法

```
/**
 * 执行登陆验证,将操作结果转换为 JSON 字符串传递给浏览器
 * @return
 */
@ResponseBody
@RequestMapping("/dologin")
public Object dologin(HttpSession session, String loginacct, String userpswd, String usertype) {
    // JAVA(Object) ==> NET(json string) ==> IE(JSON Object)
    start();
    Map<String, Object> paramMap = new HashMap<String, Object>();
    paramMap.put("loginacct", loginacct);
    paramMap.put("userpswd", MD5Util.digest(userpswd));
    // 返回查询结果
    User user = userService.queryUser(paramMap);
    if ( user == null ) {
        message ("用户账号或密码不正确");
        success(false);
    } else {
        // 保存登陆用户信息
        session.setAttribute(Const.LOGIN_USER, user);
        success(true);
        List<Permission> loginPermissions = userService.queryPermissionByUserid(user.getId());
        Permission rootPermission = null;
        Map<Integer,Permission> map = new HashMap<Integer,Permission>();
        Set<String> authUris = new HashSet<String>();
```

```
            for (Permission permission : loginPermissions) {
                map.put(permission.getId(), permission);
                if(StringUtil.isNotEmpty(permission.getUrl())){
                    authUris.add("/"+permission.getUrl());
                }
            }
            for (Permission permission : loginPermissions) {
                if(permission.getPid()==0){
                    rootPermission = permission ;
                }else{
                    Permission parent = map.get(permission.getPid());
                    parent.setOpen(true);
                    parent.getChildren().add(permission);
                }
            }

            session.setAttribute(Const.LOGIN_ROOT_PERMISSION, rootPermission);
            session.setAttribute(Const.LOGIN_AUTH_PERMISSION_URI, authUris);
        }
        return end();
}
```

```
public static final String LOGIN_AUTH_PERMISSION_URI = "authUris";
```

### 4.1.4 注意事项

所有角色都应该拥有对[控制面板:main.htm] 访问权限.

# 第 5 章 访问权限拦截器-改善性能(监听器)

- 拦截器使用到的系统所有许可信息是每一次请求都会使用到的数据.不必每次都查询数据库,否则,效率很低.
- 使用监听器,在服务器一启动时,将所有的许可信息存放到 application 域/二级缓存中,提供给拦截器使用.

## 5.1 修改监听器

```
public class ServerStartupListener implements ServletContextListener {
    public void contextDestroyed(ServletContextEvent arg0) {}

    public void contextInitialized(ServletContextEvent evt) {
        // 将当前 web 应用路径保存到指定的范围中，让所有的用户可以访问
        ServletContext application = evt.getServletContext();
        String path = application.getContextPath();

        // 在应用范围中保存用户共享的数据
        application.setAttribute("APP_PATH", path);

        //-----------------------------------------

        //获取系统中所有权限许可菜单
        ApplicationContext ioc = WebApplicationContextUtils.getWebApplicationContext(application);
        PermissionService permissionService = ioc.getBean(PermissionService.class); //可能报空指针异常,监听器配置顺序问题.
        List<Permission> allPermission = permissionService.queryAllPermission();
        Set<String> allPermissionUrls = new HashSet<String>();
```

```
            for (Permission permission : allPermission) {
                if(StringUtil.isNotEmpty(permission.getUrl())){
                    allPermissionUrls.add("/"+permission.getUrl());
                }
            }
            application.setAttribute(Const.ALL_PERMISSION_URI, allPermissionUrls);
            //---------------------------------------------
            //加载广告信息
            //加载热点项目信息
            //加载分类项目信息
        }
}
```

## 5.2 修改访问权限拦截器

```
//权限拦截器,未授权禁止访问系统相关资源
public class AuthInterceptor extends HandlerInterceptorAdapter {
    @Autowired
    private PermissionService permissionService;

    @Override
    public boolean preHandle(HttpServletRequest request,
    HttpServletResponse response, Object handler) throws Exception {
        System.out.println("AuthInterceptor - preHandle");

        String requestURI = request.getRequestURI();

        //获取系统所有许可权限菜单,请求路径必须包含于许可菜单中,否则拒绝访问
```

```
        /*List<Permission> allPermission = permissionService.queryAllPermission();
        Set<String> allPermissionUrls = new HashSet<String>();
        for (Permission permission : allPermission) {
            if(StringUtil.isNotEmpty(permission.getUrl())){
                allPermissionUrls.add("/"+permission.getUrl());
            }
        }*/
        ServletContext application = request.getSession().getServletContext();
        Set<String> allPermissionUrls =
(Set<String>)application.getAttribute(Const.ALL_PERMISSION_URI);

        if(allPermissionUrls.contains(requestURI)){
            //获取当前用户角色的许可菜单,请求路径必须包含于许可菜单中,否则拒绝访问
            HttpSession session = request.getSession();

            Set<String> authUris =
(Set<String>)session.getAttribute(Const.LOGIN_AUTH_PERMISSION_URI);

            if(authUris.contains(requestURI)){
                return true ;
            }else{
                response.sendRedirect(request.getContextPath()+"/error.htm");
                return false ;
            }

        }else{ //不需要权限控制的,直接放行
            return true ;
```

```
        }
    }
}
```

## 5.3 注意事项

- 自定义监听器放置在框架监听器后执行.
- 首页数据存放到 application 中
  - 广告
  - 热点项目(关注度高的项目)
  - 分类项目

# 第 6 章 广告模块-文件上传-功能实现

## 6.1 定义表单

```
<form id="advertForm" method="post" action="" enctype="multipart/form-data">
   <div class="form-group">
<label for="name">广告名称</label>
<input type="text" class="form-control" id="name" name="name" placeholder="请输入广告名称">
   </div>
   <div class="form-group">
<label for="url">广告地址</label>
<input type="text" class="form-control" id="url" name="url" placeholder="请输入广告地址">
   </div>
   <div class="form-group">
<label for="advpic">广告图片</label>
<input type="file" class="form-control" id="advpic" name="advpic" placeholder="请输入广告图片">
```

```
  </div>
  <button id="saveBtn" type="button" class="btn btn-success"><i class="glyphicon glyphicon-plus"></i>新增</button>
  <button type="button" class="btn btn-danger"><i class="glyphicon glyphicon-refresh"></i> 重置</button>
</form>
```

## 6.2 提交表单;采用同步方式提交表单

```
$(function(){
        $("#saveBtn").click(function(){
                $("#advertForm").attr("action","${APP_PATH}/advert/doAdd.do");
                $("#advertForm").submit();
    });
});
```

## 6.3 处理请求

```
/**
 * 新增
 * @return
 */
@ResponseBody
@RequestMapping("/doAdd")
public Object doAdd(HttpServletRequest request, Advert advert ,HttpSession session) {
    start();
    try {
        MultipartHttpServletRequest mreq = (MultipartHttpServletRequest)request;
```

```
            MultipartFile mfile = mreq.getFile("advpic");
            String name = mfile.getOriginalFilename();//java.jpg
            String extname = name.substring(name.lastIndexOf(".")); // .jpg
            String iconpath = UUID.randomUUID().toString()+extname; //232243343.jpg
            ServletContext servletContext = session.getServletContext();
            String realpath = servletContext.getRealPath("/pic");
            String path =realpath+ "\\adv\\"+iconpath;
            mfile.transferTo(new File(path));
            User user = (User)session.getAttribute(Const.LOGIN_USER);
            advert.setUserid(user.getId());
            advert.setStatus("1");
            advert.setIconpath(iconpath);

            int count = advertService.insertAdvert(advert);
            success(count==1);
        } catch ( Exception e ) {
            e.printStackTrace();
            success(false);
        }
        return end();
}
```

# 第 7 章 Jquery 插件

## 7.1 JQuery 概述

- 优秀的 JavaScript 代码库（或 JavaScript 框架）
- 封装 JavaScript 常用的功能代码

- 优化 HTML 文档操作、事件处理、动画设计和 Ajax 交互
- 具有高效灵活的 css 选择器，并且可对 CSS 选择器进行扩展
- 拥有便捷的插件扩展机制和丰富的插件

## 7.2 JQuery 插件（类似 Eclipse 工具中的插件）

- jquery 插件就是一些程序员基于 JQuery 框架所写的一些 JS 工具方法。
- 可以实现 JQuery 框架未实现的功能或对某些功能进行封装。
- 使用插件时，必须先引入 JQuery 框架，然后引入插件代码。
- 在程序中访问插件提供的方法即可。

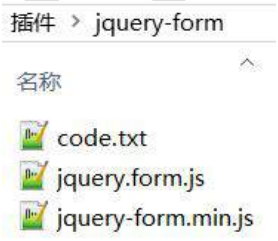
## 7.3 表单提交插件:jquery-form

## 7.4 分页页码插件:Pagination

## 7.5 页面的分页页码的展示效果可以采用插件实现:zTree

# 第 8 章 jquery-form 插件

## 8.1 jquery-form 插件




## 8.2 引入 jquery.form.js 插件

```
<script src="${APP_PATH }/jquery/jquery.form.js"></script>
```

## 8.3 Code.txt 参考

```
var options = {
    target: '#output',              //把服务器返回的内容放入 id 为 output 的元素中
    beforeSubmit: showRequest,    //提交前的回调函数
    success: showResponse,          //提交后的回调函数
    //url: url,                     //默认是 form 的 action，  如果申明，则会覆盖
    //type: type,                    //默认是 form 的 method（get or post），如果申明，则会覆盖
    //dataType: null,               //html(默认), xml, script, json...接受服务端返回的类型
    //clearForm: true,              //成功提交后，清除所有表单元素的值
    //resetForm: true,              //成功提交后，重置所有表单元素的值
    timeout: 3000                     //限制请求的时间，当请求大于 3 秒后，跳出请求
}

function showRequest(formData, jqForm, options){
    //formData: 数组对象，提交表单时，Form 插件会以 Ajax 方式自动提交这些数据，格式如：
[{name:user,value:val },{name:pwd,value:pwd}]
    //jqForm:    jQuery 对象，封装了表单的元素
    //options:   options 对象
    var queryString = $.param(formData);     //name=1&address=2
    var formElement = jqForm[0];                    //将 jqForm 转换为 DOM 对象
    var address = formElement.address.value;   //访问 jqForm 的 DOM 元素
    return true;   //只要不返回 false，表单都会提交,在这里可以对表单元素进行验证
};

function showResponse(responseText, statusText){
    //dataType=xml
```

```
    var name = $('name', responseXML).text();
    var address = $('address', responseXML).text();
    $("#xmlout").html(name + "    " + address);
    //dataType=json
    $("#jsonout").html(data.name + "    " + data.address);
};

$("#formID").ajaxSubmit(options);//jQuery 中没有这个函数，是通过插件后增加的，所以说插件是对
jQuery 框架的补充
```

# 第 9 章 广告模块-文件上传(jquery-form 插件)

## 9.1 异步提交文件上传表单

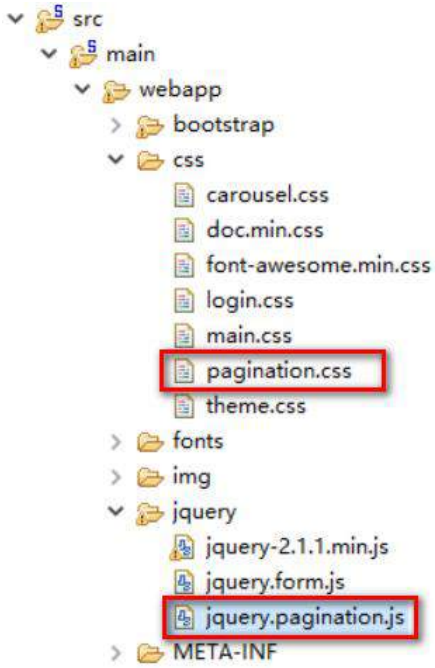
```
<script src="${APP_PATH }/jquery/jquery.form.js"></script>
<script type="text/javascript">
            $(function(){
                $("#saveBtn").click(function(){

                        var options = {
                            url:"${APP_PATH}/advert/doAdd.do",
                            beforeSubmit : function(){
                                loadingIndex = layer.msg('数据正在保存中', {icon: 6});
                                return true ;
                            },
                            success : function(result){
                                layer.close(loadingIndex);
```

```
                                if(result.success){
                                    layer.msg("广告数据保存成功", {time:1000, icon:6});
                                    window.location.href="${APP_PATH}/advert/index.htm";
                                }else{
                                    layer.msg("广告数据保存失败", {time:1000, icon:5, shift:6});
                                }
                            }
                        };
                        //通过 jquery 的 form 表单异步提交表单
                        $("#advertForm").ajaxSubmit(options);
                        return ;

                    });
                });
</script>
```

# 第 10 章 使用 pagination 插件分页

## 10.1 引用 pagination 插件的样式和 js 文件

```
<link rel="stylesheet" href="${PATH}/css/pagination.css">
<script src="${PATH}/jquery/jquery.pagination.js"></script>
```




## 10.2 页面/WEB-INF/jsp/advert/index.jsp

```
<tfoot>
<tr>
<td colspan="6" align="center">
<!-- <ul class="pagination">


</ul> -->
<div id="Pagination" class="pagination"><!-- 这里显示分页 --></div>
</td>
</tr>
</tfoot>
```

```
// 创建分页
var num_entries = pageObj.totalsize ;
$("#Pagination").pagination(num_entries, {
```

```
    num_edge_entries: 2, //边缘页数
    num_display_entries: 4, //主体页数
    callback: queryPage,
    items_per_page:pageObj.pagesize, //每页显示 1 项
    current_page:(pageObj.pageno-1), //当前页,索引从 0 开始
    prev_text:"上一页",
    next_text:"下一页"
});
```

## 10.3 死循环递归调用

**解决：**

jquery.pagination.js　注释调用 158 行

```
144                 }
145             }
146             this.nextPage = function(){
147                 if(current_page < numPages()-1) {
148                     pageSelected(current_page+1);
149                     return true;
150                 }
151                 else {
152                     return false;
153                 }
154             }
155             // 所有初始化完成，绘制链接
156             drawLinks();
157             // 回调函数
158             //opts.callback(current_page, this);
159         });
160 }
```

# 附录 1 树型

## 1.1 树形结构的同步与异步加载

### 1.1.1 同步：一次将树型结构全部加载

### 1.1.2 异步：点击父菜单加载子菜单进行逐级加载

同步读取数据

```
var setting = {
        view: {
                selectedMulti: false,
        },
        callback: {
                onClick : function(event, treeId, json) {
                }
        }
}
Var d = [{}, {}…];
$.fn.zTree.init($("#treeDemo"), setting, d);
```

异步读取数据

```
var setting = {
        view: {
                selectedMulti: false,
        },
        async: {
                enable: true,
                url:"xxxxx",
                autoParam:["id", "name=n", "level=lv"]
        },
        callback: {
                onClick : function(event, treeId, json) {
                }
        }
}
$.fn.zTree.init($("#treeDemo"), setting);
```

## 1.2 鼠标悬停的实现原理

```
addHoverDom: function(treeId, treeNode){
```

```
            var aObj = $("#" + treeNode.tId + "_a"); // #treeDemo_1_a
            aObj.attr("href", "javascript:;");
            var s = '<span id="btnGroup'+treeNode.tId+'">';
            if ( treeNode.level == 0 ) {
                s += '<a>xxxx</a>';
            }
            s += '</span>';
            aObj.after(s);
}
```

## 1.3 树型链接字体图片实现原理

```
addDiyDom: function(treeId, treeNode){
    var icoObj = $("#" + treeNode.tId + "_ico");
    if ( treeNode.icon ) {
        icoObj.removeClass("button ico_docu ico_open").addClass("fa fa-fw " + treeNode.icon).css("background","");
    }
}
```

## 1.4 树型链接鼠标悬停后离开效果实现原理

```
removeHoverDom: function(treeId, treeNode){
    $("#btnGroup"+treeNode.tId).remove();
}
```

## 1.5 树型的操作

**读取当前树对象**

```
var treeObj = $.fn.zTree.getZTreeObj("treeDemo");
```

刷新当前树对象的数据

```
treeObj.reAsyncChildNodes(null, "refresh");
```

**查询节点**

```
var nodes = treeObj.getSelectedNodes(); // 获取选择的节点
var node = treeObj.getNodeByParam("id", 1, null);// 根据参数获取节点
var nodes = treeObj.getCheckedNodes(true); // 获取被选中的节点
```

**增加新节点**

```
var newNodes = [{name:"newNode1"}, {name:"newNode2"}, {name:"newNode3"}];
newNodes = treeObj.addNodes(node, newNodes);
```

**修改节点**

```
var treeObj = $.fn.zTree.getZTreeObj("tree");
var nodes = treeObj.getNodes();
if (nodes.length>0) {
nodes[0].name = "test";
treeObj.updateNode(nodes[0]);
}
```

**删除节点**

```
treeObj.removeChildNodes(parentNode);
treeObj.removeNode(currentNode);
```

# 附录 2 文件上传

## 2.1 SpringMVC 框架的文件上传

### 2.1.1 前台

使用表单文件域完成文件上传<input type="file" name="xxxx">

表单必须采用 POST 方式提交数据，因为文件数据放置在请求体中传递，GET 方式处理不了，没有请求体

文件上传采用特殊的方式传递数据，而不是普通的名值对方式，

采用多部件方式提交， 所以在表单标签中需要增加特定属性 enctype="multipart/form-data"

### 2.1.2 配置

springMVC 框架需要提供文件上传解析器，需要在配置文件中进行配置

### 2.1.3 后台

springMVC 框架对请求进行了封装，所以可以通过封装后的请求对象来获取上传的文件数据

## 2.2 HTML5 文件上传

浏览器客户端判断上传文件的大小。

```
$("#文件域对象 ID").change(function(event){
        // 获取当前选择的文件 event.target.files
        var files = event.target.files, file;
        if (files && files.length > 0) {
            file = files[0];
        }
```

```
    // 判断上传文件的大小 file.size ，单位字节 Byte
                // 判断上传文件的类型 file.type
                // 本地生成上传文件后的临时文件地址
                var URL = window.URL || window.webkitURL;
                var imgURL = URL.createObjectURL(file);
});
```

# 作业

## 1.完成许可模块

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

1. FreeMarker 模板引擎技术
2. 系统部署架构
3. 分布式环境

# 第 1 章 FreeMarker 模板引擎技术

## 1.1 概念

FreeMarker 是一款模板引擎：

即一种基于模板和要改变的数据， 并用来生成输出文本（HTML 网页、电子邮件、配置文件、源代码等）的通用工具。

它不是面向最终用户的，而是一个 Java 类库，是一款程序员可以嵌入他们所开发产品的组件。

FreeMarker 是免费的，基于 Apache 许可证 2.0 版本发布。

其模板编写为 FreeMarker Template Language（FTL），属于简单、专用的语言。

需要准备数据在真实编程语言中来显示，比如数据库查询和业务运算， 之后模板显示已经准备好的数据。

在模板中，主要用于如何展现数据， 而在模板之外注意于要展示什么数据。

## 1.2 工作原理

所有的模板视图技术的工作原理基本类似，也就意味着 FreeMarker 和 JSP 基本差不多。

模板文件和数据模型是模板视图技术用来生成 HTML 页面所必须的组成部分。

数据模型（Java） + 模板文件（.ftl .jsp 文件）= 输出（HTML,XML,源码文件）

JSP 在 Web 系统中弥补了 servlet 生成 HTML 页面的不足，但只能应用于 WEB 系统，生成 HTML 页面。而 FreeMarker 不仅仅应用于 Web 系统，也可以应用于 JAVA 系统，还能生成 JAVA, XML 等文件，所以应用面更广。使用时，需要在项目 pom.xml 文件中增加依赖关系。

```
<dependencies>
    ...
    <dependency>
        <groupId>org.freemarker</groupId>
        <artifactId>freemarker</artifactId>
        <version>2.3.19</version>
    </dependency>
    ...
</dependencies>
```

如果应用在 SpringMVC 框架中，还需要增加相应的视图解析器。

```
<bean id="freemarkerConfig"
class="org.springframework.web.servlet.view.freemarker.FreeMarkerConfigurer">
        <!-- <property name="templateLoaderPath" value="/WEB-INF/ftl/" /> -->
        <property name="templateLoaderPaths">
            <list>
                <value>/WEB-INF/ftl/</value>
                <value>classpath:/ftl/</value>
            </list>
        </property>
```

```
        <property name="freemarkerSettings">
            <props>
                <prop key="defaultEncoding">UTF-8</prop>
<!--
FreeMarker 默认每隔 5 秒检查模板是否被更新，如果已经更新了，就会重新加载并分析模板。 但经常检查模板是否更新可能比较耗时。如果你的应用运行在生产模式下，而且你预期模板不会经常更新，则可以将更新的延迟时间延长至一个小时或者更久。 可以通过为 freemarkerSettings 属性设置 template_update_delay 达到这一目的,0 表示每次都重新加载
-->
                <prop key="template_update_delay">0</prop>
                <prop key="default_encoding">UTF-8</prop>
                <prop key="number_format">0.##########</prop>
                <prop key="datetime_format">yyyy-MM-dd HH:mm:ss</prop>
                <prop key="classic_compatible">true</prop>
                <prop key="template_exception_handler">ignore</prop>
            </props>
        </property>
</bean>

<bean class="org.springframework.web.servlet.view.freemarker.FreeMarkerViewResolver"
p:prefix="/" p:suffix=".ftl">
        <property name="cache" value="false" />
        <property name="viewClass"
 value="org.springframework.web.servlet.view.freemarker.FreeMarkerView" />
        <property name="contentType" value="text/html;charset=UTF-8"></property>
        <property name="exposeRequestAttributes" value="true" />
        <property name="exposeSessionAttributes" value="true" />
```

```
        <property name="exposeSpringMacroHelpers" value="true" />
        <property name="requestContextAttribute" value="base"></property>
        <property name="order" value="0"></property>
</bean>
```

## 1.3 基本语法

${...}：

FreeMarker 将会输出真实的值来替换大括号内的表达式，这样的表达式被称为 interpolation（插值）。

注释：

注释和 HTML 的注释也很相似，但是它们使用<#-- and -->来标识。不像 HTML 注释那样，FTL 注释不会出现在输出中（不出现在访问者的页面中），因为 FreeMarker 会跳过它们。

FTL 标签（FreeMarker 模板的语言标签）：

FTL 标签和 HTML 标签有一些相似之处，但是它们是 FreeMarker 的指令，是不会在输出中打印的。这些标签的名字以#开头。（用户自定义的 FTL 标签则需要使用@来代替#）

## 1.4 一个简单的 FreeMarker 例子

### 1.4.1 bean.ftl 代码

```
package ${packageName}.bean;
public class ${className} {
}
```

### 1.4.2 JAVA 代码

```
...
// 创建 Freemarker 对象的配置对象
Configuration cfg = new Configuration();
```

```
// 设定默认的位置（路径）
cfg.setDirectoryForTemplateLoading(new File("D:\\ftl"));
cfg.setDefaultEncoding("UTF-8");
cfg.setTemplateExceptionHandler(TemplateExceptionHandler.RETHROW_HANDLER);

// 数据
Map paramMap = new HashMap();

String className = "User";

paramMap.put("packageName", "com.atguigu.crowdfunding");
paramMap.put("className", className);

Template t = cfg.getTemplate("bean.ftl");
// 组合生成
Writer out = new OutputStreamWriter(new FileOutputStream(new File("D:\\User.java")), "UTF-8");
// 执行
t.process(paramMap, out);
...
```

## 1.5 常用语法

### 1.5.1 条件

```
<#if condition>
    ...
<#elseif condition2>
    ...
```

```
<#elseif condition3>
    ...
<#else>
    ...
</#if>
```

### 1.5.2 循环

```
<#list users as user>
    ${user.name}
</#list>
```

**索引:** ${user_index}

**序号:**临时变量+"_index"

### 1.5.3 包含

```
<#include "xxxx.ftl">
```

### 1.5.4 表达式

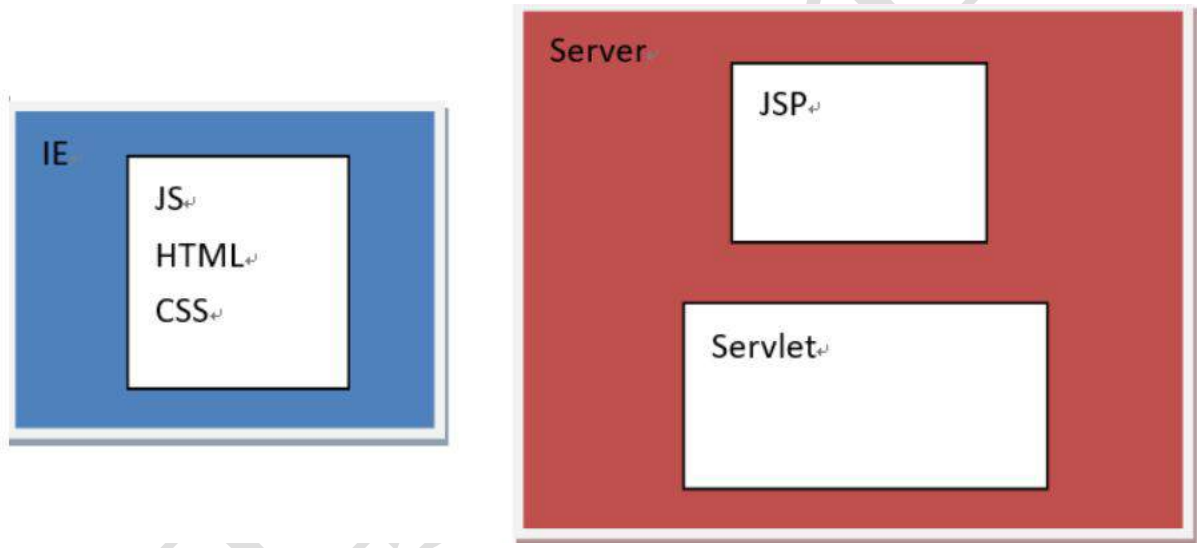
| 目录&文件 | 说明 |
| --- | --- |
| ${book.name?if_exists } | 用于判断如果存在,就输出这个值 |
| ${book.name?default(‘xxx’)} | 默认值 xxx |
| ${book.name!"xxx"} | 默认值 xxx |
| ${book.date?string('yyyy-MM-dd')} | 日期格式 |
| s?html | 对字符串进行 HTML 编码 |
| s?cap_first | 使字符串第一个字母大写 |

| | |
|---|---|
| s?lower_case | 将字符串转换成小写 |
| s?trim | 去掉字符串前后的空白字符 |
| s?size | 获得序列中元素的数目 |

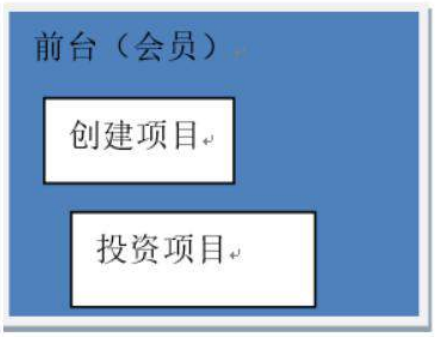
# 第 2 章 系统架构变化

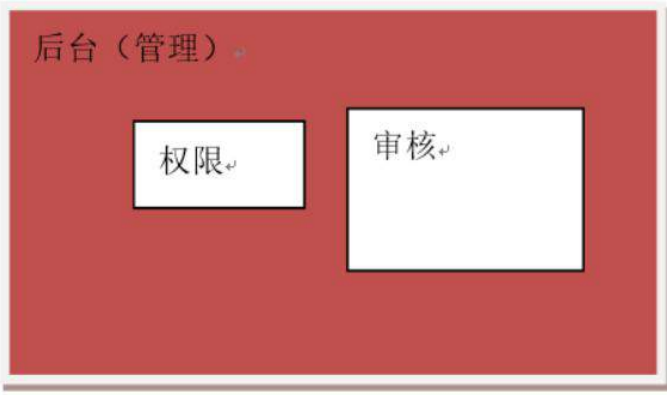
## 2.1 前台/后台-含义

### 2.1.1 代码

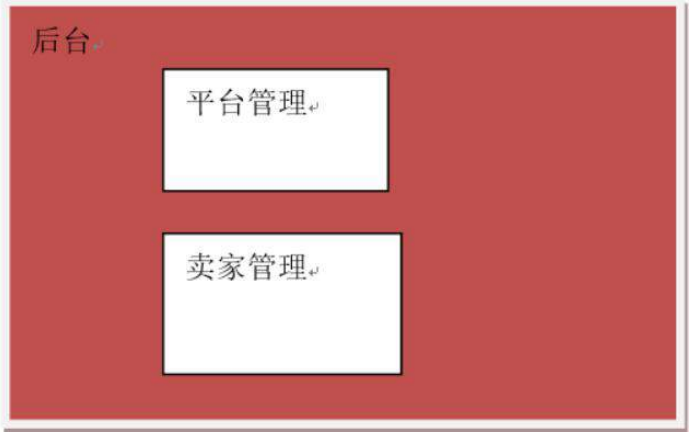


### 2.1.2 业务




### 2.1.3 系统


## 2.2 前台系统业务介绍

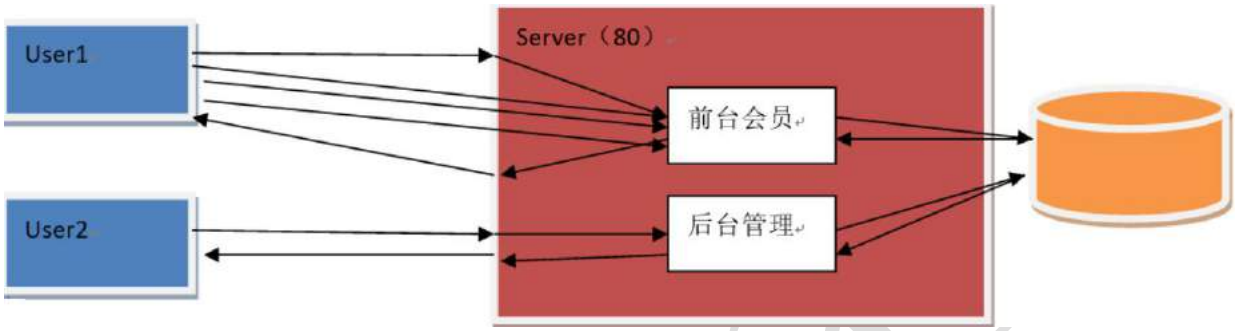
1. 发布众筹项目业务介绍
2. 审核发布项目
3. 查询众筹项目业务介绍
4. 投资众筹项目业务介绍
5. 实名认证申请业务介绍
6. 实名认证审核

## 2.3 当前系统架构

前台系统访问量大,大量线程被前台系统所占用.

后台系统访问量小,可能存在无法获取线程,系统无法使用情况

前台会员和后台管理其实还是作为一个完整的系统进行部署的.代码间调用还是处于一个进程中.

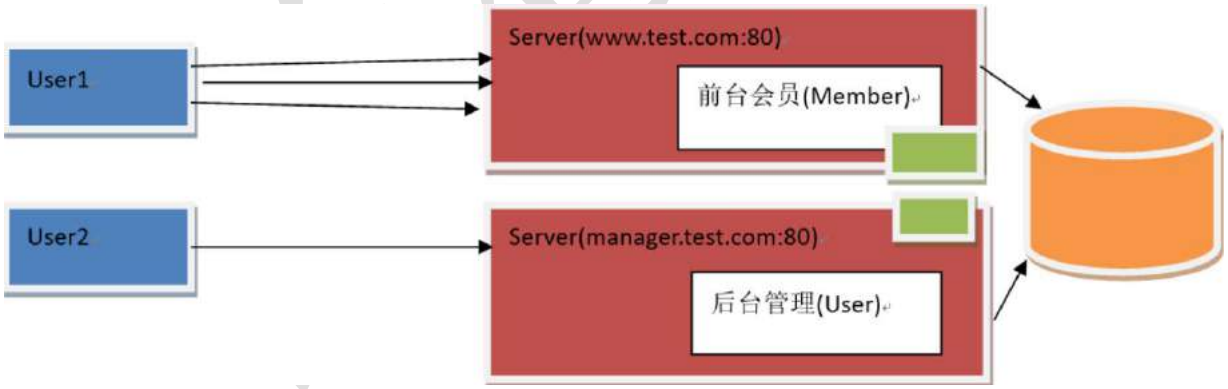



## 2.4 分布式系统架构

前台会员和后台管理两个系统独立部署

两个系统独立开,代码之间调用属于进程间调用.

Spring =>>EJB ==> RMI (Remote Method Invoke)　两个 JVM 之间的交互.



# 第 3 章 分布式环境概念介绍

## 3.1 概念

### 3.1.1 分布式

分布式系统 & 分布式用户

### 3.1.2 集群

多个服务器构成了一个群组，以单一系统的模式加以管理， 用以提高服务的可用性和可缩放性。

### 3.1.3 负载均衡 & 反向代理

负载： web 系统中一般指系统的用户承受能力

负载均衡：用多个服务器分担用户的访问，使多个服务器的负载达到均衡

反向代理 ： 不考虑具体提供的的某一台服务器，而是通过外置的代理服务器进行统一访问

### 3.1.4 Session 共享

服务器是独立的，一般情况下，无法共享 JVM 数据，所以 Session 无法共享，必须采用第三方组件(Memcached,Redis)实现 Session 共享

### 3.1.5 动静资源分离

动态资源和静态资源都会向服务器发送请求，那么大量的静态资源请求会导致服务器效率变低，运行变慢，所以进行资源的动静分离

## 3.2 软件

### 3.2.1 分布式-WebService

### 3.2.2 集群-N * Tomcat

### 3.2.3 负载均衡-Nginx

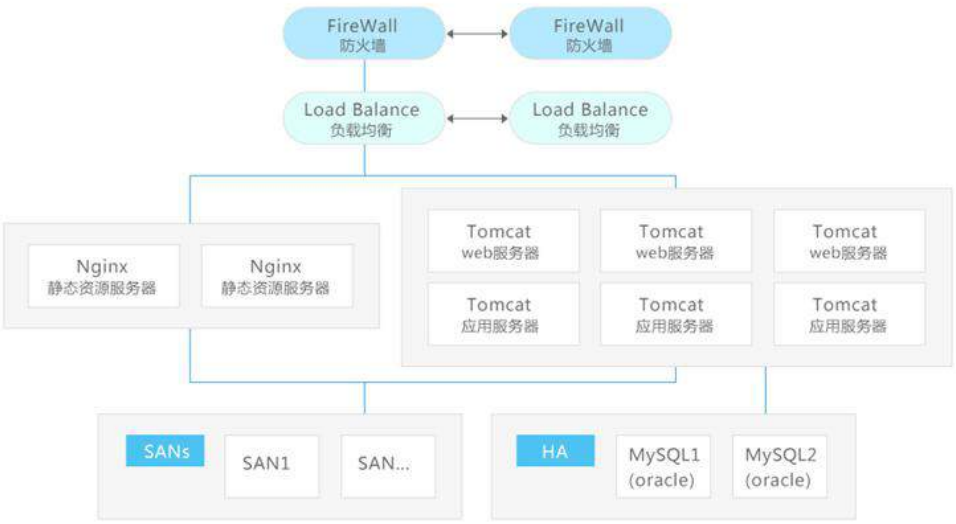
### 3.2.4 Session 共享-memcached

### 3.2.5 动静资源分离-Nginx

# 第 4 章 系统部署架构

## 4.1 系统部署架构

系统部署



### 4.1.1 HA(双机集群(HA)系统简称)

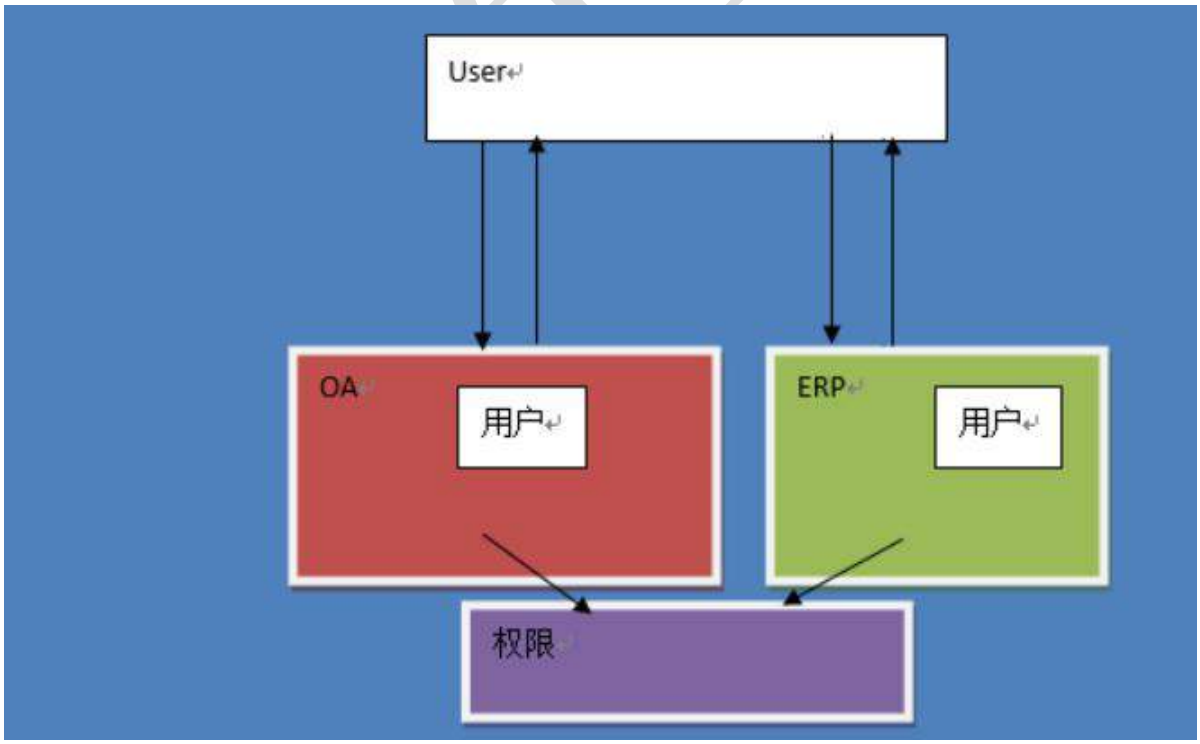
HA(High Available)，高可用性集群，是保证业务连续性的有效解决方案，一般有两个或两个以上的节点，且分为活动节点及备用节点。通常把正在执行业务的称为活动节点，而作为活动节点的一个备份的则称为备用节点。当活动节点出现问题，导致正在运行的业务（任务）不能正常运行时，备用节点此时就会侦测到，并立即接续活动节点来执行业务。从而实现业务的不中断或短暂中断

### 4.1.2 SAN(存储区域网络)

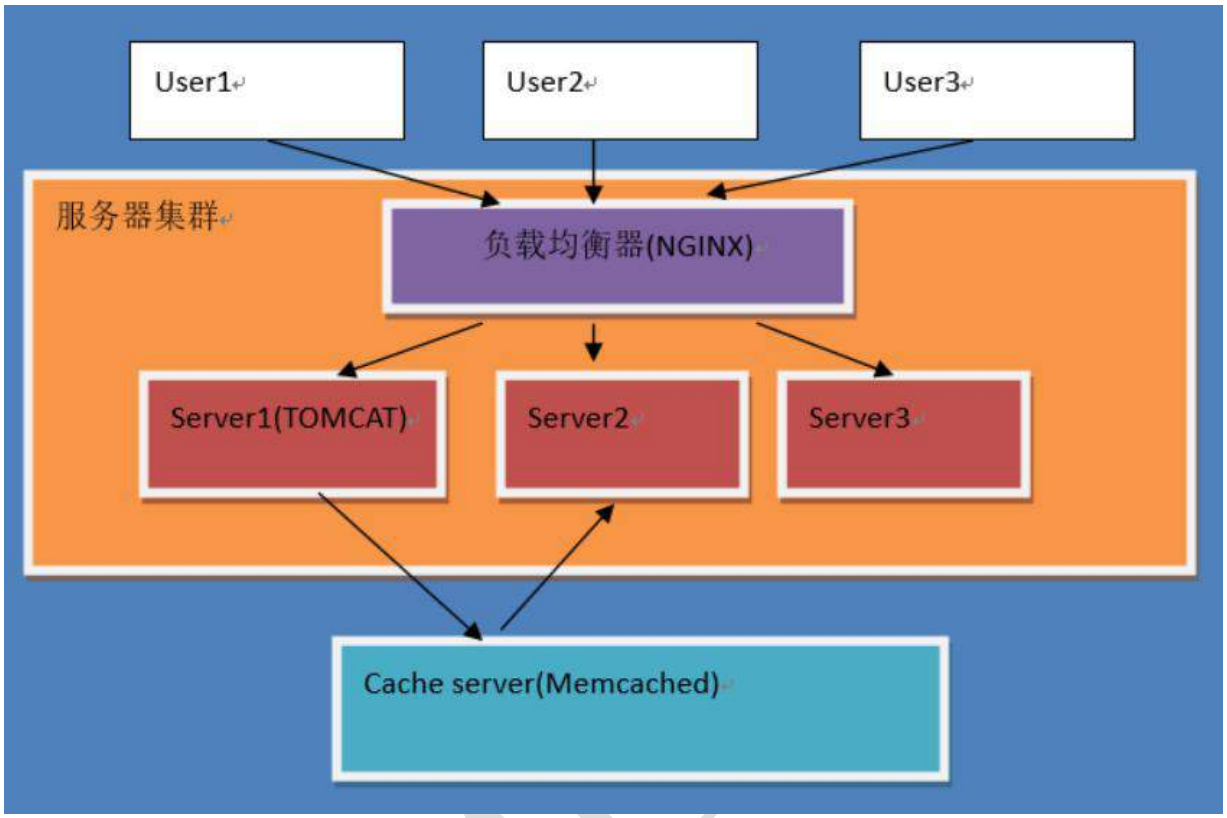
SAN 是 Storage Area Network 的缩写，即“存储区域网络”。SAN 专注于企业级存储的特有问题。

# 第 5 章 分布式环境-演示

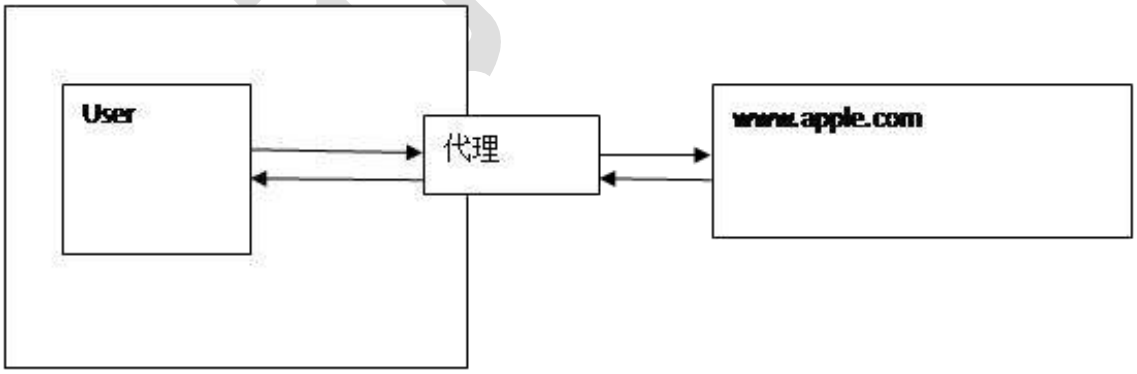
## 5.1 分布式 系统/业务



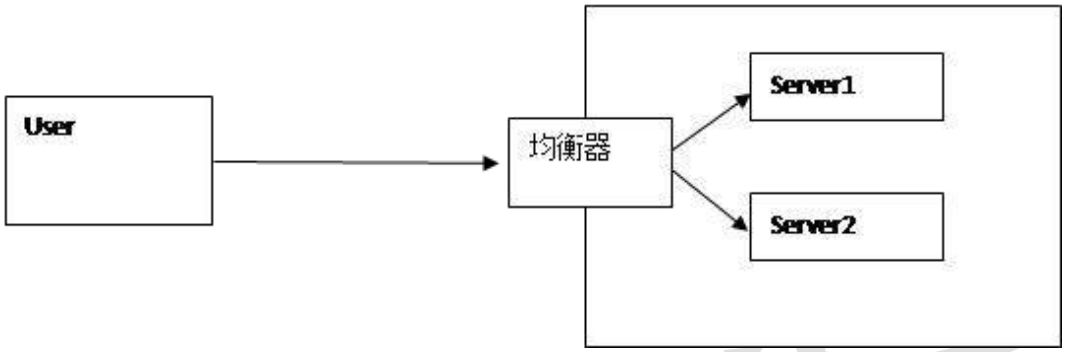
## 5.2 分布式用户




## 5.3 正向代理



## 5.4 反向代理




# 第 6 章 分布式环境搭建演示

## 6.1 安装 Nginx

解压缩 nginx-1.9.2.zip， 路径中不能包含中文名称

将【nginx-1.9.2.zip】解压到 【D:\Server\nginx-1.9.2】

## 6.2 安装 Tomcat 服务器

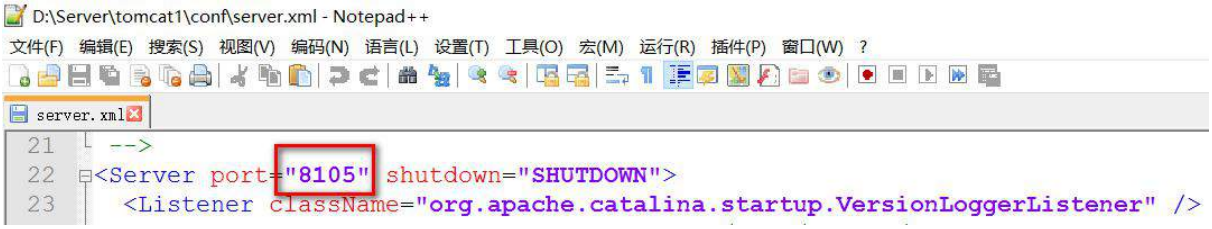
解压缩 tomcat.zip，建立两个服务器，端口号设定为分别为 8080,8181

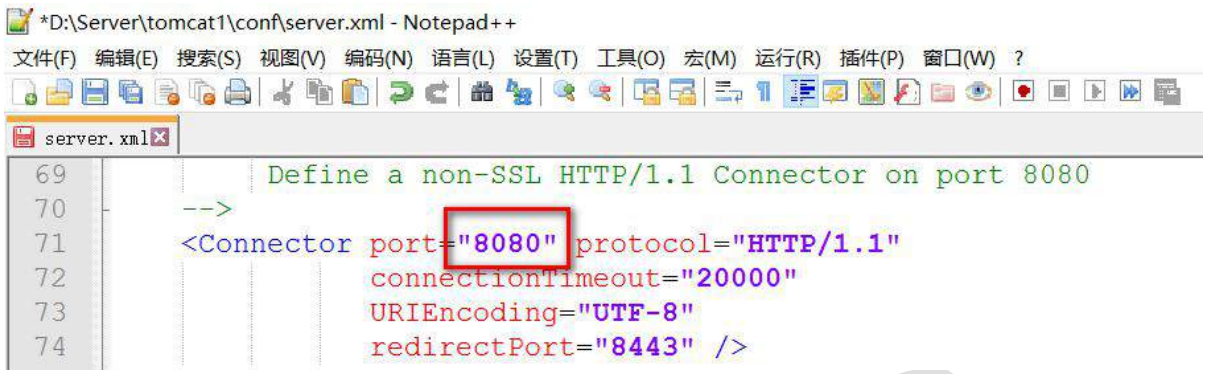
将【tomcat.zip】解压到【D:\Server\tomcat1】【D:\Server\tomcat2】

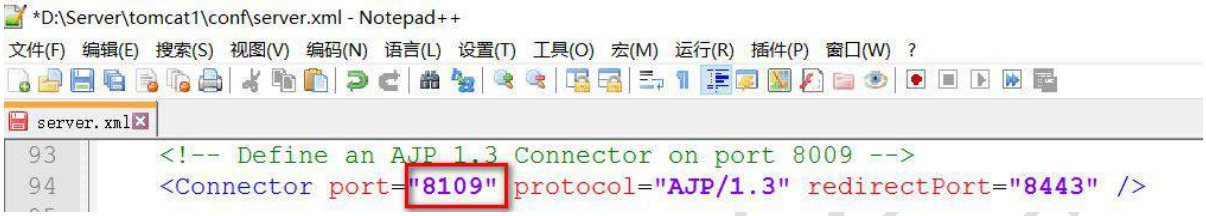
修改其中一个 Tomcat 的端口配置：分别修改为

8005,8080,8009;

8105,8181,8109;






小提示:

<Server port="8005" shutdown="SHUTDOWN"> 远程停服务端口

<Connector port="8080" protocol="HTTP/1.1" connectionTimeout="20000" redirectPort="8443" URIEncoding="UTF-8"/> 其中 8080 为 HTTP 端口，8443 为 HTTPS 端口

<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" /> 8009 为 AJP 端口

## 6.3 修改 Nginx 配置文件

- ./conf/nginx.conf
- 解除 Line34 注释，增加对应端口号。配置多服务器集群

```
upstream localhost {
    #ip_hash;
server 127.0.0.1:8181;
server 127.0.0.1:8080;
}
```

- L48 行，监听服务端口，当用户访问此端口时，进行反向代理服务

```
listen 80;
```

- 修改 L66, L70 行，设定静态资源访问路径

root D:/resources/atcrowdfunding;

- 拷贝某个图片放置在静态资源路径下：D:\resources\atcrowdfunding\java.jpg

## 6.4 安装 memcached

以管理员身份打开命令行窗口

### 6.4.1 安装 memcached 服务器

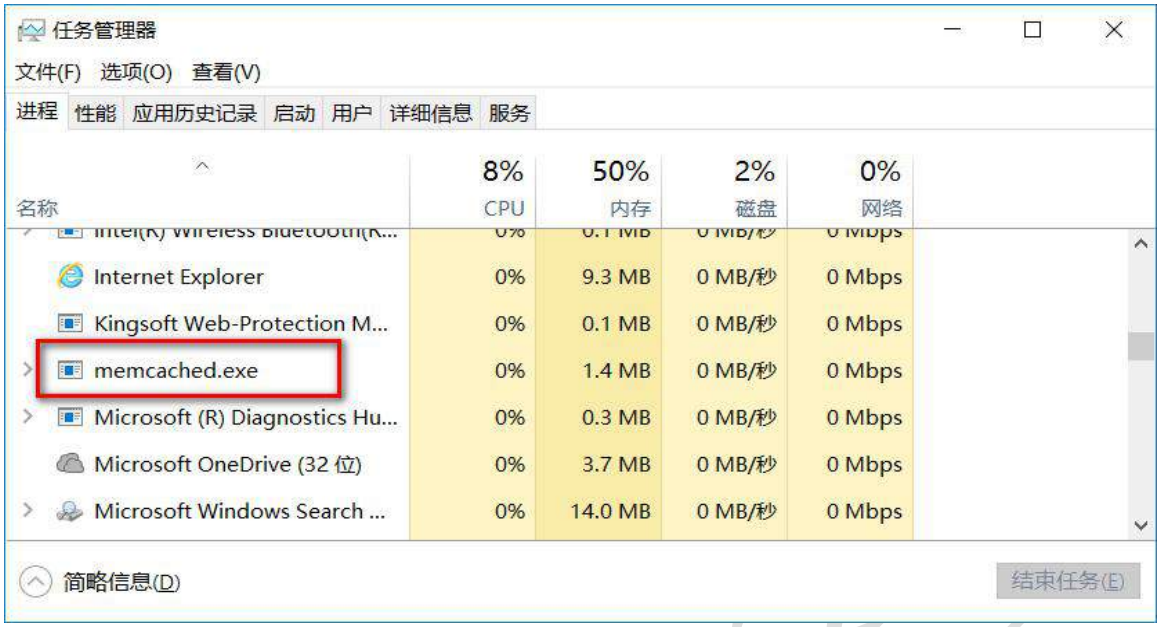
D:\Server\memcached\memcached.exe -d install

### 6.4.2 启动 memcached 服务器

D:\Server\memcached\memcached.exe -d start

### 6.4.3 关闭 memcached 服务器

D:\Server\memcached\memcached.exe -d stop|shutdown




## 6.4.4 配置 memcached 服务器

~/emcached/cmd.txt

安装 memcached D:\memcached\memcached.exe -d install

启动 memcached 的服务 D:\memcached\memcached.exe -d start

如果需要关闭 memcached 的服务 D:\memcached\memcached.exe -d stop|shutdown

设置 memcached，启动该服务后，memcached 服务默认占用的端口是 11211，占用的最大内存默认是 64M

D:\memcached\memcached.exe -p 10000 -m 512 -d start ， -p 表示要修改的端口， -m 表示占用的最大内存(单位为 M)。

常用命令：

1.telnet 到 memcache 服务器，如:telnet 192.168.1.120 11211(11211 是 memcache 的默认端口)

2.stats 查看基本信息

3.stats items 查看 items

4.get key（key 为 item 后面的字符串即键）

5.-c 最大同时连接数，默认是 1024

6.-f 块大小增长因子，默认是 1.25

7.-n 最小分配空间，key+value+flags 默认是 48

8.-h 显示帮助

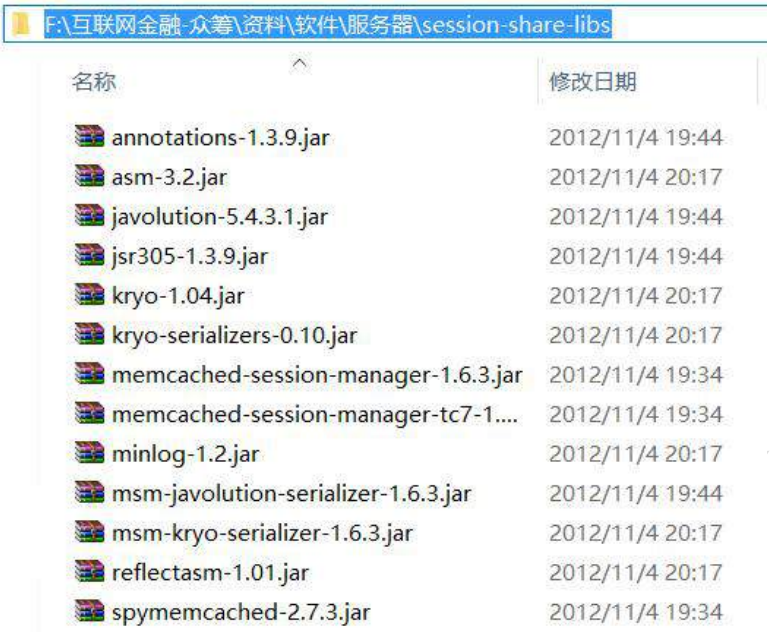
## 6.5 Tomcat 集成 Memcached

1. 服务器配置文件 context.xml

```
<WatchedResource>WEB-INF/web.xml</WatchedResource>
<Manager  className="de.javakaffee.web.msm.MemcachedBackupSessionManager"
                                        memcachedNodes="n1:127.0.0.1:11211"
                                        sticky="false"
                                        sessionBackupAsync="false"

requestUriIgnorePattern=".*\.(ico|png|gif|jpg|jpeg|bmp|css|js|html|htm)$"
transcoderFactoryClass="de.javakaffee.web.msm.serializer.kryo.KryoTranscoderFactory"
/>
```

2. 拷贝集成 jar 包

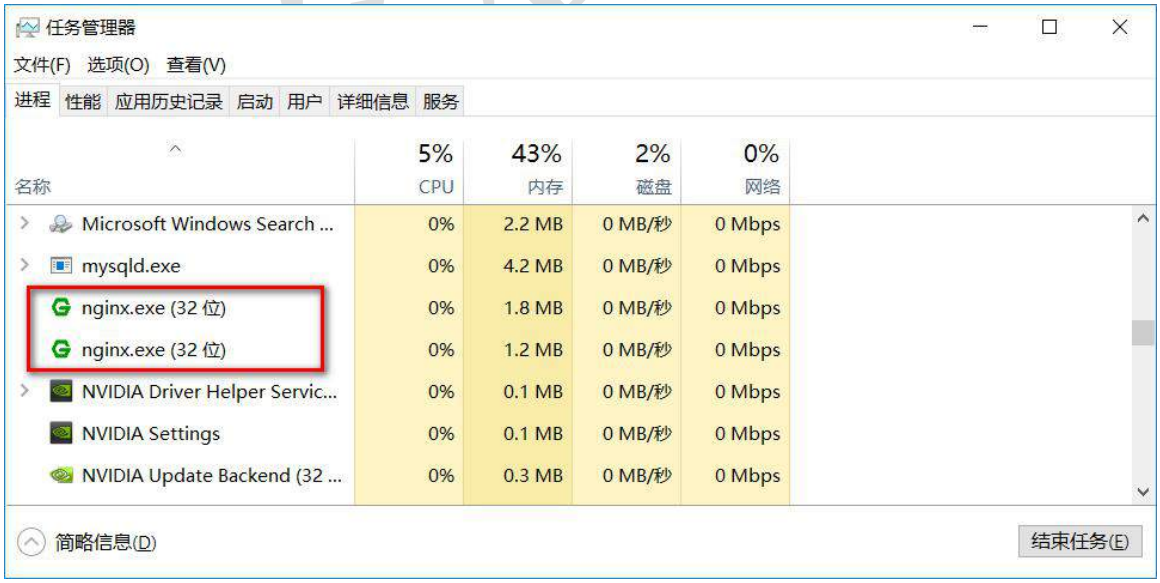
session-share-libs 到 Tomcat 服务器 lib 文件夹中




## 6.6 启动服务

### 6.6.1 启动 Nginx

- 任务管理器中出现 nginx.exe ， 启动成功



### 6.6.2 分别启动 Tomcat

双击启动：（不能设置 Tomcat 系统变量 CATALINDA_HOME 环境变量，否则启动时查找的 Tomcat 为环境变量所设置的 Tomcat）

D:\Server\tomcat1\bin\startup.bat

D:\Server\tomcat2\bin\startup.bat

## 6.7 测试 负载均衡（nginx 反向代理）

在两个 Tomcat 下增加 jsp 页面

D:\Server\tomcat1\webapps\ROOT\index.jsp

内容：Tomcat 1 JSP

D:\Server\tomcat2\webapps\ROOT\index.jsp

内容：Tomcat 2 JSP

打开浏览器：多次刷新页面，显示的页面内容为不同的 Tomcat 服务器下的部署项目资源

## 6.8 测试 session 共享

### 6.8.1 创建 testSession.jsp

在 D:\Server\tomcat2\webapps\ROOT 目录下不需要创建 testSession.jsp

```
<%
session.setAttribute("username","zhangsan");
%>
```

### 8.8.2 获取 session 中的 username

在两个 Tomcat 下 index.jsp 页面

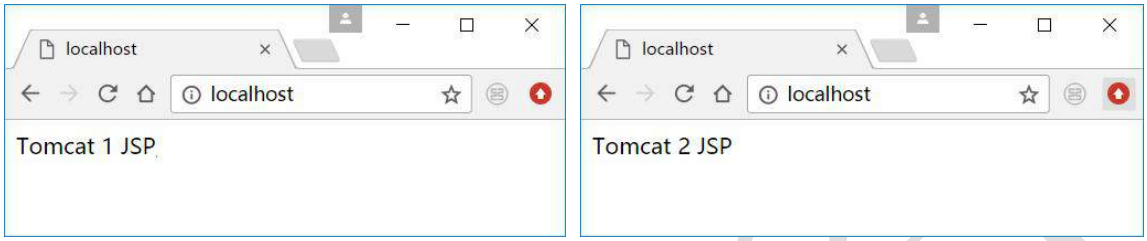
D:\Server\tomcat1\webapps\ROOT\index.jsp

内容：Tomcat 1 JSP; ${username}

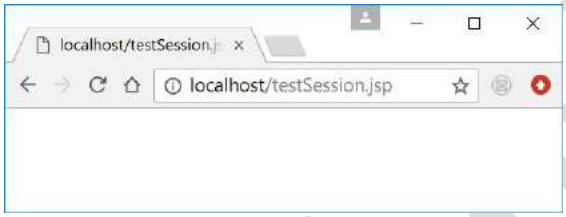
D:\Server\tomcat2\webapps\ROOT\index.jsp

内容：Tomcat 2 JSP; ${username}

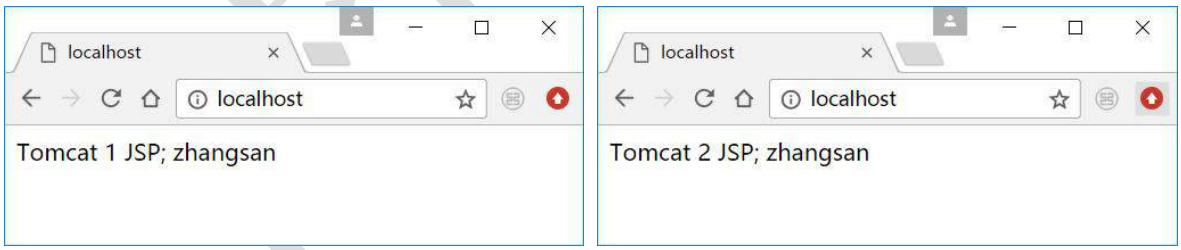
### 8.8.3 都无法获取到 username 值情况




### 8.8.4 执行 testSession.jsp 将 username 存放到 session 域中




### 8.8.5 两个服务器的 index.jsp 都可以获取到 username 值情况

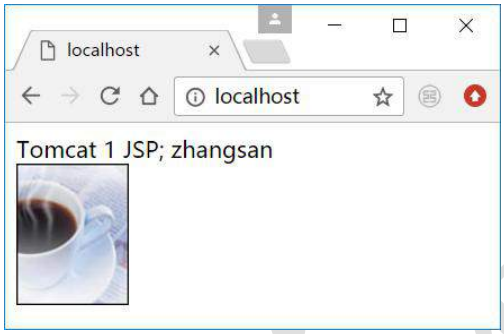


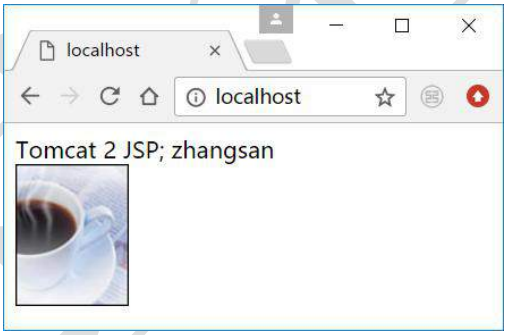
## 6.9 测试静态分离

### 6.9.1 在两个 index.jsp 中分别增加



### 6.9.2 访问服务器

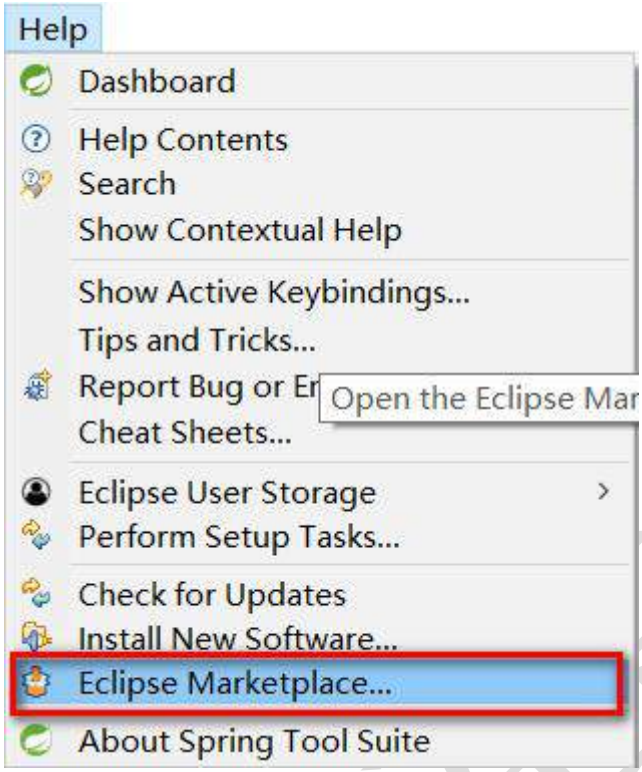


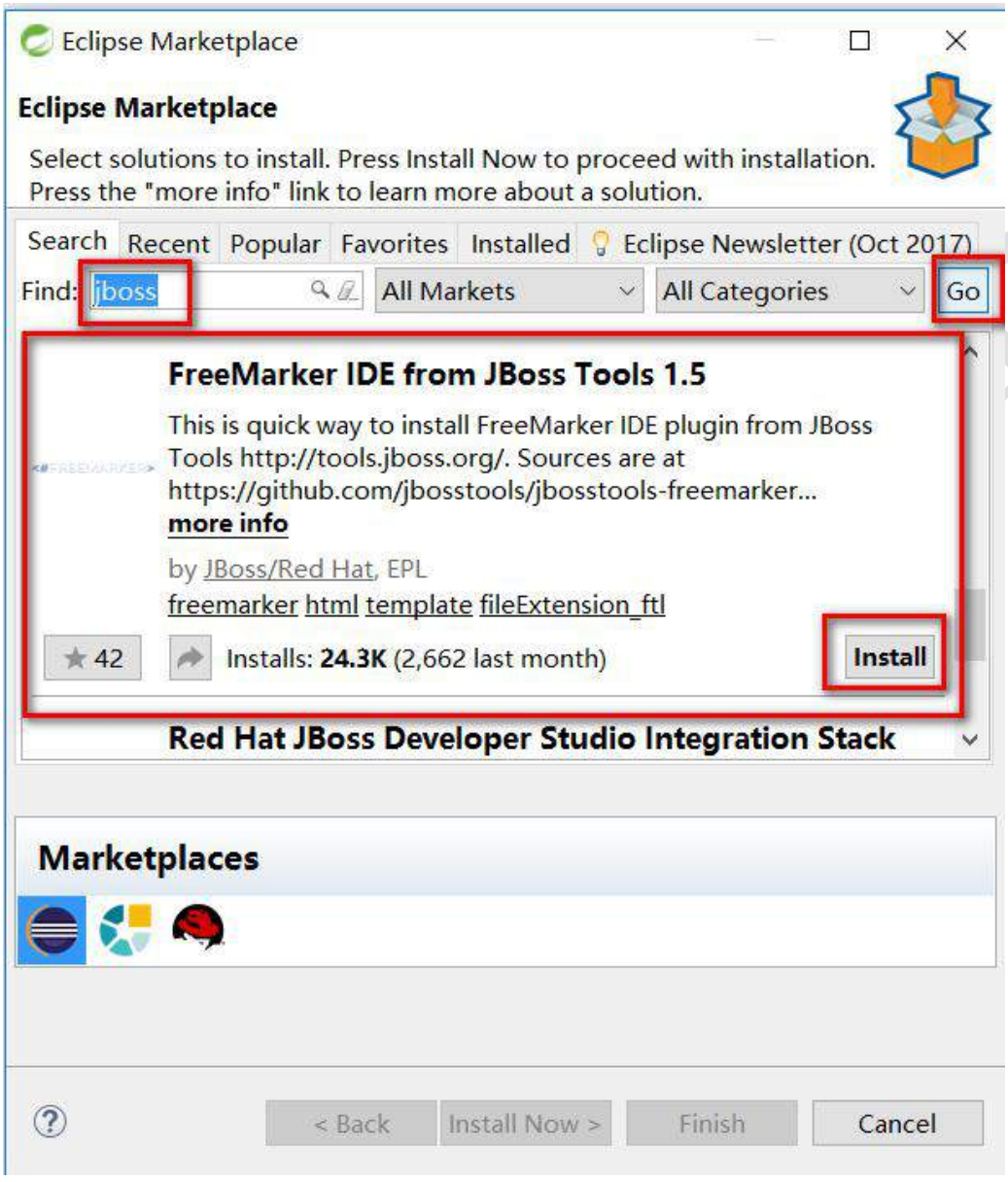

# 附录 1 Freemarker - IDE 插件

STS(Eclipse)安装 Freemarker 插件，按照步骤进行安装

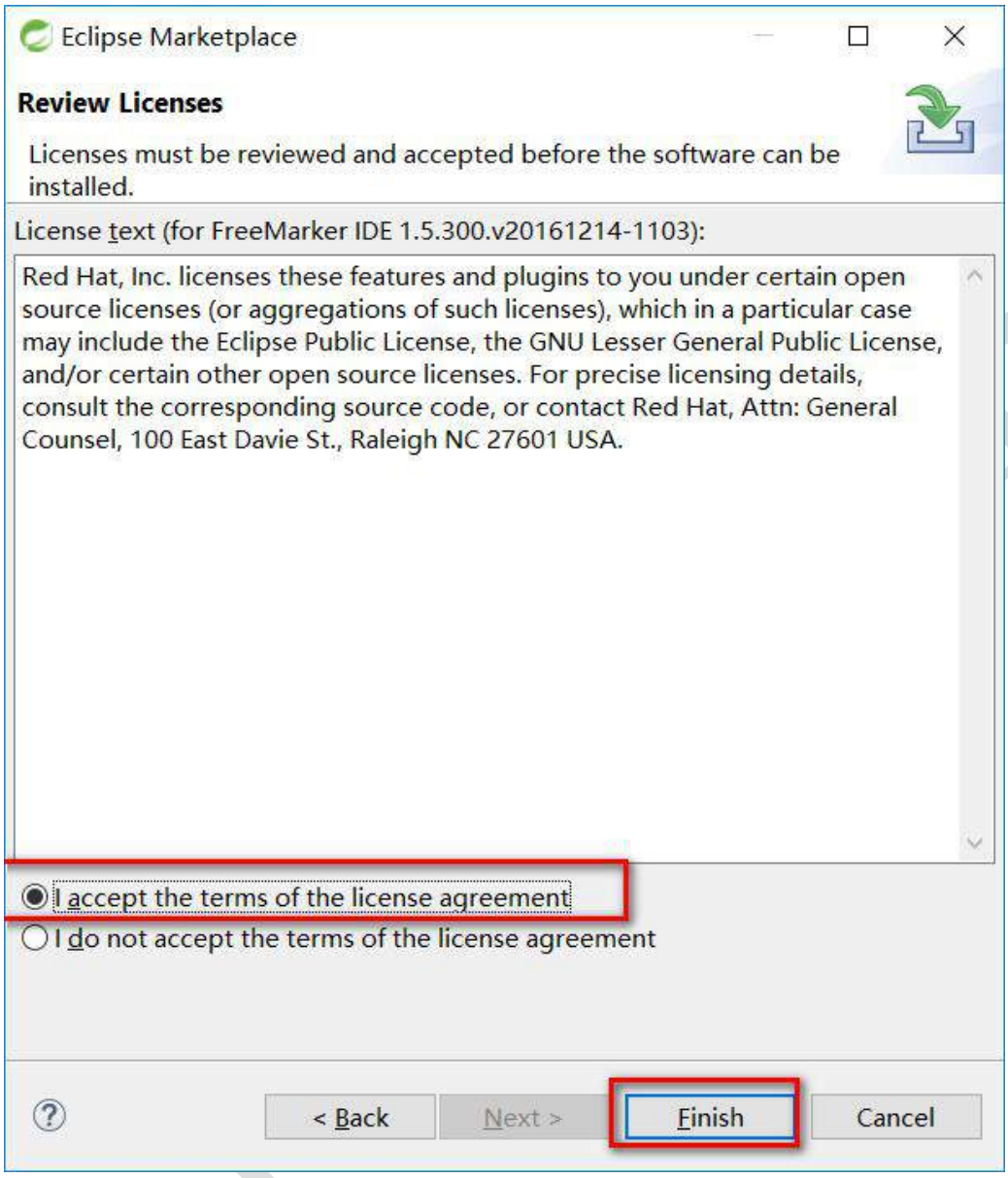
## 1.1 点击 help->Eclipse Marketplace 菜单项



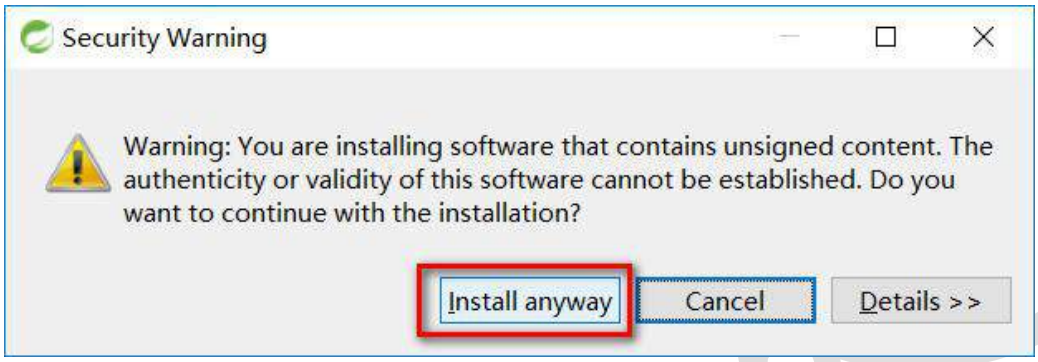
1.2 在 Find: 文本框中输入 jboos, 点击 [Go] 按钮搜索到 Freemarker 插件,点击[Install]按钮.



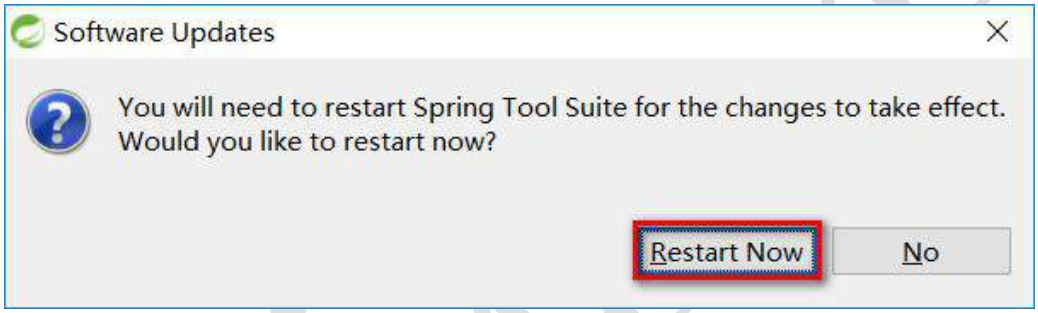
## 1.3 点击同意协议单选按钮,点击[Finish]按钮



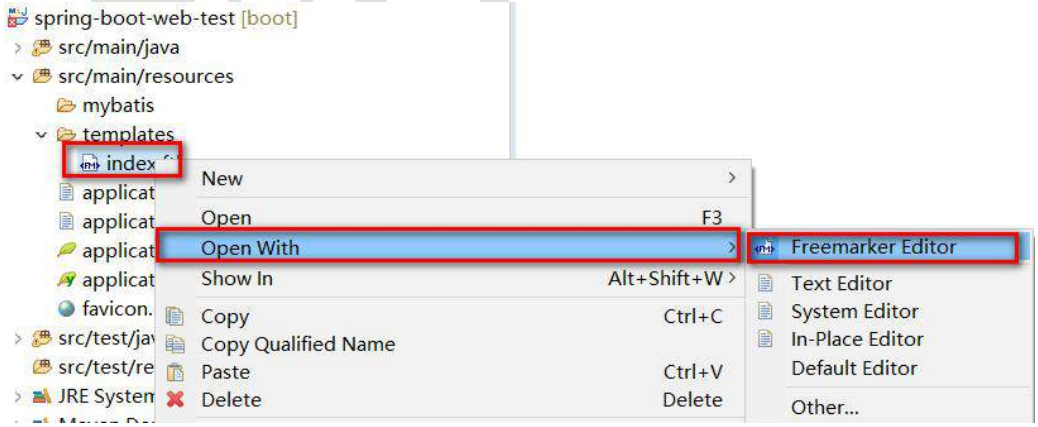
## 1.4 点击[Install anyway]按钮



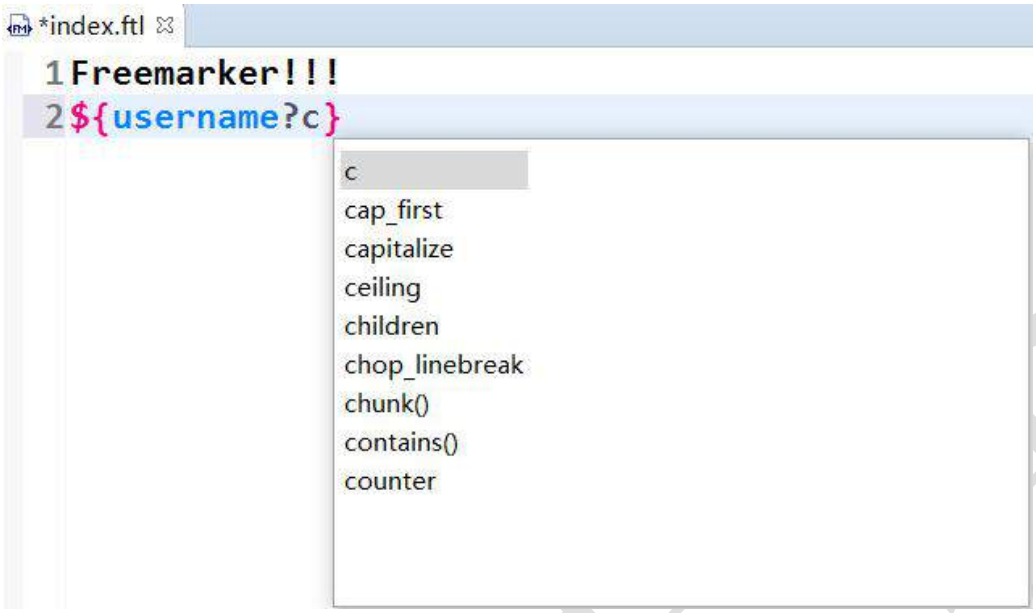

## 1.5 点击[Restart Now]按钮




## 1.6 通过 Freemarker Editor 插件打开文件



## 1.7 编写代码时有提示:Atl + /




# 附录 2 FreeMarker 如何获取域的数据

${Request.member.name}

${Session.member.name}

${Application.member.name}

${RequestParameters['id']} 取请求参数值

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 第 1 章 分布式架构

## 1.1 流行分布式架构

## 1.2 Spring 分布式架构




# 第 2 章 SpringBoot

## 2.1 概述

Spring Boot 是由 Pivotal 团队提供的全新框架，其设计目的是用来简化新 Spring 应用的初始搭建以及开发过程。
该框架使用了特定的方式来进行配置，从而使开发人员不再需要定义样板化的配置。
通过这种方式，Spring Boot 致力于在蓬勃发展的快速应用开发领域(rapid application development)成为领导者。
本章内容只是在 web 项目中应用基本的 Spring Boot 技术，如果想要学习完整 Spring Boot 课程内容，请访问 Spring Boot 官网进行学习。
此课件基于 Maven 创建项目，如果对于 Maven 工具不是很了解，请自行学习相关课件

## 2.2 为什么使用 Spring Boot?

说到为什么使用 Spring Boot，就不得不提到 Spring 框架的前世今生
Spring 框架由于其繁琐的配置，一度被人认为“配置地狱”，各种 XML、Annotation 配置混合使用，让人眼花缭乱，而且如果出错了也很难找出原因。
通过 SpringMVC 框架部署和发布 web 程序，需要和系统外服务器进行关联，操作繁琐不方便。

Spring Boot 是由 Spring 官方推出的一个新框架，对 Spring 进行了高度封装，是 Spring 未来的发展方向。使用 Spring Boot 框架后，可以帮助开发者快速搭建 Spring 框架，也可以帮助开发者快速启动一个 Web 服务，无须依赖外部 Servlet 容器，使编码变得简单，使配置变得简单，使部署变得简单，使监控变得简单。

## 2.3 Spring 前世今生

### 2.1.1 Spring1.x 时代

在 Spring1.x 时代，都是通过 xml 文件配置 bean
随着项目的不断扩大，需要将 xml 配置分放到不同的配置文件中
需要频繁的在 java 类和 xml 配置文件中切换。

### 2.1.2 Spring2.x 时代

随着 JDK 1.5 带来的注解支持，Spring2.x 可以使用注解对 Bean 进行申明和注入，大大的减少了 xml 配置文件，同时也大大简化了项目的开发。
那么，问题来了，究竟是应该使用 xml 还是注解呢？

| |
|---|
| 最佳实践：<br>应用的基本配置用 xml，比如：数据源、资源文件等；<br>业务开发用注解，比如：Service 中注入 bean 等； |

### 2.1.3 Spring3.x 到 Spring4.x

从 Spring3.x 开始提供了 Java 配置方式，使用 Java 配置方式可以更好的理解你配置的 Bean，
现在我们就处于这个时代，并且 Spring4.x 和 Spring boot 都推荐使用 java 配置的方式。

| |
|---|
| Spring 1.X<br>使用基本的框架类及配置文件（.xml）实现对象的声明及对象关系的整合。<br>org.springframework.core.io.ClassPathResource<br>org.springframework.beans.factory.xml.XmlBeanFactory<br>org.springframework.context.support.ClassPathXmlApplicationContext<br>Spring 2.X<br>使用注解代替配置文件中对象的声明。简化配置。<br>org.springframework.stereotype.@Component<br>org.springframework.stereotype.@Controller<br>org.springframework.stereotype.@Service<br>org.springframework.stereotype.@Repository<br>org.springframework.stereotype.@Scope<br>org.springframework.beans.factory.annotation.@Autowired |

Spring 3.X
使用更强大的注解完全代替配置文件。
org.springframework.context.annotation.AnnotationConfigApplicationContext
org.springframework.context.annotation.@Configuration
org.springframework.context.annotation.@Bean
org.springframework.context.annotation.@Value
org.springframework.context.annotation.@Import
Spring 4.X
使用条件注解强化之前版本的注解。
org.springframework.context.annotation.@Conditional

## 2.1.4 Spring 作者

Rod Johnson 在 2002 年编著的《Expert one on one J2EE design and development》一书中，对 Java EE 系统框架臃肿、低效、脱离现实的种种现状提出了质疑，并积极寻求探索革新之道。
以此书为指导思想，他编写了 interface21 框架，这是一个力图冲破 J2EE 传统开发的困境，从实际需求出发，着眼于轻便、灵巧，易于开发、测试和部署的轻量级开发框架。
Spring 框架即以 interface21 框架为基础，经过重新设计，并不断丰富其内涵，于 2004 年 3 月 24 日，发布了 1.0 正式版。
同年他又推出了一部堪称经典的力作《Expert one-on-one J2EE Development without EJB》，该书在 Java 世界掀起了轩然大波，不断改变着 Java 开发者程序设计和开发的思考方式。
在该书中，作者根据自己多年丰富的实践经验，对 EJB 的各种笨重臃肿的结构进行了逐一的分析和否定，并分别以简洁实用的方式替换之。
至此一战功成，Rod Johnson 成为一个改变 Java 世界的大师级人物。

## 2.1.5 案例

- **Spring1.x**

```
public static void main(String[] args) {
ApplicationContext beanFactory = new ClassPathXmlApplicationContext("beans.xml");
System.out.println("ApplicationContext..."); //立即初始化
User user = (User)beanFactory.getBean("user");
System.out.println(user);
}
```

```
public static void main(String[] args) {
Resource resource = new ClassPathResource("beans.xml") ;
XmlBeanFactory beanFactory = new XmlBeanFactory(resource);
System.out.println("XmlBeanFactory..."); //延迟初始化
User user = (User)beanFactory.getBean("user");
```

```
System.out.println(user);
}
```

- **Spring2.x**

@Controller
@Service
@Repository
@Scope
@Component

- **Spring3.x**

```
import org.springframework.context.annotation.Bean; //@since 3.0
import org.springframework.context.annotation.Configuration; //@since 3.0

@Configuration
public class BeanConfig {
    @Bean
    public User user() { //方法名称作为 bean 的 id
        return new User();
    }
}
```

```
ApplicationContext beanFactory = new AnnotationConfigApplicationContext("com.atguigu");
```

## 2.4 自动创建一个 Spring Boot 项目

- 通过 STS 快速创建一个 Spring Boot 项目,运行查看结果: Spring Banner

## 2.5 手动创建 Spring Boot_HelloWorld

### 2.5.1 创建 Maven 项目

- 在 Spring Tool Suite 中创建 Maven jar 项目，无需配置 web.xml 文件。
- 修改项目中的 pom.xml 文件，设置 JDK 编译版本为 1.8

```
<project>
    ...
    <build>
        <plugins>
            <!-- 修改 maven 默认的 JRE 编译版本，1.8 代表 JRE 编译的版本，根据自己的安装版本选择 1.7 或 1.8 -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
```

```
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
        </plugins>
    </build>
    ...
</project>
```

- Spring Boot 框架最低 JDK 版本要求 1.6，但是 Spring Boot 官方公布的一些功能使用 1.8 性能会高很多，所以本章我们选用 JDK1.8 版本
- SSM 基础架构中，需要生成 web.xml 文件，Spring Boot 框架中为什么没有？
  Spring Boot 框架开发 web 系统，是基于 servlet3.0 或以上规范，无需 web.xml 文件

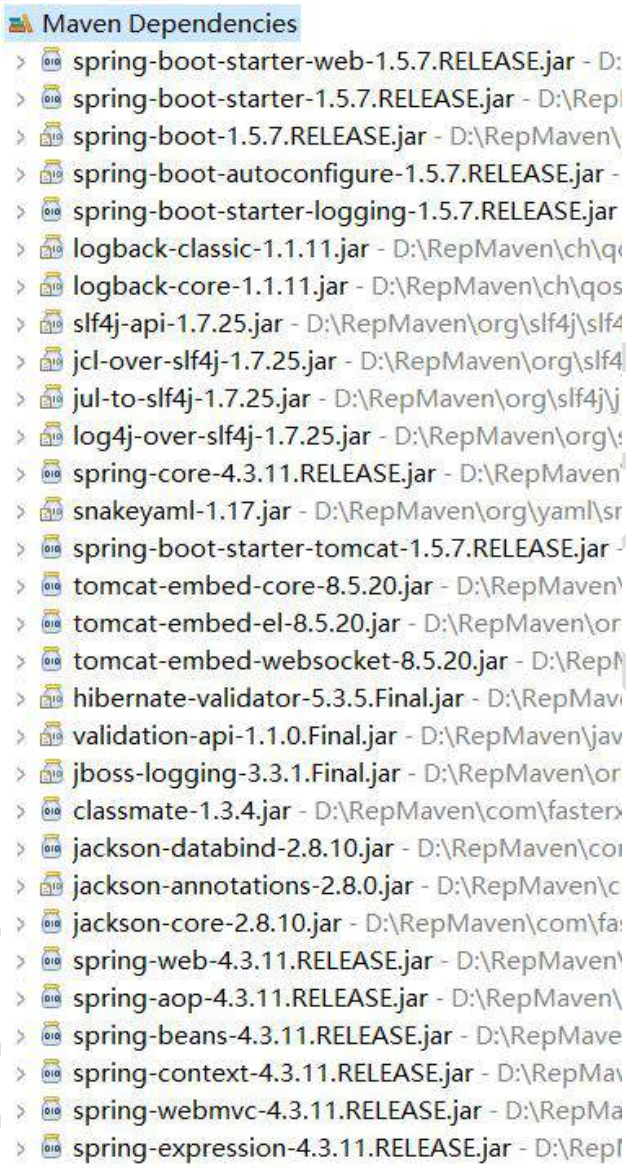
## 2.5.2 集成 Spring Boot 框架

- 修改 pom.xml 文件，增加 Spring Boot 框架的依赖关系及对 Web 环境的支持。

```
<project>
    ...
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.8.RELEASE</version>
    </parent>
    ...
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>
    ...
</project>
```

- Spring Boot 版本为官方最新正式版 1.5.8.RELEASE
- 以往的项目中，所有类库的依赖关系都需要我们自己导入到 pom.xml 文件中，但是 Spring Boot 项目增加 spring-boot-starter-web 依赖后，会自动加载 web 环境配置相关依赖(SpringMVC, Tomcat)，简化了我们的操作。
- spring-boot-starter-parent：继承 Spring Boot 的相关参数
- spring-boot-starter-xxx：代表一个 Spring Boot 模块(参考附录 1.Spring Boot 相关模块)

- spring-boot-starter-web：代表 Web 模块，在这个模块中包含了许多依赖的 JAR 包




附录 2. 项目报小红叉

## 2.5.3 增加程序代码

- 集成环境,启动服务器
- 在 src/main/java 目录中增加类 com.atguigu.crowdfunding.AtCrowdfundingApplication，并增加相应代码。

```
package com.atguigu.crowdfunding;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class AtCrowdfundingApplication { //有网的时候可以自动创建,没有网的时候,我们可以手动创建.
```

```
    public static void main(String[] args) {
        SpringApplication.run(AtCrowdfundingApplication.class, args);
    }
}
```

- Spring Boot 项目中都会有一个以 Application 结尾的应用类，然后有一个标准的 Java 入口方法 main 方法。通过这个方法启动 Spring Boot 项目，方法中无需放入任何业务逻辑。
- @SpringBootApplication 注解是 Spring Boot 核心注解
- 右键点击项目或项目中的 AtCrowdfundingApplication 类，选择菜单 Run as Spring Boot App，控制台出现下面内容表示服务启动成功。

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.8.RELEASE)
2017-08-21 14:47:42.686  INFO 7320 --- [           main] c.a.c.AtCrowdfundingApplication          : Starting AtCrowdfundingApplication on DESKTOP-9UGKHC7 with PID 7320 (D:\KnowledgeHierarchy\workspace\sts-test\Web\target\classes started by 18801 in D:\KnowledgeHierarchy\workspace\sts-test\Web)
2017-08-21 14:47:42.688  INFO 7320 --- [           main] c.a.c.AtCrowdfundingApplication          : No active profile set, falling back to default profiles: default
2017-08-21 14:47:42.727  INFO 7320 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing

...

2017-08-21 14:47:44.525  INFO 7320 --- [           main] o.s.j.e.a.AnnotationMBeanExporter        : Registering beans for JMX exposure on startup
2017-08-21 14:47:44.586  INFO 7320 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat started on port(s): 8080 (http)
2017-08-21 14:47:44.590  INFO 7320 --- [           main] c.a.c.AtCrowdfundingApplication          : Started AtCrowdfundingApplication in 2.131 seconds (JVM running for 2.394)
</java.util.map
```

## 2.5.4 集成了 Tomcat 服务器

- 当增加 Web 依赖后执行 main 方法，等同于启动 Tomcat 服务器，默认端口号为 8080。
- 如果想要修改默认的 Tomcat 服务器端口号，可以通过全局配置文件进行配置，在 src/main/resources/目录中增加 application.properties 文件。

```
server.context-path=/
server.port=80
```

```
server.session.timeout=60
server.tomcat.max-threads=800
server.tomcat.uri-encoding=UTF-8
```

- Spring Boot 会自动读取 src/main/resources/路径或类路径下/config 路径中的 application.properties 文件或 application.yml 文件。

### 2.5.5 为什么还会有配置文件

Spring Boot 我们称之为微框架，这里的“微”不是小和少的意思，而是“简”的意思，简单，简洁。项目中大部分的基础配置由 Spring Boot 框架帮我们自动集成，简化了我们的配置，但是框架自身为了扩展性，依然需要提供配置文件。
上面的代码中只是简单的应用了 Spring Boot 框架，但是我们真正要做的是将 Spring Boot 应用到项目中，所以接下来我们增加对 SpringMVC 框架，Mybatis 框架的集成。并通过源码的解读对 Spring Boot 框架有一个更深层次的理解。

# 第 3 章 SpringBoot 集成 Spring & SpringMVC

- 基本的 Spring Boot 环境已经构建好了，现在需要配置 Spring 框架及 SpringMVC 框架的业务环境

## 3.1 @ComponentScan 注解

```
package com.atguigu.crowdfunding;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
@ComponentScan(basePackages="com.atguigu")
@SpringBootApplication
public class AtCrowdfundingApplication {
    public static void main(String[] args) {
        SpringApplication.run(AtCrowdfundingApplication.class, args);
    }
}
```

## 3.2 默认扫描

默认扫描当前包 com.atguigu.crowdfunding 和子包 com.atguigu.crowdfunding.*
如果还需要扫描其他的包，那么需要增加@ComponentScan 注解,指定包名进行扫描。

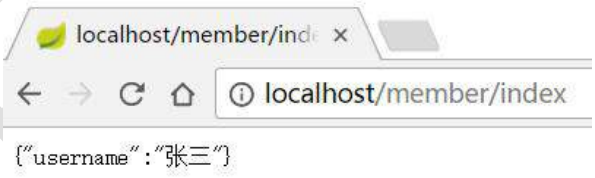
## 3.3 增加控制器代码

在 src/main/java 目录中增加类 com.atguigu.crowdfunding.controller.MemberController，并增加相应代码。

```
package com.atguigu.crowdfunding.controller;
import java.util.HashMap;
import java.util.Map;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
@Controller
@RequestMapping("/member")
public class MemberController {
    @ResponseBody
    @RequestMapping("/index")
    public Object index() {
        Map map = new HashMap();
        map.put("username", "张三");
        return map;
    }
}
```

## 3.4 执行 main 方法启动应用

- 访问路径 http://127.0.0.1:8080[/应用路径名称]/member/index 页面打印 JSON 字符串即可




## 3.5 @Controller 和@RestController 区别

官方文档：**@RestController** is a stereotype annotation that combines **@ResponseBody and @Controller**.
表示@RestController 等同于@Controller + @ResponseBody，所以上面的代码可以变为：

```
package com.atguigu.crowdfunding.controller;
import java.util.HashMap;
import java.util.Map;
```

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
@RestController
@RequestMapping("/member")
public class MemberController {
    @RequestMapping("/index")
    public Object index() {
        Map map = new HashMap();
        map.put("username", "张三");
        return map; //JSON => JavaScript Object Notation
    }
}
```

## 3.6 增加服务层代码

Service 接口，ServiceImpl 实现类的使用和 SSM 架构中的使用方式完全相同。

```
package com.atguigu.crowdfunding.service;

public interface MemberService {

}
```

```
package com.atguigu.crowdfunding.service.impl;

import org.springframework.stereotype.Service;
import com.atguigu.crowdfunding.service.MemberService;

@Service
public class MemberServiceImpl implements MemberService {

}
```

```
@RestController
public class MemberController {
    @Autowired
    private MemberService memberService ;
}
```

# 第 4 章 SpringBoot 集成 Mybatis

Spring Boot 框架在集成 Mybatis 框架的时候依然要扫描 Dao 接口，SQL 映射文件以及依赖数据库连接

池，但是和传统 SSM 框架集成时稍微有一些不同。

## 4.1 增加 Mybaits 依赖

```
<project>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <!-- 数据库连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.1.1</version>
        </dependency>
        ...
    </dependencies>
    ...
</project>
```

## 4.2 Druid VS C3P0

在实际应用中，C3P0 偶尔会出现连接超时，或自动断开的情况，虽然会自动重新连接，但是运行稳定性还是存在问题。从性能对比上来说，阿里巴巴开源平台的 Druid 连接池表现会更好一些。

## 4.3 增加 application.yml 配置

在 src/main/resources/目录下，增加 application.yml 配置文件，增加连接池和 mybatis 相关配置

```
---
spring:
  datasource:
    name: mydb
```

```
    type: com.alibaba.druid.pool.DruidDataSource
    url: jdbc:mysql://127.0.0.1:3306/atcrowdfunding
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
mybatis:
  mapper-locations: classpath*:/mybatis/mapper-*.xml
  type-aliases-package: com.atguigu.**.bean
```

## 4.4 properties 和.yml 区别

在 Spring Boot 中，有两种配置文件， 一种是 application.properties，另一种是 application.yml，两种都可以配置 Spring Boot 项目中的一些变量的定义，参数的设置等。
application.properties 配置文件在写的时候要写完整，yml 文件在写的时候层次感强，而且少写了代码。但是从严格意义上来讲，区别不大。

## 4.5 扫描 Dao 接口和开启声明式事务

需要在 AtCrowdfundingApplication 类中增加扫描注解@MapperScan("com.atguigu.**.dao")及事务管理@EnableTransactionManagement

## 4.6 增加 Dao 代码

```
package com.atguigu.crowdfunding.dao;
import org.apache.ibatis.annotations.Insert;
import org.apache.ibatis.annotations.Select;
import com.atguigu.crowdfunding.bean.Member;
public interface MemberDao {
    @Select("select * from t_member where id = #{id}")
    public Member queryById(Integer id);

    @Insert("insert into t_member (loginacct) values (#{loginacct})")
    public int insertMember(Member member);
}
```

## 4.7 增加 Member 实体类

```
package com.atguigu.crowdfunding.bean;
```

```
public class Member {
    public Integer id;
    private String name;

    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name= name;
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }

}
```

## 4.8 增加事务注解@Transactional

传统的 SSM 架构中采用的是声明式事务，需要在配置文件中增加 AOP 事务配置，Spring Boot 框架中简化了这种配置，可以在 Service 接口中增加注解@Transactional

```
package com.atguigu.crowdfunding.service.impl;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.atguigu.crowdfunding.bean.Member;
import com.atguigu.crowdfunding.dao.MemberDao;
import com.atguigu.crowdfunding.service.MemberService;
@Service
@Transactional(readOnly=true)
public class MemberServiceImpl implements MemberService {
    @Autowired
    private MemberDao memberDao;

    public Member queryById(Integer id) {
        return memberDao.queryById(id);
    }
    @Transactional
    public int insertMember(Member member) {
```

```
            return memberDao.insertMember(member);
        }
}
```

## 4.9 @Transactional 放在类前和方法前区别

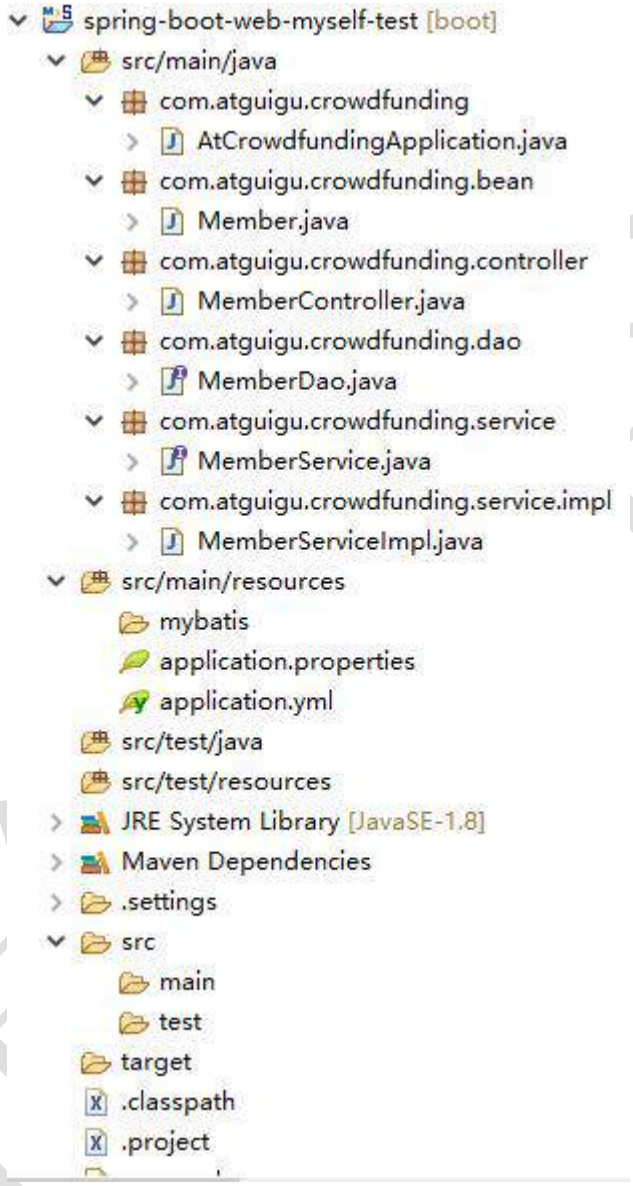
- 如果放置在类前，表示全局配置，对所有的方法起作用
- 如果放置在方法前，表示只对这一个方法起作用，且覆盖全局配置。

## 4.10 修改 MemberController 进行测试

```
package com.atguigu.crowdfunding.controller;
import java.util.HashMap;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.transaction.annotation.EnableTransactionManagement;
import com.atguigu.crowdfunding.bean.Member;
import com.atguigu.crowdfunding.service.MemberService;
@RestController
@RequestMapping("/member")
public class MemberController {
    @Autowired
    private MemberService memberService;

    @RequestMapping("/index/{id}")
    public Object index(@PathVariable("id") Integer id ) {
            Member member = memberService.queryById(id);
            return member;
    }

    @RequestMapping("/insert")
    public Object insert( Member member ) {
            memberService.insertMember(member);
            return new HashMap();
    }
}
```

## 4.11 测试

重启服务，访问路径 http://127.0.0.1:8080[/应用路径名称]/member/index 观察效果




# 第 5 章 SpringBoot 集成 freemarker

## 5.1 增加 freemarker 依赖

```
<project>
    ...
```

```
    <dependencies>
        ...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-freemarker</artifactId>
        </dependency>
        ...
    </dependencies>
    ...
</project>
```

## 5.2 增加 freemarker 相关配置

```
...
spring.freemarker.allow-request-override=false
spring.freemarker.allow-session-override=false
spring.freemarker.cache=true
spring.freemarker.charset=UTF-8
spring.freemarker.check-template-location=true
spring.freemarker.content-type=text/html
spring.freemarker.enabled=true
spring.freemarker.expose-request-attributes=false
spring.freemarker.expose-session-attributes=false
spring.freemarker.expose-spring-macro-helpers=true
spring.freemarker.prefer-file-system-access=false
spring.freemarker.suffix=.ftl
spring.freemarker.template-loader-path=classpath:/templates/
spring.freemarker.settings.template_update_delay=0
spring.freemarker.settings.default_encoding=UTF-8
spring.freemarker.settings.classic_compatible=true
spring.freemarker.order=1
...
```

## 5.3 修改 MemberController 测试

```
package com.atguigu.crowdfunding.controller;
import java.util.HashMap;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.transaction.annotation.EnableTransactionManagement;
import com.atguigu.crowdfunding.bean.Member;
import com.atguigu.crowdfunding.service.MemberService;
@Controller
@RequestMapping("/member")
public class MemberController {

    @Autowired
    private MemberService memberService;

    @RequestMapping("/index/{id}")
    public String index(@PathVariable("id") Integer id,Map<String, Object> map) {
        Member member = memberService.getMember(id);
        map.put("member", member);
        return "member";
    }

}
```

## 5.4 创建 ftl 视图文件

- 在 src/main/resources/目录中增加 templates 文件夹，并增加.ftl 文件
- 文件内容:
  FREEMARKER INDEX PAGE!!! ${Request.member.name}

## 5.5 不采用 JSP 作为视图模板原因

Spring Boot 框架一般是打包为 JAR 执行，而 JSP 在 web 工程(war 包)中可以被 java 程序读取和识别，但是在 JAR 包中是比较困难的。所以需要采用其他的模板视图技术。

# 第 6 章 源码分析

## 6.1 SpringApplication 对象创建

- SpringApplication.run(AtCrowdfundingApplication.class, args);

查看源码

```
public static ConfigurableApplicationContext run(Object[] sources, String[] args) {
   return new SpringApplication(sources).run(args);
```

```
}
```

- 自己创建 SpringApplication 对象,可以进行相关设置.

```
SpringApplication springApplication = new SpringApplication(AtCrowdfundingApplication.class);
springApplication.setBannerMode(Banner.Mode.OFF);
springApplication.run(args);
```

## 6.2 SpringApplication 对象初始化

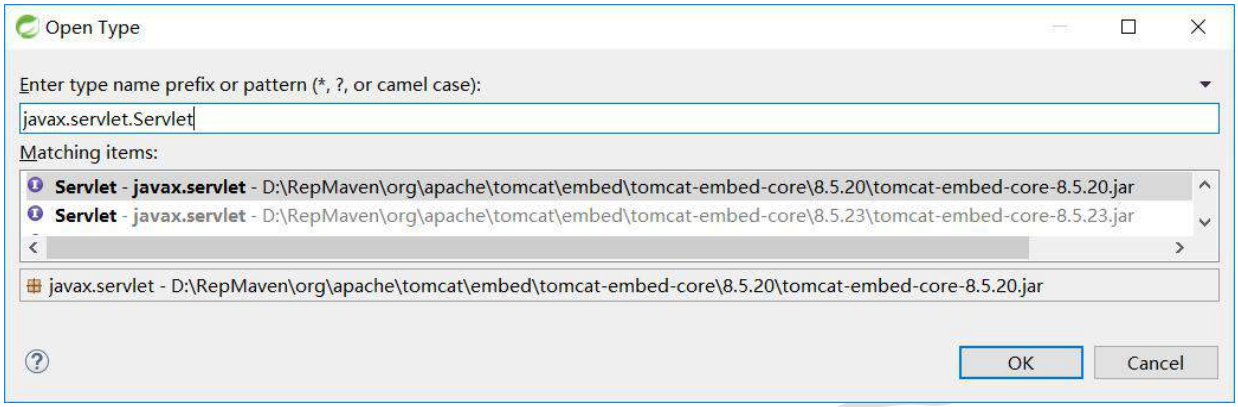
- 源码分析-构造 SpringApplication 对象,进行初始化.

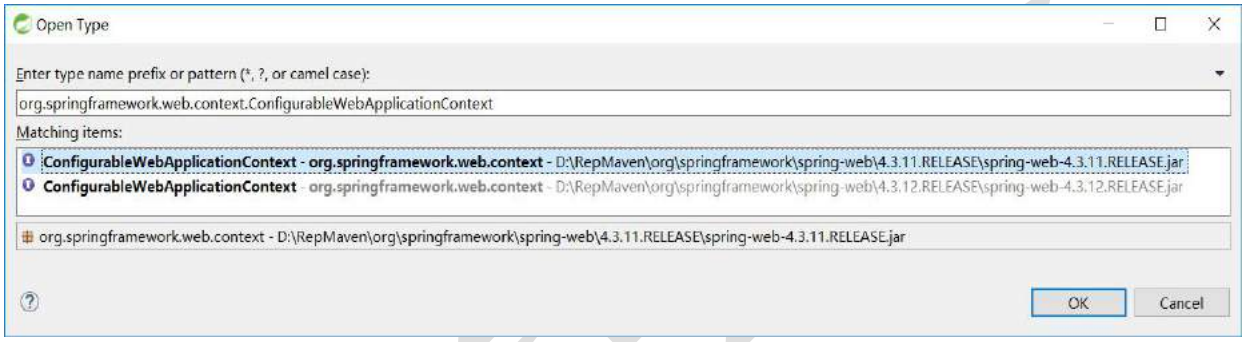
```
@SuppressWarnings({ "unchecked", "rawtypes" })
private void initialize(Object[] sources) {
   if (sources != null && sources.length > 0) {
      this.sources.addAll(Arrays.asList(sources)); //将启动类放置到集合中
   }
   this.webEnvironment = deduceWebEnvironment(); //判断是否为 web 环境
   //从"META-INF/spring.factories"文件中读取配置,将字符串分解后得到加载的类的名称,并创建对象存放到集合中.
   setInitializers((Collection) getSpringFactoriesInstances(ApplicationContextInitializer.class));
   //同样,加载并创建对象.
   setListeners((Collection) getSpringFactoriesInstances(ApplicationListener.class));
   this.mainApplicationClass = deduceMainApplicationClass();
}
```

- deduceWebEnvironment() : 判断是否为 web 环境

```
private boolean deduceWebEnvironment() {
   for (String className : WEB_ENVIRONMENT_CLASSES) {
      if (!ClassUtils.isPresent(className, null)) { //判断是否为 web 环境
         return false;
      }
   }
   return true; //两个类都能找到,就是 web 环境.
}
```

```
//判断是否为 web 环境,这两个类都在,说明是 web 环境.
private static final String[] WEB_ENVIRONMENT_CLASSES =
  { "javax.servlet.Servlet","org.springframework.web.context.ConfigurableWebApplicationContext" };
```




- 决定加载哪些类:

```
public static Class<?> forName(String name, ClassLoader classLoader) throws ClassNotFoundException,
LinkageError {
Assert.notNull(name, "Name must not be null");

Class<?> clazz = resolvePrimitiveClassName(name);
if (clazz == null) {
clazz = commonClassCache.get(name);
}
if (clazz != null) {
return clazz;
}

// "java.lang.String[]" style arrays     是否为字符串数组
if (name.endsWith(ARRAY_SUFFIX)) {
String elementClassName = name.substring(0, name.length() - ARRAY_SUFFIX.length());
Class<?> elementClass = forName(elementClassName, classLoader);
return Array.newInstance(elementClass, 0).getClass();
}

// "[Ljava.lang.String;" style arrays 是否为数组
if (name.startsWith(NON_PRIMITIVE_ARRAY_PREFIX) && name.endsWith(";")) {
String elementName = name.substring(NON_PRIMITIVE_ARRAY_PREFIX.length(), name.length() - 1);
```

```
Class<?> elementClass = forName(elementName, classLoader);
return Array.newInstance(elementClass, 0).getClass();
}

// "[[I" or "[[Ljava.lang.String;" style arrays  是否为多维数组
if (name.startsWith(INTERNAL_ARRAY_PREFIX)) {
String elementName = name.substring(INTERNAL_ARRAY_PREFIX.length());
Class<?> elementClass = forName(elementName, classLoader);
return Array.newInstance(elementClass, 0).getClass();
}

ClassLoader clToUse = classLoader;
if (clToUse == null) {
clToUse = getDefaultClassLoader();//获取默认类加载器.
}
try {
return (clToUse != null ? clToUse.loadClass(name) : Class.forName(name)); //加载类.
}
catch (ClassNotFoundException ex) {
int lastDotIndex = name.lastIndexOf(PACKAGE_SEPARATOR);
if (lastDotIndex != -1) {
String innerClassName =
name.substring(0, lastDotIndex) + INNER_CLASS_SEPARATOR + name.substring(lastDotIndex + 1);
try {
return (clToUse != null ? clToUse.loadClass(innerClassName) : Class.forName(innerClassName));
}
catch (ClassNotFoundException ex2) {
// Swallow - let original exception get through
}
}
throw ex;
}
}
```

setInitializers((Collection) getSpringFactoriesInstances(ApplicationContextInitializer.class));

非常重要的类:

```
ApplicationContextInitializer
```

加载类,并创建实例.

```
private <T> Collection<? extends T> getSpringFactoriesInstances(Class<T> type,
Class<?>[] parameterTypes, Object... args) {
   ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
```

```
    // Use names and ensure unique to protect against duplicates
    //从"META-INF/spring.factories"文件中读取配置,将字符串分解后得到加载的类的名称
    Set<String> names =
            new LinkedHashSet<String>(SpringFactoriesLoader.loadFactoryNames(type, classLoader));
    //根据类的名称反射创建对象.
    List<T> instances = createSpringFactoriesInstances(type, parameterTypes,classLoader, args, names);
    AnnotationAwareOrderComparator.sort(instances);
    return instances;
}
```

- 加载哪些类,获取这些类的名称.

SpringFactoriesLoader.loadFactoryNames(type, classLoader)

```
public static List<String> loadFactoryNames(Class<?> factoryClass, ClassLoader classLoader) {
String factoryClassName = factoryClass.getName(); //ApplicationContextInitializer
try {
Enumeration<URL> urls = (classLoader != null ?
classLoader.getResources(FACTORIES_RESOURCE_LOCATION) :
ClassLoader.getSystemResources(FACTORIES_RESOURCE_LOCATION));
List<String> result = new ArrayList<String>();
while (urls.hasMoreElements()) {
URL url = urls.nextElement();
Properties properties = PropertiesLoaderUtils.loadProperties(new UrlResource(url));
String factoryClassNames = properties.getProperty(factoryClassName);
//分解字符串 delimitedListToStringArray(str, ",")
result.addAll(Arrays.asList(StringUtils.commaDelimitedListToStringArray(factoryClassNames)));
}
return result;
}
catch (IOException ex) {
throw new IllegalArgumentException("Unable to load [" + factoryClass.getName() +
"] factories from location [" + FACTORIES_RESOURCE_LOCATION + "]", ex);
}
}
```

- public static final String FACTORIES_RESOURCE_LOCATION = "META-INF/spring.factories";
- 在这两个 jar 包中可以找到"META-INF/spring.factories"文件

**spring-boot-1.5.7.RELEASE.jar**

**spring-boot-autoconfigure-1.5.7.RELEASE.jar**


```
# Application Context Initializers
org.springframework.context.ApplicationContextInitializer=\
org.springframework.boot.context.ConfigurationWarningsApplicationContextInitializer,\
org.springframework.boot.context.ContextIdApplicationContextInitializer,\
org.springframework.boot.context.config.DelegatingApplicationContextInitializer,\
org.springframework.boot.context.embedded.ServerPortInfoApplicationContextInitializer
```

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration=\
org.springframework.boot.autoconfigure.admin.SpringApplicationAdminJmxAutoConfiguration,\
org.springframework.boot.autoconfigure.aop.AopAutoConfiguration,\
org.springframework.boot.autoconfigure.amqp.RabbitAutoConfiguration,\
org.springframework.boot.autoconfigure.batch.BatchAutoConfiguration,\
org.springframework.boot.autoconfigure.cache.CacheAutoConfiguration,\
…
org.springframework.boot.autoconfigure.data.solr.SolrRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.redis.RedisAutoConfiguration,\
org.springframework.boot.autoconfigure.data.redis.RedisRepositoriesAutoConfiguration,\
org.springframework.boot.autoconfigure.data.rest.RepositoryRestMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.data.web.SpringDataWebAutoConfiguration,\
org.springframework.boot.autoconfigure.elasticsearch.jest.JestAutoConfiguration,\
org.springframework.boot.autoconfigure.freemarker.FreeMarkerAutoConfiguration,\
org.springframework.boot.autoconfigure.gson.GsonAutoConfiguration,\
org.springframework.boot.autoconfigure.h2.H2ConsoleAutoConfiguration,\
org.springframework.boot.autoconfigure.hateoas.HypermediaAutoConfiguration,\
org.springframework.boot.autoconfigure.hazelcast.HazelcastAutoConfiguration,\
org.springframework.boot.autoconfigure.hazelcast.HazelcastJpaDependencyAutoConfiguration,\
org.springframework.boot.autoconfigure.info.ProjectInfoAutoConfiguration,\
org.springframework.boot.autoconfigure.integration.IntegrationAutoConfiguration,\
org.springframework.boot.autoconfigure.jackson.JacksonAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.JdbcTemplateAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.JndiDataSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.XADataSourceAutoConfiguration,\
org.springframework.boot.autoconfigure.jdbc.DataSourceTransactionManagerAutoConfiguration,\
```

```
org.springframework.boot.autoconfigure.jms.JmsAutoConfiguration,\
org.springframework.boot.autoconfigure.jmx.JmxAutoConfiguration,\
...
org.springframework.boot.autoconfigure.transaction.TransactionAutoConfiguration,\
org.springframework.boot.autoconfigure.transaction.jta.JtaAutoConfiguration,\
org.springframework.boot.autoconfigure.validation.ValidationAutoConfiguration,\
org.springframework.boot.autoconfigure.web.DispatcherServletAutoConfiguration,\
org.springframework.boot.autoconfigure.web.EmbeddedServletContainerAutoConfiguration,\
org.springframework.boot.autoconfigure.web.ErrorMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.web.HttpEncodingAutoConfiguration,\
org.springframework.boot.autoconfigure.web.HttpMessageConvertersAutoConfiguration,\
org.springframework.boot.autoconfigure.web.MultipartAutoConfiguration,\
org.springframework.boot.autoconfigure.web.ServerPropertiesAutoConfiguration,\
org.springframework.boot.autoconfigure.web.WebClientAutoConfiguration,\
org.springframework.boot.autoconfigure.web.WebMvcAutoConfiguration,\
org.springframework.boot.autoconfigure.websocket.WebSocketAutoConfiguration,\
org.springframework.boot.autoconfigure.websocket.WebSocketMessagingAutoConfiguration,\
org.springframework.boot.autoconfigure.webservices.WebServicesAutoConfiguration
```

```
# Application Listeners
org.springframework.context.ApplicationListener=\
org.springframework.boot.ClearCachesApplicationListener,\
org.springframework.boot.builder.ParentContextCloserApplicationListener,\
org.springframework.boot.context.FileEncodingApplicationListener,\
org.springframework.boot.context.config.AnsiOutputApplicationListener,\
org.springframework.boot.context.config.ConfigFileApplicationListener,\
org.springframework.boot.context.config.DelegatingApplicationListener,\
org.springframework.boot.liquibase.LiquibaseServiceLocatorApplicationListener,\
org.springframework.boot.logging.ClasspathLoggingApplicationListener,\
org.springframework.boot.logging.LoggingApplicationListener
```

- 加载 application.properties 属性文件.

```
public class ConfigFileApplicationListener
        implements EnvironmentPostProcessor, SmartApplicationListener, Ordered {

    private static final String DEFAULT_PROPERTIES = "defaultProperties";

    // Note the order is from least to most specific (last one wins)
    private static final String DEFAULT_SEARCH_LOCATIONS = "classpath:/,classpath:/config/,file:./,file:./config/";

    private static final String DEFAULT_NAMES = "application";
```

- 通过抛异常获取堆栈,查看是否含有 main 方法

```
private Class<?> deduceMainApplicationClass() {
    try {
        StackTraceElement[] stackTrace = new RuntimeException().getStackTrace();
        for (StackTraceElement stackTraceElement : stackTrace) {
```

```
                if ("main".equals(stackTraceElement.getMethodName())) {
                    return Class.forName(stackTraceElement.getClassName()); //加载 main 方法的类
                }
            }
        }
        catch (ClassNotFoundException ex) {
            // Swallow and continue
        }
        return null;
}
```

- 如果堆栈中有 main 方法,则反射获取@SpringBootApplication 注解.

```
@SpringBootApplication
public class AtCrowdfundingApplication {

   public static void main(String[] args) {
      //SpringApplication.run(AtCrowdfundingApplication.class, args);

      SpringApplication springApplication = new SpringApplication(AtCrowdfundingApplication.class);
      springApplication.setBannerMode(Banner.Mode.OFF);
      springApplication.run(args);
   }
}
```

## 6.3 @SpringBootApplication 注解

如果堆栈中有 main 方法,则反射获取@SpringBootApplication 注解.

SpringBootApplication.class

```
46 @Target(ElementType.TYPE)
47 @Retention(RetentionPolicy.RUNTIME)
48 @Documented
49 @Inherited
50 @SpringBootConfiguration
51 @EnableAutoConfiguration
52 @ComponentScan(excludeFilters = {
53         @Filter(type = FilterType.CUSTOM, classes = TypeExcludeFilter.class),
54         @Filter(type = FilterType.CUSTOM, classes = AutoConfigurationExcludeFilter.class) })
55 public @interface SpringBootApplication {
```

### 6.3.1 @SpringBootConfiguration

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
```

```
@Configuration
public @interface SpringBootConfiguration {


}
```

说明:
@SpringBootConfiguration 所修饰的类,也是一个配置类.
SpringBoot 框架与 Spring 框架区分开,所以在 springBoot 项目中,推荐使用@SpringBootConfiguration

## 6.3.2 @ComponentScan

注意:
Spring 和 SpringMVC 项目集成分别扫描组件
SpringBoot 会自动扫描运行类所在的包,以及子包下的类.

## 6.3.3 @EnableAutoConfiguration

- 自动配置相关模块,例如:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

- 加载@Import(EnableAutoConfigurationImportSelector.class)

```
@SuppressWarnings("deprecation")
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Inherited
@AutoConfigurationPackage
@Import(EnableAutoConfigurationImportSelector.class)
public @interface EnableAutoConfiguration {
```

- 启动自动装配,选择性导入.

EnableAutoConfigurationImportSelector

```
@Deprecated
public class EnableAutoConfigurationImportSelector
extends AutoConfigurationImportSelector {

@Override
protected boolean isEnabled(AnnotationMetadata metadata) {
if (getClass().equals(EnableAutoConfigurationImportSelector.class)) {
```

```
return getEnvironment().getProperty(
EnableAutoConfiguration.ENABLED_OVERRIDE_PROPERTY, Boolean.class,true);
}
return true;
}
}
```

**AutoConfigurationImportSelector**

```
@Override
public String[] selectImports(AnnotationMetadata annotationMetadata) { //选择性导入
if (!isEnabled(annotationMetadata)) {
return NO_IMPORTS;
}
try {
AutoConfigurationMetadata autoConfigurationMetadata = AutoConfigurationMetadataLoader
.loadMetadata(this.beanClassLoader);
AnnotationAttributes attributes = getAttributes(annotationMetadata);
List<String> configurations = getCandidateConfigurations(annotationMetadata,attributes);//选择性导
入
configurations = removeDuplicates(configurations);
configurations = sort(configurations, autoConfigurationMetadata);
Set<String> exclusions = getExclusions(annotationMetadata, attributes);
checkExcludedClasses(configurations, exclusions);
configurations.removeAll(exclusions);
configurations = filter(configurations, autoConfigurationMetadata);
fireAutoConfigurationImportEvents(configurations, exclusions);
return configurations.toArray(new String[configurations.size()]);
}
catch (IOException ex) {
throw new IllegalStateException(ex);
}
}

protected List<String> getCandidateConfigurations(AnnotationMetadata metadata,AnnotationAttributes
attributes) {
List<String> configurations =
SpringFactoriesLoader.loadFactoryNames(getSpringFactoriesLoaderFactoryClass(),
getBeanClassLoader());
Assert.notEmpty(configurations,"No auto configuration classes found in META-INF/spring.factories. If you
"
+ "are using a custom packaging, make sure that file is correct.");
return configurations;
```

```
}
//启用自动装配
protected Class<?> getSpringFactoriesLoaderFactoryClass() {
    return EnableAutoConfiguration.class;
}
```

- 自动装配对应的配置文件.

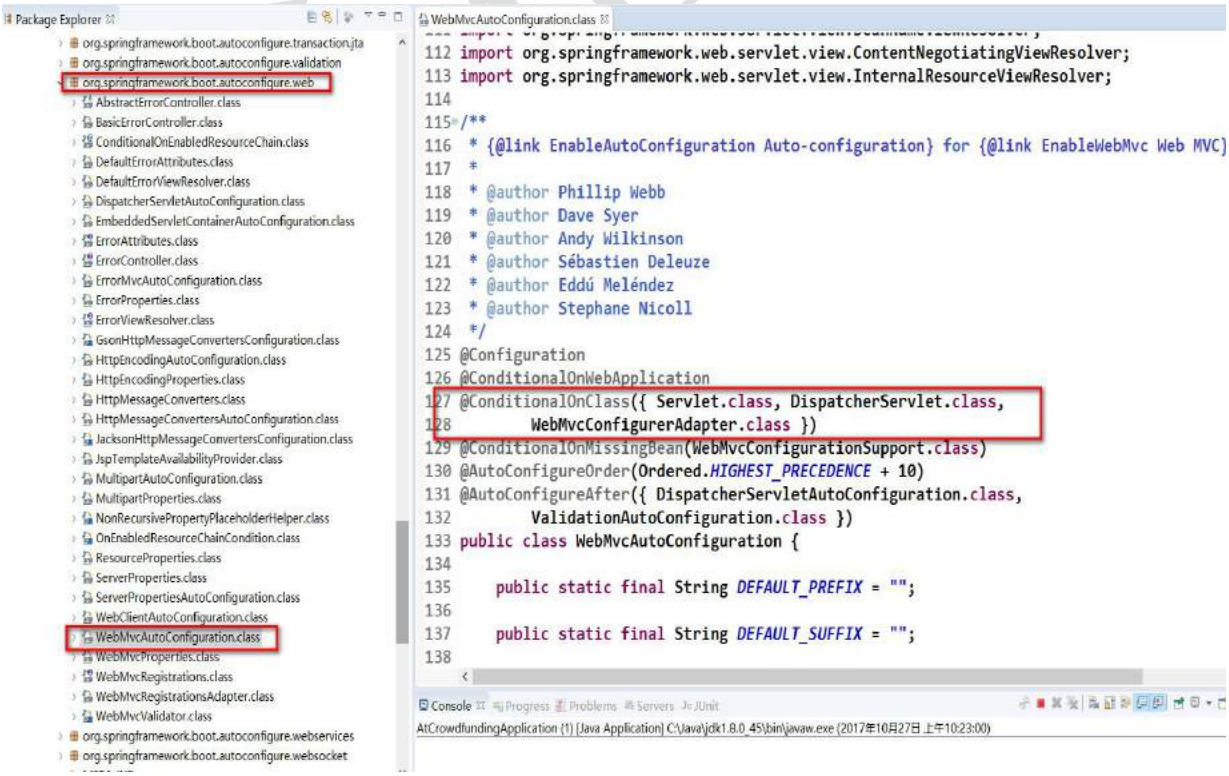
从 spring-boot-autoconfigure-1.5.8.RELEASE.jar 包中找到"META-INF/spring.factories"搜找:

```
# Auto Configure
org.springframework.boot.autoconfigure.EnableAutoConfiguration='
...
org.springframework.boot.autoconfigure.data.redis.RedisAutoConfiguration,\
org.springframework.boot.autoconfigure.data.redis.RedisRepositoriesAutoConfiguration,\
```

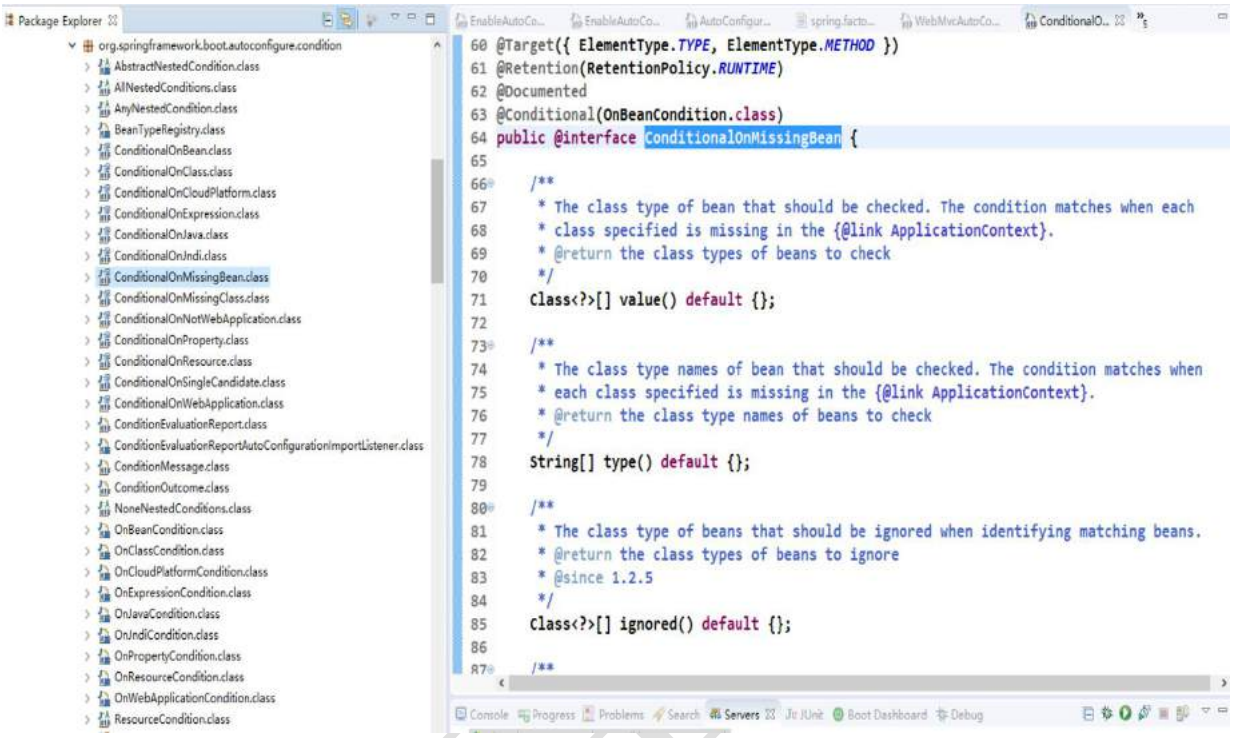
```
@Configuration
@ConditionalOnClass({ JedisConnection.class, RedisOperations.class, Jedis.class }) //根据条件进行自动配置
@EnableConfigurationProperties(RedisProperties.class)
public class RedisAutoConfiguration {
...
```

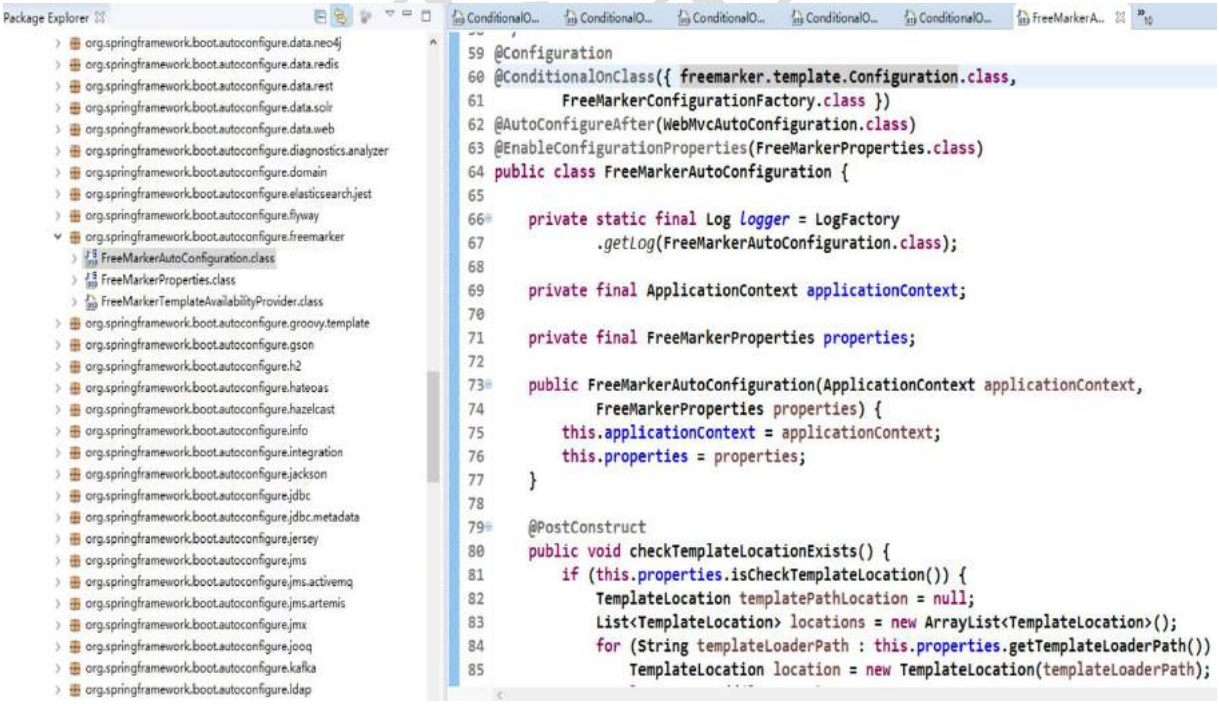
- SpringMVC 之所以可以加载,就是因为 jar 中含有:

Servlet.class, DispatcherServlet.class,WebMvcConfigurerAdapter.class



- 条件注解,参考　附录 4.相关注解




## 6.4 SpringApplication 对象 run 方法

- 运行 Spring Boot 项目:springApplication.run(args);

```
public ConfigurableApplicationContext run(String... args) {
StopWatch stopWatch = new StopWatch(); //秒表开始
stopWatch.start();
ConfigurableApplicationContext context = null; //IOC 容器
FailureAnalyzers analyzers = null;
configureHeadlessProperty();
SpringApplicationRunListeners listeners = getRunListeners(args); //创建监听器对象
listeners.starting(); //启动监听器
try {
ApplicationArguments applicationArguments = new DefaultApplicationArguments(args); //获取命令行参数.
ConfigurableEnvironment environment = prepareEnvironment(listeners,applicationArguments);
Banner printedBanner = printBanner(environment); //打印 Banner
// 如果是 web 环境,则加载 AnnotationConfigEmbeddedWebApplicationContext 类,否则加载 AnnotationConfigApplicationContext 类.
context = createApplicationContext();
analyzers = new FailureAnalyzers(context);
prepareContext(context, environment, listeners, applicationArguments,printedBanner);
refreshContext(context);//初始化 IOC 容器
afterRefresh(context, applicationArguments);
listeners.finished(context, null);
stopWatch.stop();//秒表结束
if (this.logStartupInfo) {
new StartupInfoLogger(this.mainApplicationClass)
.logStarted(getApplicationLog(), stopWatch);//记录启动的时间
}
return context;
}
catch (Throwable ex) {
handleRunFailure(context, listeners, analyzers, ex);
throw new IllegalStateException(ex);
}
}
```

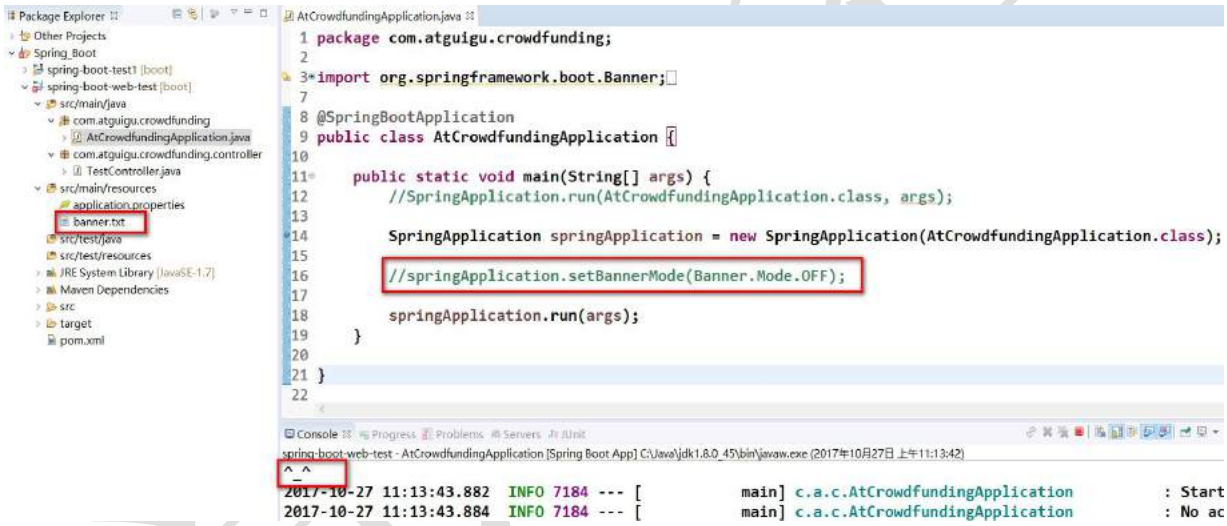
## 6.4.1 自定义 Banner

### 1 打印指定的 banner 或默认的 banner

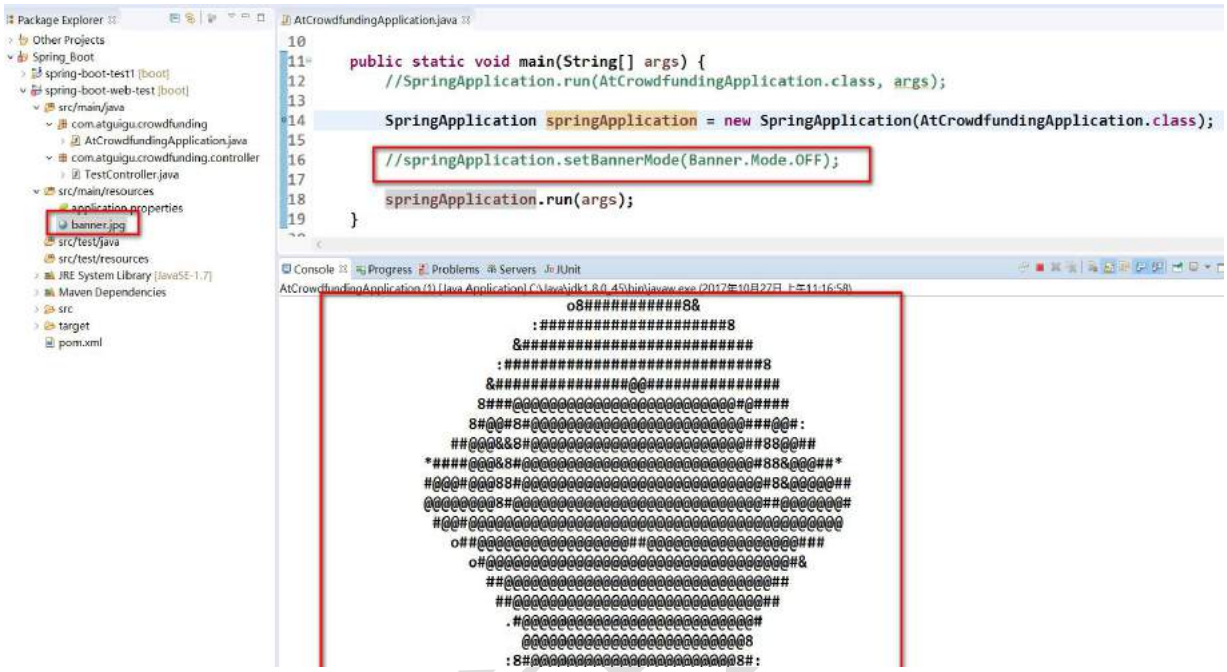
```
private Banner printBanner(ConfigurableEnvironment environment) {
if (this.bannerMode == Banner.Mode.OFF) { //是否打印 banner
```

```
return null;
}
ResourceLoader resourceLoader = this.resourceLoader != null ? this.resourceLoader
: new DefaultResourceLoader(getClassLoader());
SpringApplicationBannerPrinter bannerPrinter = new SpringApplicationBannerPrinter(
resourceLoader, this.banner);
if (this.bannerMode == Mode.LOG) {
return bannerPrinter.print(environment, this.mainApplicationClass, logger);
}
return bannerPrinter.print(environment, this.mainApplicationClass, System.out);
}
```

## 2 文本方式



3 图片方式




## 6.4.2 创建 IOC 容器对象

```
protected ConfigurableApplicationContext createApplicationContext() {
Class<?> contextClass = this.applicationContextClass;
if (contextClass == null) {
try {
// 如果是 web 环境，则加载 AnnotationConfigEmbeddedWebApplicationContext 类，否则加载
AnnotationConfigApplicationContext 类.
//public static final String DEFAULT_WEB_CONTEXT_CLASS = "org.springframework."+
"boot.context.embedded.AnnotationConfigEmbeddedWebApplicationContext";
public static final String DEFAULT_CONTEXT_CLASS = "org.springframework.context."+
"annotation.AnnotationConfigApplicationContext";

contextClass = Class.forName(this.webEnvironment? DEFAULT_WEB_CONTEXT_CLASS :
DEFAULT_CONTEXT_CLASS);
}
catch (ClassNotFoundException ex) {
throw new IllegalStateException(
"Unable create a default ApplicationContext, "
+ "please specify an ApplicationContextClass",ex);
}
```

```
}
return (ConfigurableApplicationContext) BeanUtils.instantiate(contextClass);
}
```

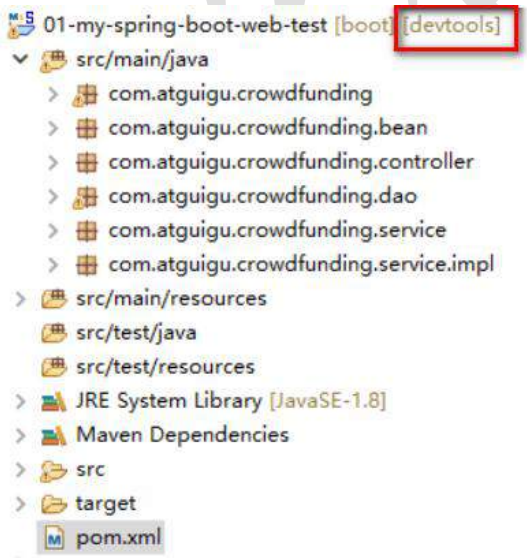
# 第 7 章 热部署

项目开发过程中，常常会改动页面数据或者修改数据结构，为了显示改动效果，往往需要重启应用查看改变效果，其实就是重新编译生成了新的 Class 文件，这个文件里记录着和代码等对应的各种信息，然后 Class 文件将被虚拟机的 ClassLoader 加载。而热部署正是利用了这个特点，它监听到如果有 Class 文件改动了，就会创建一个新的 ClaassLoader 进行加载该文件，经过一系列的过程，最终将结果呈现在我们眼前。

1.Spring Boot 实现热部署很简单，在 pom.xml 中增加相关依赖即可

```
<dependencies>
    ...
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <optional>true</optional><!-- 这个需要为 true 热部署才有效-->
    </dependency>
    ...
</dependencies>
```

2.项目名称后会增加:[devtools]




3.修改类之后,服务器自动重新启动，加载类。在开发环境下使用,在生产环境下屏蔽。

# 第 8 章 动态启用服务器端口

Spring Boot 项目在不同的环境下（开发，测试，生产），服务端口不一致怎么办？
Spring Boot 针对于不同的环境提供了 Profile 支持。通过 profile 的设定来查找不同的配置文件，
如在 application.properties 配置文件中增加配置参数 --spring.profiles.active=test，
那么框架会自动读取 application-test.properties 文件，
在不同的配置文件中设定不同的端口号（8181，8282。。。）即可。

## 8.1 演示 1 - 开发工具中运行

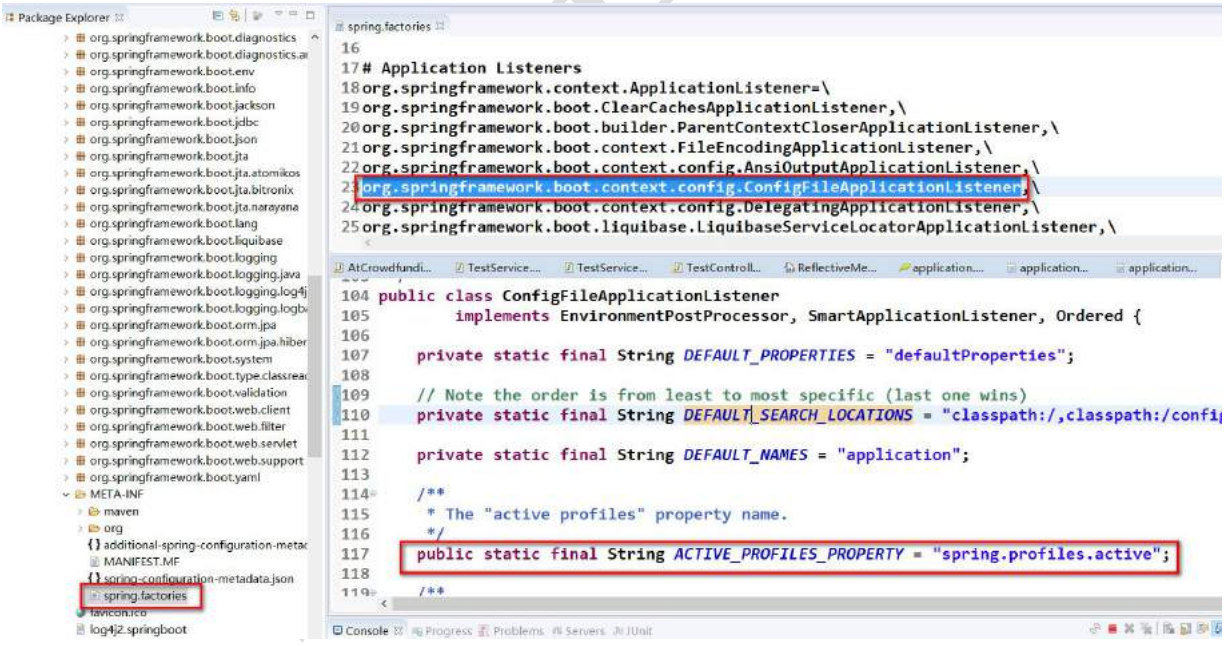
- application-product.properties

```
server.port=8181
```

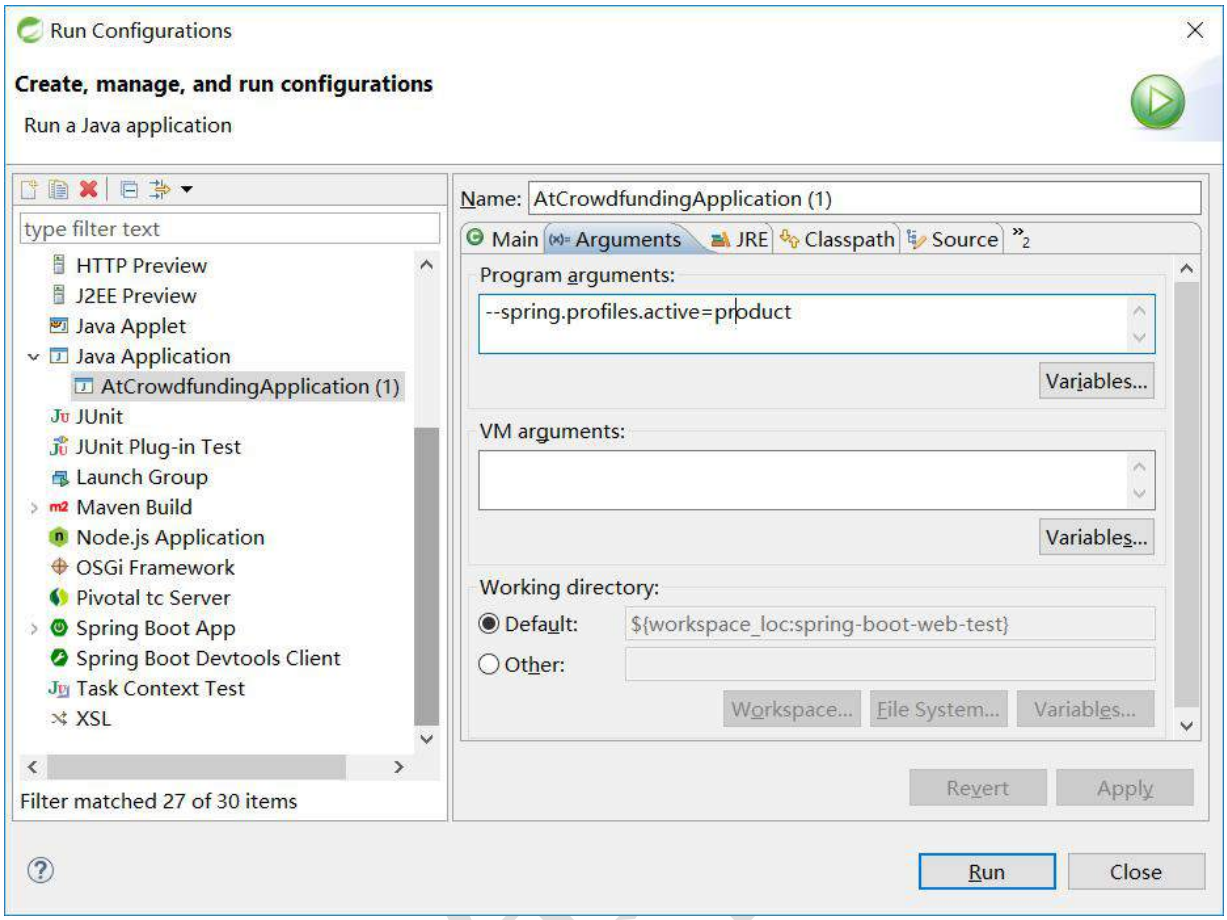
- application-test.properties

```
server.port=8282
```

- 原理

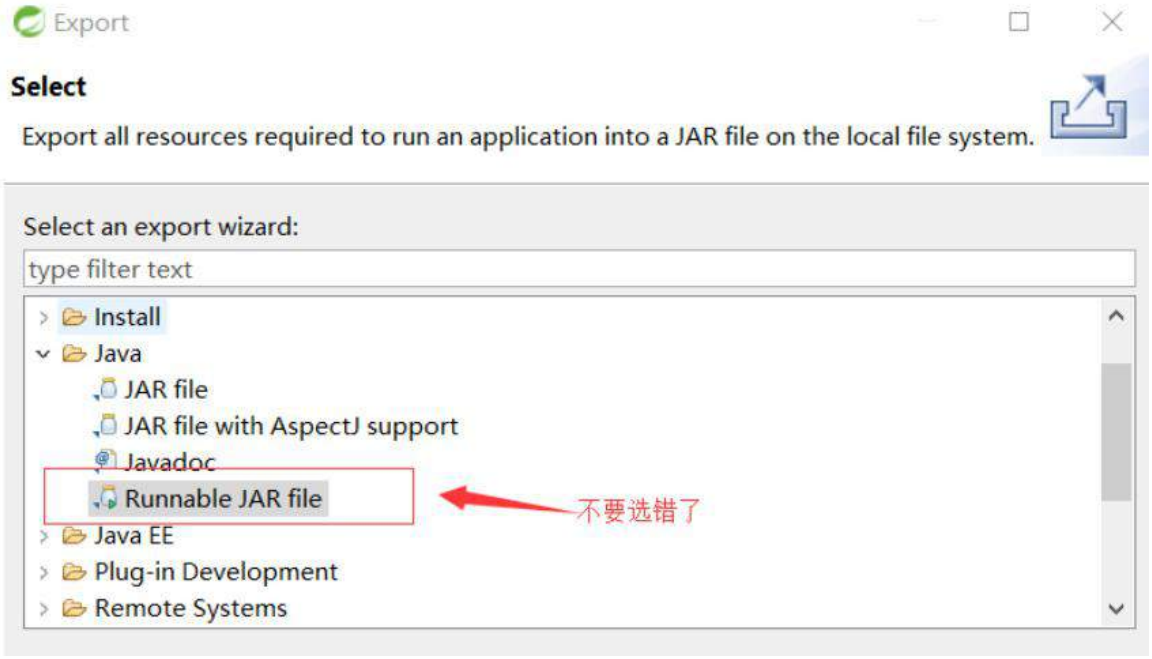



- 动态运行服务器

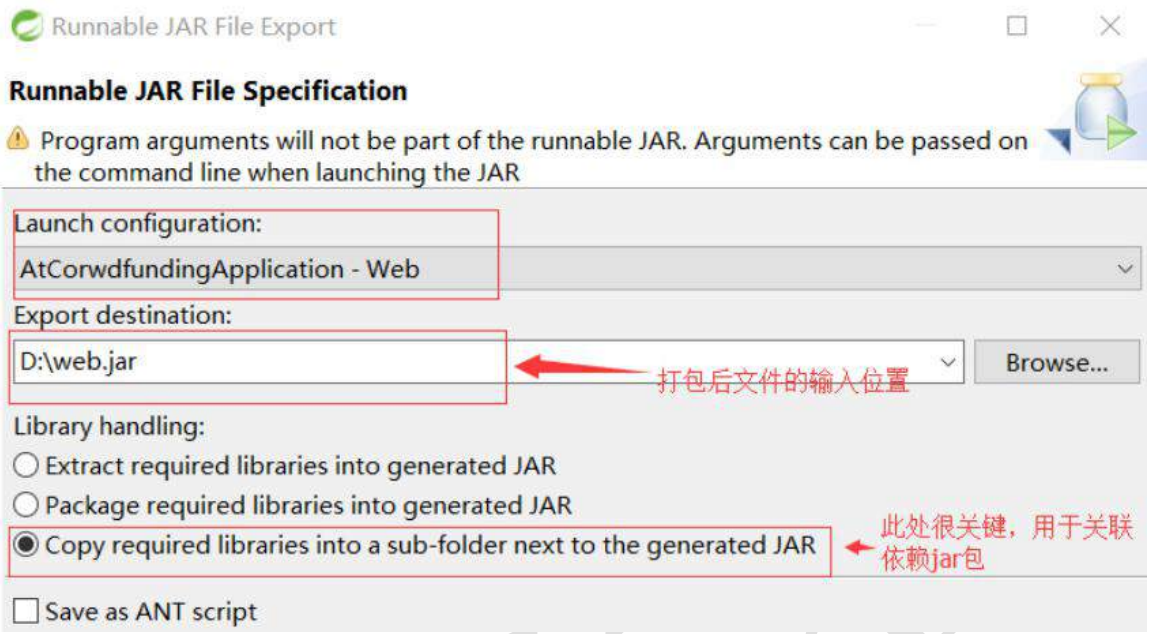



## 8.2 演示 2 - 打 jar 包运行

- 右键点击项目，选择菜单 Export...



- 点击 Next 按钮，执行下一步配置




- 点击 Finish 按钮完成。

输入命令行参数：java -jar hello.jar --spring.profiles.active=test

- 运行报错:

**数据库文件位置不对.将 src/main/resources 下的文件复制到 src/main/java 目录下,再打 jar 包即可.**

# 第 9 章 分布式集群 - 集成 Nginx

因为 Spring Boot 框架内嵌 Tomcat 服务器，所以本章的分布式集群依然是基于 Tomcat + Nginx

Nginx 作为反向代理服务器，收到请求后将通过 http 协议转发给 Tomcat，所以 Tomcat 需要从从 HTTP 头信息中去获取协议信息。

## 9.1 演示 1 - 内置 Tomcat

- 在 application.properties 中增加配置信息。

```
...
server.address=127.0.0.1
server.tomcat.remote-ip-header=x-forwarded-for
server.tomcat.protocol-header=x-forwarded-proto
server.tomcat.port-header=X-Forwarded-Port
server.use-forward-headers=true
...
```

- 注意:

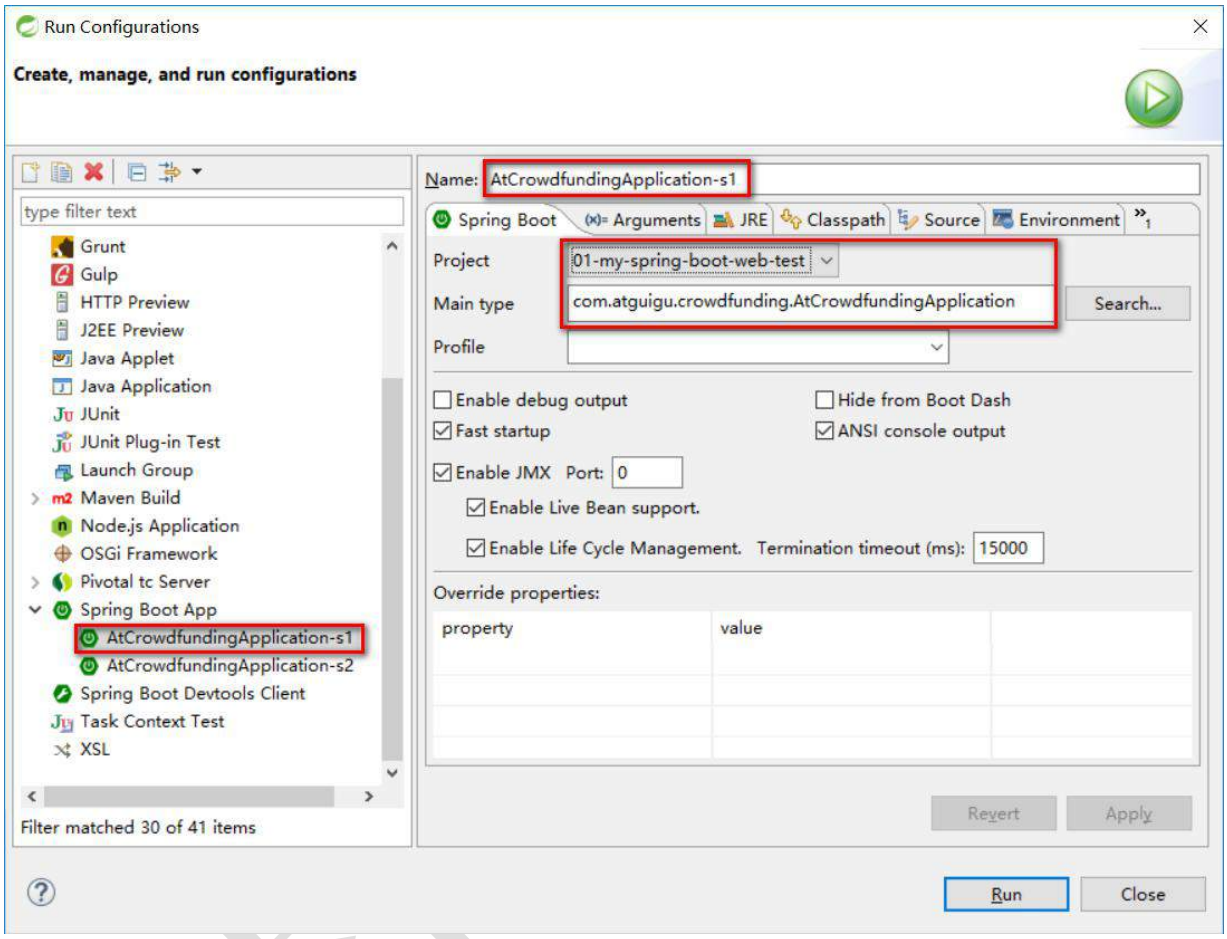
设置 127.0.0.1 不允许远程连接本服务器.这样可以限制用户必须通过反向代理服务器进行访问.

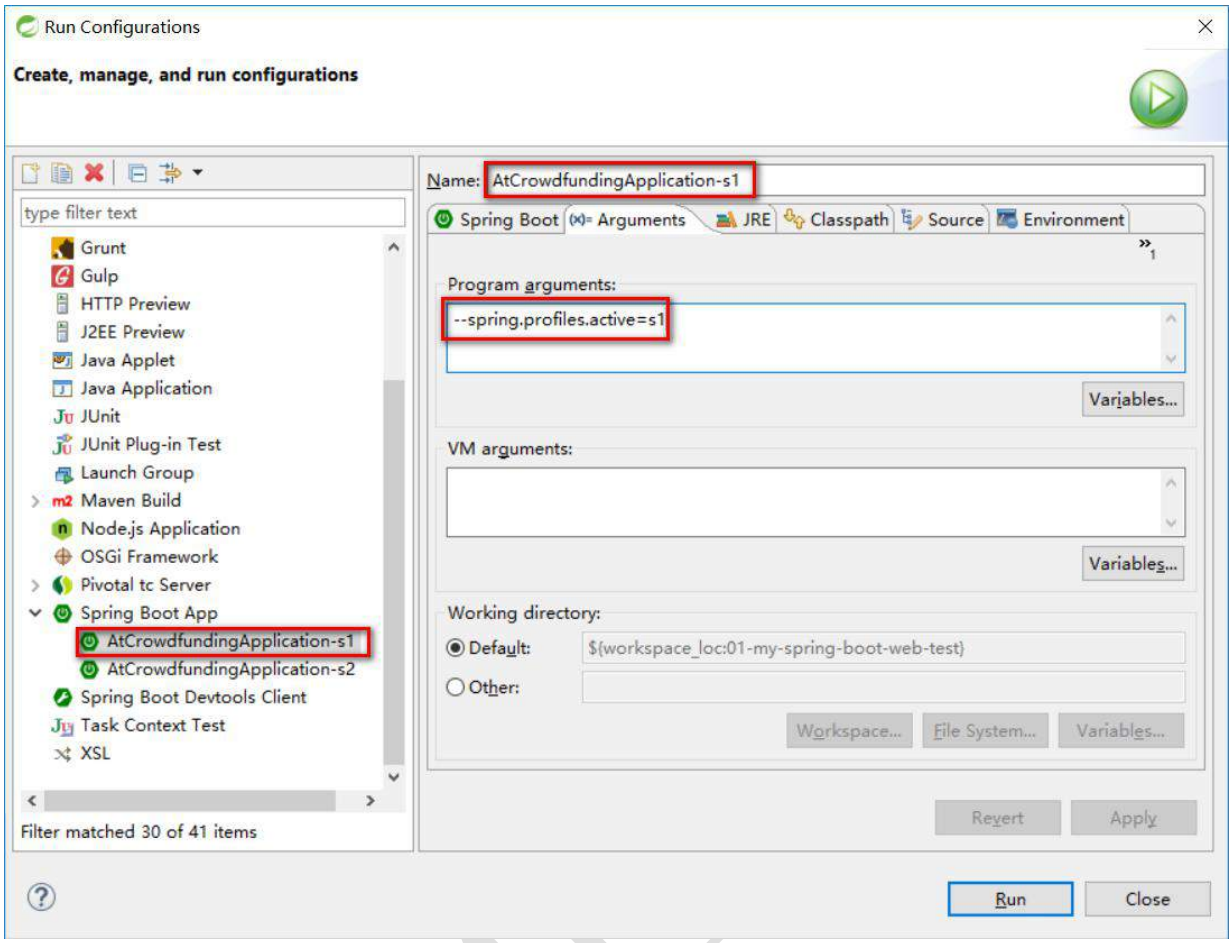
application-s1.properties

server.port=8080

application-s2.properties

server.port=8181






## 9.2 演示 2 - 外置 Tomcat

- 如何将 Spring Boot 项目发布为 war 包独立部署？
- 将当前的项目 pom.xml 中(<packaging>war</packaging>)打包方式设置为 war。(需要 Maven update, 否则,可能不起作用)
- 将项目中 tomcat 模块的依赖类库设置为 provided。

```
...
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-tomcat</artifactId>
    <scope>provided</scope>
</dependency>
...
```

- 将当前 Application 启动类继承 SpringBootServletInitializer，并重写方法 configure。

```
@Override
protected SpringApplicationBuilder configure(SpringApplicationBuilder builder) {
    builder.sources(Application.class);// Application 为启动类
```

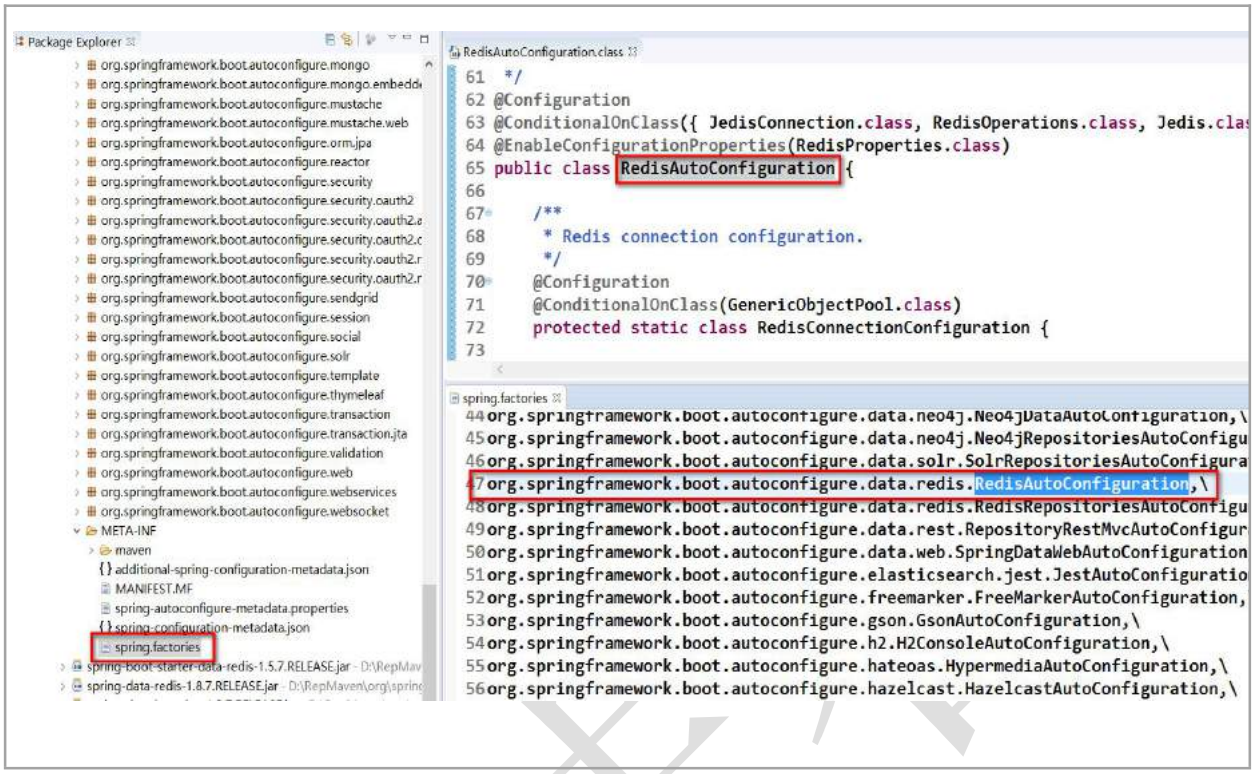
```
        return super.configure(builder);
}
```

## 9.3 演示 3 - Redis 缓存

### 9.3.1 增加 Redis 依赖关系

Spring Boot 框架在集群时需要设置 Session 共享，框架当前只支持 Redis 缓存作为 Session 共享组件。在项目 pom.xml 配置文件中增加 Redis 依赖关系。这样框架就可以自动加载 Redis 配置.

```
<project>
    ...
    <dependencies>
        ...
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.session</groupId>
            <artifactId>spring-session-data-redis</artifactId>
        </dependency>
        ...
    </dependencies>
    ...
</project>
```




## 9.3.2 增加 Redis 相关配置

```
...
spring.session.store-type=redis
spring.redis.database=1
spring.redis.host=127.0.0.1
spring.redis.password=123123
spring.redis.port=6379
spring.redis.pool.max-idle=8
spring.redis.pool.min-idle=0
spring.redis.pool.max-active=8
spring.redis.pool.max-wait=-1
spring.redis.timeout=60000
...
```

spring.session.store-type 取值要么是 redis,要么是 none

## 9.3.3 增加集成配置类

```
@Configuration
@EnableRedisHttpSession
public class RedisSessionConfig {
```

```
}
```

### 9.3.4 实体对象序列化

设置 Redis Session 共享后，如果向 Session 中保存数据，需要让数据对象实现可序列化接口 java.io.Serializable！！！ 分别启动 Nginx, 2 个 Web 应用，观察是否集成成功

# 第 10 章 Redis 安装启动

## 10.1 安装

将软件压缩包 Redis-x64-3.0.500.zip 解压缩放置在无中文，无空格的路径中

## 10.2 配置

修改其中的 redis.windows.conf 文件，增加数据库密码

```
requirepass 123123
```

## 10.3 启动

在当前文件夹中，执行 DOS 命令

```
redis-server.exe    redis.windows.conf
```

当出现如下内容，即表示 Redis 启动成功，窗口不要关闭


# 第 11 章 健康监控

生产环境中，需要实时或定期监控服务的可用性。Spring Boot 的 actuator（监控）功能提供了很多监控所需的接口。

## 11.1 增加依赖关系

```
...
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
...
```

## 11.2 设置端口

在 application.yml 配置文件中指明监控端口，如果未指明，那么监控端口和当前应用端口一致。

```
...
management:
  port: 8400
...
```

## 11.3 查看健康指标

输入网址 http://localhost:8400/health 查看健康指标
{"status":"UP"}

## 11.4 配置项目信息

在 application.yml 配置文件中增加环境信息，数据可从 pom.xml 文件中获取

```
...
info:
  app:
    name: "@project.name@"
    description: "@project.description@"
    version: "@project.version@"
    spring-boot-version: "@project.parent.version@"
...
```

## 11.5 查看项目信息

输入网址 http://localhost:8400/info 查看配置
{"app":{"name":"springboot-member","description":"Parent pom providing dependency and plugin management for applications built with Maven","version":"1.0","spring-boot-version":"1.5.7.RELEASE"}}

## 11.6 具体监控指标

| 监控指标 | 说明 |
| --- | --- |
| /health | 提供应用程序的健康状态 |
| /info | 显示应用程序的基本描述 |
| /env | 提供应用程序的环境变量 |
| /metrics | 显示 Metrics 子系统管理的信息 |
| /configprops | 提供不同配置对象 |
| /autoconfig | 以 web 形式对外暴露 AutoConfiguration 信息 |
| /beans | 列出所有由 Spring Boot 创建的 bean |
| /mapping | 显示当前应用支持的 URL 映射 |

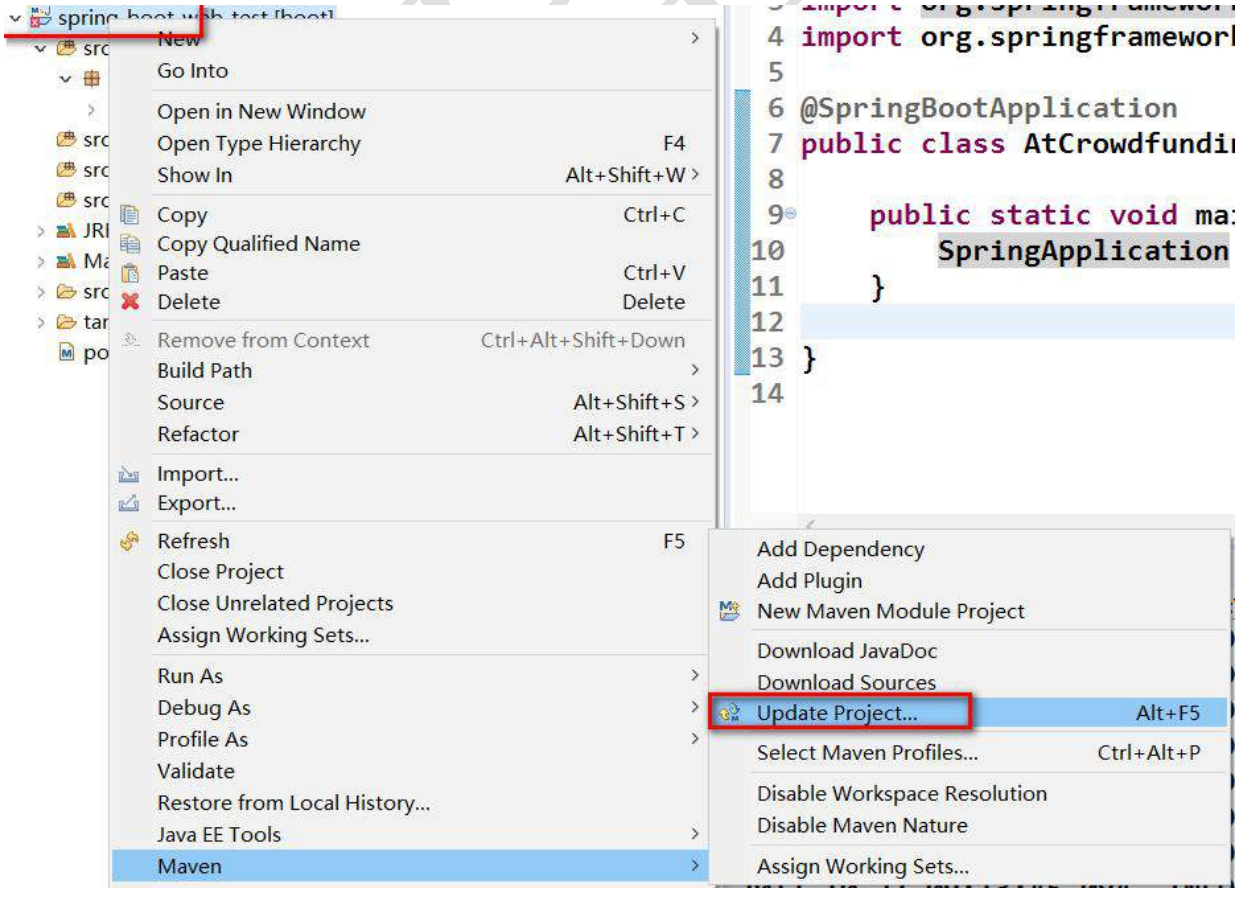
# 附录 1 Spring Boot 相关模块

表 6-1 官方提供的 starter pom

| 名　称 | 描　述 |
|---|---|
| spring-boot-starter | Spring Boot 核心 starter，包含自动配置、日志、yaml 配置文件的支持 |
| spring-boot-starter-actuator | 准生产特性，用来监控和管理应用 |
| spring-boot-starter-remote-shell | 提供基于 ssh 协议的监控和管理 |
| spring-boot-starter-amqp | 使用 spring-rabbit 来支持 AMQP |
| spring-boot-starter-aop | 使用 spring-aop 和 AspectJ 支持面向切面编程 |
| spring-boot-starter-batch | 对 Spring Batch 的支持 |
| spring-boot-starter-cache | 对 Spring Cache 抽象的支持 |
| spring-boot-starter-cloud-connectors | 对云平台（Cloud Foundry、Heroku）提供的服务提供简化的连接方式 |
| spring-boot-starter-data-elasticsearch | 通过 spring-data-elasticsearch 对 Elasticsearch 支持 |
| spring-boot-starter-data-gemfire | 通过 spring-data-gemfire 对分布式存储 GemFire 的支持 |
| spring-boot-starter-data-jpa | 对 JPA 的支持，包含 spring-data-jpa、spring-orm 和 Hibernate |
| spring-boot-starter-data-mongodb | 通过 spring-data-mongodb，对 MongoDB 进行支持 |
| spring-boot-starter-data-rest | 通过 spring-data-rest-webmvc 将 Spring Data repository 暴露为 REST 形式的服务 |
| spring-boot-starter-data-solr | 通过 spring-data-solr 对 Apache Solr 数据检索平台的支持 |
| spring-boot-starter-freemarker | 对 FreeMarker 模板引擎的支持 |
| spring-boot-starter-groovy-templates | 对 Groovy 模板引擎的支持 |
| spring-boot-starter-hateoas | 通过 spring-hateoas 对基于 HATEOAS 的 REST 形式的网络服务的支持 |
| spring-boot-starter-hornetq | 通过 HornetQ 对 JMS 的支持 |
| spring-boot-starter-integration | 对系统集成框架 spring-integration 的支持 |
| spring-boot-starter-jdbc | 对 JDBC 数据库的支持 |
| spring-boot-starter-jersey | 对 Jersery REST 形式的网络服务的支持 |
| spring-boot-starter-jta-atomikos | 通过 Atomikos 对分布式事务的支持 |
| spring-boot-starter-jta-bitronix | 通过 Bitronix 对分布式事务的支持 |
| spring-boot-starter-mail | 对 javax.mail 的支持 |
| spring-boot-starter-mobile | 对 spring-mobile 的支持 |
| spring-boot-starter-mustache | 对 Mustache 模板引擎的支持 |
| spring-boot-starter-redis | 对键值对内存数据库 Redis 的支持，包含 spring-redis |

| | |
|---|---|
| spring-boot-starter-security | 对 spring-security 的支持 |
| spring-boot-starter-social-facebook | 通过 spring-social-facebook 对 Facebook 的支持 |
| spring-boot-starter-social-linkedin | 通过 spring-social-linkedin 对 Linkedin 的支持 |
| spring-boot-starter-social-twitter | 通过 spring-social-twitter 对 Twitter 的支持 |
| spring-boot-starter-test | 对常用的测试框架 JUnit、Hamcrest 和 Mockito 的支持，包含 spring-test 模块 |
| spring-boot-starter-thymeleaf | 对 Thymeleaf 模板引擎的支持，包含于 Spring 整合的配置 |
| spring-boot-starter-velocity | 对 Velocity 模板引擎的支持 |
| spring-boot-starter-web | 对 Web 项目开发的支持，包含 Tomcat 和 spring-webmvc |
| spring-boot-starter-Tomcat | Spring Boot 默认的 Servlet 容器 Tomcat |
| spring-boot-starter-Jetty | 使用 Jetty 作为 Servlet 容器替换 Tomcat |
| spring-boot-starter-undertow | 使用 Undertow 作为 Servlet 容器替换 Tomcat |
| spring-boot-starter-logging | Spring Boot 默认的日志框架 Logback |
| spring-boot-starter-log4j | 支持使用 Log4J 日志框架 |
| spring-boot-starter-websocket | 对 WebSocket 开发的支持 |
| spring-boot-starter-ws | 对 Spring Web Services 的支持 |

# 附录 2 项目报小红叉



# 附录 3 yml 详细配置

# 附录 4 相关注解

## 4.1 @ImportResource

导入 XML 文件.
@ImportResource({"classpath:some-context.xml","classpath:other-context.xml"})

## 4.2 java -jar xx.jar --server.port=9090

## 4.3 常规属性配置

application.properties
book.author=zhangsan
book.price=22.0
在类的成员变量上使用@Value 注解

```
@Value("${book.author}")
private String bookAuthor
```

采用@ConfigurationProperties 自动注入
org.springframework.boot.context.properties.ConfigurationProperties

```
@Components
```

```
@ConfigurationProperties(prefix="book",locations="classpath:config/application.properties",ignoreUnknownFields = true)
```

```
Public class User{
private String author ;
private double price ;
//    Get/set…
}
```

## 4.4 日志配置

Spring boot 使用 Logback 作为日志框架.
logging.file=D:/temp/log.log
logging.level.org.springframework.web=DEBUG

## 4.5 自动扫描

@EnableAutoConfiguration 默认扫描 main 类所在的包及子包.

## 4.6 条件注解

@ConditionalOnBean：当容器里有指定的 Bean 的条件下。

@ConditionalOnClass：当类路径下有指定的类的条件下。

@ConditionalOnExpression：基于 SpEL 表达式作为判断条件。

@ConditionalOnJava：基于 JVM 版本作为判断条件。

@ConditionalOnJndi：在 JNDI 存在的条件下查找指定的位置。

@ConditionalOnMissingBean：当容器里没有指定 Bean 的情况下。

@ConditionalOnMissingClass：当类路径下没有指定的类的条件下。

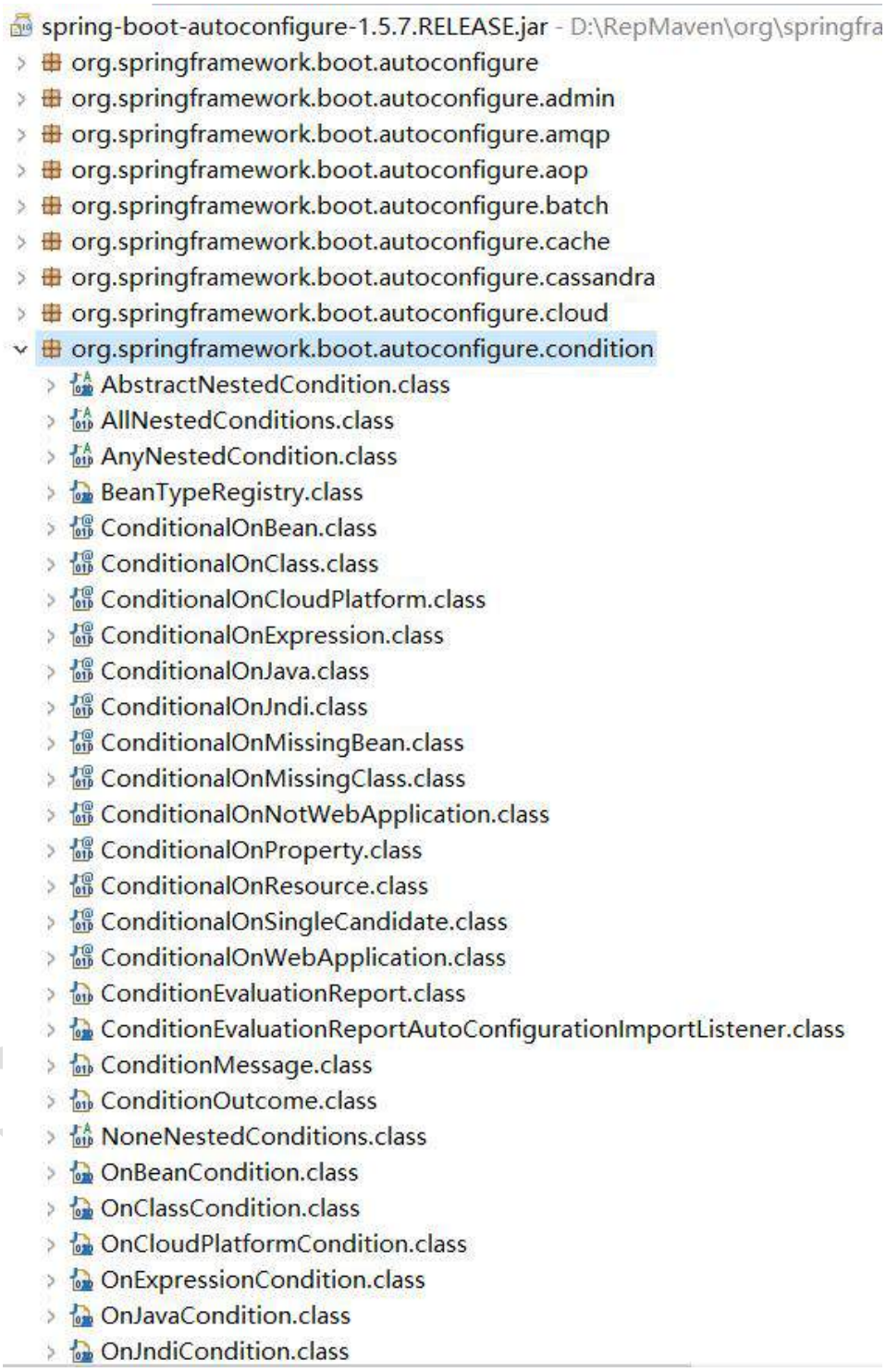
@ConditionalOnNotWebApplication：当前项目不是 Web 项目的条件下。

@ConditionalOnProperty：指定的属性是否有指定的值。

@ConditionalOnResource：类路径是否有指定的值。

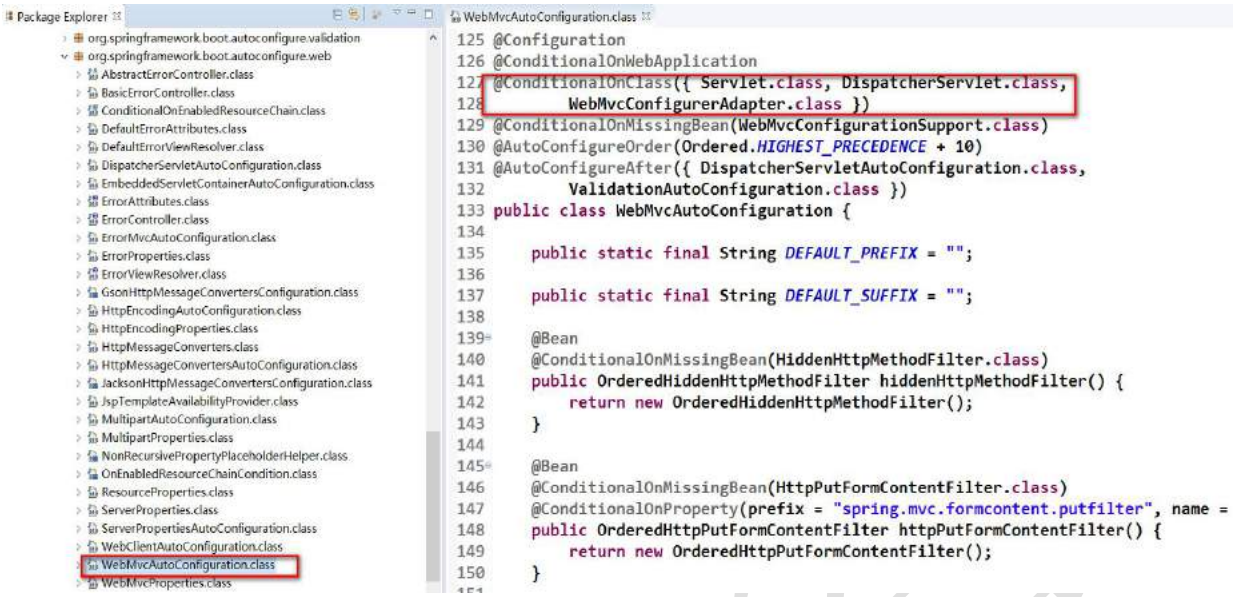
@ConditionalOnSingleCandidate：当指定 Bean 在容器中只有一个，或者虽然有多个但是指定首选的 Bean。

@ConditionalOnWebApplication：当前项目是 Web 项目的条件下。




- 案例:

是否自动加载 SpringMVC 相关配置.




## 4.7 @RestController

```
@Target(ElementType.TYPE)
@Retention(RetentionPolicy.RUNTIME)
@Documented
@Controller
@ResponseBody
public @interface RestController {
    String value() default "";
}
```

## 4.8 修改 Favicon(偏爱图标；网站图标)配置

Spring.mvc.favicon.enabled=false　默认 true 开启.
在 META_INF/resources/下,类路径 resources/下,类路径 static/下或类路径 public 下,增加 favicon.ico 文件就可以修改掉默认图标

# 附录 5 字符编码

## 5.1 字符编码过滤器

### 5.1.1 之前配置

```
<!-- 配置字符集 -->
<filter>
    <filter-name>encodingFilter</filter-name>
        <filter-class>org.springframework.web.filter.CharacterEncodingFilter</filter-class>
    <init-param>
    <param-name>encoding</param-name>
        <param-value>UTF-8</param-value>
    </init-param>
    <init-param>
    <!--Spring 里的字符过滤器 CharacterEncodingFilter 是针对请求的，forceEncoding=true 是意思是指无论客户端请求是否包含了编码，都用过滤器里的编码来解析请求 -->
        <param-name>forceEncoding</param-name>
        <param-value>true</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>encodingFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

### 5.1.2 现在配置

- 默认配置

```
# default    UTF-8    and    true
spring.http.encoding.charset=UTF-8
spring.http.encoding.force=true
```

- **org.springframework.boot.autoconfigure.web.HttpEncodingProperties**

```
/*
 * Copyright 2012-2016 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
```

```
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
 *          http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.springframework.boot.autoconfigure.web;

import java.nio.charset.Charset;
import java.util.Locale;
import java.util.Map;

import org.springframework.boot.context.properties.ConfigurationProperties;

/**
 * Configuration properties for http encoding.
 *
 * @author Stephane Nicoll
 * @author Brian Clozel
 * @since 1.2.0
 */
@ConfigurationProperties(prefix = "spring.http.encoding")
public class HttpEncodingProperties {

public static final Charset DEFAULT_CHARSET = Charset.forName("UTF-8");

/**
 * Charset of HTTP requests and responses. Added to the "Content-Type" header if not
 * set explicitly.
 */
private Charset charset = DEFAULT_CHARSET;

/**
 * Force the encoding to the configured charset on HTTP requests and responses.
 */
```

```
private Boolean force;

/**
 * Force the encoding to the configured charset on HTTP requests. Defaults to true
 * when "force" has not been specified.
 */
private Boolean forceRequest;

/**
 * Force the encoding to the configured charset on HTTP responses.
 */
private Boolean forceResponse;

/**
 * Locale to Encoding mapping.
 */
private Map<Locale, Charset> mapping;

public Charset getCharset() {
return this.charset;
}

public void setCharset(Charset charset) {
this.charset = charset;
}

public boolean isForce() {
return Boolean.TRUE.equals(this.force);
}

public void setForce(boolean force) {
this.force = force;
}

public boolean isForceRequest() {
return Boolean.TRUE.equals(this.forceRequest);
}

public void setForceRequest(boolean forceRequest) {
this.forceRequest = forceRequest;
}
```

```
public boolean isForceResponse() {
return Boolean.TRUE.equals(this.forceResponse);
}

public void setForceResponse(boolean forceResponse) {
this.forceResponse = forceResponse;
}

public Map<Locale, Charset> getMapping() {
return this.mapping;
}

public void setMapping(Map<Locale, Charset> mapping) {
this.mapping = mapping;
}

boolean shouldForce(Type type) {
Boolean force = (type == Type.REQUEST ? this.forceRequest : this.forceResponse);
if (force == null) {
force = this.force;
}
if (force == null) {
force = (type == Type.REQUEST);
}
return force;
}

enum Type {

REQUEST, RESPONSE

}

}
```

- **org.springframework.boot.autoconfigure.web.HttpEncodingAutoConfiguration**

```
/*
 * Copyright 2012-2016 the original author or authors.
 *
 * Licensed under the Apache License, Version 2.0 (the "License");
 * you may not use this file except in compliance with the License.
 * You may obtain a copy of the License at
 *
```

```
 *        http://www.apache.org/licenses/LICENSE-2.0
 *
 * Unless required by applicable law or agreed to in writing, software
 * distributed under the License is distributed on an "AS IS" BASIS,
 * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
 * See the License for the specific language governing permissions and
 * limitations under the License.
 */

package org.springframework.boot.autoconfigure.web;

import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.condition.ConditionalOnClass;
import org.springframework.boot.autoconfigure.condition.ConditionalOnMissingBean;
import org.springframework.boot.autoconfigure.condition.ConditionalOnProperty;
import org.springframework.boot.autoconfigure.condition.ConditionalOnWebApplication;
import org.springframework.boot.autoconfigure.web.HttpEncodingProperties.Type;
import org.springframework.boot.context.embedded.ConfigurableEmbeddedServletContainer;
import org.springframework.boot.context.embedded.EmbeddedServletContainerCustomizer;
import org.springframework.boot.context.properties.EnableConfigurationProperties;
import org.springframework.boot.web.filter.OrderedCharacterEncodingFilter;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.Ordered;
import org.springframework.web.filter.CharacterEncodingFilter;

/**
 * {@link EnableAutoConfiguration Auto-configuration} for configuring the encoding to use
 * in web applications.
 *
 * @author Stephane Nicoll
 * @author Brian Clozel
 * @since 1.2.0
 */
@Configuration
@EnableConfigurationProperties(HttpEncodingProperties.class)
@ConditionalOnWebApplication
@ConditionalOnClass(CharacterEncodingFilter.class)
@ConditionalOnProperty(prefix = "spring.http.encoding", value = "enabled", matchIfMissing = true)
public class HttpEncodingAutoConfiguration {
```

```
private final HttpEncodingProperties properties;

public HttpEncodingAutoConfiguration(HttpEncodingProperties properties) {
this.properties = properties;
}

@Bean
@ConditionalOnMissingBean(CharacterEncodingFilter.class)
public CharacterEncodingFilter characterEncodingFilter() {
CharacterEncodingFilter filter = new OrderedCharacterEncodingFilter();
filter.setEncoding(this.properties.getCharset().name());
filter.setForceRequestEncoding(this.properties.shouldForce(Type.REQUEST));
filter.setForceResponseEncoding(this.properties.shouldForce(Type.RESPONSE));
return filter;
}

@Bean
public LocaleCharsetMappingsCustomizer localeCharsetMappingsCustomizer() {
return new LocaleCharsetMappingsCustomizer(this.properties);
}

private static class LocaleCharsetMappingsCustomizer
implements EmbeddedServletContainerCustomizer, Ordered {

private final HttpEncodingProperties properties;

LocaleCharsetMappingsCustomizer(HttpEncodingProperties properties) {
this.properties = properties;
}

@Override
public void customize(ConfigurableEmbeddedServletContainer container) {
if (this.properties.getMapping() != null) {
container.setLocaleCharsetMappings(this.properties.getMapping());
}
}

@Override
public int getOrder() {
return 0;
}
```

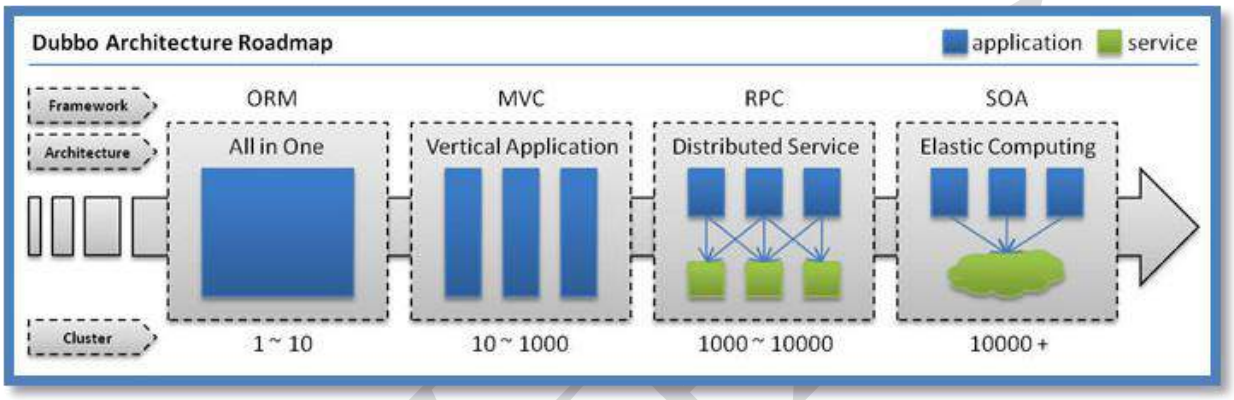
```
}

}
```

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 第 1 章 软件系统架构演变




**单一应用架构**

当网站流量很小时，只需一个应用，将所有功能都部署在一起，以减少部署节点和成本。此时，用于简化增删改查工作量的**数据访问框架(ORM)是关键**。

**垂直应用架构**

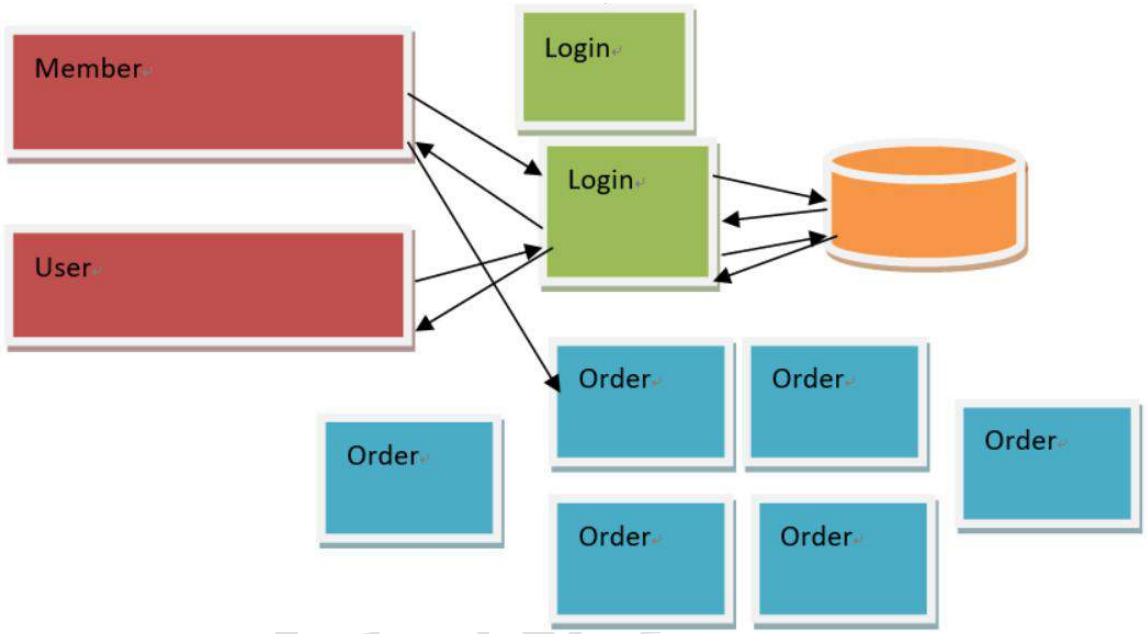
当访问量逐渐增大，单一应用增加机器带来的加速度越来越小，将应用拆成互不相干的几个应用，以提升效率。此时，用于加速前端页面开发的 **Web 框架(MVC)是关键**。

**分布式服务架构**

当垂直应用越来越多，应用之间交互不可避免，将核心业务抽取出来，作为独立的服务，逐渐形成稳定的服务中心，使前端应用能更快速的响应多变的市场需求。此时，用于提高业务复用及整合的**分布式服务框架(RPC)是关键**。

**流动计算架构**

当服务越来越多，容量的评估，小服务资源的浪费等问题逐渐显现，此时需增加**一个调度中心**基于访问压力实时管理集群容量，提高集群利用率。此时，用于提高机器利用率的**资源调度和治理中心(SOA)是关键**。




# 第 2 章 Spring Cloud 概念

## 2.1 概念

Spring Cloud 是一系列框架的有序集合。它利用 Spring Boot 的开发便利性巧妙地简化了分布式系统基础设施的开发，如服务发现注册、配置中心、消息总线、负载均衡、断路器、数据监控等，都可以用 Spring Boot 的开发风格做到一键启动和部署。Spring 并没有重复制造轮子，它只是将目前各家公司开发的比较成熟、经得起实际考验的服务框架组合起来，通过 Spring Boot 风格进行再封装屏蔽掉了复杂的配置和实现原理，最终给开发者留出了一套简单易懂、易部署和易维护的分布式系统开发工具包。

“微服务架构”在这几年非常的火热，以至于关于微服务架构相关的开源产品被反复的提及（比如：netflix、dubbo），Spring Cloud 也因 Spring 社区的强大知名度和影响力也被广大架构师与开发者备受

关注。

那么什么是“微服务架构”呢？简单的说，微服务架构就是将一个完整的应用从数据存储开始垂直拆分成多个不同的服务，每个服务都能独立部署、独立维护、独立扩展，服务与服务间通过诸如 RESTful API 的方式互相调用。

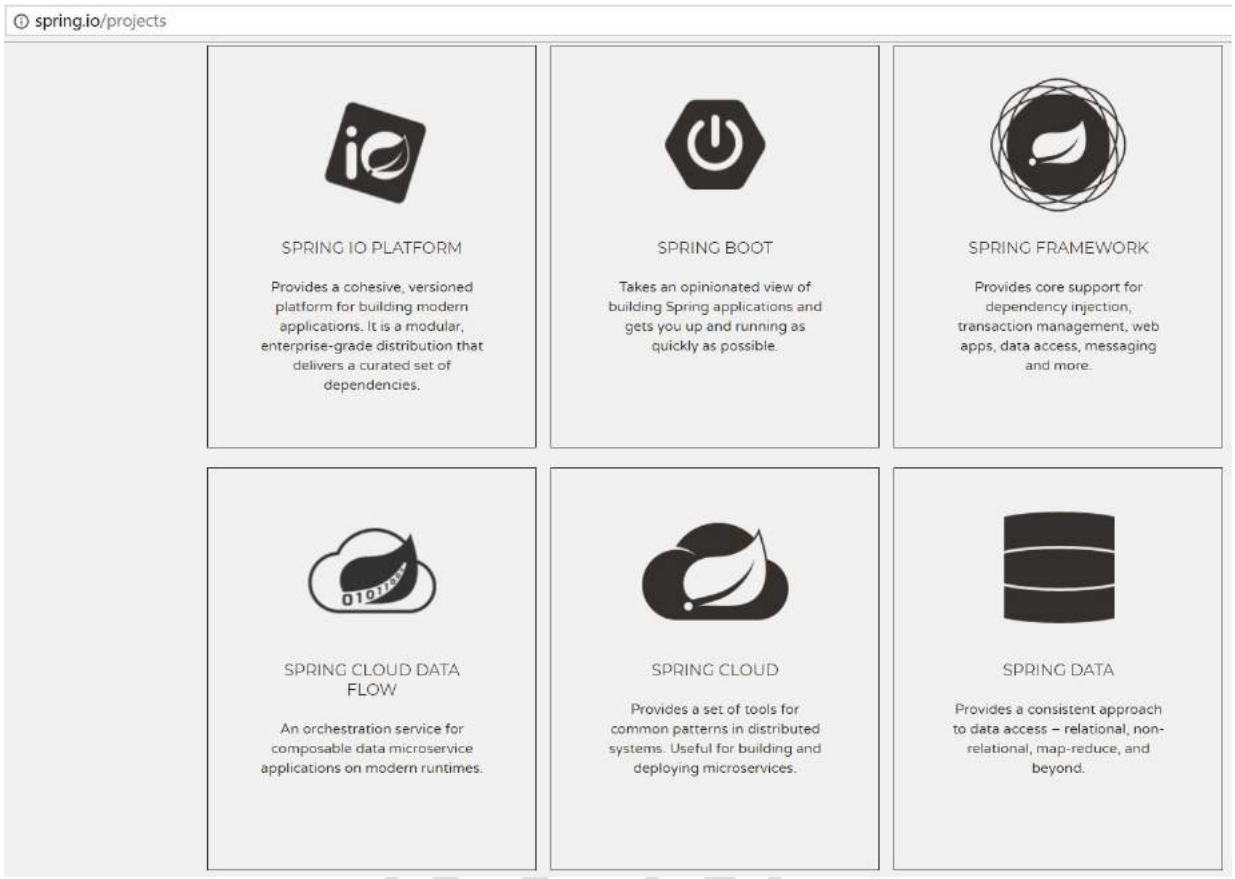
## 2.2 为什么使用 Spring Cloud

Spring Cloud 对于中小型互联网公司来说是一种福音，因为这类公司往往没有实力或者没有足够的资金投入去开发自己的分布式系统基础设施，使用 Spring Cloud 一站式解决方案能在从容应对业务发展的同时大大减少开发成本。同时，随着近几年微服务架构和 Docker 容器概念的火爆，也会让 Spring Cloud 在未来越来越“云”化的软件开发风格中立有一席之地，尤其是在目前五花八门的分布式解决方案中提供了标准化的、全站式的技术方案，意义可能会堪比当前 Servlet 规范的诞生，有效推进服务端软件系统技术水平的进步。

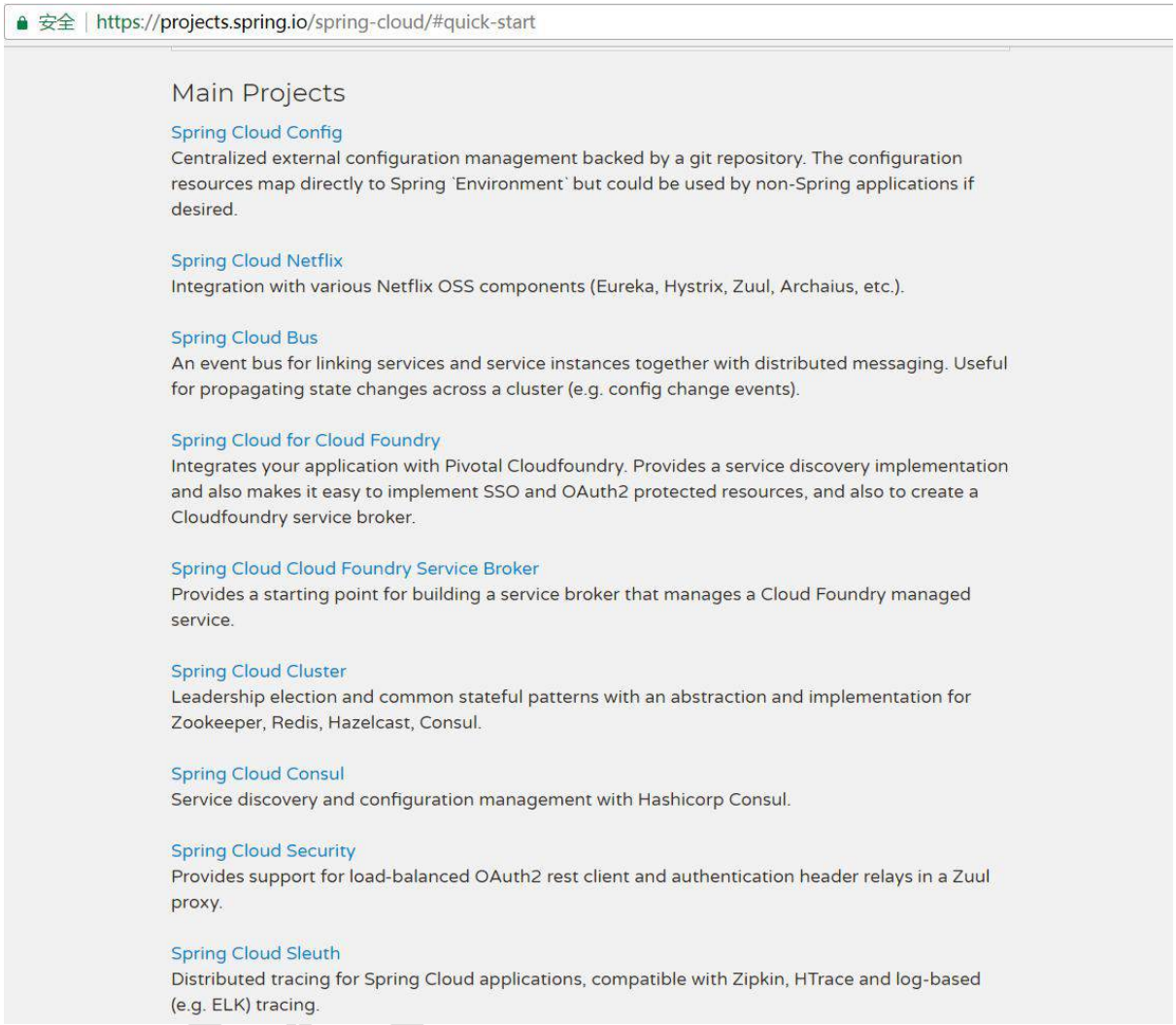
## 2.3 应用 Spring Cloud

Spring Cloud Netflix 项目是 Spring Cloud 的子项目之一，主要内容是对 Netflix 公司一系列开源产品的包装，它为 Spring Boot 应用提供了自配置的 Netflix OSS 整合。通过一些简单的注解，开发者就可以快速的在应用中配置一下常用模块并构建庞大的分布式系统。它主要提供的模块包括：服务发现（Eureka），断路器（Hystrix），智能路由（Zuul），客户端负载均衡（Ribbon）等。

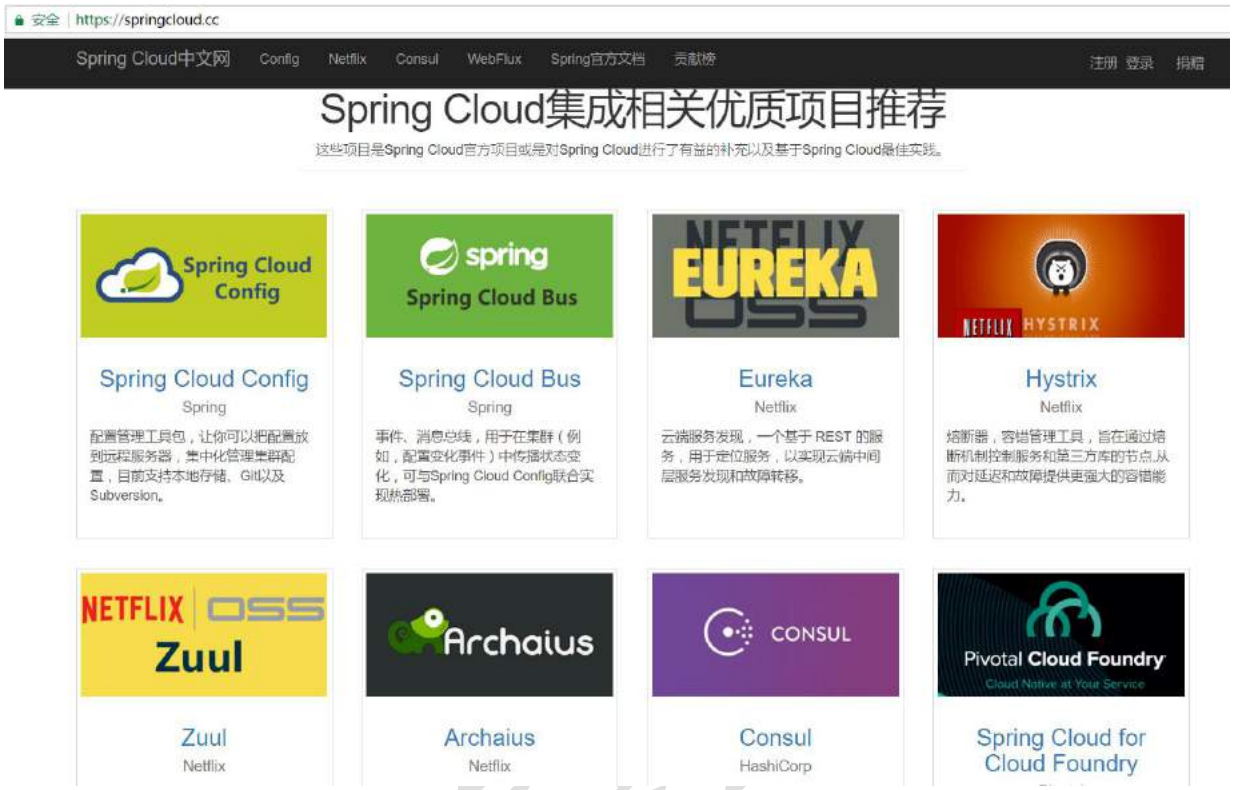
http://spring.io/projects




https://projects.spring.io/spring-cloud/#quick-start

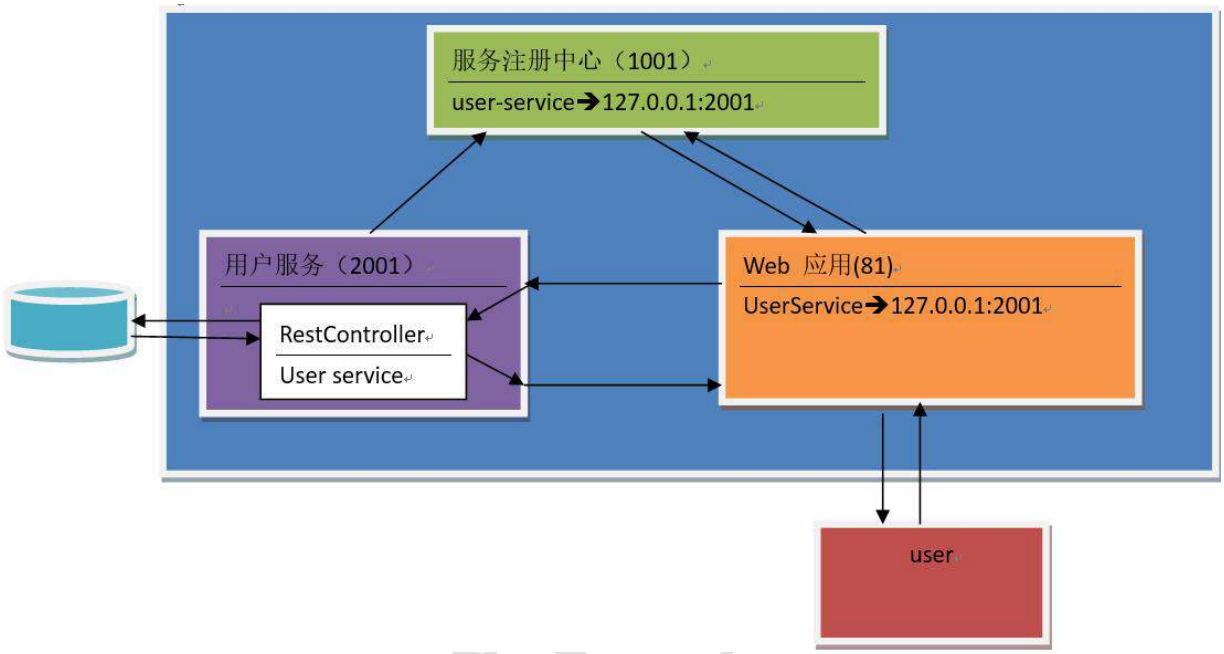



https://springcloud.cc/



# 第 3 章 分布式运行流程

## 3.1 图解




## 3.2 执行过程

- 启动服务注册中心（1001）
- 启动会员服务（2001），向注册中心（1001）中注册当前会员服务。（生产者）
- 用户向 web 服务器（80）发送请求，请求会员服务数据（2001）
- web 服务器（80）接收请求后从注册中心（1001）中获取会员服务（2001）的地址
- 通过这个地址向会员服务器（2001）发送请求，获取数据
- 会员服务器（2001）对请求进行响应，向 web 服务器（80）返回处理数据。
- Web 服务器（80）获取返回的数据后，再向用户返回处理结果

# 第 4 章 Spring Cloud Eureka-案例

Spring Cloud 可以将服务模块从系统中剥离出来，形成独立的业务组件。

通过服务发现注册功能，将业务服务组件发布到服务注册中心，让其他应用进行访问。

Spring Cloud 为服务治理做了一层抽象接口，所以在 Spring Cloud 应用中可以支持多种不同的服务治理框架，

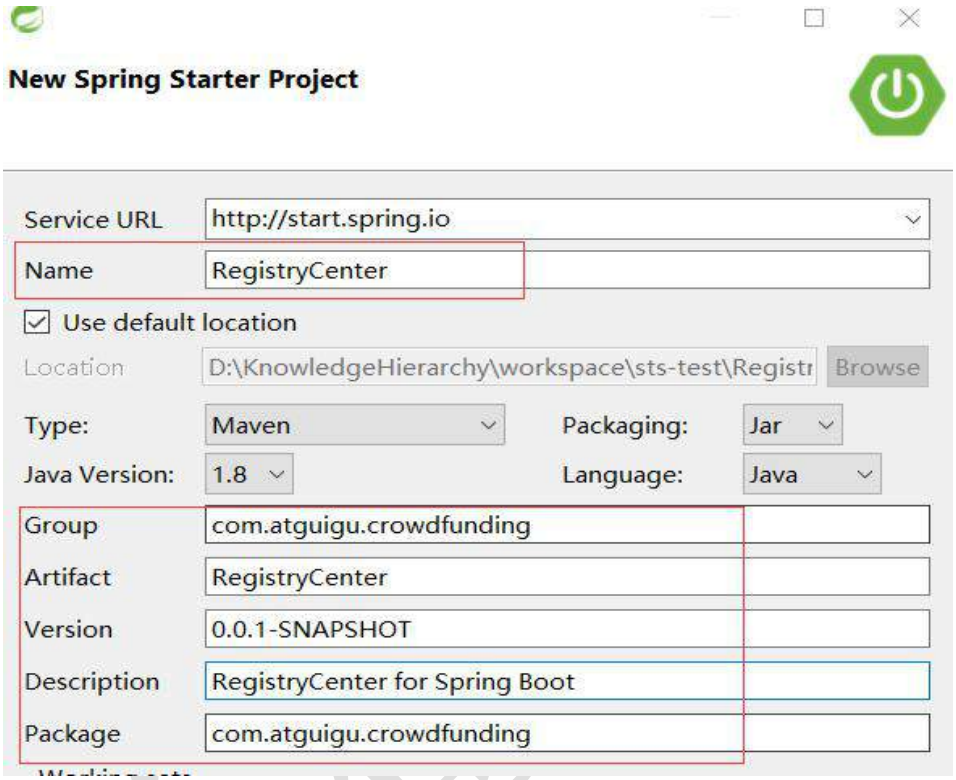
比如：**Netflix Eureka**、Consul、Zookeeper。

在 Spring Cloud 服务治理抽象层的作用下，我们可以无缝地切换服务治理实现，并且不影响任何其他的服务注册、服务发现、服务调用等逻辑。

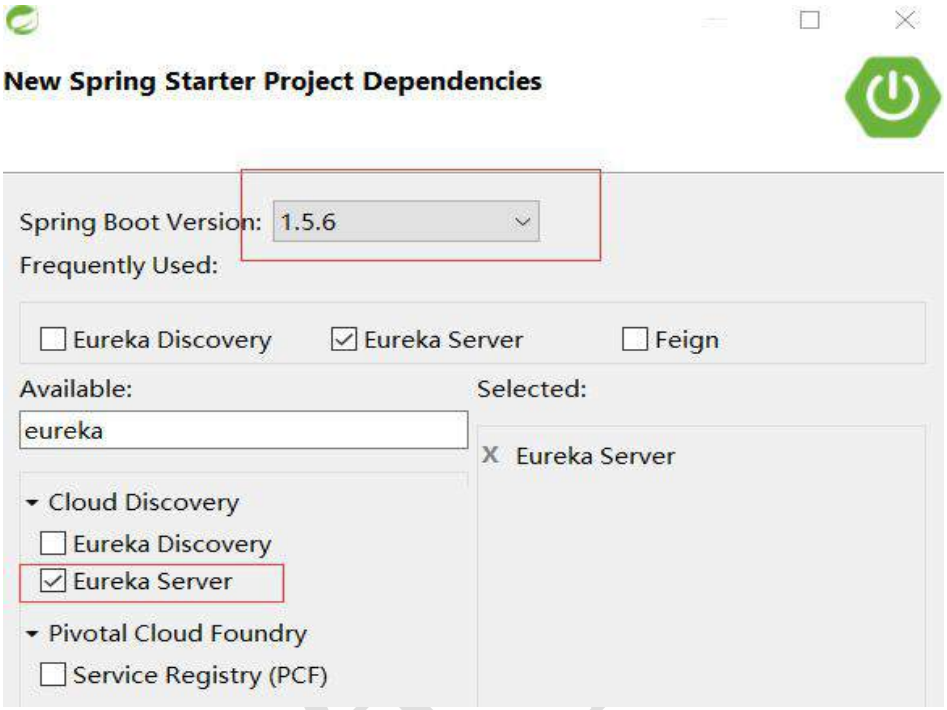
## 4.1 创建服务注册中心

本章学习中，我们使用 Spring Cloud Eureka 来实现服务治理，Spring Cloud Eureka 是 Spring Cloud Netflix 项目下的服务治理模块。

4.1.1 在 **Spring Tool Suite** 工具中，右键点击项目视图空白处，选择菜单 **New Spring Starter Project** 后，填写项目信息。



### 4.1.2 点击 Next 按钮后，选择依赖的类库，Spring Boot Version 请选择最新版




### 4.1.3 点击 Finish 按钮即可

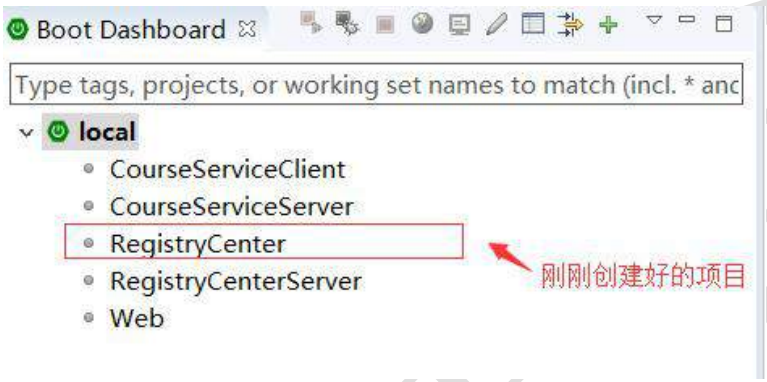
### 4.1.4 项目创建完成后，自动创建 RegistryCenterApplication 类，在类前增加注解@EnableEurekaServer

在 application.properties 文件中，增加配置信息

```
...
spring.application.name=eureka-server
server.port=1001
eureka.instance.hostname=127.0.0.1
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
```

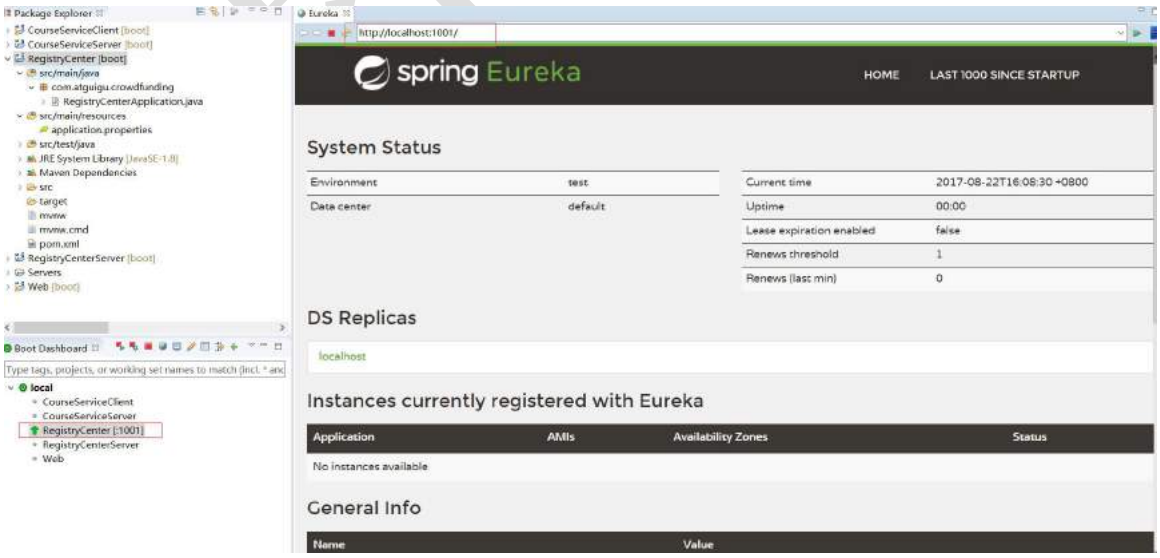
```
eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/
...
```

### 4.1.5 在 Spring Tool Suite 工具的 Boot Dashboard 视图中，可以看见刚刚创建的项目




### 4.1.6 选择项目，点击视图中右上角的启动按钮

后。双击项目查看效果

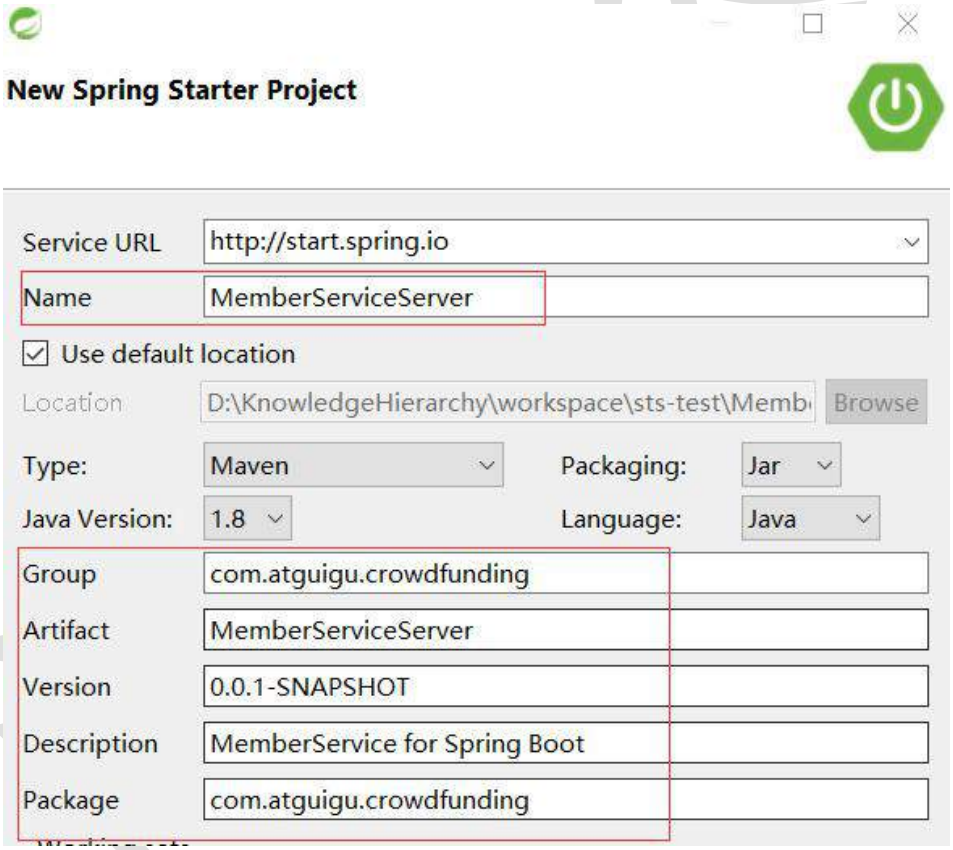


### 4.1.7 如果 eureka 服务页面正常显示，那么注册中心创建成功。

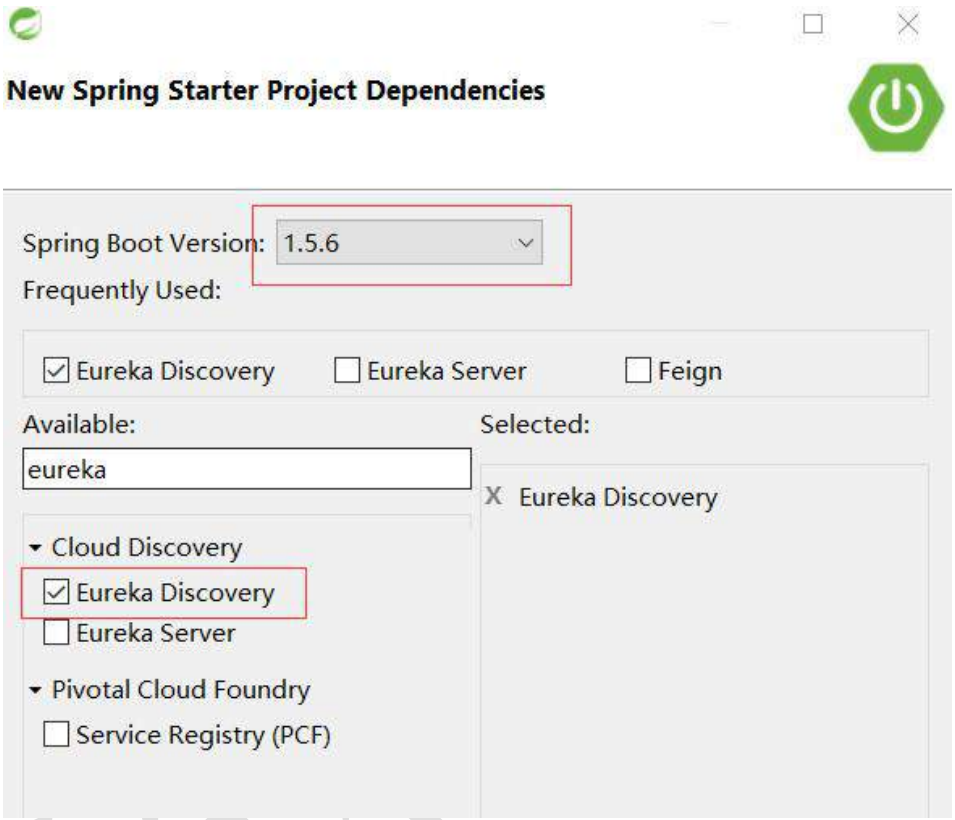
## 4.2 增加会员服务

### 4.2.1 创建会员服务，在注册中心中进行注册。

在 Spring Tool Suite 工具中，右键点击项目视图空白处，选择菜单 New Spring Starter Project 后，填写项目信息。



4.2.2 点击 **Next** 按钮后，选择依赖的类库，**Spring Boot Version** 请选择最新版




4.2.3 点击 **Finish** 按钮即可

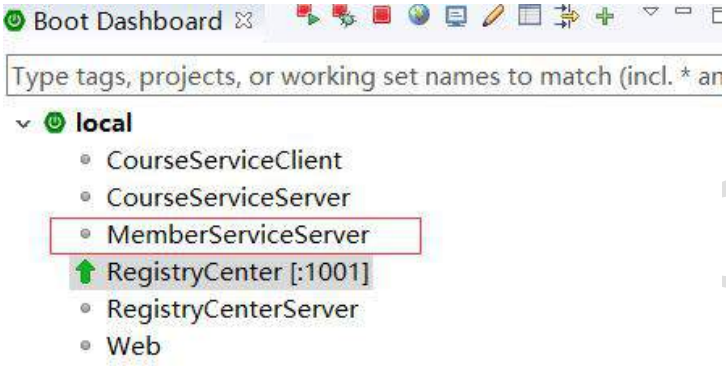
4.2.4 项目创建完成后，自动创建 **MemberServiceServerApplication** 类，在类前增加注解**@EnableDiscoveryClient**

在 application.properties 文件中，增加配置信息

```
...
spring.application.name=eureka-member-service
server.port=2001
eureka.client.serviceUrl.defaultZone=http://127.0.0.1:1001/eureka/
```

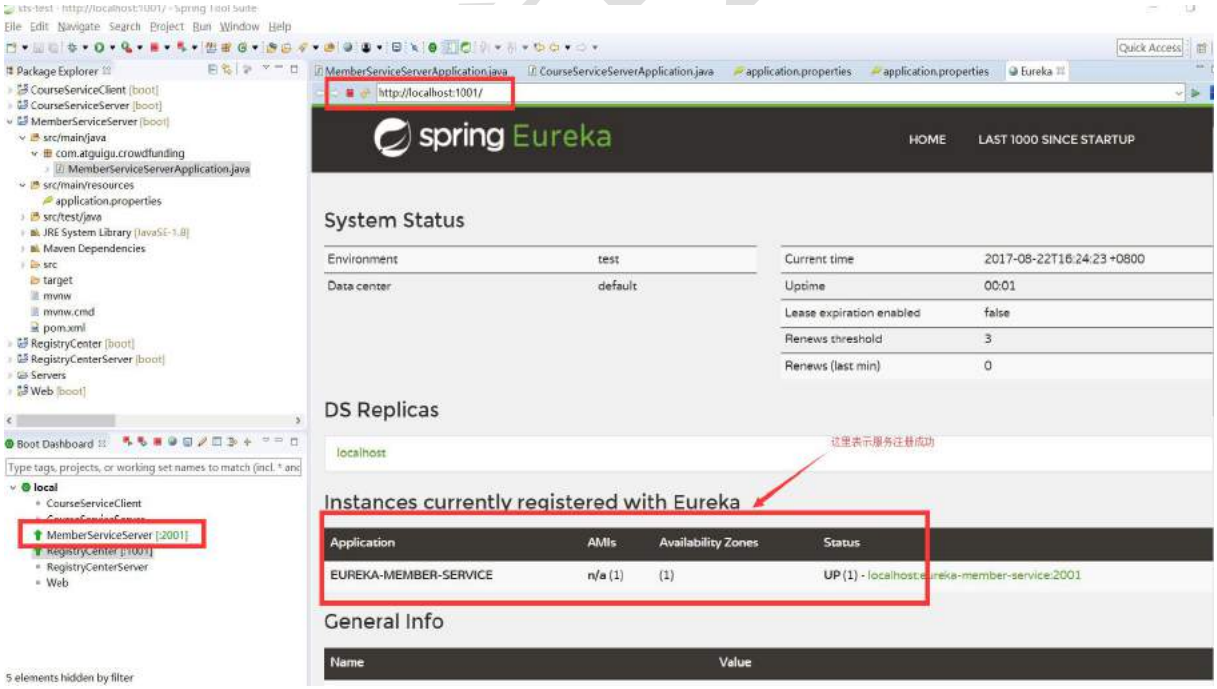
...

在 Spring Tool Suite 工具的 Boot Dashboard 视图中，可以看见刚刚创建的项目




选择项目，点击视图中左上角的启动按钮

后。双击 RegistryCenter 项目查看效果




如果 eureka 服务页面增加了 eureka-member-service 服务，那么会员服务注册成功。

在 MemberServiceServer 项目中，添加业务类 DispatcherController

```
package com.atguigu.crowdfunding.controller;

import org.springframework.web.bind.annotation.RequestMapping;
```

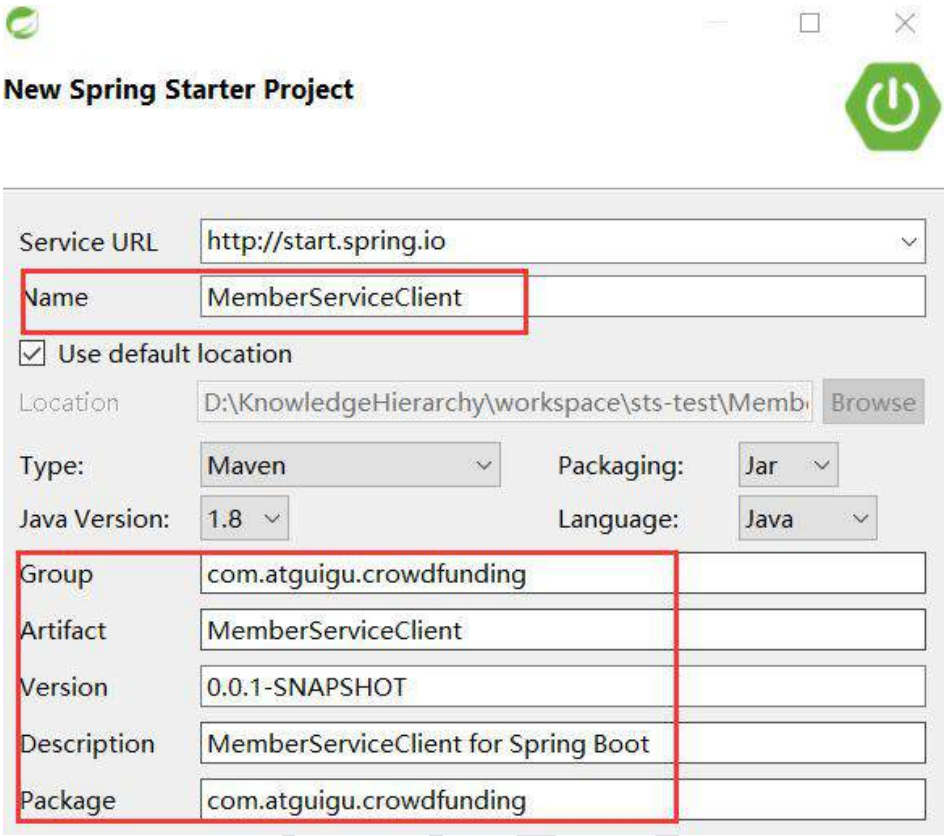
```
import org.springframework.web.bind.annotation.RestController;
@RestController
public class DispatcherController {
    @RequestMapping("/test")
    public String test() {
        return "member service !";
    }
}
```

重启项目后，访问 http://127.0.0.1:2001/test。页面中显示 member service !表示成功。

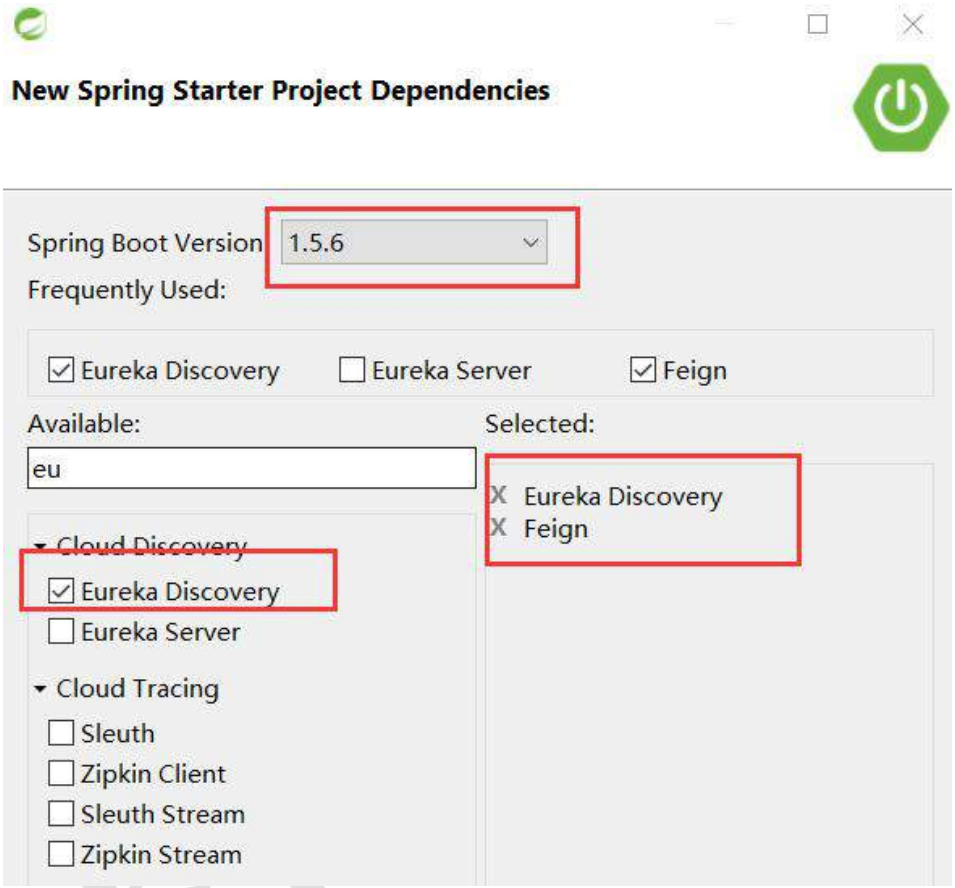
## 4.3 访问会员服务的客户端项目

### 4.3.1 当在注册中心注册了会员服务后，其他的任何应用都可以通过 feign 方式从注册中心中获取会员服务。

在 Spring Tool Suite 工具中，右键点击项目视图空白处，选择菜单 New Spring Starter Project 后，填写项目信息。



4.3.2 点击 **Next** 按钮后，选择依赖的类库，**Spring Boot Version** 请选择最新版




4.3.3 点击 **Finish** 按钮即可

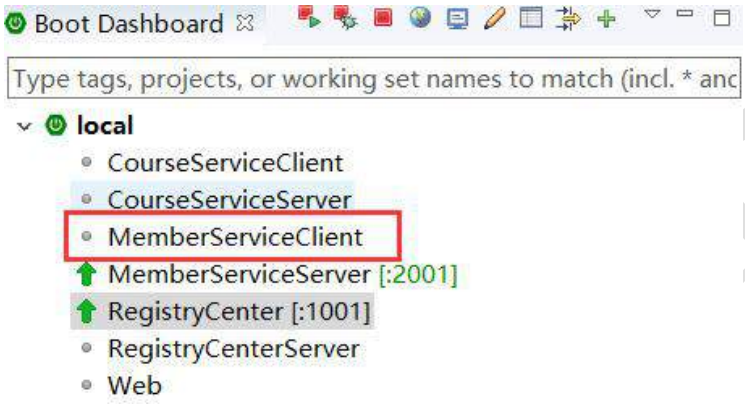
4.3.4 项目创建完成后，自动创建 **MemberServiceClientApplication** 类，在类前增加注解 **@EnableDiscoveryClient, @EnableFeignClients**

在 application.properties 文件中，增加配置信息

```
...
```

```
spring.application.name=eureka-member-client

server.port=80

eureka.client.serviceUrl.defaultZone=http://localhost:1001/eureka/

...
```

在 Spring Tool Suite 工具的 Boot Dashboard 视图中，可以看见刚刚创建的项目




在 MemberServiceClient 项目中，添加服务访问类 MemberClient

```
package com.atguigu.crowdfunding.client;

import org.springframework.cloud.netflix.feign.FeignClient;

import org.springframework.web.bind.annotation.GetMapping;

@FeignClient("eureka-member-service")

public interface MemberClient {

    @GetMapping("/test")

    String test(); //与服务的方法返回类型一致

}
```

@FeignClient("eureka-member-service")中的字符串内容应该和会员服务在注册中心注册的名称相同

eureka-member-service

在 MemberServiceClient 项目中，添加业务类 DispatcherController 调用会员服务 MemberClient。

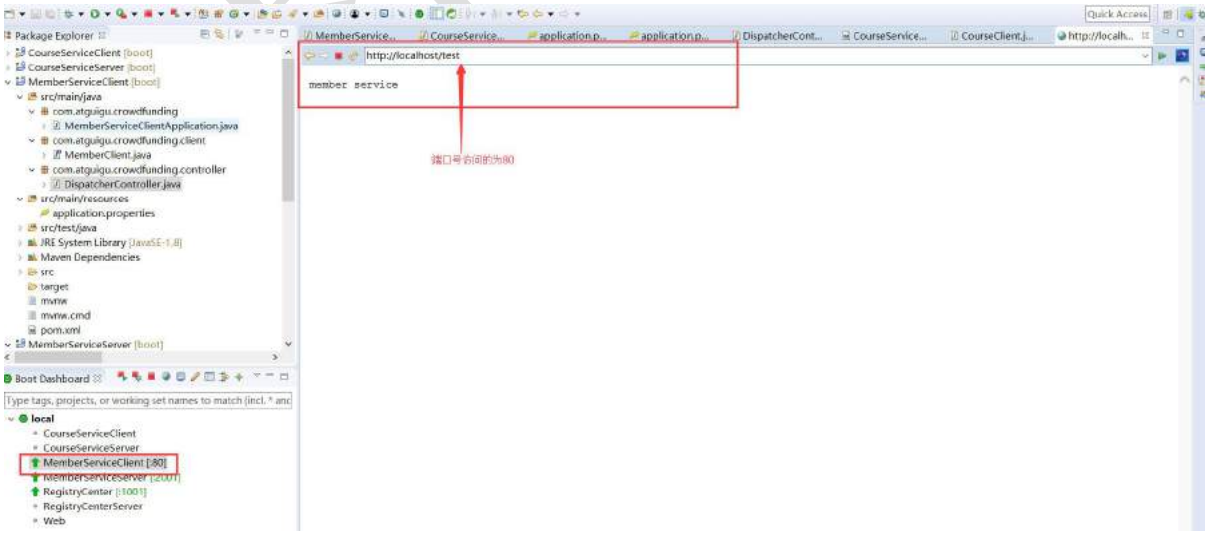
```
package com.atguigu.crowdfunding.controller;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.RequestMapping;
```

```
import org.springframework.web.bind.annotation.RestController;

import com.atguigu.crowdfunding.client.MemberClient;

@RestController

public class DispatcherController {

    @Autowired

    private MemberClient memberClient;

    @RequestMapping("/test")

    public String test() {

        return memberClient.test();

    }

}
```

选择项目，点击视图中左上角的启动按钮

后。

访问 http://127.0.0.1/test。页面中显示 member service !表示客户端访问会员服务成功。



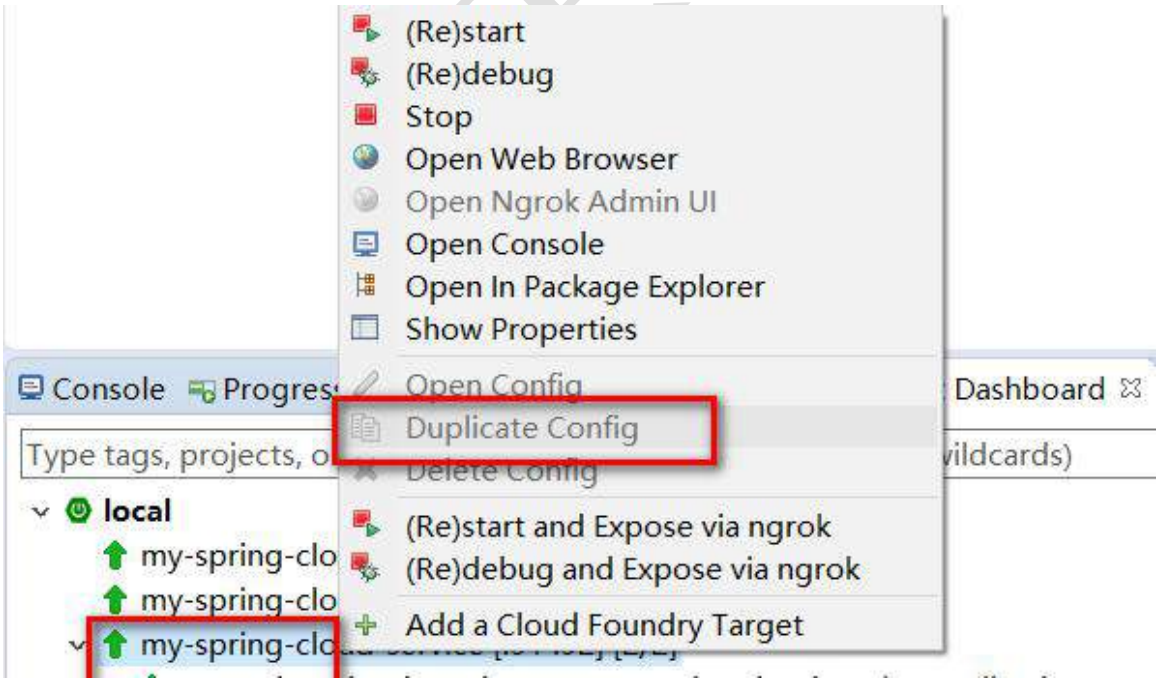
# 第 5 章 Spring Cloud Ribbon

在服务发现 Eureka 中。我们使用了 Feign 方式读取 Service 应用中的数据,那如果多个 Service 应用提供相同的服务，这时候 feign 服务客户端访问哪一个 Service 应用就不确定了，所以此时需要一个类似于 Nginx 负载均衡器一样的软件或服务来进行调度。

Ribbon 就是这样的一种技术，不过和 Nginx 区别在于，Ribbon 是在 feign 客户端获取 Eureka 注册服务列表后进行的负载均衡（软负载均衡）。

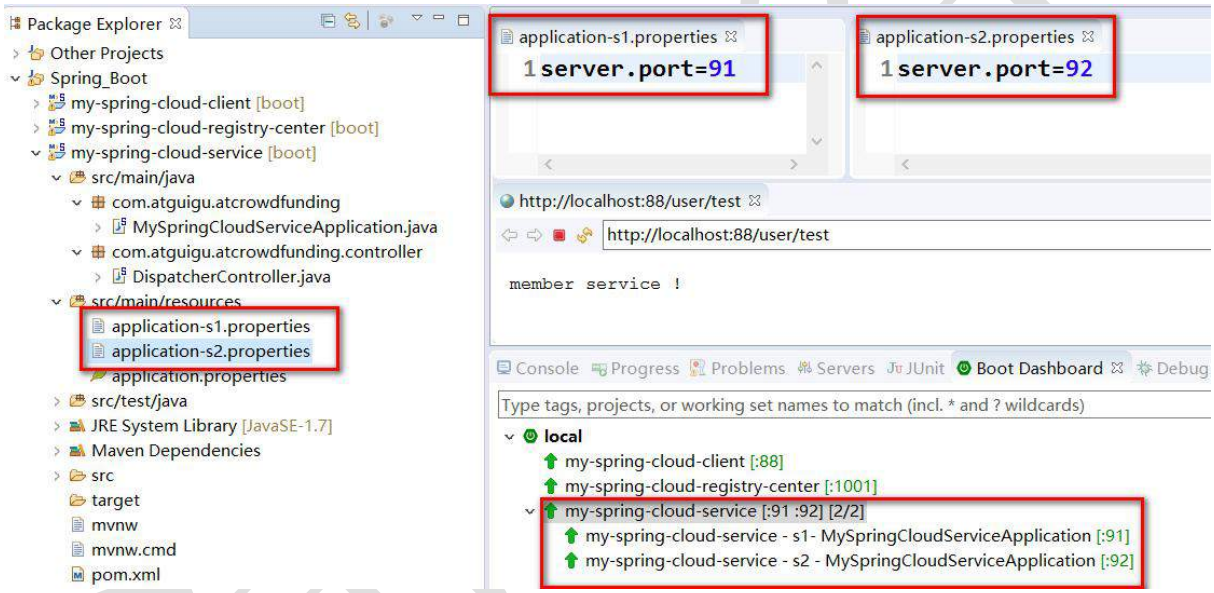
feign 服务中已经内置了 Ribbon, 所以不需要额外的操作。

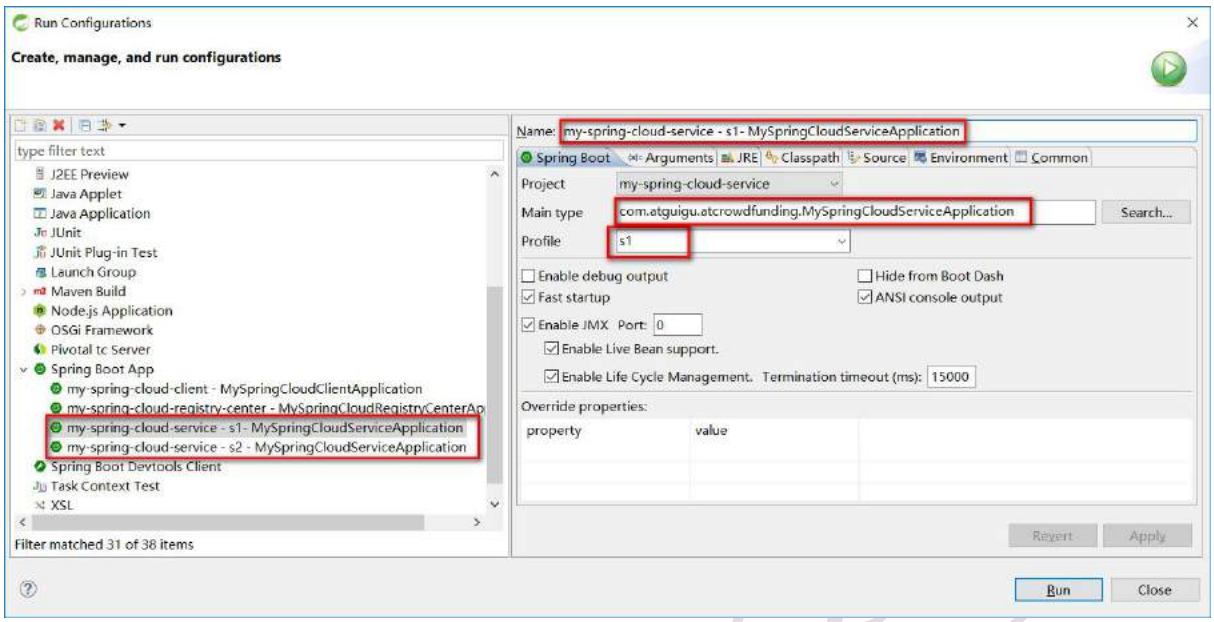
## 5.1 打开 Boot Dashboard 视图,点击服务项目,右键[Duplicate Config],复制出一个配置.



## 5.2 分别配置多个服务


- 只需要修改端口号,服务名称不需要改,否则,客户端调用时有两个名称就不方便调用了.






## 5.3 启动多个服务,通过客户端访问,观察服务控制台变化.

## 5.4 根据集群部署,@FeignClient 会从注册中心获取多个同名的服务,这样 Ribbon 可以根据一些算法进行负载均衡.

```
@FeignClient("eureka-user-service")
public interface UserClient {
    @GetMapping("/test")
    String test();
}
```

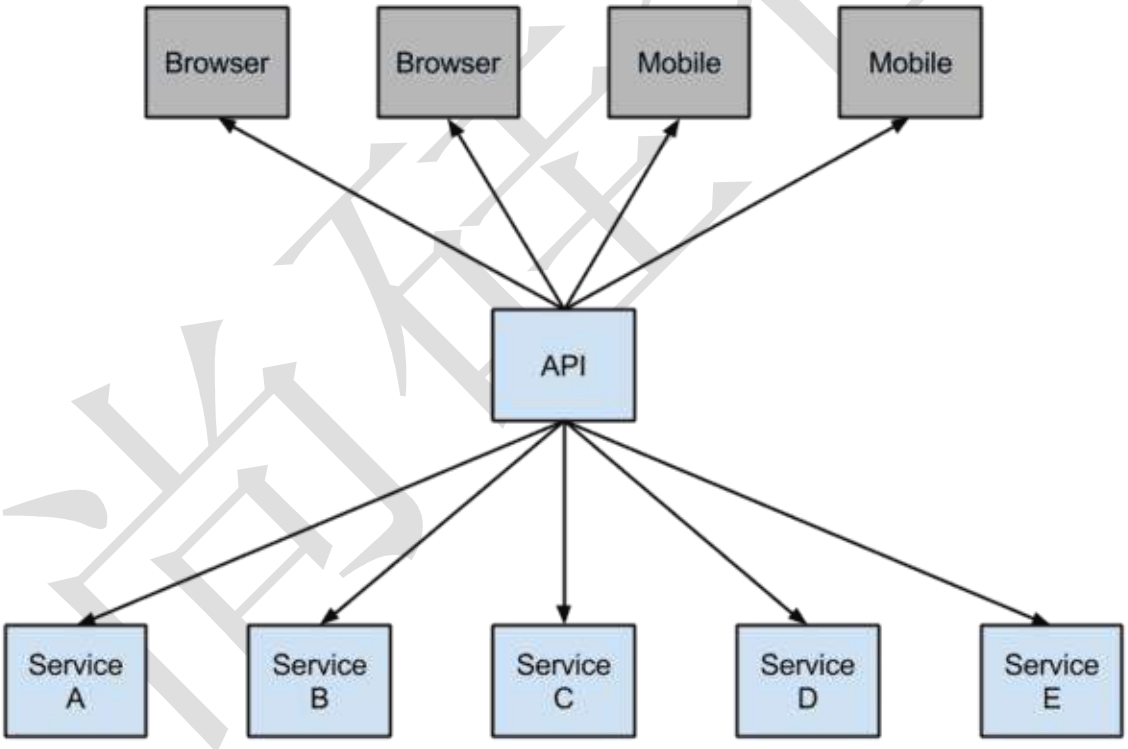
# 第 6 章 Spring Cloud Hystrix

在微服务架构中，根据业务来拆分成一个个的服务，服务与服务之间可以相互调用（RPC），在 Spring Cloud 可以用 Feign 来调用。为了保证其高可用，单个服务通常会集群部署。由于网络原因或者自身

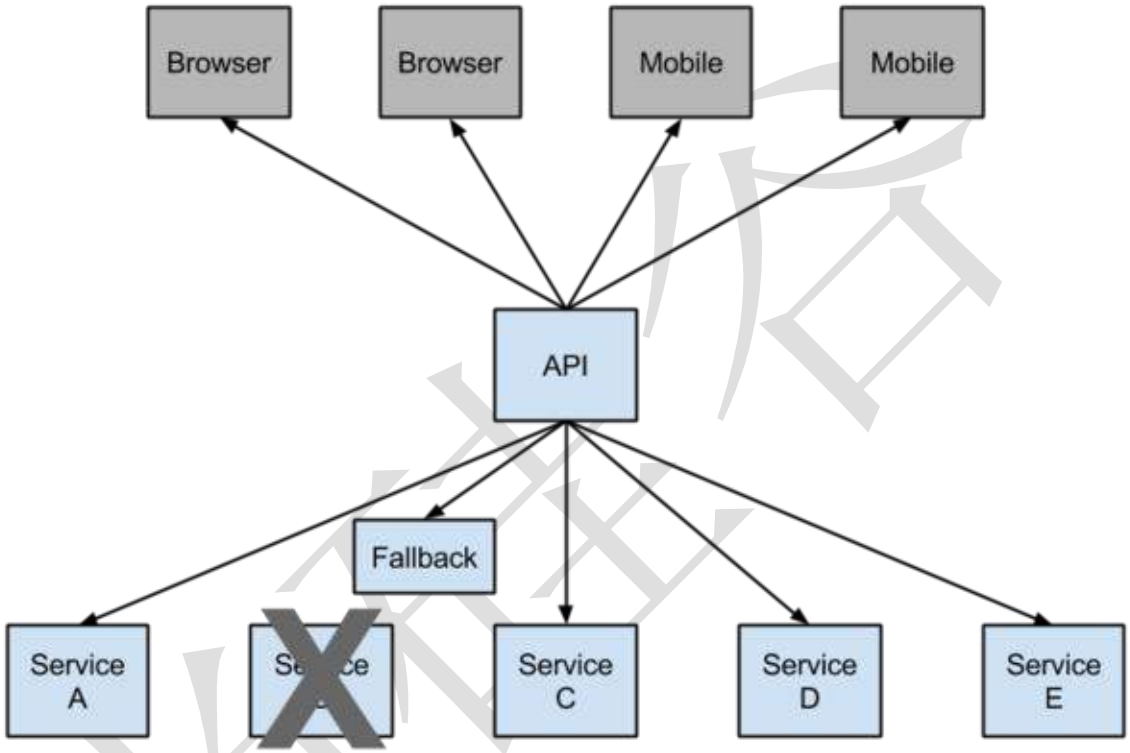
的原因，服务并不能保证 100%可用，如果单个服务出现问题，调用这个服务就会出现线程阻塞，此时若有大量的请求涌入，Servlet 容器的线程资源会被消耗完毕，导致服务瘫痪。服务与服务之间的依赖性，故障会传播，会对整个微服务系统造成灾难性的严重后果，这就是服务故障的“雪崩”效应。

Netflix has created a library called Hystrix that implements the circuit breaker pattern. In a micros\ervice architecture it is common to have multiple layers of service calls.

## 6.1 在微服务架构中，一个请求需要调用多个服务是非常常见的



6.2 较底层的服务如果出现故障，会导致连锁故障。当对特定的服务的调用的不可用达到一个阀值（Hystrix 是 5 秒 20 次） 断路器将会被打开




6.3 断路打开后，可避免连锁故障。

6.4 修改客户端项目代码,演示熔断器:

6.4.1 代码实现时，首先需要在 pom.xml 文件中增加依赖关系

```
...
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-hystrix</artifactId>
```

```
</dependency>
...
```

### 6.4.2 在程序的启动类中增加@EnableHystrix 注解开启 Hystrix。

### 6.4.3 在调用 feign 服务的 Controller 方法前增加注解 @HystrixCommand(fallbackMethod="服务失败时调用的方法名")

```
@HystrixCommand(fallbackMethod="testError")
@RequestMapping("/test")
public String test() {
    return memberClient.test();
}

public String testError() {
    return "Server Error ... ";
}
```

6.4.4 服务调用失败时回调的方法声明应该和 feign 服务调用的方法声明保持一致(返回类型和参数必须一致,作用域无所谓)

6.4.5 正确启动应用后，再关闭服务，测试断路器是否工作正常。

# 第 7 章 完善案例-增加业务层和持久化层

## 7.1 spring-cloud-service

### 7.1.1 定义实体 bean

实现 Serializable 接口


### 7.1.2 增加相关配置

| | |
|---|---|
| <dependency><br><groupId>mysql</groupId> | 创建 application.yml 文件,并增加配置:<br>spring: |

```
<artifactId>mysql-connector-java</artifactId>
</dependency>
<!-- 数据库连接池 -->
<dependency>
<groupId>com.alibaba</groupId>
<artifactId>druid</artifactId>
<version>1.0.5</version>
</dependency>
<dependency>
<groupId>org.mybatis.spring.boot</groupId>
<artifactId>mybatis-spring-boot-starter</artifactId>
<version>1.1.1</version>
</dependency>
```

```
  datasource:
    name: mydb
    type: com.alibaba.druid.pool.DruidDataSource
    url: jdbc:mysql://127.0.0.1:3306/test
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
mybatis:
  mapper-locations:
classpath*:/mybatis/mapper-*.xml
  type-aliases-package: com.atguigu.**.bean
```

### 7.1.3 增加业务层

```
package com.atguigu.atcrowdfunding.service;

import com.atguigu.atcrowdfunding.bean.User;

public interface UserService {
    User queryUserById(Integer id);
}
```

```
package com.atguigu.atcrowdfunding.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;
import com.atguigu.atcrowdfunding.bean.User;
import com.atguigu.atcrowdfunding.dao.UserDao;
import com.atguigu.atcrowdfunding.service.UserService;

@Service
@Transactional(readOnly=true)
public class UserServiceImpl implements UserService {
```

```
    @Autowired
    private UserDao userDao ;

    @Transactional
    public User queryUserById(Integer id) {
        return userDao.queryUserById(id);
    }
}
```

### 7.1.4 增加持久化层

```
package com.atguigu.atcrowdfunding.dao;

import org.apache.ibatis.annotations.Select;
import com.atguigu.atcrowdfunding.bean.User;

public interface UserDao {
    @Select(value="select * from tt_user where id=#{id}")
    User queryUserById(Integer id);
}
```

### 7.1.5 修改启动类

```
package com.atguigu.atcrowdfunding;

import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
```

```
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.transaction.annotation.EnableTransactionManagement;


@MapperScan("com.atguigu.**.dao")
@EnableTransactionManagement
@EnableDiscoveryClient
@SpringBootApplication
public class MySpringCloudServiceApplication {


    public static void main(String[] args) {
        SpringApplication.run(MySpringCloudServiceApplication.class, args);
    }
}
```

### 7.1.6 修改控制器类

```
package com.atguigu.atcrowdfunding.controller;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;


import com.atguigu.atcrowdfunding.bean.User;
import com.atguigu.atcrowdfunding.service.UserService;


@RestController
public class DispatcherController {
```

```
    @Autowired
    private UserService userService ;
    @RequestMapping("/test/{id}")
    public User queryUserById(@PathVariable("id") Integer id) {
        return userService.queryUserById(id);
    }
}
```

### 7.1.7 测试:服务层是否可以单独被访问

## 7.2 spring-cloud-client

### 7.2.1 定义实体 bean

实现 Serializable 接口


### 7.2.2 增加相关配置

| | |
|---|---|
| <dependency><br><groupId>org.springframework.boot</groupI | spring.freemarker.allow-request-override=false<br>spring.freemarker.allow-session-override=false |

| | |
|---|---|
| d><br><artifactId>spring-boot-starter-freemarker</artifactId><br></dependency> | spring.freemarker.cache=true<br>spring.freemarker.charset=UTF-8<br>spring.freemarker.check-template-location=true<br>spring.freemarker.content-type=text/html<br>spring.freemarker.enabled=true<br>spring.freemarker.expose-request-attributes=false<br>spring.freemarker.expose-session-attributes=false<br>spring.freemarker.expose-spring-macro-helpers=true<br>spring.freemarker.prefer-file-system-access=false<br>spring.freemarker.suffix=.ftl<br>spring.freemarker.template-loader-path=classpath:/templates/<br>spring.freemarker.settings.template_update_delay=0<br>spring.freemarker.settings.default_encoding=UTF-8<br>spring.freemarker.settings.classic_compatible=true<br>spring.freemarker.order=1 |

### 7.2.3 增加客户端方法

```
package com.atguigu.atcrowdfunding.client;


import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;


import com.atguigu.atcrowdfunding.bean.User;


@FeignClient("eureka-user-service")
```

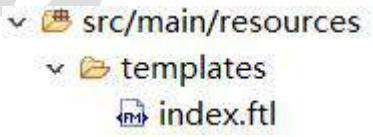
```
public interface UserClient {
    @RequestMapping("/test/{id}")
    public User queryUserById(@PathVariable("id") Integer id);
}
```

### 7.2.4 增加控制器

```
package com.atguigu.atcrowdfunding.controller;

import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;

import com.atguigu.atcrowdfunding.bean.User;
import com.atguigu.atcrowdfunding.client.UserClient;
import com.netflix.hystrix.contrib.javanica.annotation.HystrixCommand;
 @Controller
public class DispatcherController {

    @Autowired
    private UserClient userClient ;

    @RequestMapping("/user/query/{id}")
    public String query(@PathVariable("id") Integer id,Map<String, Object> map) {
        User user = userClient.queryUserById(id);
```

```
        map.put("user", user);
        return "index";
    }

    @RequestMapping("/user/test")
    @HystrixCommand(fallbackMethod="error")
    public String test() {
        System.out.println("1111111111111111");
        return userClient.test();
    }

    public String error() {
        System.out.println("~~~~~~~~~~~~~~error~~~~~~~~~~~~~~");
        return "Error...";
    }
}
```

### 7.2.5 增加 Freemarker 视图文件



# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

1. 项目划分重构
2. 搭建分布式项目环境
3. 实名认证审核流程
4. Activiti 流程框架
5. Activiti5 流程框架介绍
6. JBPM4 ==》 Activiti5
7. 框架概念
    1) 流程定义
    2) 流程实例
    3) 流程变量
    4) 任务
    5) 开始、结束
    6) 网关
8. 框架核心对象 ProcessEngine 和 7 个核心服务对象
    1) 持久化服务
    2) 运行时服务
    3) 任务服务
    4) 历史服务
    5) …

9. 流程框架的使用

   1) 引入类库

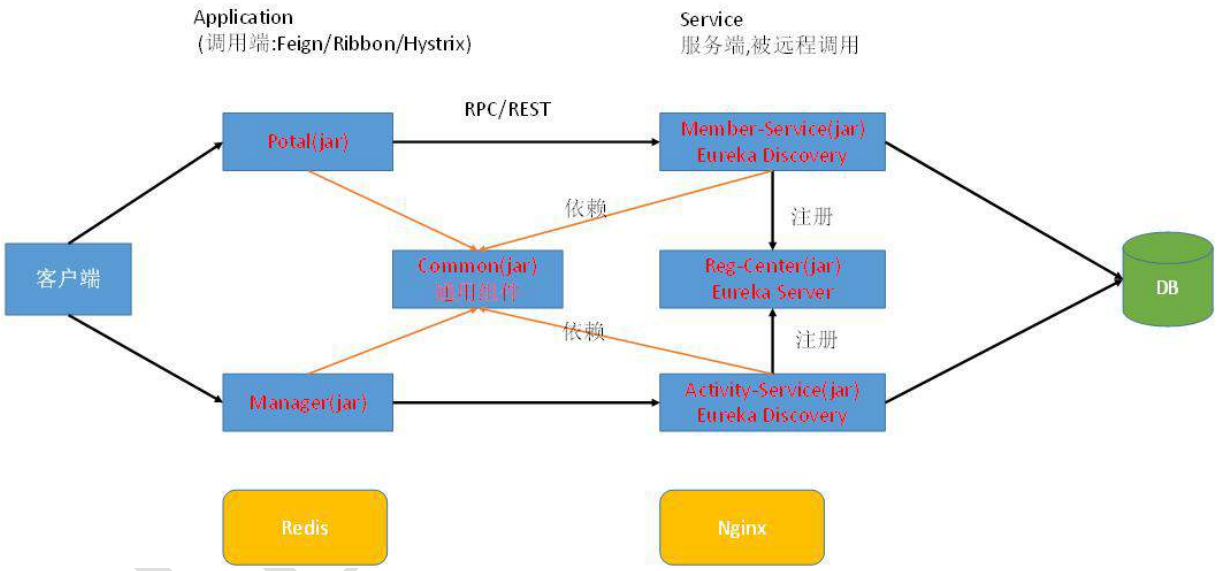
   2) 数据库创建表

   3) 获取核心对象

   4) 获取相关服务，操作流程业务

# 第一章 重构尚筹网项目

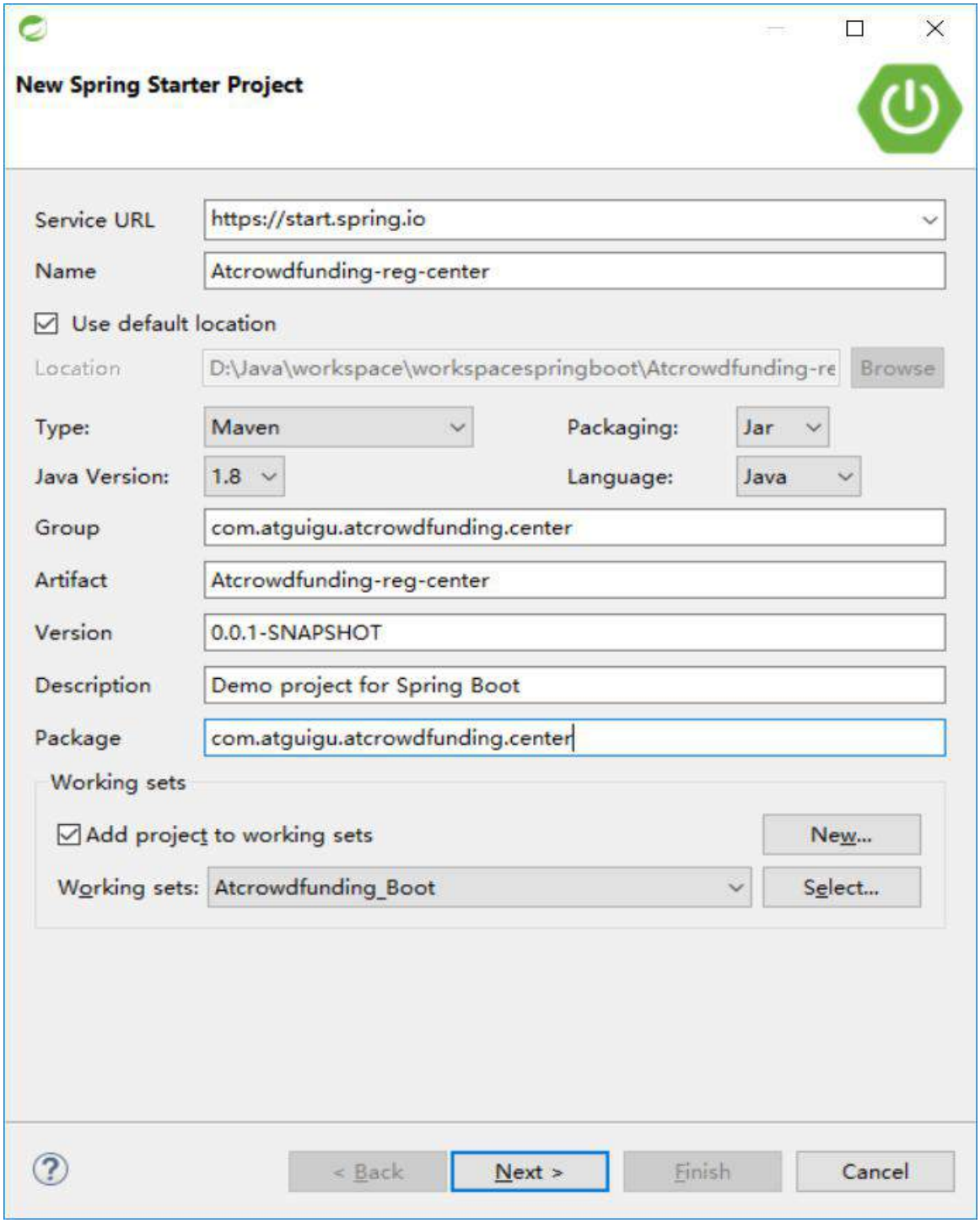
## 1.1 分布式架构

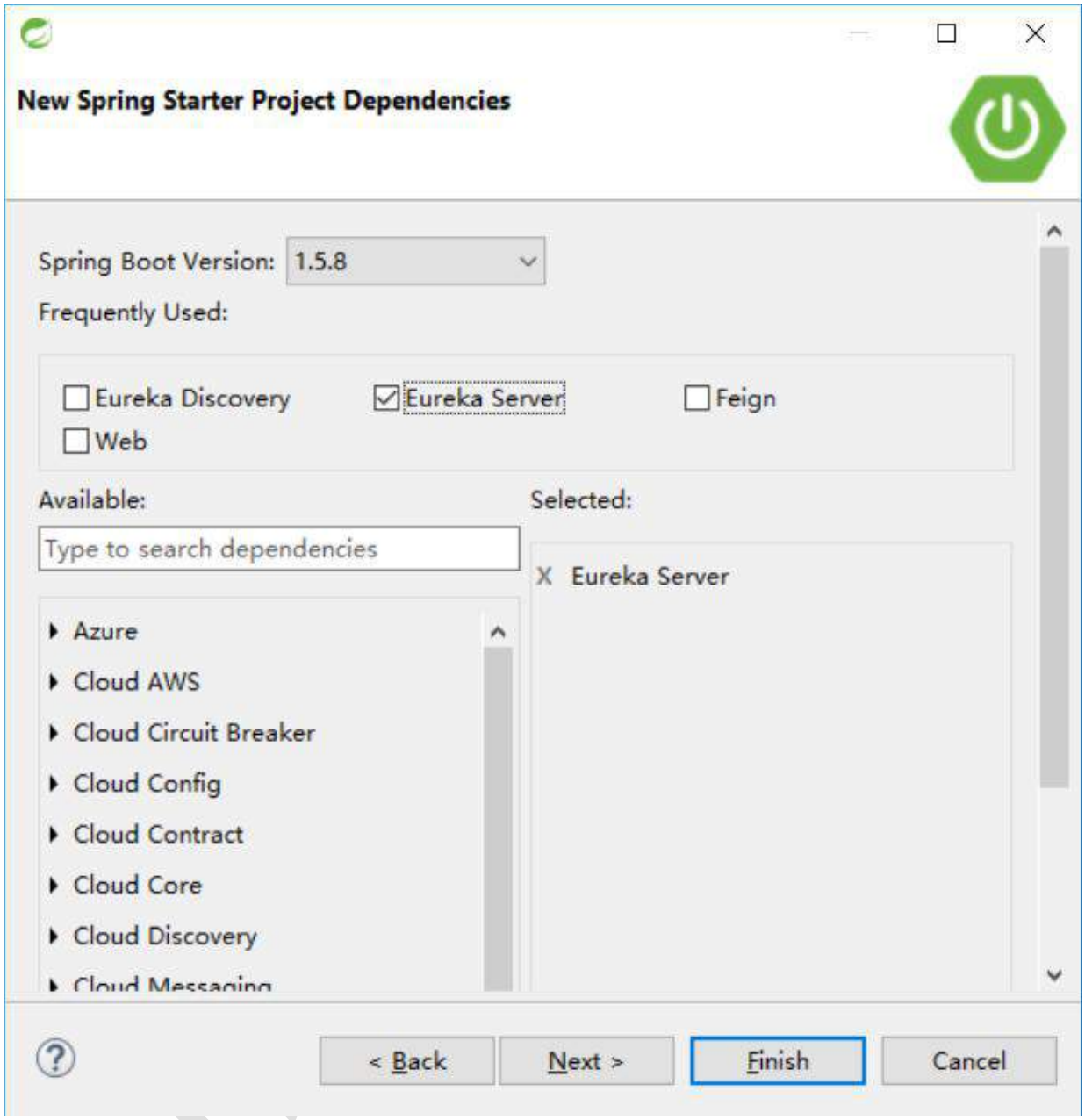



- atcrowdfunding-reg-center 1001
- atcrowdfunding-boot-common
- atcrowdfunding-boot-manager 9090
- atcrowdfunding-boot-portal 8080
- atcrowdfunding-boot-member-service 2001
- atcrowdfunding-boot-activiti-service 3001

## 1.2 搭建分布式环境

### 2.1 创建注册中心:atcrowdfunding-reg-center





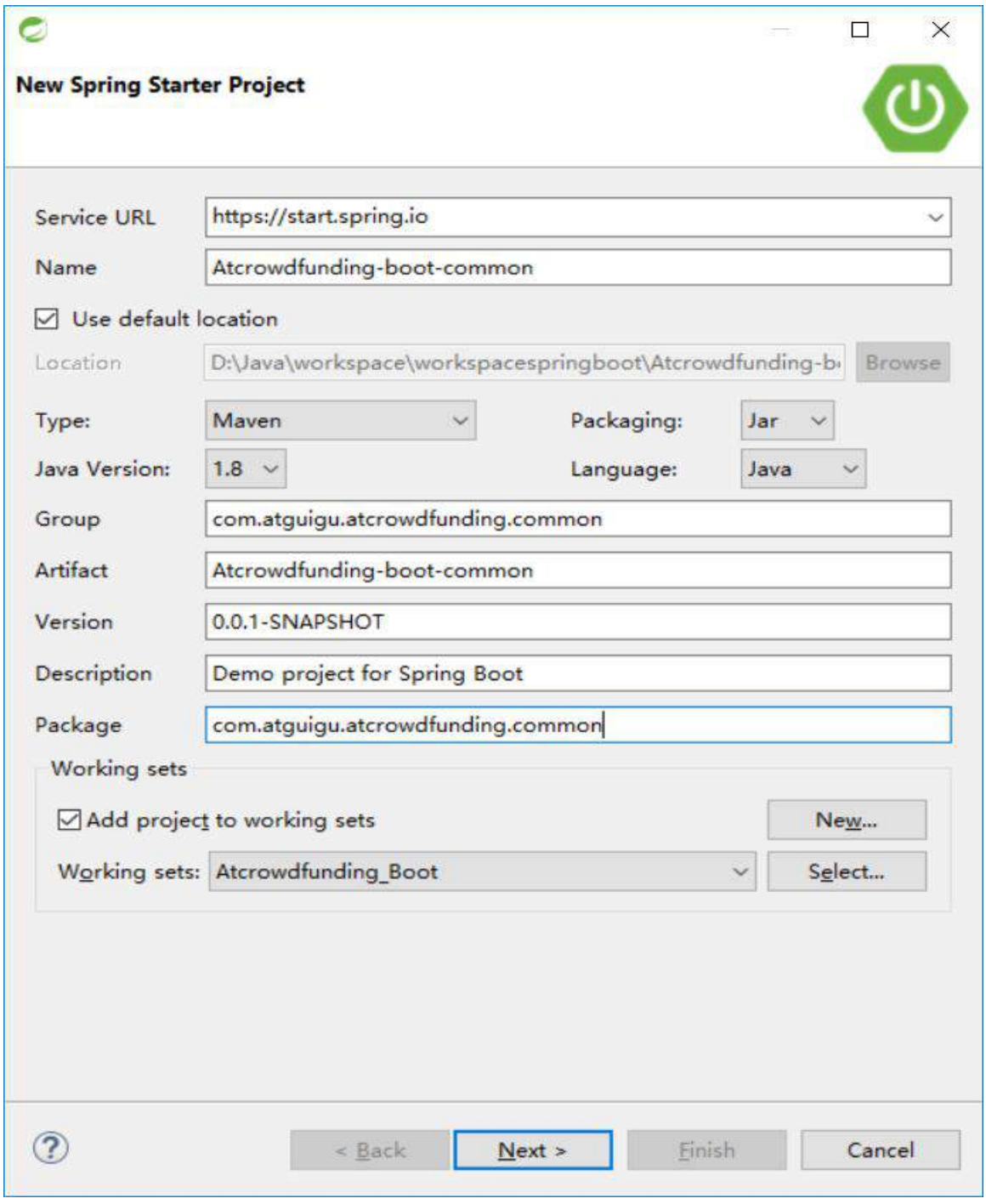

```
package com.atguigu.atcrowdfunding.center;


import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;


@EnableEurekaServer
```

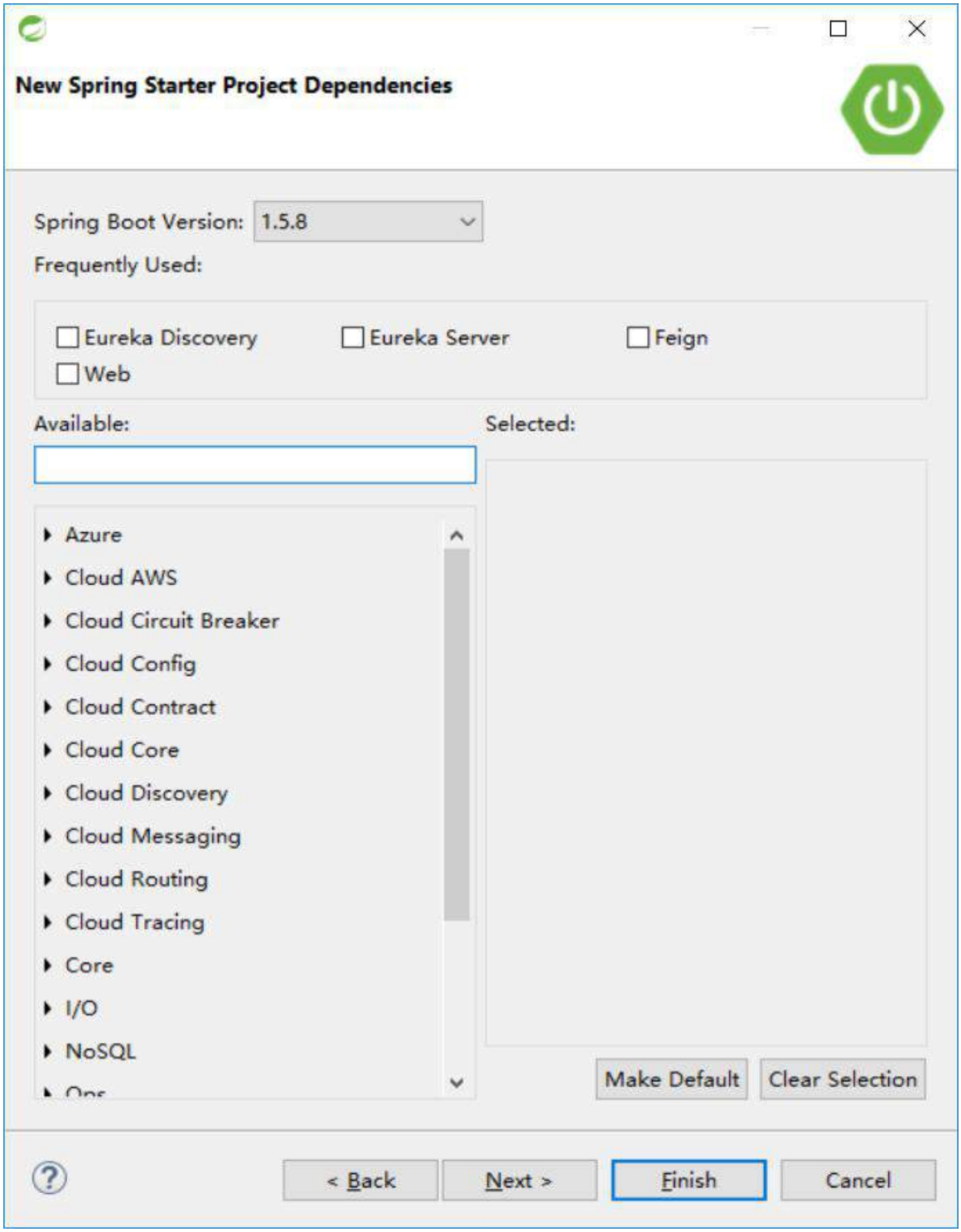
```
@SpringBootApplication
public class AtcrowdfundingRegCenterApplication {
    public static void main(String[] args) {
        SpringApplication.run(AtcrowdfundingRegCenterApplication.class, args);
    }
}
```

```
spring.application.name=atcrowdfunding-reg-center
server.port=1001
eureka.instance.hostname=127.0.0.1
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false
eureka.client.serviceUrl.defaultZone=http://localhost:${server.port}/eureka/
```

## 2.2 创建 Common 项目:atcrowdfunding-boot-common

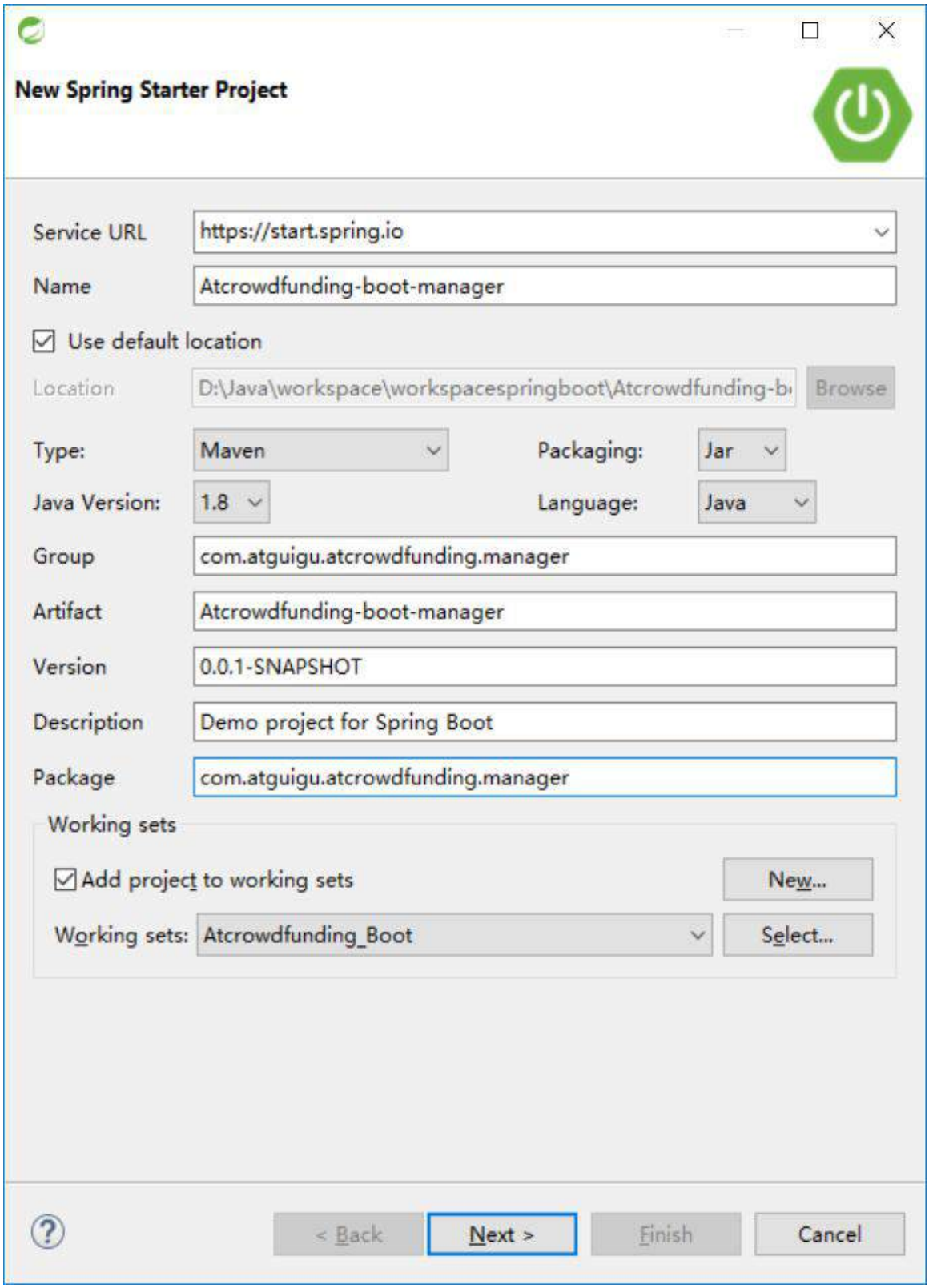
- 工具类,实体 bean

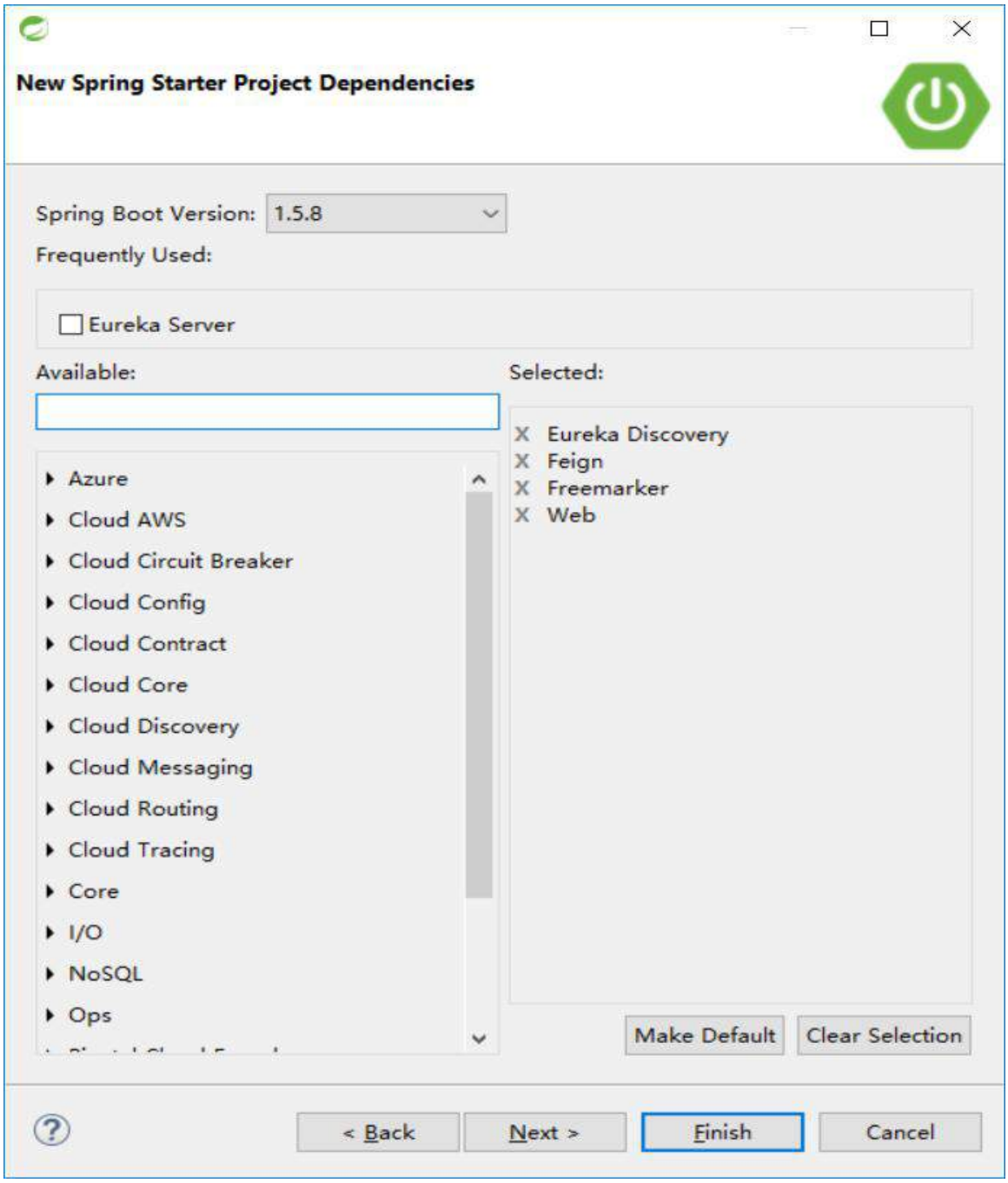

New Spring Starter Project
Service URL
https://start.spring.io
Name
Atcrowdfunding-boot-common
Use default location
Location
D:\Java\workspace\workspacespringboot\Atcrowdfunding-b
Browse
Type:
Maven
Packaging:
Jar
Java Version:
1.8
Language:
Java
Group
com.atguigu.atcrowdfunding.common
Artifact
Atcrowdfunding-boot-common
Version
0.0.1-SNAPSHOT
Description
Demo project for Spring Boot
Package
com.atguigu.atcrowdfunding.common
Working sets
Add project to working sets
New...
Working sets:
Atcrowdfunding_Boot
Select...
< Back
Next >
Finish
Cancel




- com.atguigu.atcrowdfunding.common.bean
- com.atguigu.atcrowdfunding.common.util
- 删除 AtcrowdfundingBootCommonApplication 类,用不上.

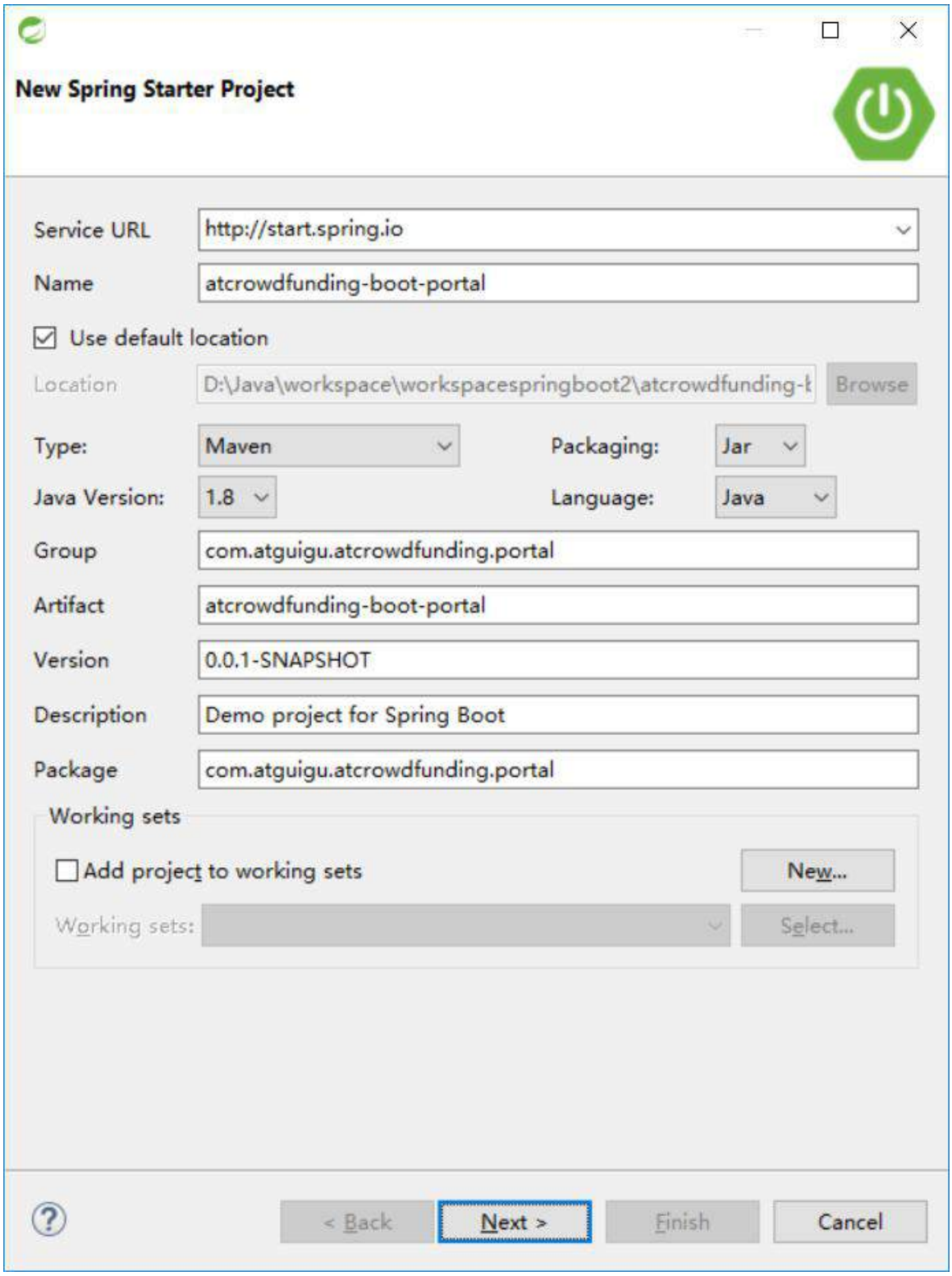
## 2.3 创建 manager 客户端项目:atcrowdfunding-boot-manager

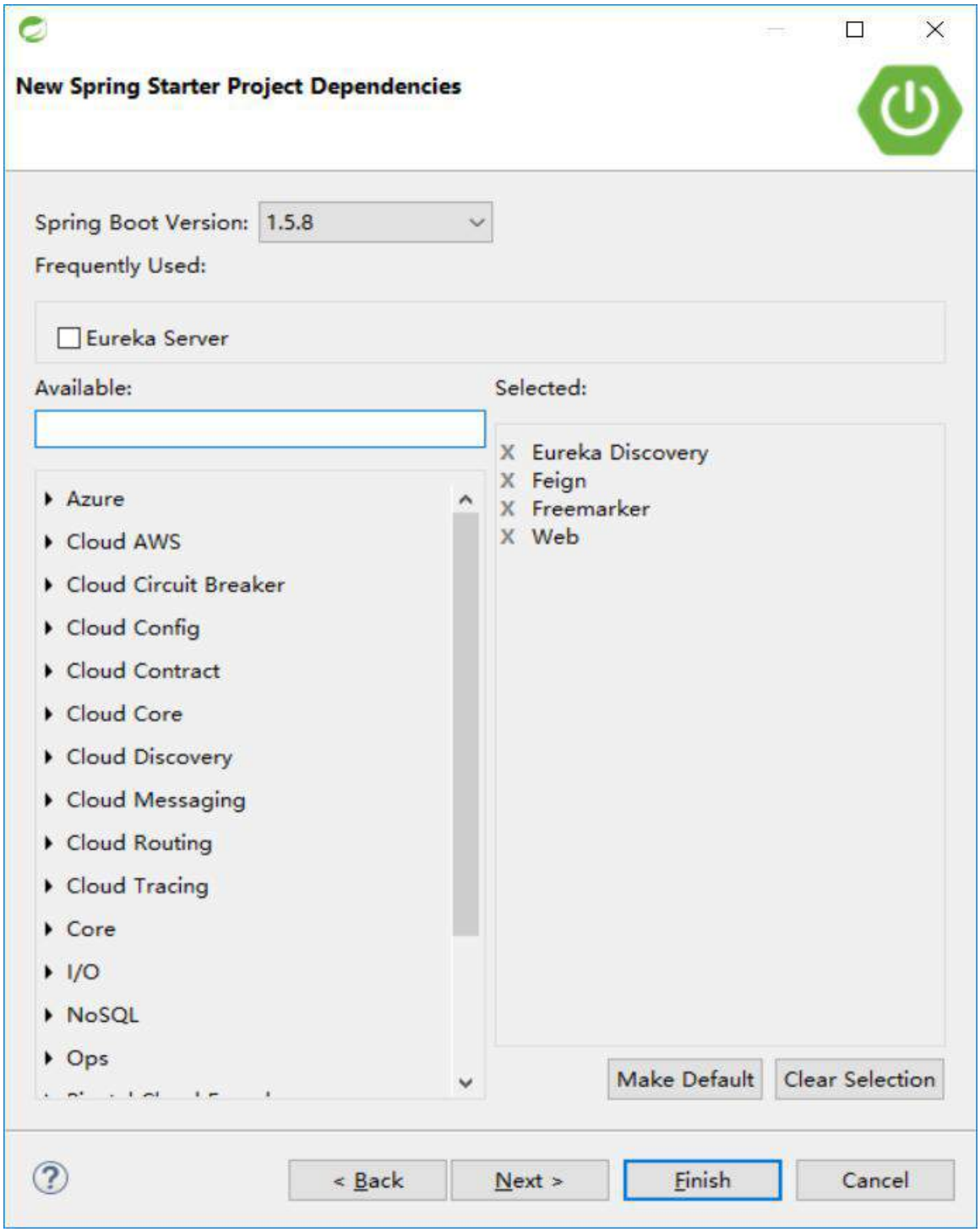





- 依赖于 common
- 主类上增加相应注解:
  - @EnableFeignClients
  - @EnableDiscoveryClient

## 2.4 创建 Portal 客户端项目:atcrowdfunding-boot-portal

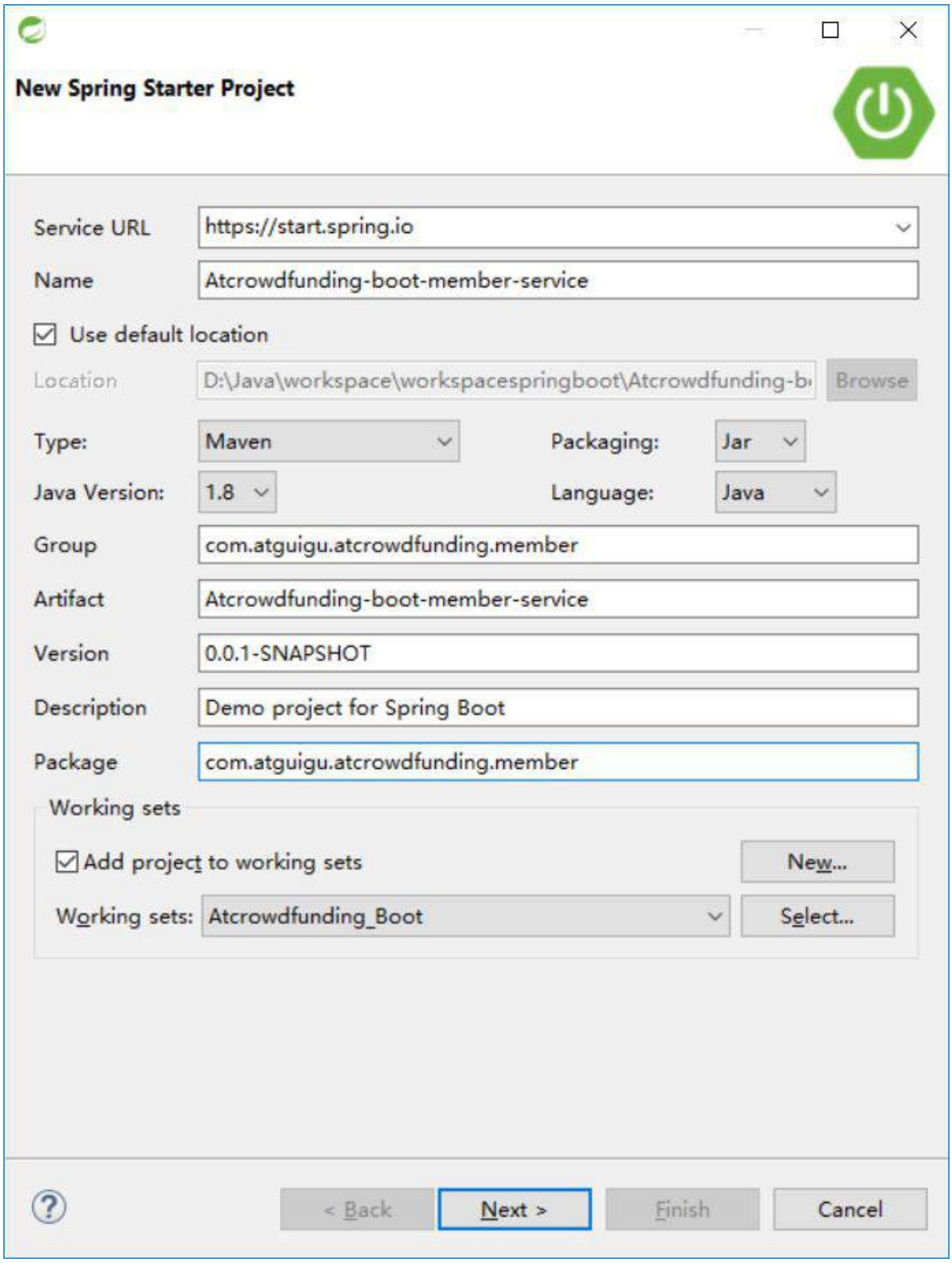




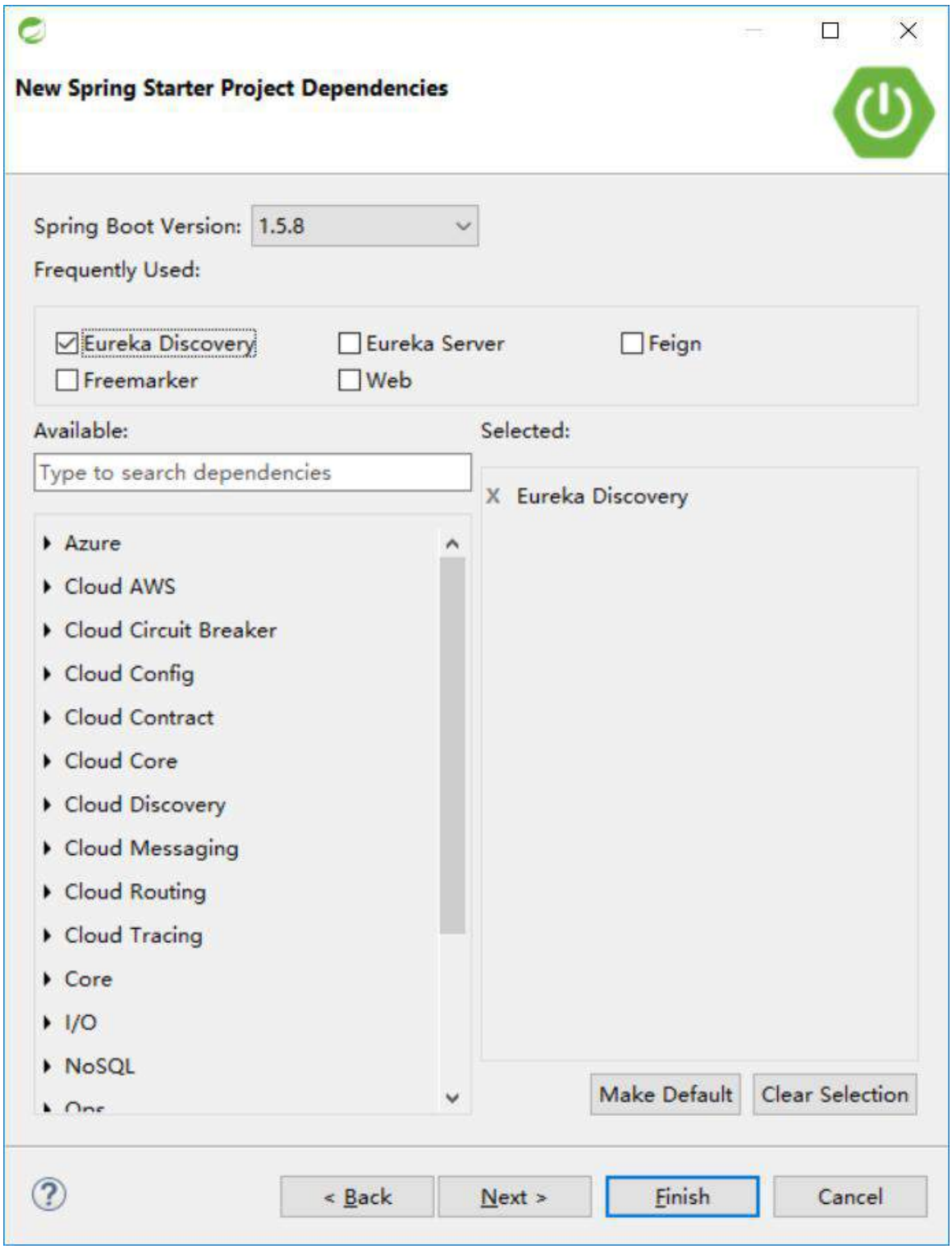

- 依赖于 common
- 主类上增加相应注解:
  - @EnableFeignClients
  - @EnableDiscoveryClient

## 2.5 创建 member 会员服务项目

atcrowdfunding-boot-member-service



- 增加 Eureka Discovery，也可以将 feign 增加进来,因为有 可能进行服务端项目间调用.




- 依赖于 common
- 依赖 MyBatis

```
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <!-- 数据库连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.1.1</version>
        </dependency>
```

在 src/main/resources/目录下，增加 application.yml 配置文件，增加连接池和 mybatis 相关配置

```
---
spring:
  datasource:
    name: mydb
    type: com.alibaba.druid.pool.DruidDataSource
    url: jdbc:mysql://127.0.0.1:3306/atcrowdfunding
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
mybatis:
  mapperLocations: classpath*:mybatis/mapper-*.xml
```

```
  typeAliasesPackage: com.atguigu.**.bean
```

- application.properties

```
spring.application.name=atcrowdfunding-member-service
server.port=2001
eureka.client.serviceUrl.defaultZone=http://127.0.0.1:1001/eureka/
```

在 XxxApplication 类中增加@EnableDiscoveryClient

扫描 Dao 接口，需要在 XxxApplication 类中增加扫描注解@MapperScan("com.atguigu.**.dao")及事务管理@EnableTransactionManagement

```
package com.atguigu.atcrowdfunding.member;


import org.mybatis.spring.annotation.MapperScan;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.transaction.annotation.EnableTransactionManagement;


@EnableTransactionManagement
@MapperScan("com.atguigu.**.dao")
@EnableDiscoveryClient
@SpringBootApplication
public class AtcrowdfundingBootMemberServiceApplication {


    public static void main(String[] args) {
        SpringApplication.run(AtcrowdfundingBootMemberServiceApplication.class, args);
    }
}
```

### 2.5.1 添加包结构

com.atguigu.atcrowdfunding.member

com.atguigu.atcrowdfunding.member.controller

com.atguigu.atcrowdfunding.member.dao

com.atguigu.atcrowdfunding.member.service.impl

### 2.5.2 增加 Member (存放到 common 项目中)

```
package com.atguigu.atcrowdfunding.common.bean;


import java.io.Serializable;


public class Member implements Serializable {
        private Integer id;
        private String loginacct;
        private String userpswd;
        private String username;
        private String email;
        private String authstatus;
        private String usertype;
        private String realname;
        private String cardnum;
        private String accttype;
...
}
```

### 2.5.3 增加 Controller

```
package com.atguigu.atcrowdfunding.member.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

import com.atguigu.atcrowdfunding.common.bean.Member;
import com.atguigu.atcrowdfunding.member.service.MemberService;

@RestController
public class MemberController {

    @Autowired
    private MemberService memberService;

    @RequestMapping("/query/{loginacct}")
    public Member queryMemberByLoginacct(@PathVariable("loginacct")String loginacct) {
        return memberService.queryByLoginacct(loginacct);//在 java 程序中做密码判断验证
    }
}
```

### 2.5.4 增加 Service

```
package com.atguigu.atcrowdfunding.member.service;
```

```
import com.atguigu.atcrowdfunding.common.bean.Member;


public interface MemberService {

Member queryByLoginacct(String loginacct);

}
```

```
package com.atguigu.atcrowdfunding.member.service.impl;


import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;


import com.atguigu.atcrowdfunding.common.bean.Member;

import com.atguigu.atcrowdfunding.member.dao.MemberDao;


@Service

@Transactional(readOnly=true)

public class MemberServiceImpl implements MemberService {


@Autowired

private MemberDao memberDao;


@Override

public Member queryByLoginacct(String loginacct) {

return memberDao.queryByLoginacct(loginacct);

}

}
```

### 2.5.5 增加 DAO

```
package com.atguigu.atcrowdfunding.member.dao;


import org.apache.ibatis.annotations.Select;
import com.atguigu.atcrowdfunding.common.bean.Member;


public interface MemberDao {
@Select("select * from t_member where loginacct = #{loginacct}")
Member queryByLoginacct(String loginacct);
}
```

### 2.5.6 测试独立的服务,没有问题,再继续…

## 2.6 配置 Portal 项目

### 2.6.1 增加包结构

com.atguigu.atcrowdfunding.portal

com.atguigu.atcrowdfunding.portal.config

com.atguigu.atcrowdfunding.portal.controller

com.atguigu.atcrowdfunding.portal.service

com.atguigu.atcrowdfunding.portal.web.listener

### 2.6.2 增加静态资源

### 2.6.3 增加 Freemarker 配置及相关

```
server.context-path=/
server.port=8080
server.session.timeout=60
server.tomcat.max-threads=800
server.tomcat.uri-encoding=UTF-8

spring.freemarker.allow-request-override=false
spring.freemarker.allow-session-override=false
spring.freemarker.cache=false
spring.freemarker.charset=UTF-8
spring.freemarker.check-template-location=true
spring.freemarker.content-type=text/html
spring.freemarker.enabled=true
spring.freemarker.expose-request-attributes=false
spring.freemarker.expose-session-attributes=false
spring.freemarker.expose-spring-macro-helpers=true
spring.freemarker.prefer-file-system-access=false
spring.freemarker.suffix=.ftl
spring.freemarker.template-loader-path=classpath:/templates/
spring.freemarker.settings.template_update_delay=0
spring.freemarker.settings.default_encoding=UTF-8
spring.freemarker.settings.classic_compatible=true
spring.freemarker.order=1
```

### 2.6.4 设置项目名称

```
spring.application.name=atcrowdfunding-portal
eureka.client.serviceUrl.defaultZone=http://localhost:1001/eureka/
```

### 2.6.5 增加 Controller,跳转登录页面

```
package com.atguigu.atcrowdfunding.portal.controller;


import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;


@Controller
public class PortalController {
    @RequestMapping("/login")
    public String login() {
        return "login";
    }
}
```

### 2.6.6 增加

- \atcrowdfunding-boot-portal\src\main\resources\templates\login.ftl

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="UTF-8">
```

```
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="description" content="">
    <meta name="keys" content="">
    <meta name="author" content="">
    <link rel="stylesheet" href="/bootstrap/css/bootstrap.min.css">
    <link rel="stylesheet" href="/css/font-awesome.min.css">
    <link rel="stylesheet" href="/css/login.css">
    <style>


    </style>
  </head>
  <body>
    <nav class="navbar navbar-inverse navbar-fixed-top" role="navigation">
      <div class="container">
        <div class="navbar-header">
          <div><a class="navbar-brand" href="index.html" style="font-size:32px;">尚筹网-创意产品众筹平台</a></div>
        </div>
      </div>
    </nav>

    <div class="container">
      <form id="loginForm" class="form-signin" role="form">
        <h2 class="form-signin-heading"><i class="glyphicon glyphicon-cloud"></i> 用户登录</h2>
          <div class="form-group has-success has-feedback">
            <input type="text" class="form-control" id="loginacct" name="loginacct" value="zhangsan"
```

```
placeholder="请输入登录账号" autofocus>
                <span class="glyphicon glyphicon-user form-control-feedback"></span>
            </div>
            <div class="form-group has-success has-feedback">
                <input type="password" class="form-control" id="userpswd" name="userpswd"
value="123123" placeholder="请输入登录密码" style="margin-top:10px;">
                <span class="glyphicon glyphicon-lock form-control-feedback"></span>
            </div>
            <div class="form-group has-success has-feedback">
                <select class="form-control" >
                    <option value="member">会员</option>
                    <option value="user">管理</option>
                </select>
            </div>
          <div class="checkbox">
            <label>
                <input type="checkbox" value="remember-me"> 记住我
            </label>
            <br>
            <label>
                忘记密码
            </label>
            <label style="float:right">
                <a href="reg.html">我要注册</a>
            </label>
          </div>
          <a class="btn btn-lg btn-success btn-block" onclick="dologin()" > 登录</a>
```

```
    </form>
</div>
<script src="${APP_PATH}/jquery/jquery-2.1.1.min.js"></script>
<script src="/bootstrap/js/bootstrap.min.js"></script>
<script src="/layer/layer.js"></script>
<script>
function dologin() {
    var loginacct = $("#loginacct");
    if ( loginacct.val() == "" ) {
        layer.msg("登陆账号不能为空，请输入", {time:1500, icon:5, shift:6}, function(){
            loginacct.focus();
        });
        return;
    }

    var userpswd = $("#userpswd");
    if ( userpswd.val() == "" ) {
        layer.msg("登录密码不能为空，请输入", {time:1500, icon:5, shift:6}, function(){
            userpswd.focus();
        });
        return;
    }
    var loadingIndex = -1;
    $.ajax({
        type : "POST",
        url  : "${APP_PATH}/doMemberLogin",
        data : {
```

```
                "loginacct" : loginacct.val(),
                "userpswd"    : userpswd.val()
            },
            beforeSend : function() {
                loadingIndex = layer.load(2, {time: 10*1000});
            },
            success : function(result){
                layer.close(loadingIndex);
                if ( result.success == true ) {
                    window.location.href = "${APP_PATH}/member";
                } else {
                    layer.msg(result.message, {time:1500, icon:5, shift:6});
                }
            },
            error : function(){
            }
        });
    }
    </script>
  </body>
</html>
```

## 2.6.7 在 Controller 中完成登录操作

```
package com.atguigu.atcrowdfunding.portal.controller;

import javax.servlet.http.HttpSession;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;

import com.atguigu.atcrowdfunding.common.BaseController;
import com.atguigu.atcrowdfunding.common.bean.Member;
import com.atguigu.atcrowdfunding.portal.service.MemberService;

@Controller
public class PortalController extends BaseController {
    @Autowired
    private MemberService memberService;

    //跳转登录页面
    @RequestMapping("/login")
        public String login() {
        return "login";
    }

    //登录系统
    @ResponseBody
    @RequestMapping("/doMemberLogin")
    public Object doMemberLogin( Member member, HttpSession session ) {
        start();

        try {
```

```
            Member dbMember = memberService.queryMemberByLoginacct(member.getLoginacct());
            if ( dbMember == null ) {
                fail();
            } else {
                if ( dbMember.getUserpswd().equals(member.getUserpswd()) ) {
                    session.setAttribute("loginMember", dbMember);
                    success();
                } else {
                    fail();
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
            fail();
        }

        return end();
    }
}
```

### 2.6.8 增加 Service

```
package com.atguigu.atcrowdfunding.portal.service;


import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
```

```
import com.atguigu.atcrowdfunding.common.bean.Member;


@FeignClient("atcrowdfunding-member-service")
public interface MemberService {

    @RequestMapping("/query/{loginacct}")
    public Member queryMemberByLoginacct(@PathVariable("loginacct")String loginacct);
}
```

### 2.6.9 增加登录成功后跳转映射

```
//登录后跳转会员中心.
@RequestMapping("/member")
public String member() {
    return "member";
}
```

### 2.6.10 增加 member.ftl

```
member loginacct : ${loginMember.loginacct}!
```

## 2.7 增加监听器,解决路径问题(修改 ftl 页面路径)

```
package com.atguigu.atcrowdfunding.portal.web.listener;


import javax.servlet.ServletContext;
import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
```

```
import javax.servlet.annotation.WebListener;


@WebListener    //声明监听器对象,需要设置扫描注解@ServletComponentScan
//@WebServlet //声明 Servlet 对象
//@WebFilter //声明过滤器对象
public class PortalServetStartupListener implements ServletContextListener {
    //观察服务器启动时,监听器对象是否被创建.
    public PortalServetStartupListener() {
        System.out.println("PortalServetStartupListener...");
    }


    @Override
    public void contextInitialized(ServletContextEvent sce) {
        ServletContext application = sce.getServletContext();
        String path = application.getContextPath();
        application.setAttribute("APP_PATH", path);
    }


    @Override
    public void contextDestroyed(ServletContextEvent sce) {
        // TODO Auto-generated method stub
    }
}
```

- 将页面上所有前台路径增加上下文路径:  ${APP_PATH}/

## 2.8 增加 Redis 配置,解决 Session 共享问题

```
        <dependency>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-redis</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.session</groupId>
            <artifactId>spring-session-data-redis</artifactId>
        </dependency>
```

```
server.address=127.0.0.1
server.tomcat.remote-ip-header=x-forwarded-for
server.tomcat.protocol-header=x-forwarded-proto
server.tomcat.port-header=X-Forwarded-Port
server.use-forward-headers=true
```

```
spring.session.store-type=redis
spring.redis.database=1
spring.redis.host=127.0.0.1
spring.redis.password=123123
spring.redis.port=6379
spring.redis.pool.max-idle=8
spring.redis.pool.min-idle=0
spring.redis.pool.max-active=8
spring.redis.pool.max-wait=-1
spring.redis.timeout=60000
```

```
package com.atguigu.atcrowdfunding.portal.config;


import org.springframework.context.annotation.Configuration;
import org.springframework.session.data.redis.config.annotation.web.http.EnableRedisHttpSession;
```

```
@Configuration
@EnableRedisHttpSession
public class RedisSessionConfig {}
```

# 第二章 Activiti 流程框架

## 4.1 介绍

### 4.1.1 审核流程动态 Activiti

业务发展，需要上传更多的内容，流程会发生变化

例如：身份证号的公安部系统交互

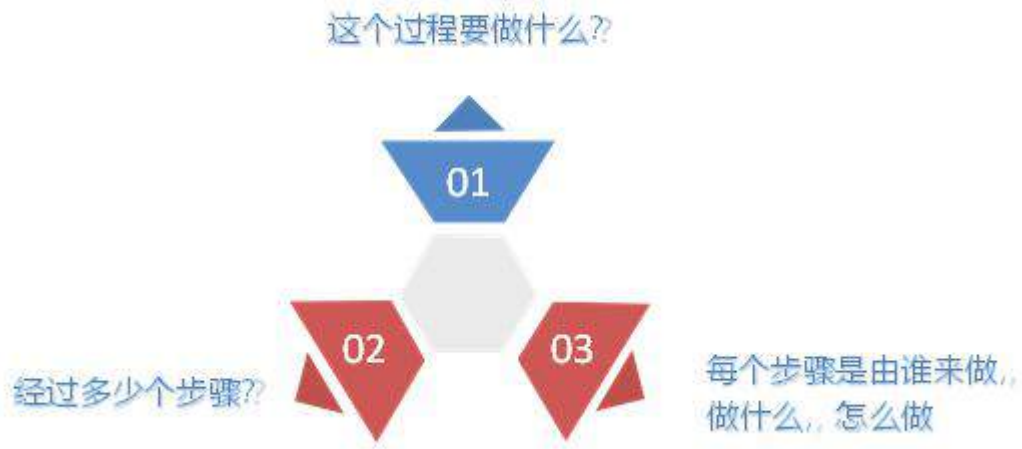
### 4.1.2 什么是工作流

简单的理解就是工作的流程，这里的流程指的是，完成一个企业中具体业务的一系列工作步骤，

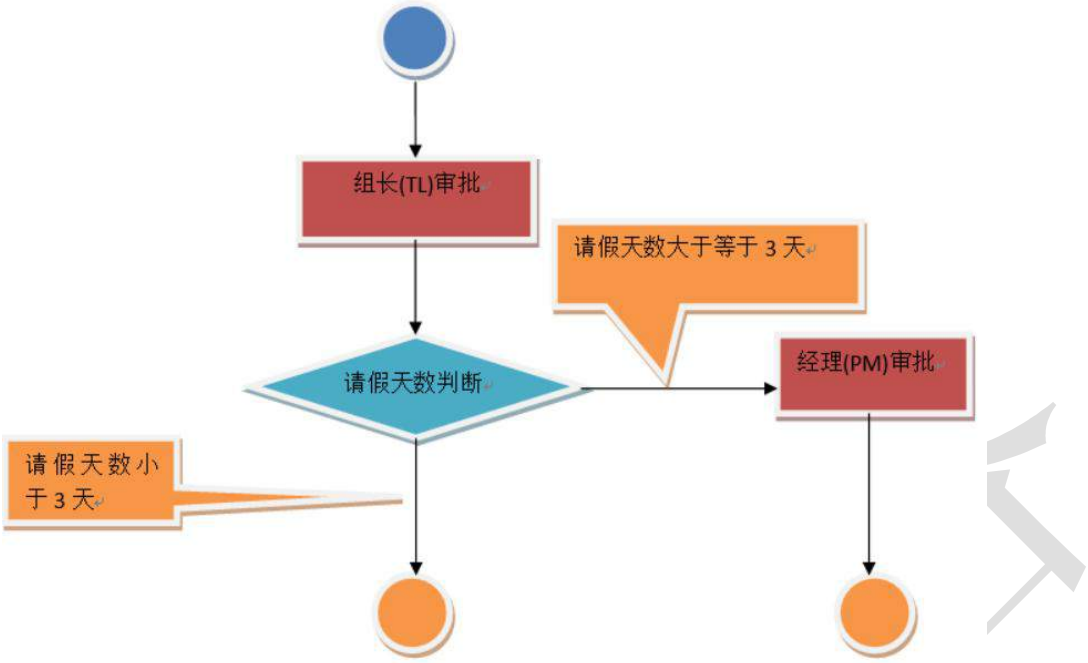
所有的步骤合在一起就是业务从开始到结束的流转过程。

从计算机系统的角度来讲，工作流系统表示：业务过程的部分和整体在计算机应用环境的自动化操作

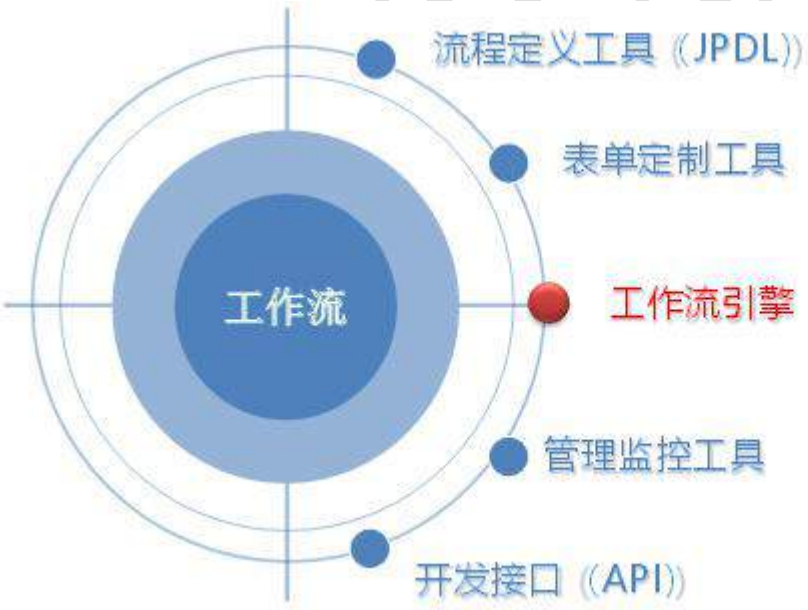

### 4.1.3 工作流的三大要素




### 4.1.4 工作流示例 - 请假




### 4.1.5 工作流系统的组成(安装 Eclipse 的 Activiti 插件)



## 4.1.6 工作流的相关概念


## 4.1.7 工作流产品


## 4.1.8 Activiti5 框架的由来

JBPM，全称是 Java Business Process Management（业务流程管理），它是覆盖了业务流程管理、工作

流、服务协作等领域的一个开源的、灵活的、易扩展的可执行流程语言框架。JBPM 是公开源代码项目

JBPM 在 2004 年 10 月 18 日，发布了 2.0 版本，并在同一天加入了 JBoss ,成为了 JBoss 企业中间件平台的一个组成部分，它的名称也改成 JBoss jBPM

在 JBPM4 之后，公司内部对于软件的规划发生了分歧，所以当时的项目架构师脱离了原来的公司，加入新的公司后，改了名称 Activiti5。

JBPM 采用 Hibernte （JBPM 核心发生了很大的变化）

Activiti5 采用 MyBatis (MyBatis 3, 1 和 2 都叫做 iBatis)

## 4.1.9 Activiti5 框架

**ProcessEngine** 这个类是 Activiti5 的核心，所有的服务都需要通过 ProcessEngine 来创建，该类是线程安全的

Activiti5 的持久化层采用的是 Mybatis，这样移植性好

Activiti5 主要包括 7 个 Service，这些 Service 都是通过 ProcessEngine 创建

repositoryService（持久化服务） 与数据库进行交互的

runtimeService（运行时服务） 与运行流程有关

formService（表单服务）

identityService（身份信息）

taskService（任务服务） 与流程中的每一个步骤有关

historyService（历史信息） 查看历史的流程步骤

managementService（管理定时任务）在固定的时间点完成固定的任务

## 4.1.10 Activiti5 框架表结构

Activiti 使用到的表都是 ACT_开头的。

ACT_RE_*: ‘RE’表示 repository(存储)，RepositoryService 接口所操作的表。带此前缀的表包含的是静态信息，如，流程定义，流程的资源（图片，规则等）。

ACT_RU_*: ‘RU’表示 runtime，运行时表-RuntimeService。这是运行时的表存储着流程变量，用户

任务，变量，职责（job）等运行时的数据。Activiti 只存储实例执行期间的运行时数据，当流程实例结束时，将删除这些记录。这就保证了这些运行时的表小且快。

ACT_ID_*: ’ID’表示 identity (组织机构)，IdentityService 接口所操作的表。用户记录，流程中使用到的用户和组。这些表包含标识的信息，如用户，用户组，等等。

ACT_HI_*: ’HI’表示 history，历史数据表，HistoryService。就是这些表包含着流程执行的历史相关数据，如结束的流程实例，变量，任务，等等

ACT_GE_*: 全局通用数据及设置(general)，各种情况都使用的数据

## 4.1.11 Activiti5 框架表结构

act_ge_bytearray 二进制数据表

act_ge_property 属性数据表存储整个流程引擎级别的数据,初始化表结构时，会默认插入三条记录，

act_hi_actinst 历史节点表

act_hi_attachment 历史附件表

act_hi_comment 历史意见表

act_hi_identitylink 历史流程人员表

act_hi_detail 历史详情表，提供历史变量的查询

act_hi_procinst 历史流程实例表

act_hi_taskinst 历史任务实例表

act_hi_varinst 历史变量表

act_id_group 用户组信息表

act_id_info 用户扩展信息表

act_id_membership 用户与用户组对应信息表

act_id_user 用户信息表

act_re_deployment 部署信息表

act_re_model 流程设计模型部署表

act_re_procdef 流程定义数据表

act_ru_event_subscr　throwEvent、catchEvent 时间监听信息表

act_ru_execution 运行时流程执行实例表

act_ru_identitylink 运行时流程人员表，主要存储任务节点与参与者的相关信息

act_ru_job 运行时定时任务数据表

act_ru_task 运行时任务节点表

act_ru_variable 运行时流程变量数据表

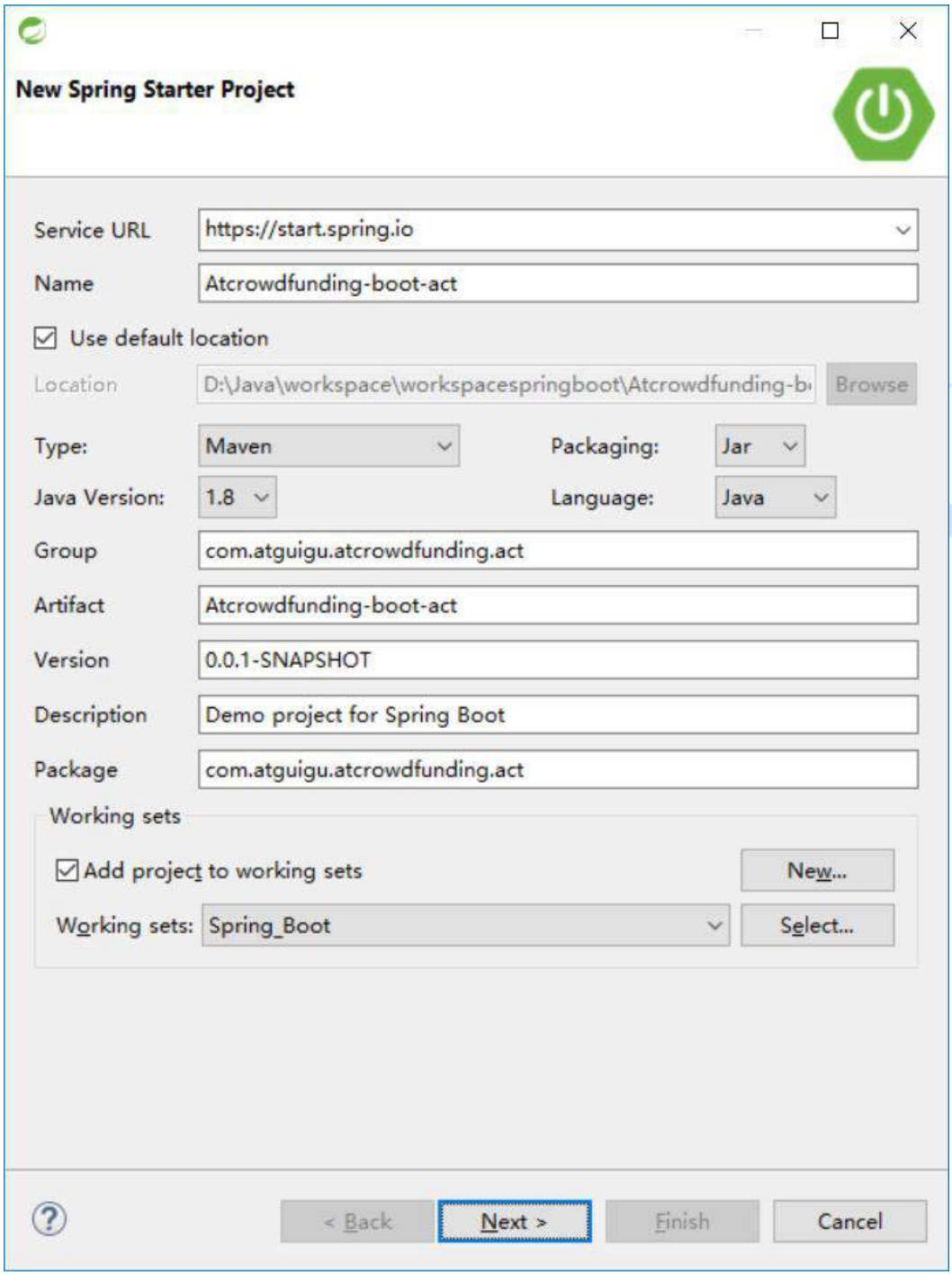
## 4.2 练习

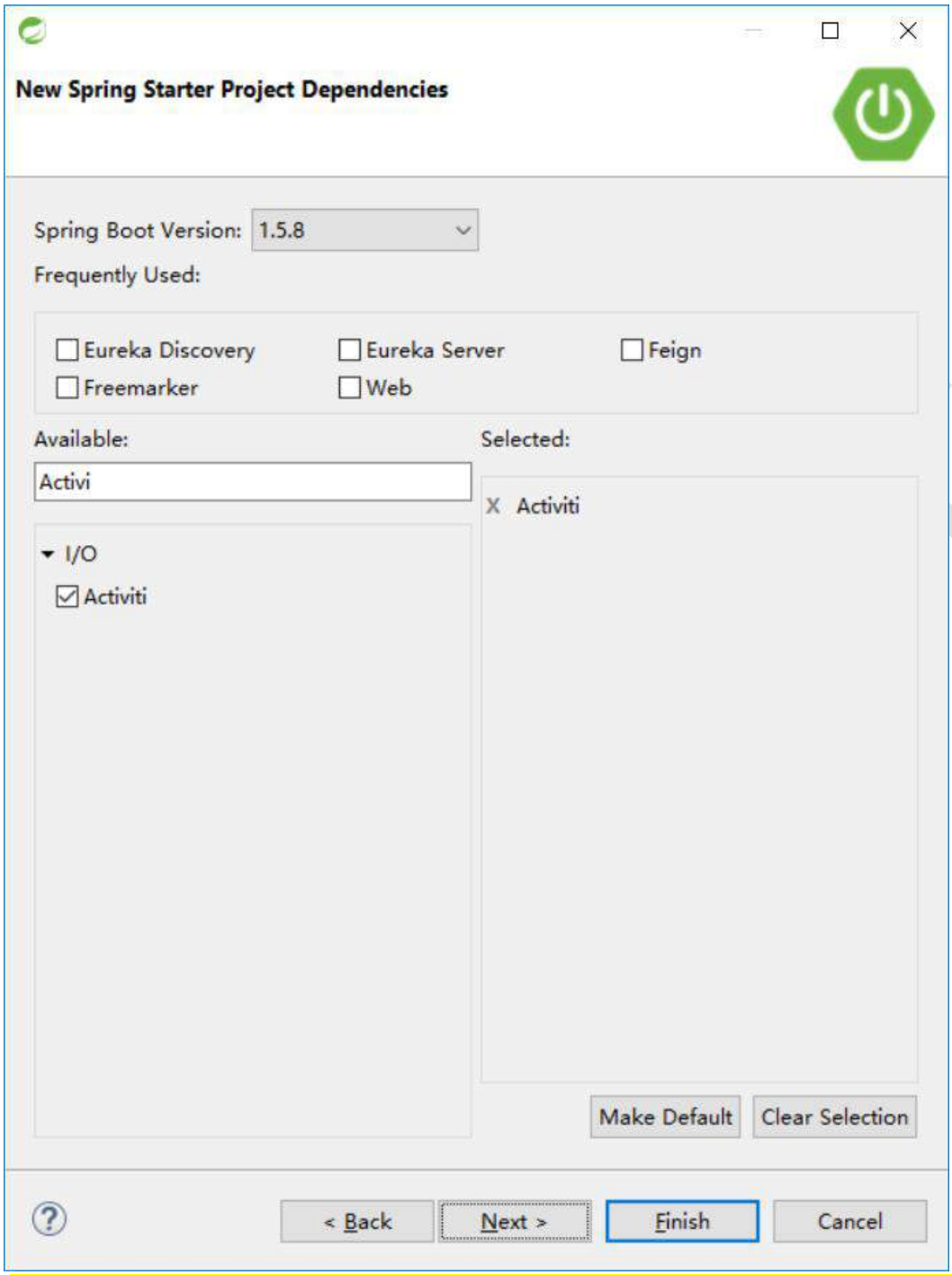
### 4.2.1 插件安装

### 4.2.2 画请假流程图

### 4.2.3 创建流程表

### 4.2.4 创建项目


New Spring Starter Project
Service URL
https://start.spring.io
Name
Atcrowdfunding-boot-act
Use default location
Location
D:\Java\workspace\workspacespringboot\Atcrowdfunding-b
Browse
Type:
Maven
Packaging:
Jar
Java Version:
1.8
Language:
Java
Group
com.atguigu.atcrowdfunding.act
Artifact
Atcrowdfunding-boot-act
Version
0.0.1-SNAPSHOT
Description
Demo project for Spring Boot
Package
com.atguigu.atcrowdfunding.act
Working sets
Add project to working sets
New...
Working sets:
Spring_Boot
Select...
< Back
Next >
Finish
Cancel




- 新版本中已不支持自动添加 Activiti 组件了,无法勾选该模块 😭 ,请自行添加依赖.

```
<dependency>
    <groupId>org.activiti</groupId>
    <artifactId>activiti-spring-boot-starter-basic</artifactId>
    <version>5.21.0</version>
```

```
        </dependency>
```

## 4.3 流程引擎对象

### 4.3.1.搭建环境

#### 1.增加依赖

```
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
        </dependency>
        <!-- 数据库连接池 -->
        <dependency>
            <groupId>com.alibaba</groupId>
            <artifactId>druid</artifactId>
            <version>1.0.5</version>
        </dependency>
        <dependency>
            <groupId>org.mybatis.spring.boot</groupId>
            <artifactId>mybatis-spring-boot-starter</artifactId>
            <version>1.1.1</version>
        </dependency>
```

#### 2.增加配置

```
---
spring:
```

```
  datasource:
    name: mydb
    type: com.alibaba.druid.pool.DruidDataSource
    url: jdbc:mysql://127.0.0.1:3306/atcrowdfunding
    username: root
    password: root
    driver-class-name: com.mysql.jdbc.Driver
mybatis:
  mapperLocations: classpath*:mybatis/mapper-*.xml
  typeAliasesPackage: com.atguigu.**.bean
```

## 3.基本代码

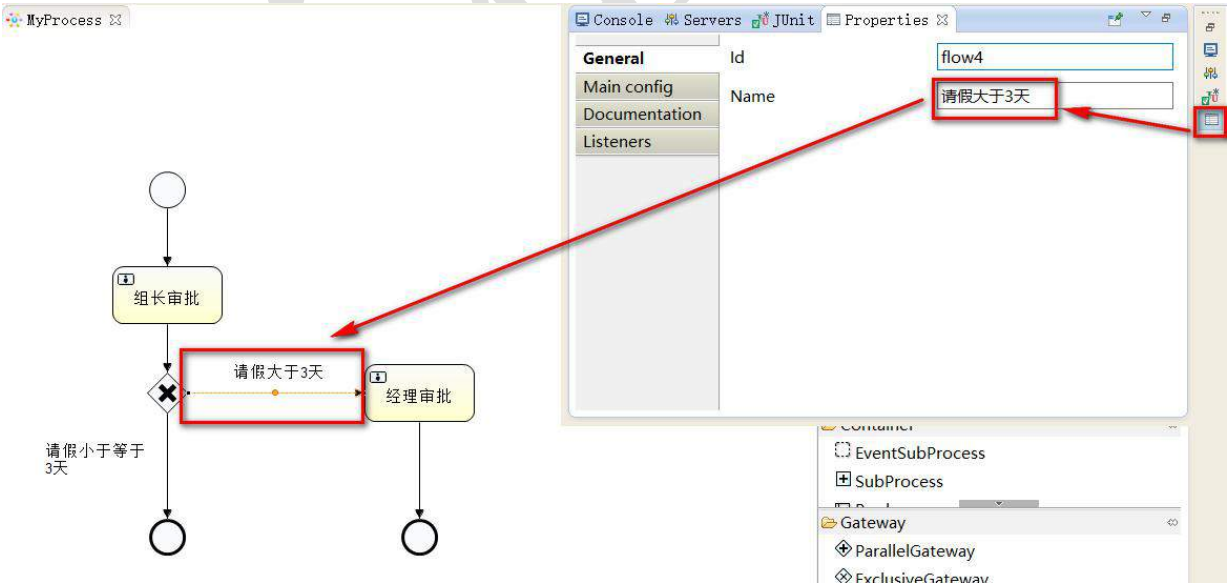
```
package com.atguigu.atcrowdfunding.act;


import org.activiti.engine.ProcessEngine;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;


@RunWith(SpringRunner.class)
@SpringBootTest
public class ApplicationTests {
    @Autowired
    private ProcessEngine processEngine ;
```

```
    @Autowired
    private RepositoryService repositoryService ;

    @Test
    public void contextLoads() {
        System.out.println( processEngine );
        System.out.println(repositoryService);
    }
}
```

## 4.4 流程定义

### 4.4.1 请假流程

### 4.4.2 定义流程分支的箭头名称



## 4.4.3 源文件：MyProcess.bpmn (了解)

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:activiti="http://activiti.org/bpmn"
xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
xmlns:omgdc="http://www.omg.org/spec/DD/20100524/DC"
xmlns:omgdi="http://www.omg.org/spec/DD/20100524/DI"
typeLanguage="http://www.w3.org/2001/XMLSchema"
expressionLanguage="http://www.w3.org/1999/XPath" targetNamespace="http://www.activiti.org/test">
  <process id="myProcess" name="My process" isExecutable="true">
    <startEvent id="startevent1" name="Start"></startEvent>
    <userTask id="usertask1" name="组长审批"></userTask>
    <sequenceFlow id="flow1" sourceRef="startevent1" targetRef="usertask1"></sequenceFlow>
    <exclusiveGateway id="exclusivegateway1" name="Exclusive Gateway"></exclusiveGateway>
    <sequenceFlow id="flow2" sourceRef="usertask1" targetRef="exclusivegateway1"></sequenceFlow>
    <endEvent id="endevent1" name="End"></endEvent>
    <sequenceFlow id="flow3" name="请假小于等于3天" sourceRef="exclusivegateway1"
targetRef="endevent1"></sequenceFlow>
    <userTask id="usertask2" name="经理审批"></userTask>
    <sequenceFlow id="flow4" name="请假大于3天" sourceRef="exclusivegateway1"
targetRef="usertask2"></sequenceFlow>
    <endEvent id="endevent2" name="End"></endEvent>
    <sequenceFlow id="flow5" sourceRef="usertask2" targetRef="endevent2"></sequenceFlow>
  </process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_myProcess">
    <bpmndi:BPMNPlane bpmnElement="myProcess" id="BPMNPlane_myProcess">
```

```
<bpmndi:BPMNShape bpmnElement="startevent1" id="BPMNShape_startevent1">
   <omgdc:Bounds height="35.0" width="35.0" x="480.0" y="130.0"></omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="usertask1" id="BPMNShape_usertask1">
   <omgdc:Bounds height="55.0" width="105.0" x="445.0" y="220.0"></omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="exclusivegateway1" id="BPMNShape_exclusivegateway1">
   <omgdc:Bounds height="40.0" width="40.0" x="477.0" y="320.0"></omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="endevent1" id="BPMNShape_endevent1">
   <omgdc:Bounds height="35.0" width="35.0" x="480.0" y="460.0"></omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="usertask2" id="BPMNShape_usertask2">
   <omgdc:Bounds height="55.0" width="105.0" x="685.0" y="313.0"></omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShape bpmnElement="endevent2" id="BPMNShape_endevent2">
   <omgdc:Bounds height="35.0" width="35.0" x="720.0" y="460.0"></omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNEdge bpmnElement="flow1" id="BPMNEdge_flow1">
   <omgdi:waypoint x="497.0" y="165.0"></omgdi:waypoint>
   <omgdi:waypoint x="497.0" y="220.0"></omgdi:waypoint>
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdge bpmnElement="flow2" id="BPMNEdge_flow2">
   <omgdi:waypoint x="497.0" y="275.0"></omgdi:waypoint>
   <omgdi:waypoint x="497.0" y="320.0"></omgdi:waypoint>
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdge bpmnElement="flow3" id="BPMNEdge_flow3">
```

```
            <omgdi:waypoint x="497.0" y="360.0"></omgdi:waypoint>
            <omgdi:waypoint x="497.0" y="460.0"></omgdi:waypoint>
            <bpmndi:BPMNLabel>
               <omgdc:Bounds height="48.0" width="100.0" x="381.0" y="379.0"></omgdc:Bounds>
            </bpmndi:BPMNLabel>
         </bpmndi:BPMNEdge>
         <bpmndi:BPMNEdge bpmnElement="flow4" id="BPMNEdge_flow4">
            <omgdi:waypoint x="517.0" y="340.0"></omgdi:waypoint>
            <omgdi:waypoint x="685.0" y="340.0"></omgdi:waypoint>
            <bpmndi:BPMNLabel>
               <omgdc:Bounds height="16.0" width="100.0" x="559.0" y="313.0"></omgdc:Bounds>
            </bpmndi:BPMNLabel>
         </bpmndi:BPMNEdge>
         <bpmndi:BPMNEdge bpmnElement="flow5" id="BPMNEdge_flow5">
            <omgdi:waypoint x="737.0" y="368.0"></omgdi:waypoint>
            <omgdi:waypoint x="737.0" y="460.0"></omgdi:waypoint>
         </bpmndi:BPMNEdge>
      </bpmndi:BPMNPlane>
   </bpmndi:BPMNDiagram>
</definitions>
```

## 4.5 部署-查询

### 4.5.1 部署流程代码

*注意: bpmn 文件存放到 processes 文件夹下,会自动部署.*

```
Deployment deploy =
repositoryService.createDeployment().addClasspathResource("MyProcess.bpmn").deploy();
```

```
//    "processes/MyProcess.bpmn"
```

## 4.5.2 部署后表数据分析

### 1.act_ge_bytearray 表

二进制数据表，存储了流程定义图形的 XML 文件和图片信息

保存流程定义的 xml 信息

保存流程定义的图片

### 2.act_re_deployment

部署信息表，存储了部署的相关信息(部署时间)

### 3.act_re_procdef

流程定义数据表，存储了当前流程图形的相关信息(id,name,版本号)

### 4.解决乱码问题

```
package com.atguigu.atcrowdfunding.act.config;


import org.activiti.spring.SpringProcessEngineConfiguration;
import org.activiti.spring.boot.ProcessEngineConfigurationConfigurer;
import org.springframework.context.annotation.Configuration;


@Configuration
public class ActivitiConfig implements ProcessEngineConfigurationConfigurer{
    @Override
```

```
    public void configure(SpringProcessEngineConfiguration processEngineConfiguration) {
        processEngineConfiguration.setActivityFontName("宋体");
        processEngineConfiguration.setLabelFontName("宋体");
    }
}
```

### 4.5.3 查询流程定义数据

```
//查询部署流程定义
@Test
public void query() {
ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();
/*List<ProcessDefinition> list = query.list();
for (ProcessDefinition pd : list) {
System.out.println(pd.getId() +" - " + pd.getName() + " - " + pd.getVersion());
}*/
//long count = query.count(); //流程定义的数量
//System.out.println(count);
//ProcessDefinition singleResult = query.latestVersion().singleResult();//获取最后一次部署的流程定义对象
//System.out.println(singleResult.getId() +" - " + singleResult.getName() + " - " + singleResult.getVersion());
/*ProcessDefinitionQuery processDefinitionQuery = query.orderByProcessDefinitionVersion().desc();//根据版本号进行倒序排序
List<ProcessDefinition> list2 = processDefinitionQuery.list();
for (ProcessDefinition pd : list2) {
System.out.println("---"+pd.getId() +" - " + pd.getName() + " - " + pd.getVersion());
}*/
```

```
//query.listPage(firstResult, maxResults);//分页的方法
ProcessDefinition singleResult = query.processDefinitionKey("myProcess").latestVersion().singleResult();
System.out.println(singleResult.getId() +" - " + singleResult.getName() + " - " + singleResult.getVersion());
}
```

## 4.6 启动流程实例

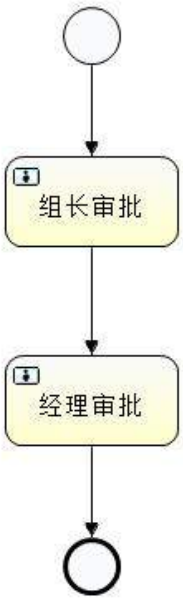
### 4.6.1 创建（启动）流程实例

```
@Test
public void testProcessInstance(){

    //查询流程定义
    ProcessDefinition processDefinition = repositoryService
    .createProcessDefinitionQuery()
    .processDefinitionKey("myProcess")
    .latestVersion()
    .singleResult();

    //运行启动流程实例
    /*
     ProcessInstance[201]
     act_hi_procinst： 历史流程实例表
    保存了流程实例的信息
     act_hi_taskinst： 历史任务实例表
    保存了流程任务的相关信息
    act_hi_actinst:历史节点表
    保存了流程执行的节点顺序
```

```
        act_ru_execution : 运行时流程执行实例表
        保存了当前流程执行的节点数据,流程开始会自动完成,直接执行第一个任务
        act_ru_task : 运行时任务节点表
        保存当前流程执行的任务数据
        */
        ProcessInstance processInstance = runtimeService.startProcessInstanceById(processDefinition.getId());
        System.out.println(processInstance);
}
```

## 4.6.2 启动流程后的分析

### 1.流程图




### 2.启动流程的数据分析

act_ru_execution : 运行时流程执行实例表

act_ru_task : 运行时任务节点表

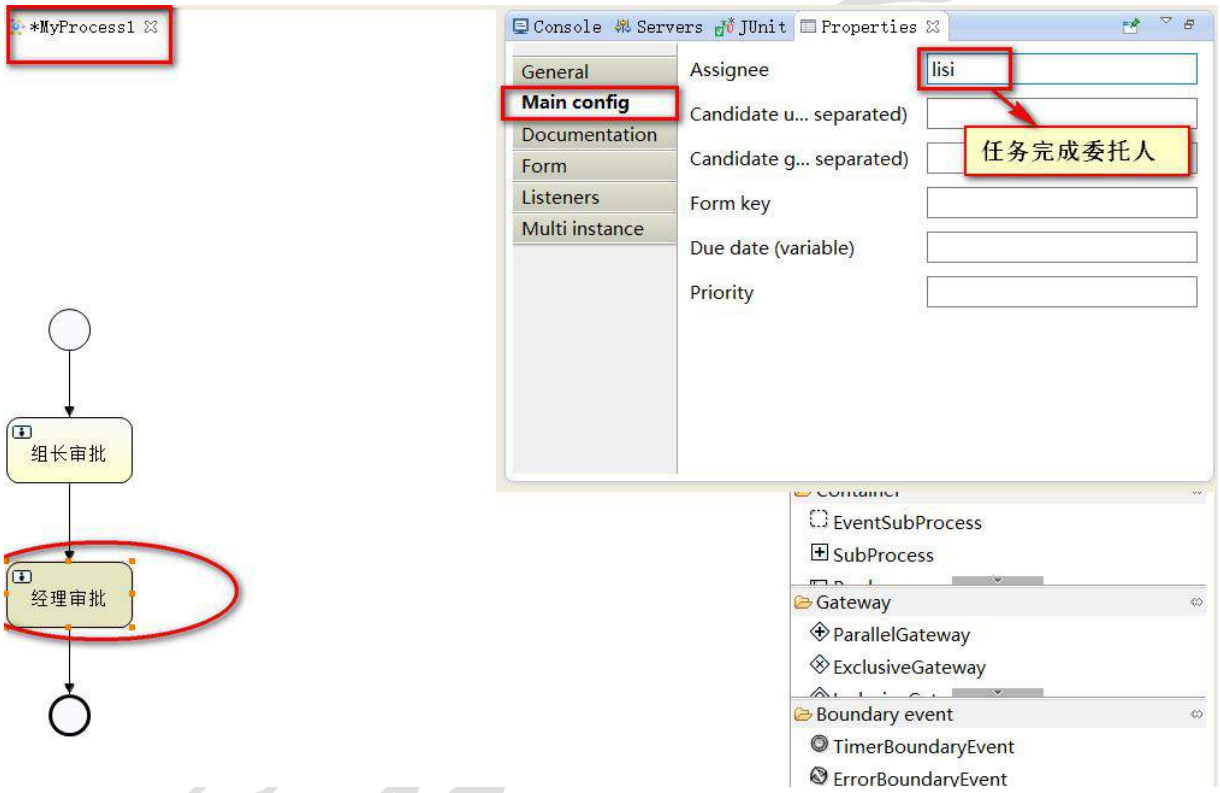
act_hi_procinst : 历史流程实例表

act_hi_taskinst：历史任务实例表

act_hi_actinst：历史节点列表,把每一个执行过的节点数据保存

## 4.7 任务

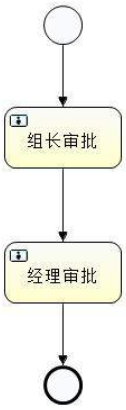
### 4.7.1 定义新的请假流程：设置任务节点的委托人



## 4.7.2 部署新的工作流程，启动流程实例，观察表数据

2.1act_ru_identitylink（增加了用户 id）


**2.2.流程任务**




2.3 获取流程任务

```
@Test
public void testTask(){
    TaskQuery taskQuery = taskService.createTaskQuery();
    List<Task> list1 = taskQuery.taskAssignee("zhangsan").list(); //获取 zhangsan 的任务列表
    List<Task> list2 = taskQuery.taskAssignee("lisi").list();//获取 lisi 的任务列表
```

```
    for (Task task : list1) {
        System.out.println("zhangsan 的任务="+task.getName());
    }
    System.out.println("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~");
    for (Task task : list2) {
        System.out.println("lisi 的任务="+task.getName());
    }
}
```

打印结果：

```
zhangsan 的任务=组长审批
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

## 2.4 查询并完成任务

```
@Test
public void testTask(){
    TaskQuery taskQuery = taskService.createTaskQuery();
    List<Task> list1 = taskQuery.taskAssignee("zhangsan").list(); //获取 zhangsan 的任务列表
    List<Task> list2 = taskQuery.taskAssignee("lisi").list();//获取 lisi 的任务列表
    for (Task task : list1) {
        System.out.println("zhangsan 的任务="+task.getName());
        taskService.complete(task.getId());
    }
}
```

完成任务的数据变化：

act_ru_execution, act_ru_task 运行服务的表的当前流程实例数据全部清空

act_hi_procinst 历史流程实例表

act_hi_taskinst 历史任务实例表

act_hi_actinst 历史节点表

act_hi_identitylink 历史流程人员表

## 4.8 历史

### 4.8.1 获取历史服务对象，查看历史流程

```
@Test
public void testHistoryService(){
    //查询流程定义
    ProcessDefinition processDefinition = repositoryService
    .createProcessDefinitionQuery()
    .processDefinitionKey("myProcess")
    .latestVersion()
    .singleResult();

    HistoricProcessInstanceQuery query = historyService.createHistoricProcessInstanceQuery();
    HistoricProcessInstance historicProcessInstance =
                query.processInstanceId("401").finished().singleResult();
    //HistoricProcessInstanceEntity[superProcessInstanceId=null]
    System.out.println(historicProcessInstance);
}
```

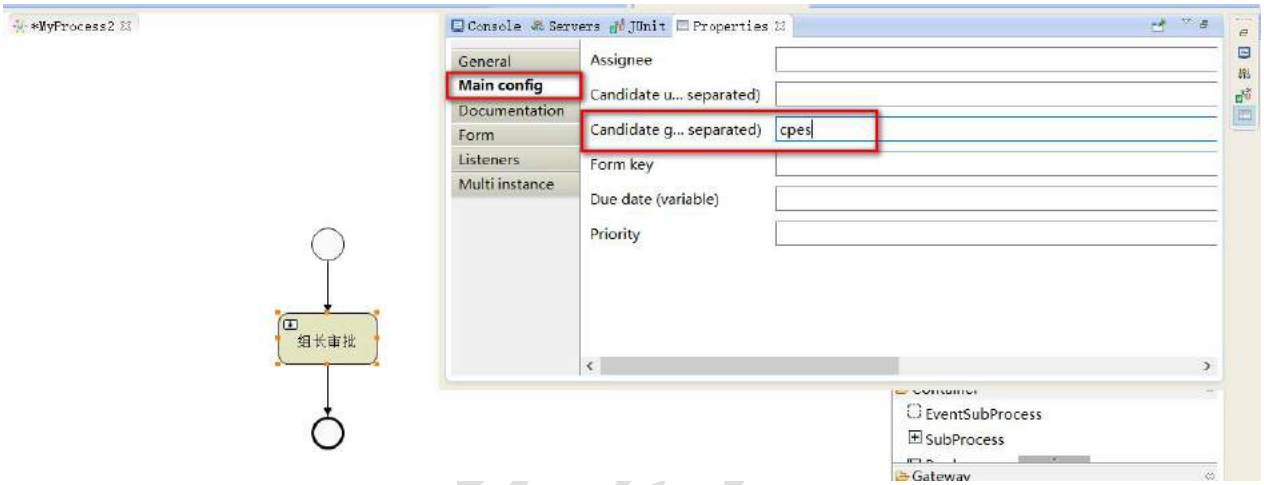
### 4.8.2 如何判断历史流程是否完全结束

finished() 判断流程实例是否完成状态，如果是完成状态会返回历史流程实例对象，否则为 null

# 4.9 任务领取

## 4.9.1 指派任务

将任务指派给 一个组，然后，由组内指定人进行任务完成操作。




## 4.9.2 领取任务

### 1.部署流程

### 2.启动流程实例

### 3.测试领取任务

```
@Test
public void testTaskClaim(){
    TaskQuery taskQuery = taskService.createTaskQuery();
    List<Task> list = taskQuery.taskCandidateGroup("cpes").list();
```

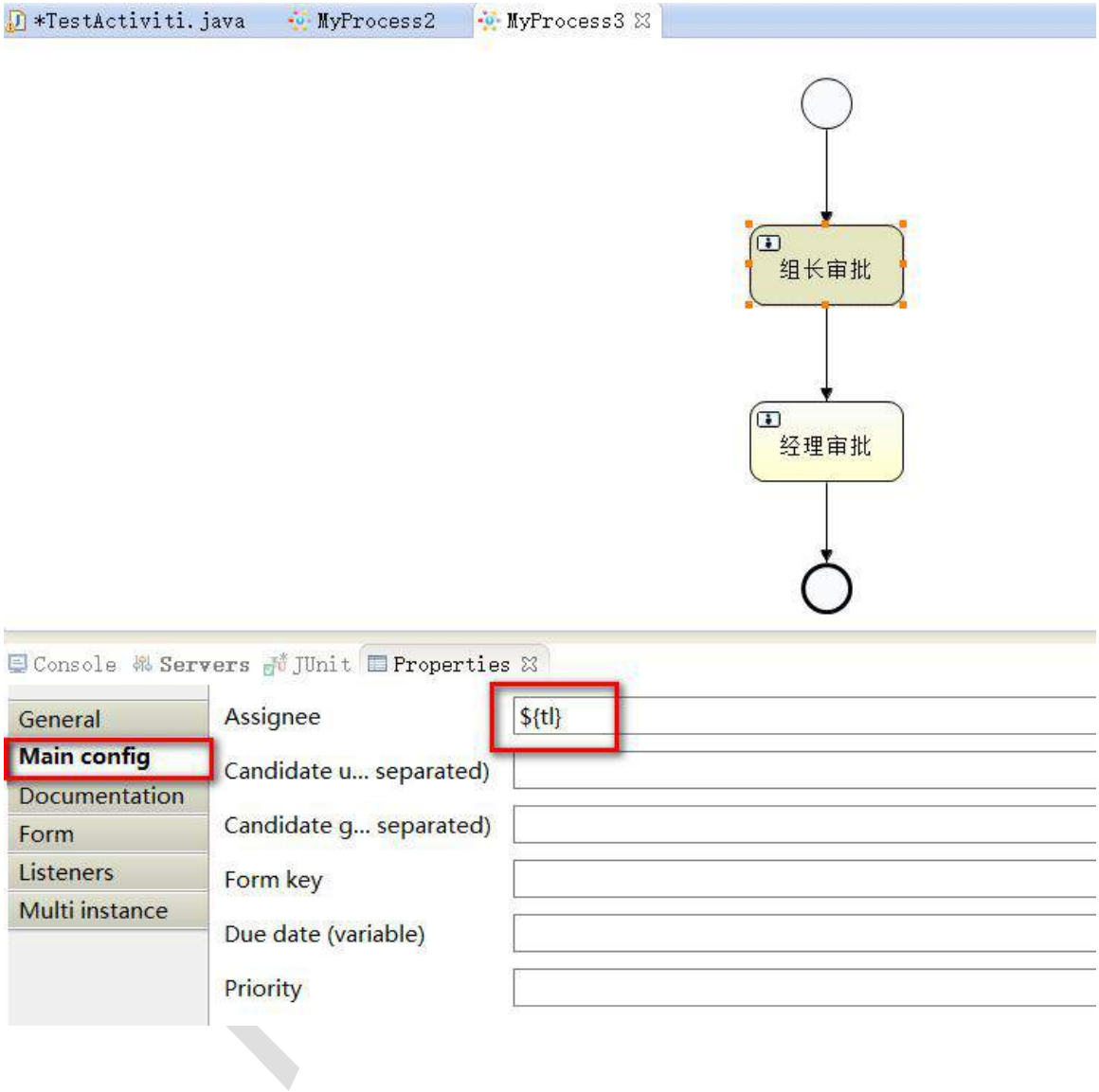
```
        long count = taskQuery.taskAssignee("zhangsan").count();
        System.out.println("zhangsan 领取之前的任务数量："+count);

        for (Task task : list) {
            taskService.claim(task.getId(), "zhangsan"); //领取任务
        }

        taskQuery = taskService.createTaskQuery();
        count = taskQuery.taskAssignee("zhangsan").count();
        System.out.println("zhangsan 领取之后的任务数量："+count);
}
```

领取任务案例：

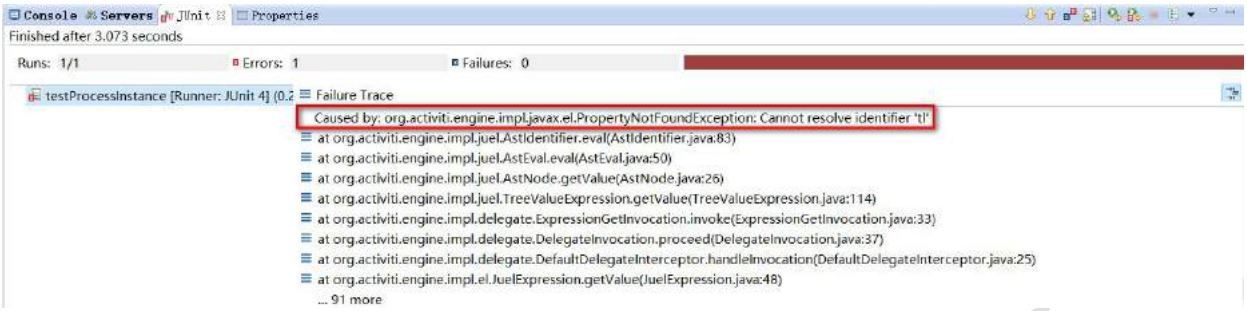
银行：排队结账

## 4.10 流程变量

### 4.10.1 重新定义流程，委托人采用流程变量获取委托人



## 4.10.2 部署，启动流程，报错



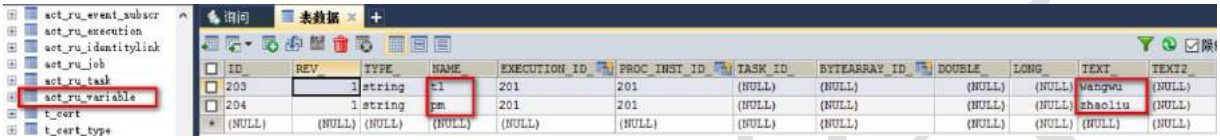

## 4.10.3 启动流程时，需要指定委托人参数

### 1.代码

```
//流程变量
@Test
public void testProcessVar(){
    //查询流程定义
    ProcessDefinition processDefinition = repositoryService
    .createProcessDefinitionQuery()
    .processDefinitionKey("myProcess")
    .latestVersion()
    .singleResult();

    Map<String,Object> variables = new HashMap<String,Object>();
    variables.put("tl", "wangwu");
    variables.put("pm", "zhaoliu");

    //ProcessInstance processInstance =
 runtimeService.startProcessInstanceById(processDefinition.getId());
```

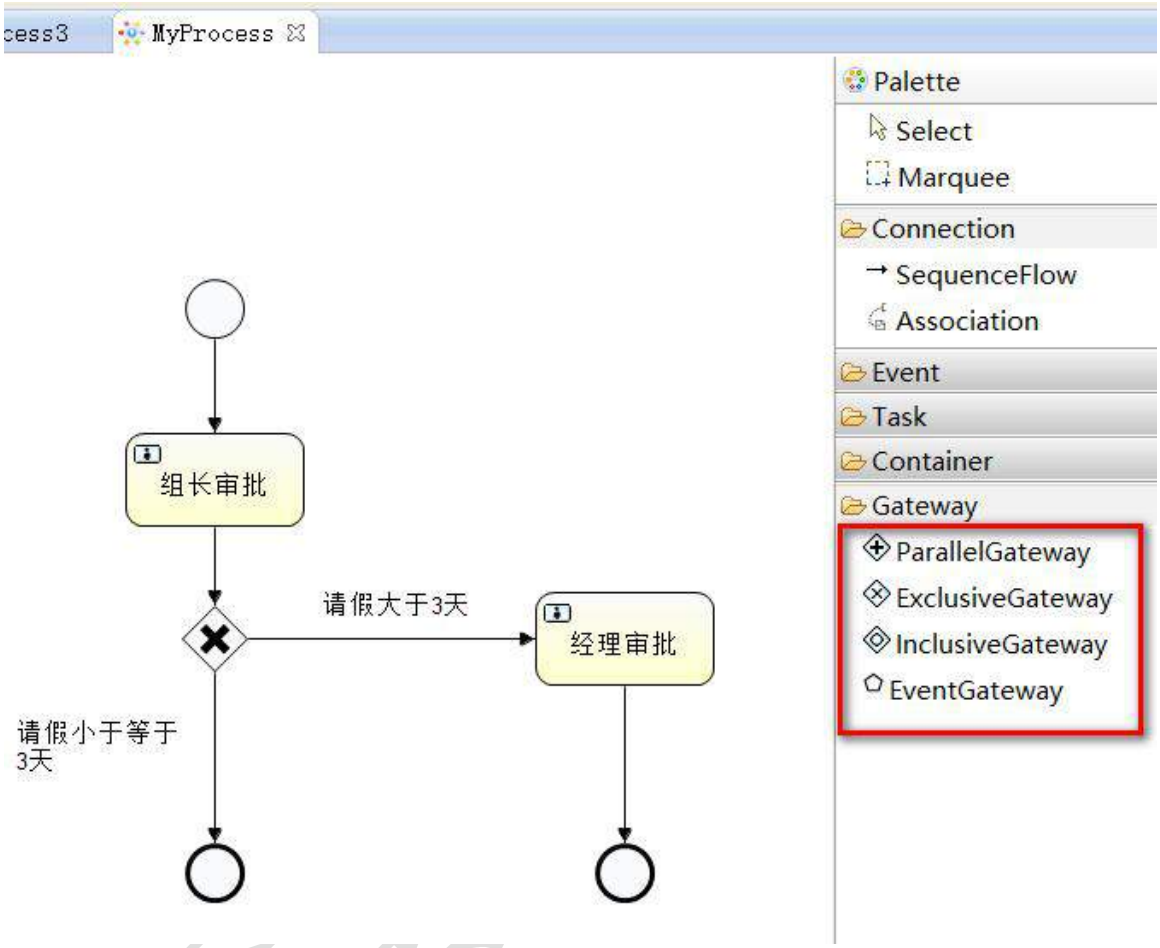
```
        ProcessInstance processInstance =
runtimeService.startProcessInstanceById(processDefinition.getId(),variables);
        System.out.println(processInstance);
}
```

## 2.数据库数据变化



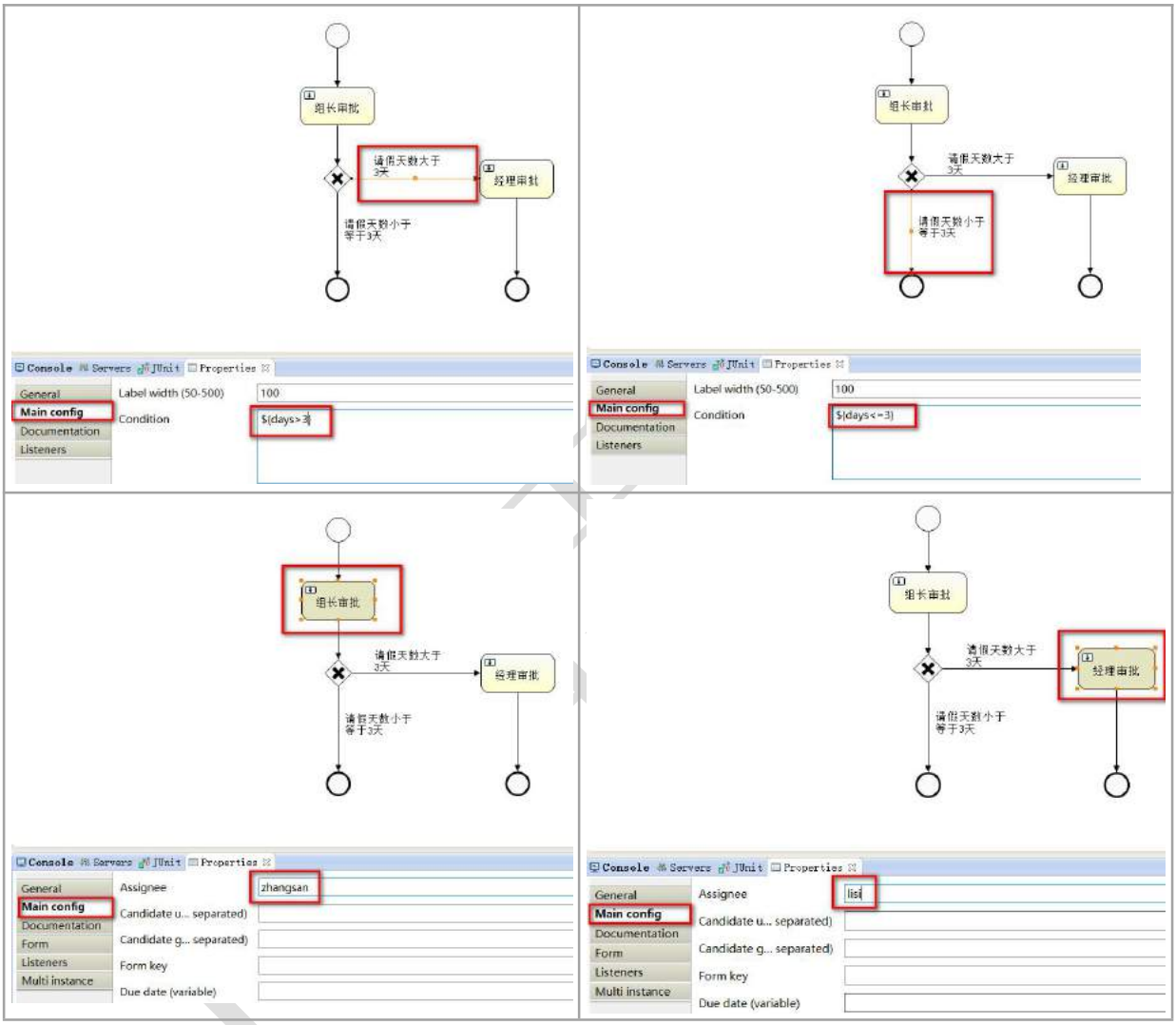
## 4.11 排他网关-决策

### 4.11.1 多个不同的网关



## 4.11.2 重新画流程：网关

### 1.网关条件




### 2.排他网关：测试

```
//网关 - 多个分支同时只能有一个被执行，称为排他网关,也称为决策.
@Test
public void testProcess(){
```

```
    Deployment deploy =
repositoryService.createDeployment().addClasspathResource("MyProcess4.bpmn").deploy();
    System.out.println(deploy);

    //查询流程定义
    ProcessDefinition processDefinition = repositoryService.createProcessDefinitionQuery()
    .processDefinitionKey("myProcess")
    .latestVersion()
    .singleResult();

    //准备流程变量
    Map<String,Object> variables = new HashMap<String,Object>();
    variables.put("days", 3);

    //启动流程，传递流程变量
    ProcessInstance processInstance =
runtimeService.startProcessInstanceById(processDefinition.getId(),variables);
    System.out.println("启动的流程实例："+processInstance.getId());

    List<Task> list = taskService.createTaskQuery().taskAssignee("zhangsan").list();
    for (Task task : list) {
        System.out.println("zhangsan 完成的任务："+task.getName());
        taskService.complete(task.getId());
    }

    HistoricProcessInstance historicProcessInstance = historyService.createHistoricProcessInstanceQuery()
    .processInstanceId(processInstance.getId())
```

```
        .finished()
        .singleResult();

        System.out.println("流程是否完成 = "+(historicProcessInstance!=null));

}
```

### 4.11.3 注意

特殊情况，条件判断分支中不包含 3，但恰恰给了 3 的值，导致两个分支都没有执行。

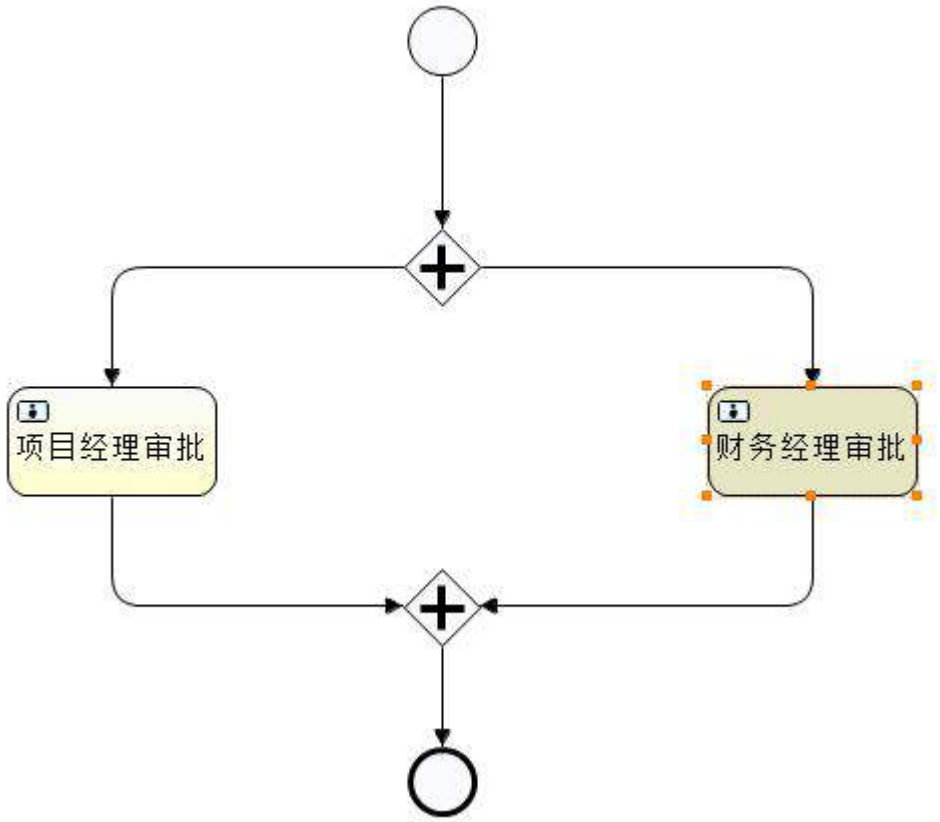
## 4.12 并行网关-会签

### 4.12.1 并行网关流程图

出差，出差天数项目经理要审批，申请出差费用财务部门要需要审批

项目经理审批：zhangsan

财务经理审批：lisi




## 4.12.2 代码

```
//网关 - 多个分支同时都被执行，流程才能往后执行 ，称为并行网关 （在其他流程框架中也称为会签）
@Test
public void testGetway2(){

Deployment deploy =
  repositoryService.createDeployment().addClasspathResource("MyProcess5.bpmn").deploy();
System.out.println(deploy);

//查询流程定义
ProcessDefinition processDefinition = repositoryService
```

```
.createProcessDefinitionQuery()
.processDefinitionKey("myProcess")
.latestVersion()
.singleResult();

//启动流程，传递流程变量
ProcessInstance processInstance = runtimeService.startProcessInstanceById(processDefinition.getId());
System.out.println("启动的流程实例："+processInstance.getId());

List<Task> list1 = taskService().createTaskQuery().taskAssignee("zhangsan").list();
List<Task> list2 = taskService().createTaskQuery().taskAssignee("lisi").list();

System.out.println("zhangsan 未完成的任务数量："+list1.size());
System.out.println("lisi 未完成的任务数量："+list2.size());

for (Task task : list1) {
System.out.println("zhangsan 完成的任务："+task.getName());
taskService().complete(task.getId());
}

System.out.println("zhangsan 完成了任务");

HistoricProcessInstance historicProcessInstance = historyService.createHistoricProcessInstanceQuery()
.processInstanceId(processInstance.getId())
.finished()
.singleResult();
```

```
System.out.println("流程是否完成 = "+(historicProcessInstance!=null));



for (Task task : list2) {
System.out.println("lisi 完成的任务："+task.getName());
processEngine.getTaskService().complete(task.getId());
}

System.out.println("lisi 完成了任务");

historicProcessInstance = historyService()
.createHistoricProcessInstanceQuery()
.processInstanceId(processInstance.getId())
.finished()
.singleResult();

System.out.println("流程是否完成 = "+(historicProcessInstance!=null));

/*
DeploymentEntity[id=1, name=null]
启动的流程实例：5
zhangsan 未完成的任务数量：1
lisi 未完成的任务数量：1
zhangsan 完成的任务：项目经理审批
zhangsan 完成了任务
流程是否完成 = false
lisi 完成的任务：财务经理审批
```

```
lisi 完成了任务
流程是否完成 = true
 */
}
```

## 4.13 包含网关（排他+并行）

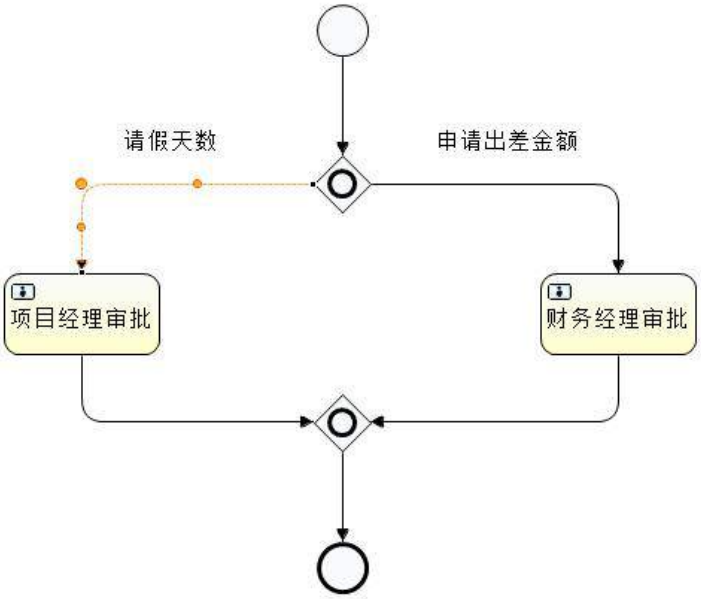
### 4.13.1 流程图

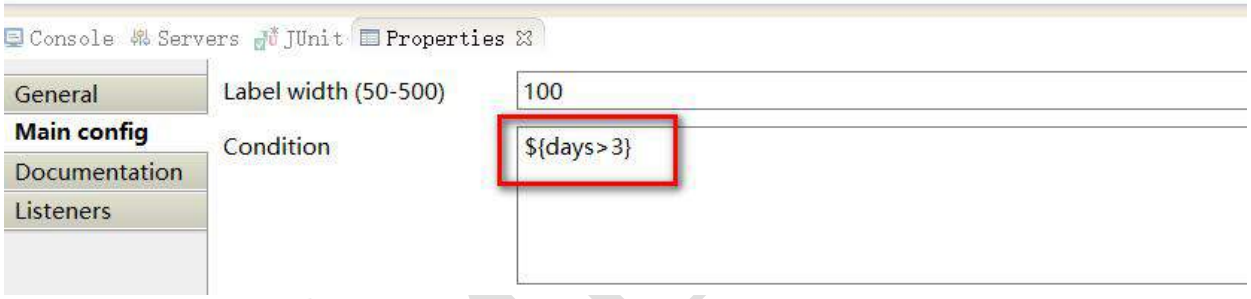
项目经理审批：zhangsan

条件：${days>3}

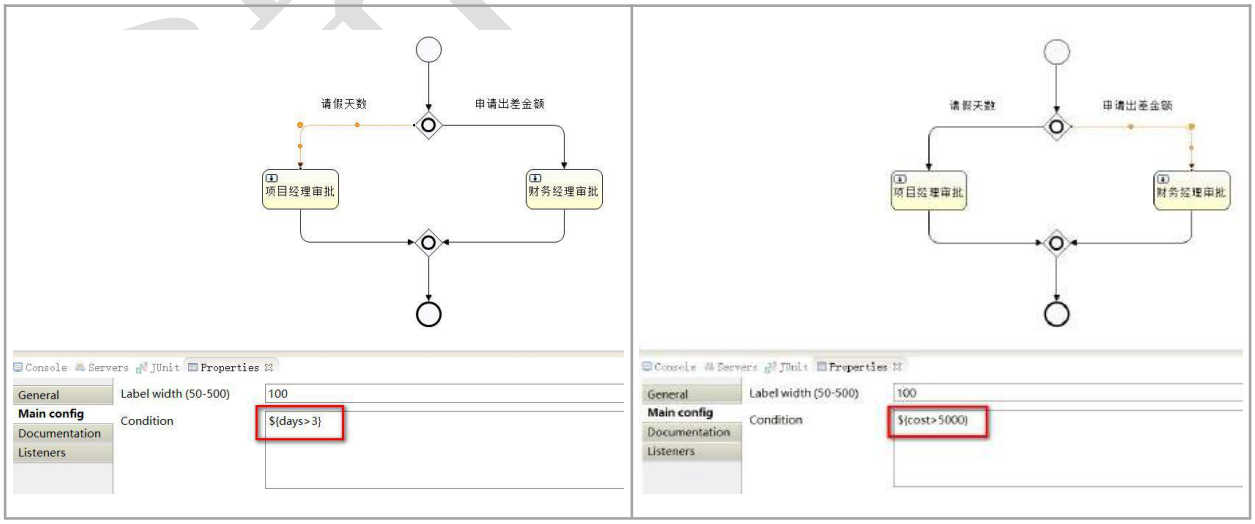
财务经理审批：lisi

条件：${cost>5000}


请假天数
申请出差金额
项目经理审批
财务经理审批


Console
Servers
JUnit
Properties
General
Main config
Documentation
Listeners
Label width (50-500)
100
Condition
${days>3}

## 4.13.2 流程图条件


请假天数
申请出差金额
项目经理审批
财务经理审批
Console
Servers
JUnit
Properties
General
Main config
Documentation
Listeners
Label width (50-500)
100
Condition
${days>3}
${cost>5000}

### 4.13.3 测试代码

```
//网关 - 包含网关 （排他 + 并行）
// 如果当前条件只有一个成立则执行后续流程-相当于排他网关
// 如果当前所有条件成立，则必须所有条件成立的任务完成才会执行后续流程-相当于并行网关
@Test
public void testGetway3(){

Deployment deploy =
repositoryService.createDeployment().addClasspathResource("MyProcess6.bpmn").deploy();
System.out.println(deploy);

//查询流程定义
ProcessDefinition processDefinition = repositoryService
.createProcessDefinitionQuery()
.processDefinitionKey("myProcess")
.latestVersion()
.singleResult();

//准备流程变量
Map<String,Object> variables = new HashMap<String,Object>();
//variables.put("days", 5);//条件：days>3
//variables.put("cost", 100); //条件：cost>5000

/*
DeploymentEntity[id=1, name=null]
启动的流程实例：5
```

```
zhangsan 完成的任务：项目经理审批
流程是否完成 = true
 */


variables.put("days", 5);//条件：days>3
variables.put("cost", 10000); //条件：cost>5000

/*
DeploymentEntity[id=101, name=null]
启动的流程实例：105
zhangsan 完成的任务：项目经理审批
流程是否完成 = false
 */

//启动流程，传递流程变量
ProcessInstance processInstance =
runtimeService.startProcessInstanceById(processDefinition.getId(),variables);
System.out.println("启动的流程实例："+processInstance.getId());

List<Task> list = taskService.createTaskQuery().taskAssignee("zhangsan").list();
for (Task task : list) {
System.out.println("zhangsan 完成的任务："+task.getName());
taskService.complete(task.getId());
}

HistoricProcessInstance historicProcessInstance =historyService.createHistoricProcessInstanceQuery()
```

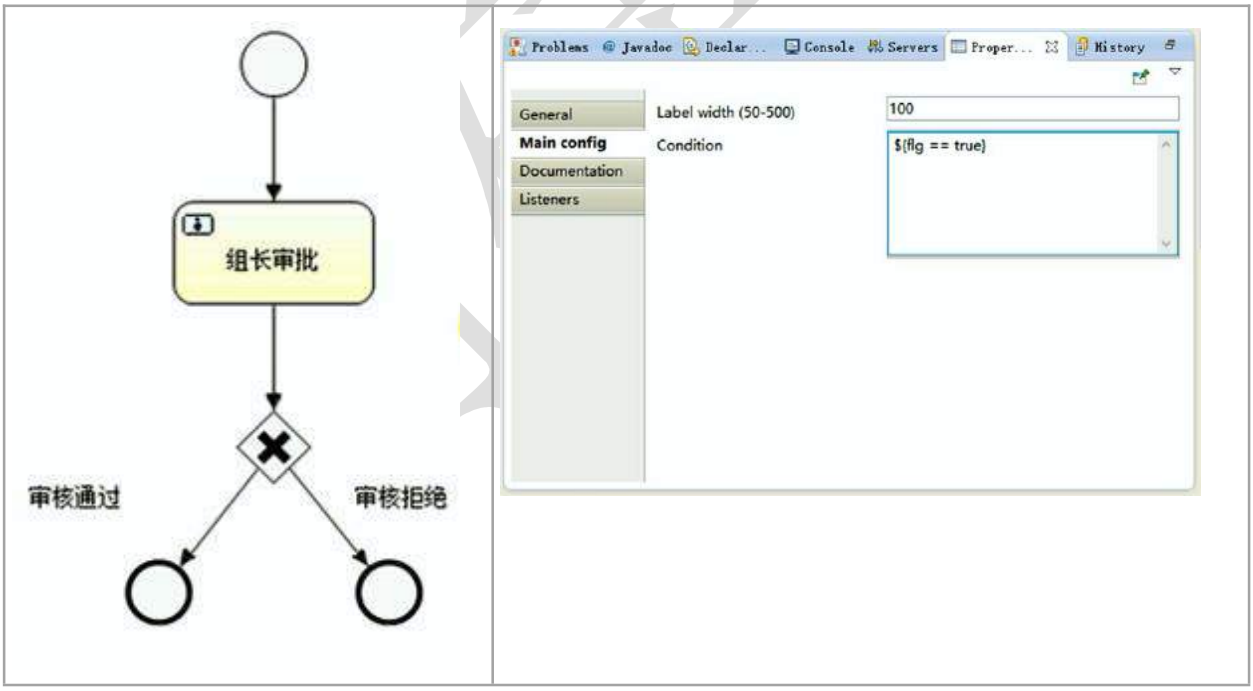
```
.processInstanceId(processInstance.getId())
.finished()
.singleResult();

System.out.println("流程是否完成 ＝ "+(historicProcessInstance!=null));
}
```

## 4.14 流程监听器

### 4.14.1 流程监听器

审核通过:${flag==true}

审核拒绝:${flag==false}




### 4.14.2 流程监听器开发应用

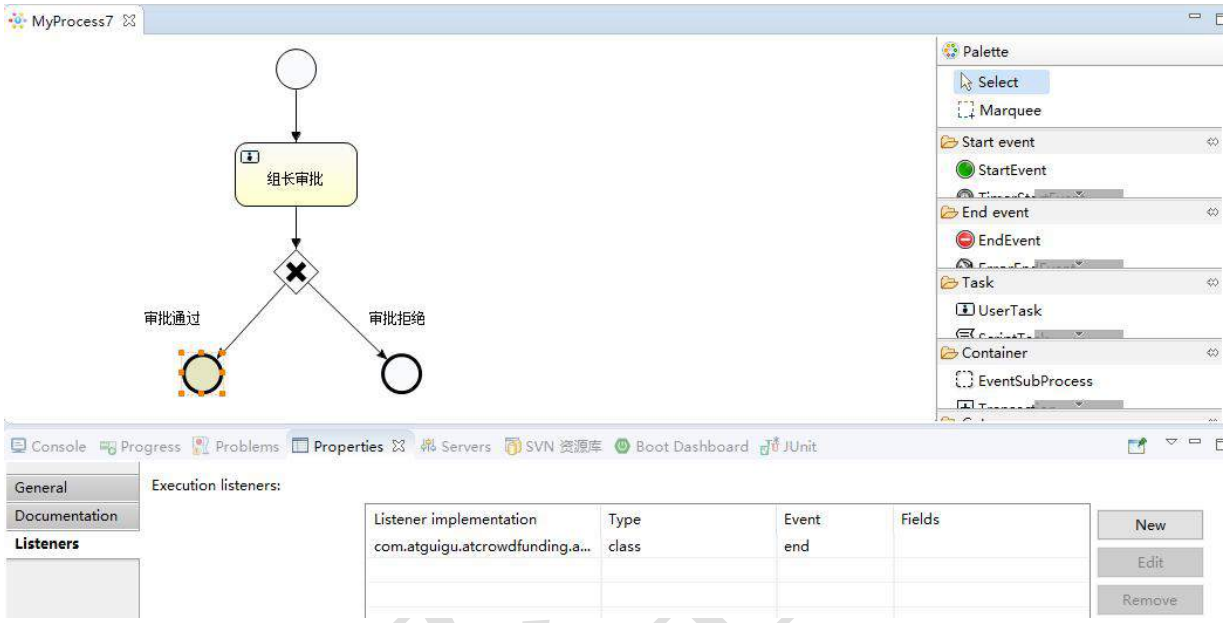
```
package com.atguigu.atcrowdfunding.act.listener;
```

```
import org.activiti.engine.delegate.DelegateExecution;
import org.activiti.engine.delegate.ExecutionListener;

public class NoListener implements ExecutionListener {
    public void notify(DelegateExecution execution) throws Exception {
        System.out.println( "流程拒绝" );
    }
}
```

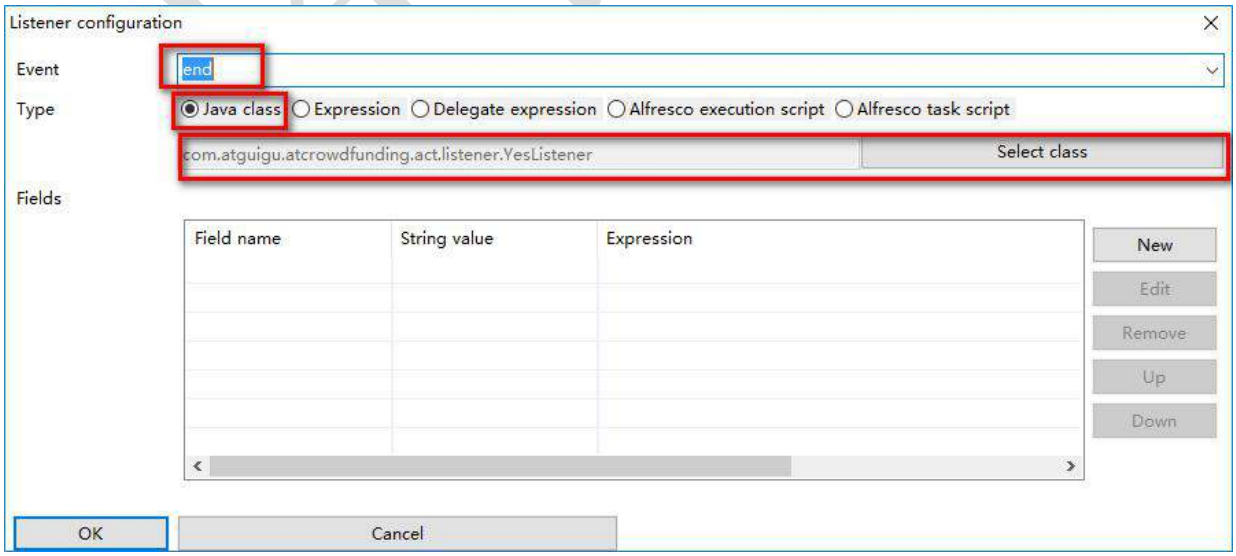
```
package com.atguigu.atcrowdfunding.act.listener;

import org.activiti.engine.delegate.DelegateExecution;
import org.activiti.engine.delegate.ExecutionListener;

public class YesListener implements ExecutionListener {
    public void notify(DelegateExecution execution) throws Exception {
        System.out.println( "流程通过" );
    }
}
```

## 4.14.3 设置监听器

1.选择结束节点,增加监听器




2.选择 Java class ,添加监听器类



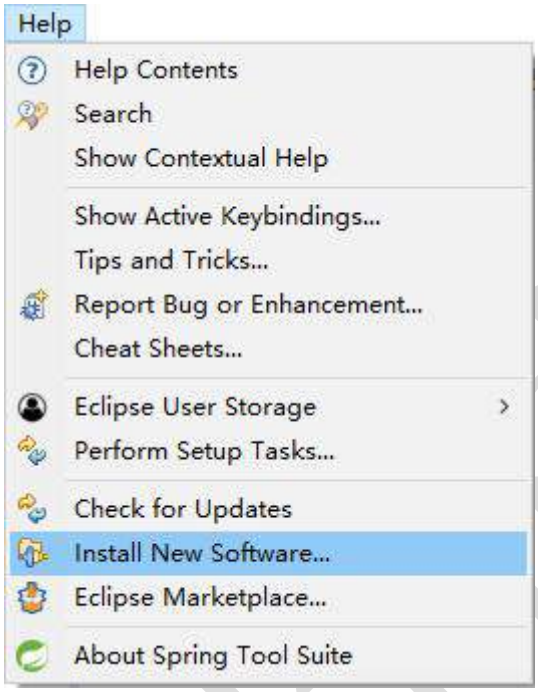
### 4.14.4 测试

```
public static void main(String[] args) {
    //部署流程
    ProcessInstance pi = processEngine.getRuntimeService().startProcessInstanceByKey("myProcess");

    TaskService taskService = processEngine.getTaskService();

    List<Task> tasks = taskService.createTaskQuery().taskAssignee("zhangsan").list();

    System.out.println( "zhangsan 的任务数量 = " + tasks.size() );

    for ( Task task : tasks ) {
        taskService.setVariable(task.getId(), "flg", false);
        taskService.complete(task.getId());
    }

    System.out.println( "流程结束" );
}
```

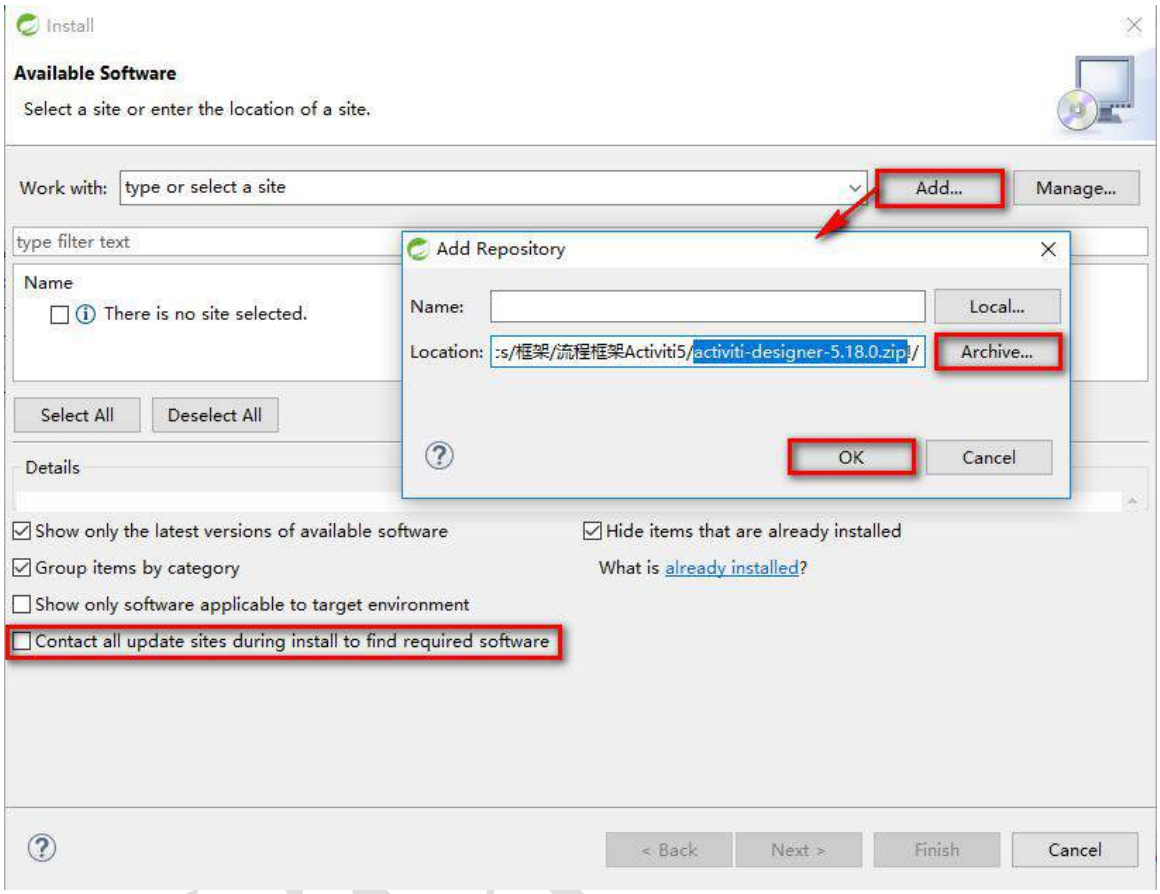
# 附录 1 Eclipse(STS)的 Activiti 插件安装

## 1.1 下载 activiti-designer-5.18.zip

## 1.2 点击 Help->Install New Software…



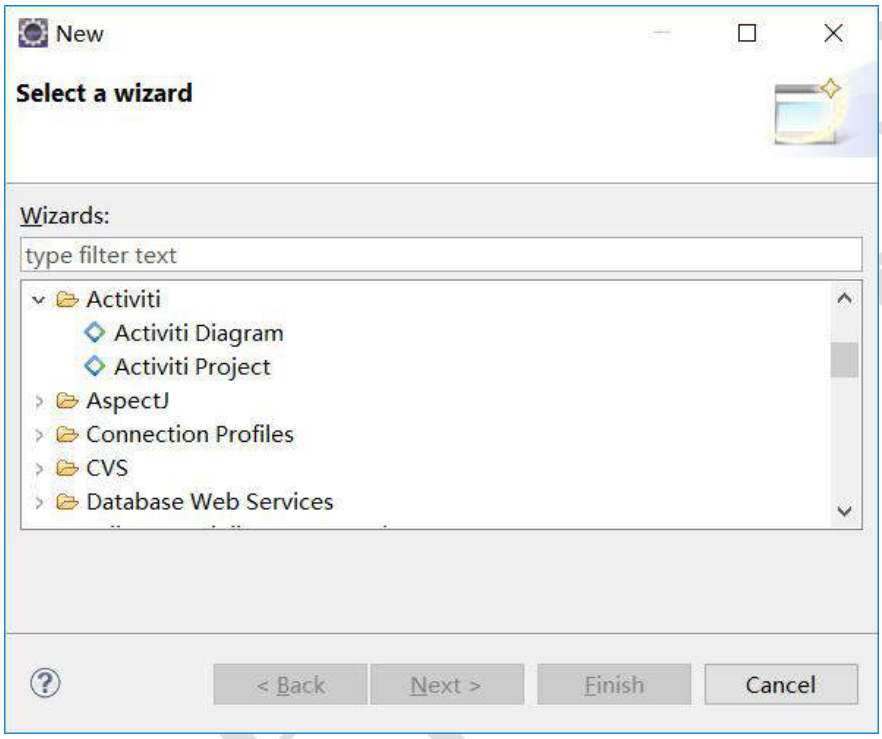
## 1.3 选择插件



## 1.4 下一步…

## 1.5 重启 Eclipse/STS

## 1.6 查看安装是否成功




# 附录 2 Activiti 表结构

# 附录 3 设计模式-单例模式

Singleton 模式主要作用是保证在 Java 应用程序中，一个类 Class 只有一个实例存在。

一般 Singleton 模式通常形式:

第一种形式: 饿汉式

定义一个类，它的构造函数为 private 的，

它有一个 static 的 private 的该类变量，

在类初始化时实例化，

通过一个 public 的 getInstance 方法获取对它的引用,继而调用其中的方法。

```
public class Singleton {
private Singleton(){}
//在自己内部定义自己一个实例，是不是很奇怪？
//注意这是 private 只供内部调用
private static Singleton instance = new Singleton();
//这里提供了一个供外部访问本 class 的静态方法，可以直接访问
public static Singleton getInstance() {
return instance;
}
}
```

第二种形式: 懒汉式

```
public class Singleton {
private static Singleton instance = null;
public static synchronized Singleton getInstance() {
//这个方法比上面有所改进，不用每次都进行生成对象，只是第一次
//使用时生成实例，提高了效率！
if (instance==null)
instance＝new Singleton();
}
return instance;
}
```

其他形式:

定义一个类，它的构造函数为 private 的，所有方法为 static 的。

一般认为第一种形式要更加安全些

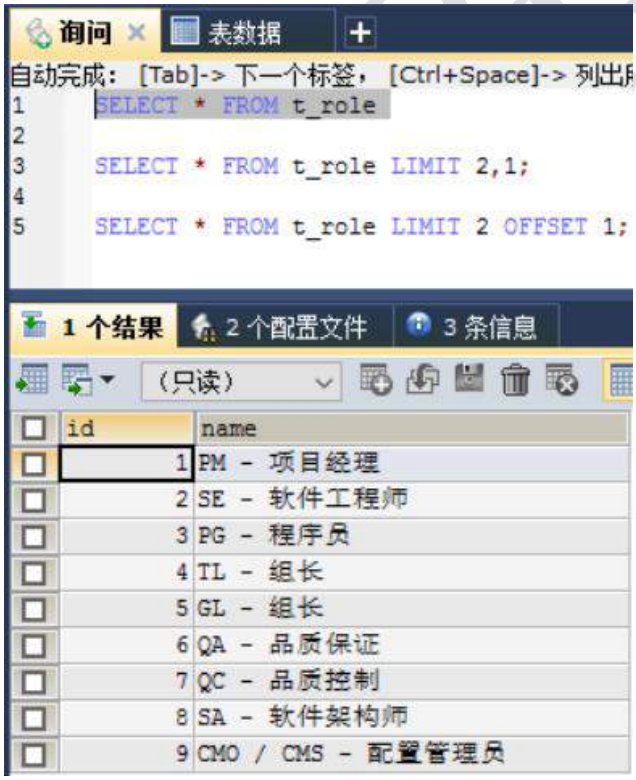
# 附录 4 删除表结构

```
DROP TABLE IF EXISTS act_ge_property          ;

DROP TABLE IF EXISTS act_hi_actinst           ;
DROP TABLE IF EXISTS act_hi_attachment        ;
DROP TABLE IF EXISTS act_hi_comment           ;
DROP TABLE IF EXISTS act_hi_identitylink      ;
DROP TABLE IF EXISTS act_hi_detail            ;
DROP TABLE IF EXISTS act_hi_procinst          ;
DROP TABLE IF EXISTS act_hi_taskinst          ;
DROP TABLE IF EXISTS act_hi_varinst           ;

DROP TABLE IF EXISTS act_id_info              ;
DROP TABLE IF EXISTS act_id_membership        ;
DROP TABLE IF EXISTS act_id_user              ;

DROP TABLE IF EXISTS act_re_model             ;

DROP TABLE IF EXISTS act_ru_event_subscr      ;
DROP TABLE IF EXISTS act_ru_identitylink      ;
DROP TABLE IF EXISTS act_ru_job               ;
DROP TABLE IF EXISTS act_ru_task              ;
DROP TABLE IF EXISTS act_ru_variable          ;

DROP TABLE IF EXISTS act_id_group             ;
```

```
DROP TABLE IF EXISTS act_ru_execution        ;


DROP TABLE IF EXISTS act_evt_log       ;
DROP TABLE IF EXISTS act_procdef_info       ;


DROP TABLE IF EXISTS act_re_procdef           ;
DROP TABLE IF EXISTS act_ge_bytearray         ;
DROP TABLE IF EXISTS act_re_deployment        ;
```

# 附录 5 limit　offset

## 5.1 原数据



## 5.2 select * from t_role limit 2,1;

表示从索引为 2 开始查询,查询数量为 1


## 5.3 select * from t_role limit 2 offset 1;

表示从索引为 1 开始查询,查询数量为 2

# 附录 6 异常

## 6.1 依赖的顺序导致的问题

```
<dependencies>


<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
</dependency>
<!-- 数据库连接池 -->
<dependency>
<groupId>com.alibaba</groupId>
<artifactId>druid</artifactId>
<version>1.0.5</version>
</dependency>
<!--依赖 MyBatis  3.4 版本  先声明者优先 -->
<dependency>
<groupId>org.mybatis.spring.boot</groupId>
<artifactId>mybatis-spring-boot-starter</artifactId>
<version>1.1.1</version>
</dependency>


<!--依赖 MyBatis 3.3 版本 -->
<dependency>
<groupId>org.activiti</groupId>
<artifactId>activiti-spring-boot-starter-basic</artifactId>
```

```
<version>5.21.0</version>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

```
2017-11-16 09:29:33.870 ERROR 10584 --- [ main]
o.s.test.context.TestContextManager : Caught exception while allowing TestExecutionListener
[org.springframework.boot.test.autoconfigure.SpringBootDependencyInjectionTestExecutionListener@b7dd
107] to prepare test instance
[com.atguigu.atcrowdfunding.act.AtcrowdfundingBootActApplicationTests@646007f4]

java.lang.NoClassDefFoundError: org/apache/ibatis/annotations/Mapper
at
org.mybatis.spring.boot.autoconfigure.MybatisAutoConfiguration$AutoConfiguredMapperScannerRegistrar.
registerBeanDefinitions(MybatisAutoConfiguration.java:175)
~[mybatis-spring-boot-autoconfigure-1.1.1.jar:1.1.1]
…
```

## 6.2 类路径下缺少 processes 文件夹

```
Caused by: org.springframework.beans.BeanInstantiationException: Failed to instantiate
[org.activiti.spring.SpringProcessEngineConfiguration]: Factory method 'springProcessEngineConfiguration'
threw exception; nested exception is java.io.FileNotFoundException: class path resource [processes/] cannot
```

be resolved to URL because it does not exist

…

# 总结

## 1.流程框架 Activiti5

Activiti5 框架源自于 JBPM4 流程框架，

集成到项目中时，会自动创建框架所需的 23 张表，5 种不同的分类

框架底层使用的是 Mybatis 作为持久层，但是对于开发人员来说，依然采用面向对象的方式操作数据库

ProcessEngine 对象为框架的核心对象，可以创建 7 种不同的 service 服务对象，完成不同的业务操作及操作不同的业务表

流程框架的主要应用目的是可以动态扩展流程中的任务及动态增加业务处理

## 2.流程的相关概念

### 2.1 流程定义

部署流程后，数据库中会有 3 张表的数据发生变化，用于存储当前部署及流程定义相关的数据，图形，文件内容

### 2.2 流程实例

流程实例其实就是流程定义的具体应用，通过 startXXXXX 方法启动流程实例，

启动流程后，会在 RU 表及 HI 表中增加相应的数据

## 2.3 任务

所谓的任务其实就是流程中的工作步骤，可以将任务委派给某个人完成，也可以委派给某个小组后再分配给个人完成，当流程中所有的任务完成后，流程结束。

## 2.4 开始，结束

流程的边界，由框架自动完成，不需要人工参与

## 2.5 变量

将流程中固定不变的内容通过变量来表示，可以让流程更具备通用性

在流程定义图中使用 EL 表达式（${tl}）增加变量，在启动流程时传递变量

Map<String, Object> varMap = new HashMap<String, Object>();

varMap.put(“tl”, “zhangsan”);

## 2.6 网关

网关就是流程中的分支判断，可以在流程定义图中增加判断条件实现

框架中的网关主要分为 3 种:

**排他网关**：多个逻辑分支同时只能执行一个，分支执行完毕，流程结束

**并行网关**：多个逻辑分支同时执行，一个分支执行完毕后，流程没有结束，需要等待其他分支的执行，所有的分支执行完毕后，流程结束

**包含网关**：多个逻辑分支如果有一个条件成立，那么等同于排他网关，如果有多个条件成立，那么等同于并行网关。

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

1. 实名认证申请-邮件任务
2. James 邮件服务器
3. foxmail 邮件客户端
4. 邮件任务-Javamail 发送邮件
5. 流程管理-流程定义-定义实名认证审核流程图

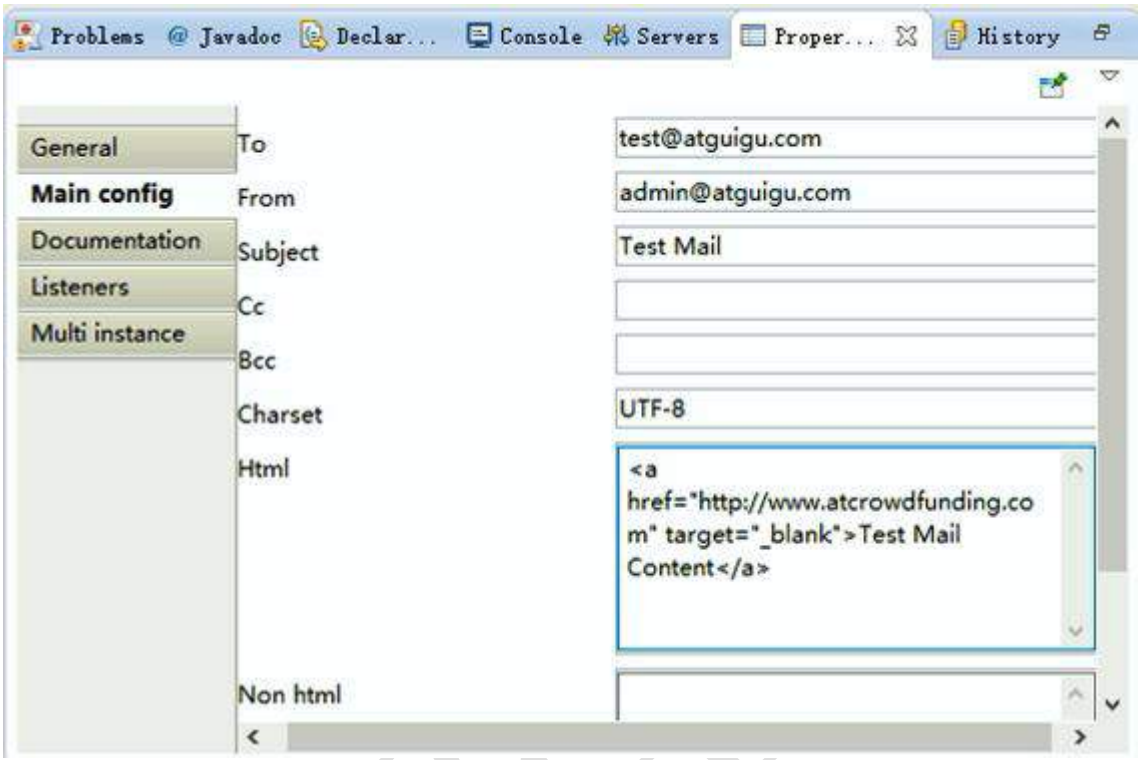
# 第 1 章 实名认证申请-邮件任务

注:通过邮箱模拟银行卡确认申请.

## 1.1 演示发送邮件流程

## 1.2 流程

## 1.3 系统自动发送邮件相关设置




## 1.4 流程设置信息介绍

- To:设置接收邮件的地址
- From:设置发送邮件的地址
- Subject:设置邮件标题
- cc（carbon copy）:抄送
- bcc（blind carbon copy）:密送
- Charset:设置字符编码
- Html:设置邮件内容,带 HTML 元素
- Non Html:设置邮件内容,不带 HTML 元素

## 1.5 注意

- 采用 Activiti 给邮件服务器发送邮件,默认访问是 localhost 本机的邮件服务器.

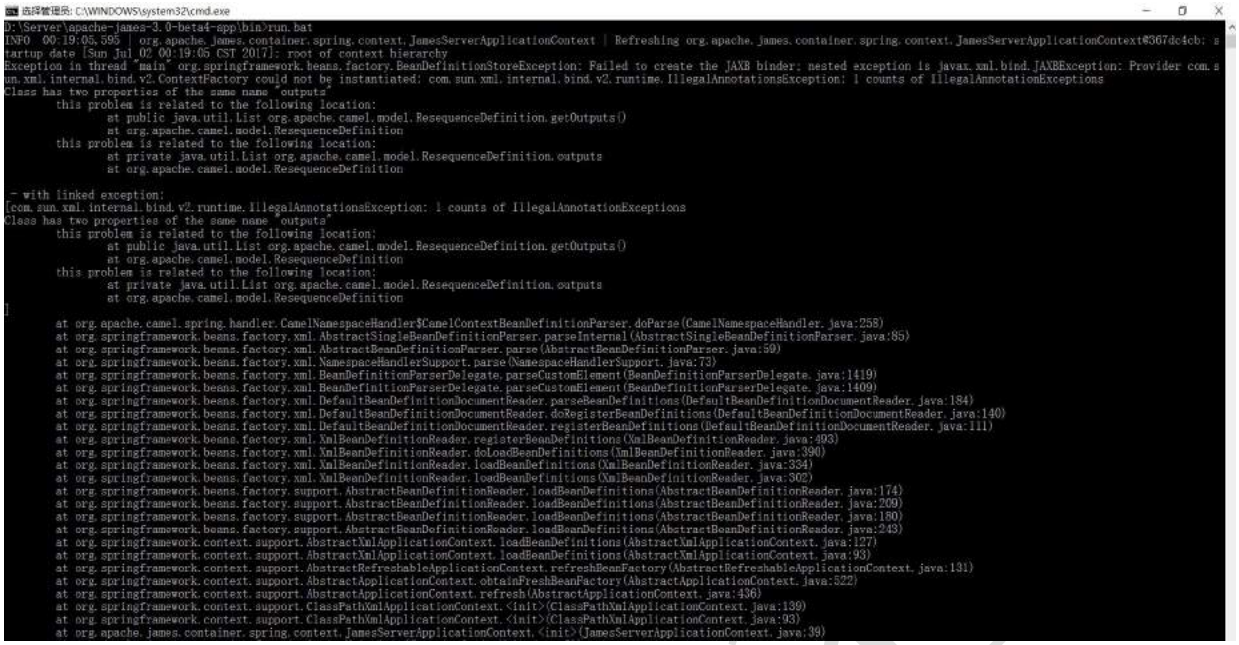

## 1.6 搭建邮件服务器,完成系统自动发送邮件的准备工作

# 第 2 章 配置邮件服务器(默认 Derby 数据库)

## 2.1 安装邮件服务器

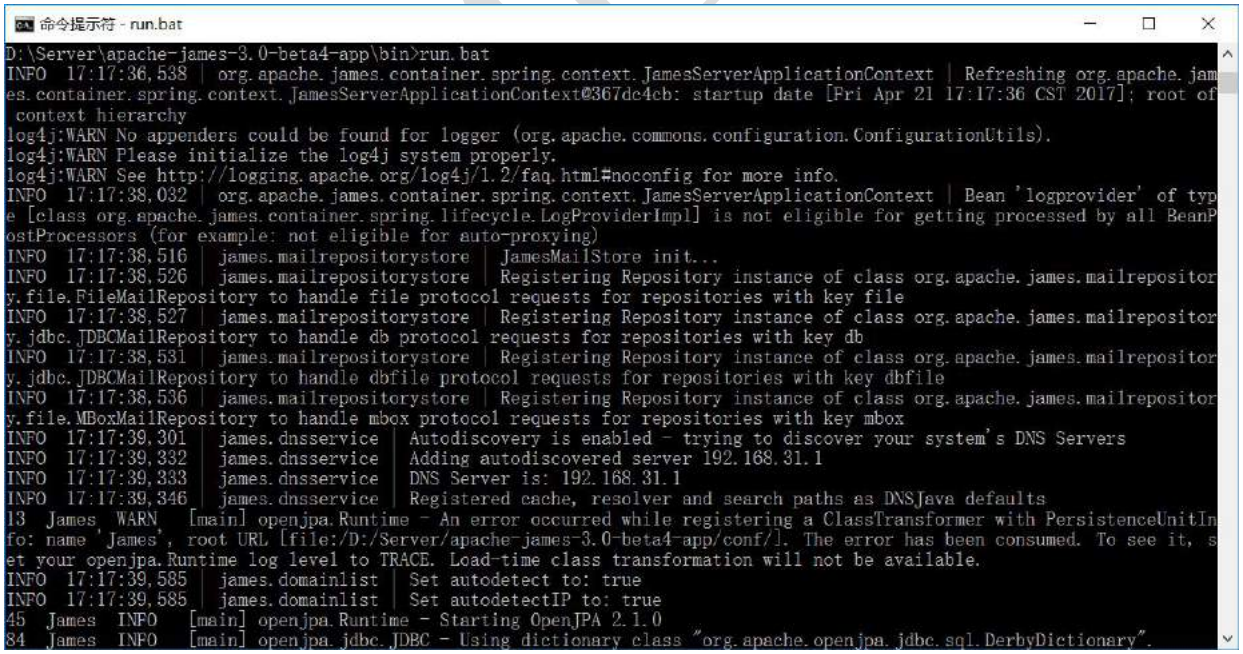
- 将 apache-james-3.0-beta4-app.zip 安装到非中文，非空格目录下“D:\Server\apache-james-3.0-beta4-app”
- 将“jaxb-impl-2.1.3.jar”拷贝到“.\apache-james-3.0-beta4-app\conf\lib\jaxb-impl-2.1.3.jar”
- JDK1.6 不需要拷贝这个 jar 包；JDK1.7,1.8 需要拷贝这个 jar 包


- 如果不拷贝 jaxb-impl-2.1.3.jar 包会报错

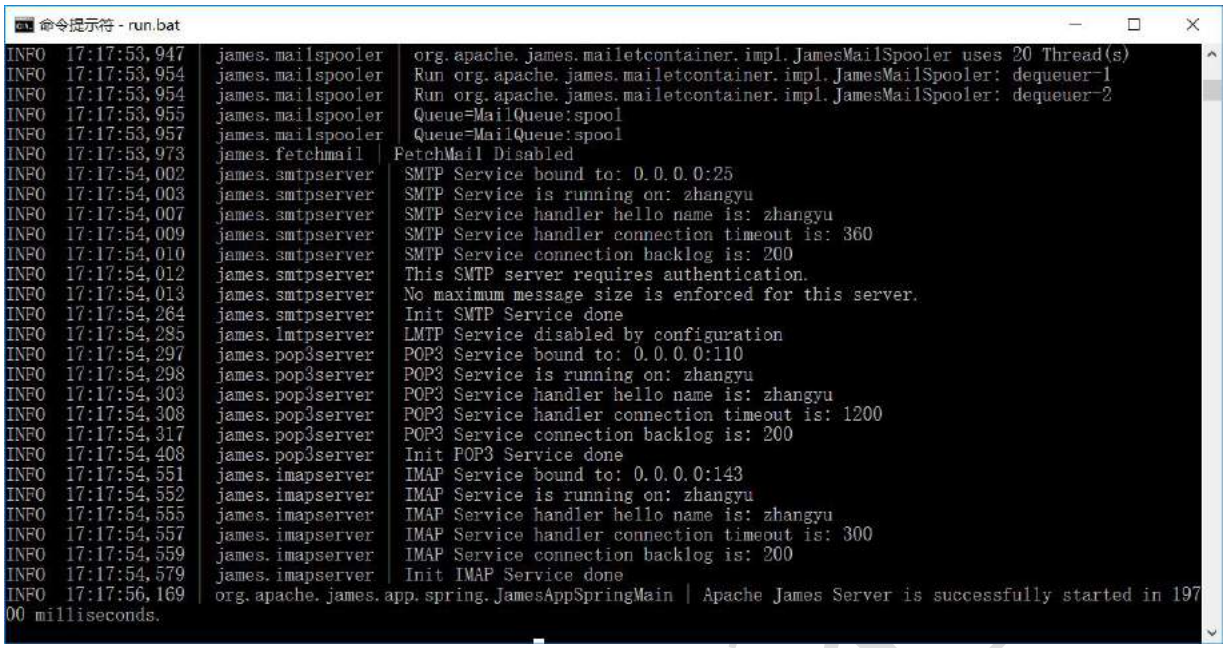



## 2.2 启动邮件服务器

- 端口默认为 9999
- 通过 DOS 窗口，进入到 james 服务器的 bin 文件目录， 然后执行下面的 DOS 命令
- 启动




- 启动成功




## 2.3 James 邮件服务器

### 2.3.1 创建邮箱的域名

james-cli.bat -h localhost -p 9999 adddomain atguigu.com

### 2.3.2 创建邮箱用户

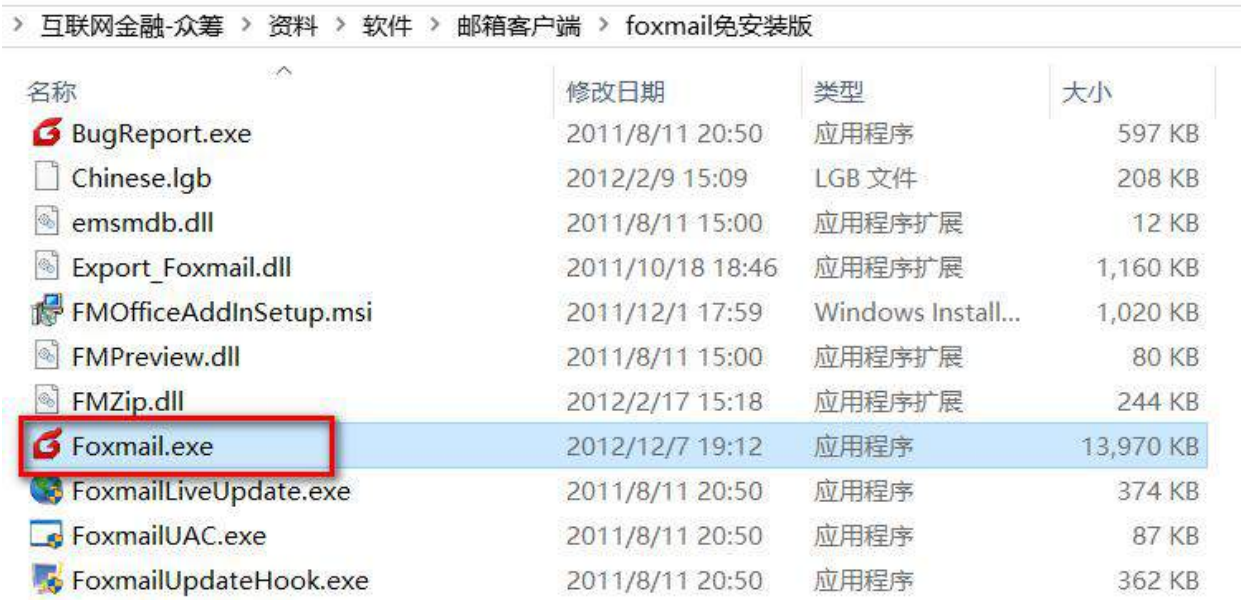
james-cli.bat -h localhost -p 9999 adduser test@atguigu.com test
james-cli.bat -h localhost -p 9999 adduser admin@atguigu.com admin

# 第 3 章 foxmail 邮件客户端

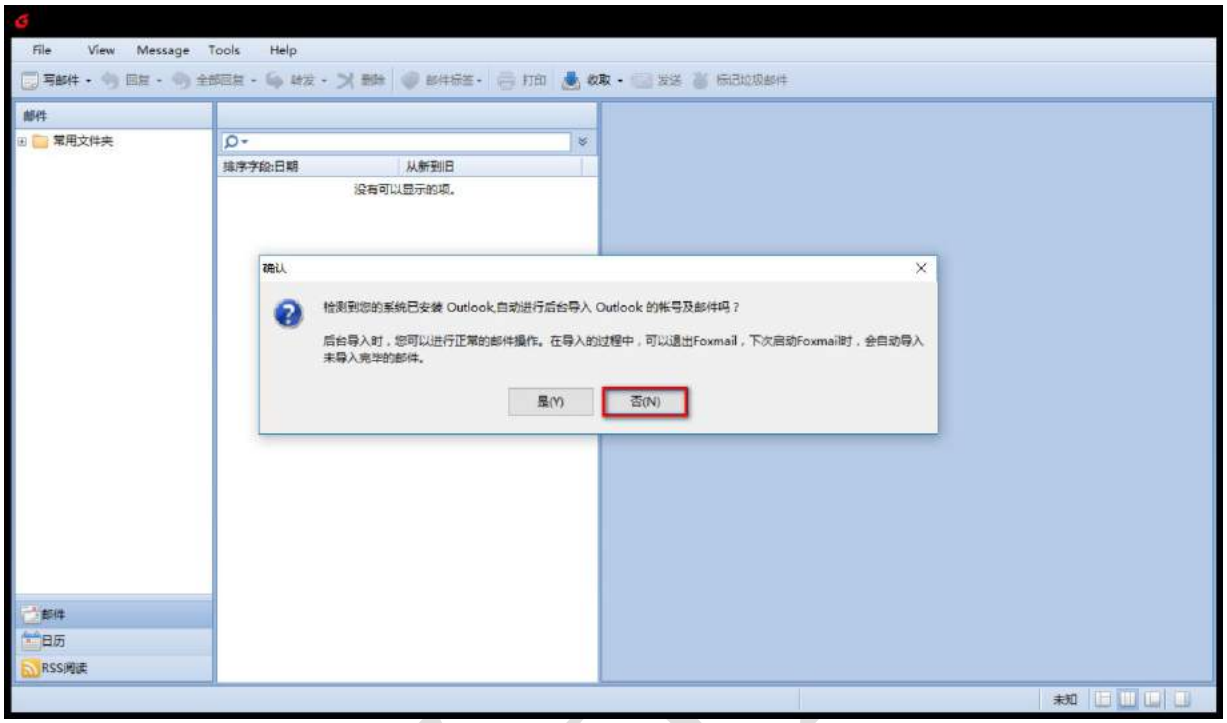
## 3.1 解压

15 年前，张小龙是中国 Top10 的程序员。
他一个人写代码，完成了 Foxmail 的头三个版本。
Foxmail 如此受到欢迎，以至于他不得不外放了语言包，让各种忠实粉丝翻译为十几种语言。
直至今日，Foxmail 被收购，QQmail 推出，Foxmail 的企业用户依然有数百万。
QQ 邮箱 漂流瓶 微信

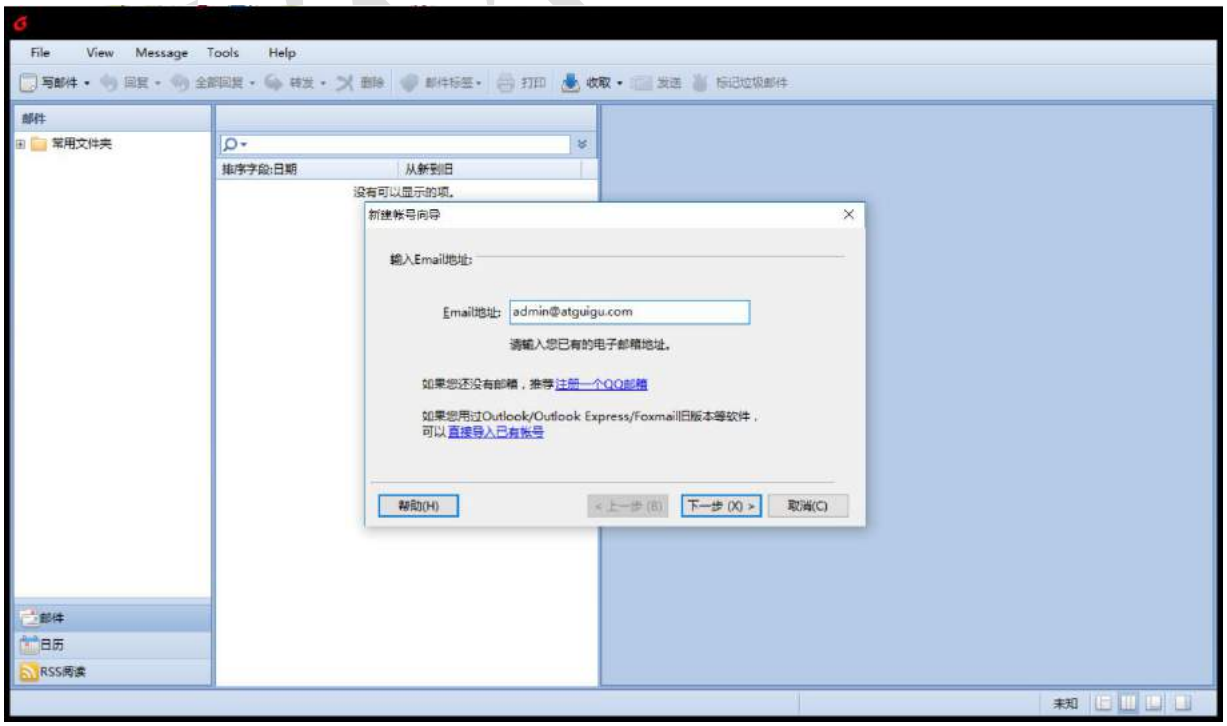
## 3.2 启动 foxmail 邮件客户端



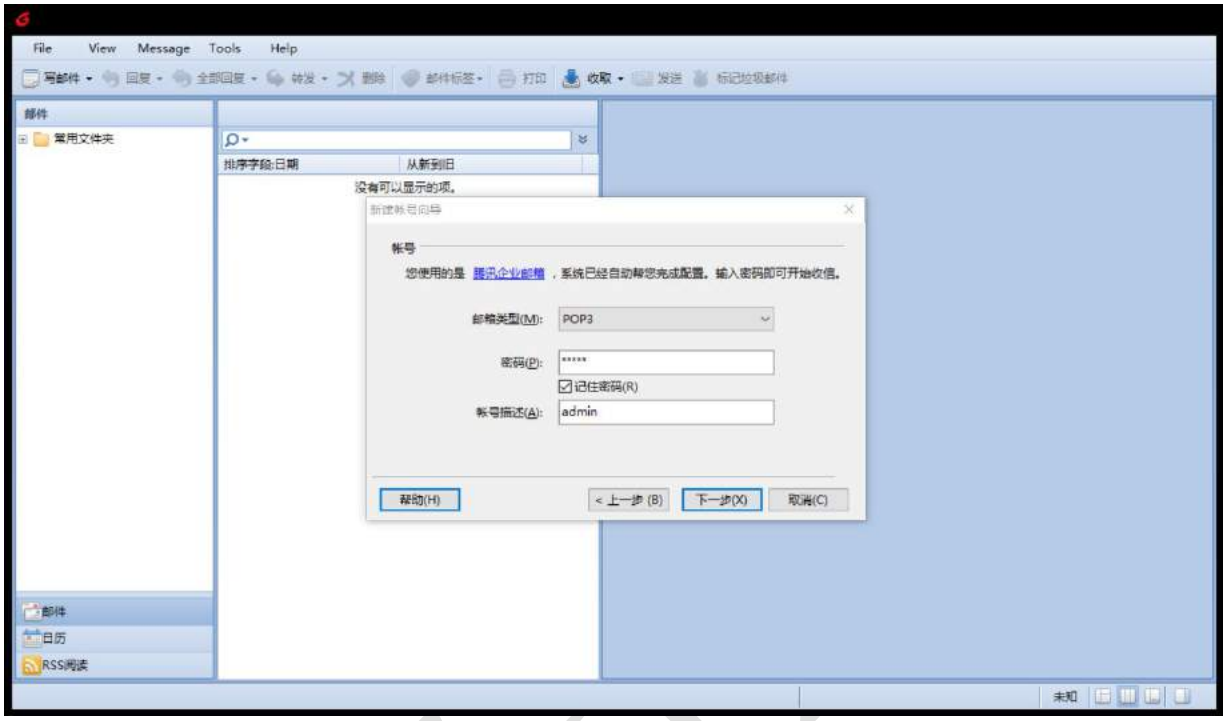
### 3.2.1 是否导入 Outlook，选择【否】




### 3.2.2 设置账号



## 3.2.3 设置密码




## 3.2.4 设置服务器地址及端口

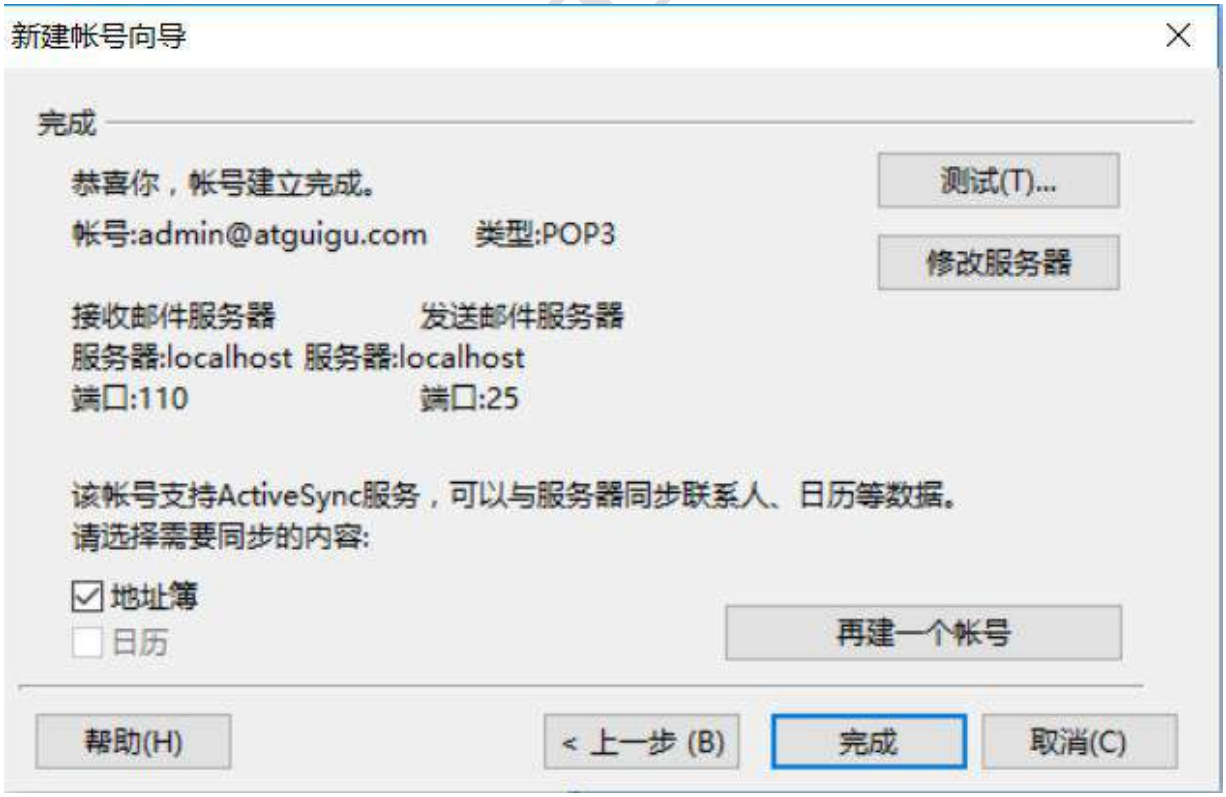
- 设置接收服务器类型:POP3

POP3，全名为“Post Office Protocol - Version 3”，即“邮局协议版本 3”。是 TCP/IP 协议族中的一员，由 RFC1939 定义。本协议主要用于支持使用客户端远程管理在服务器上的电子邮件。提供了 SSL 加密的 POP3 协议被称为 POP3S。

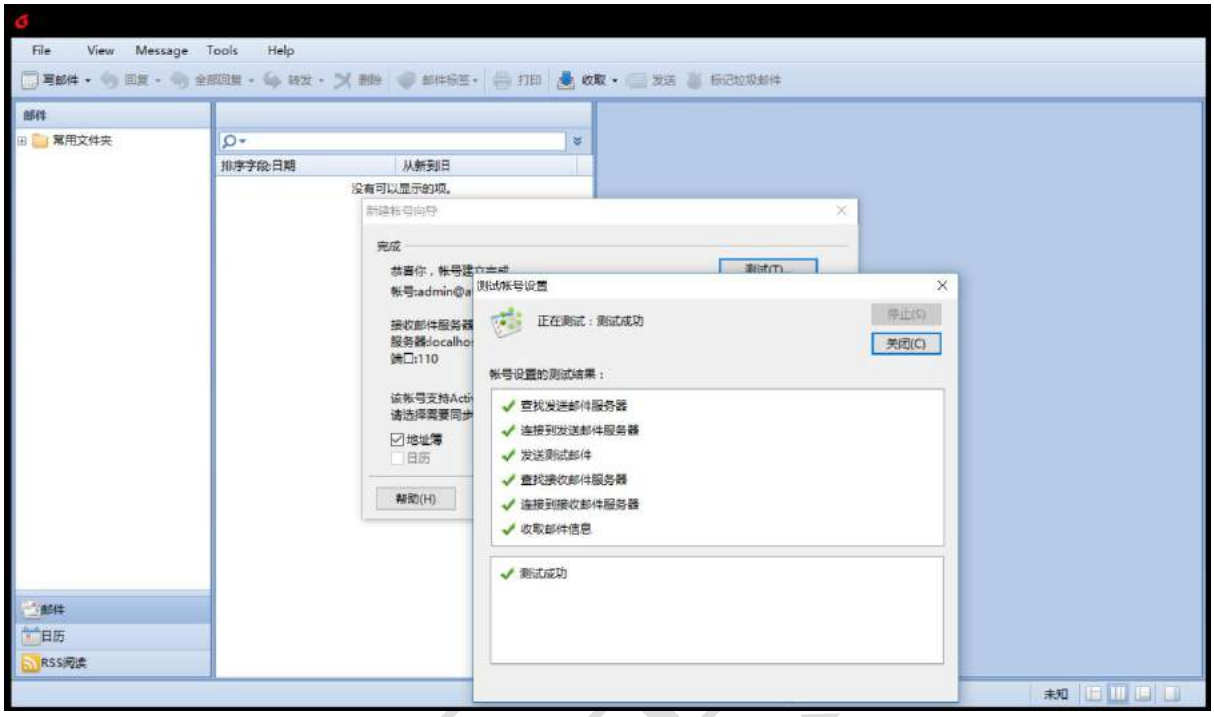
- 设置接收邮件服务器(R)的:localhost
- 端口:110,不使用 SSL 来连接服务器
- 设置发送邮件服务器(S):localhost
- 端口:25,不使用 SSL 来连接服务器


SSL(Secure Sockets Layer 安全套接层),及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全及数据完整性的一种安全协议。TLS 与 SSL 在传输层对网络连接进行加密。

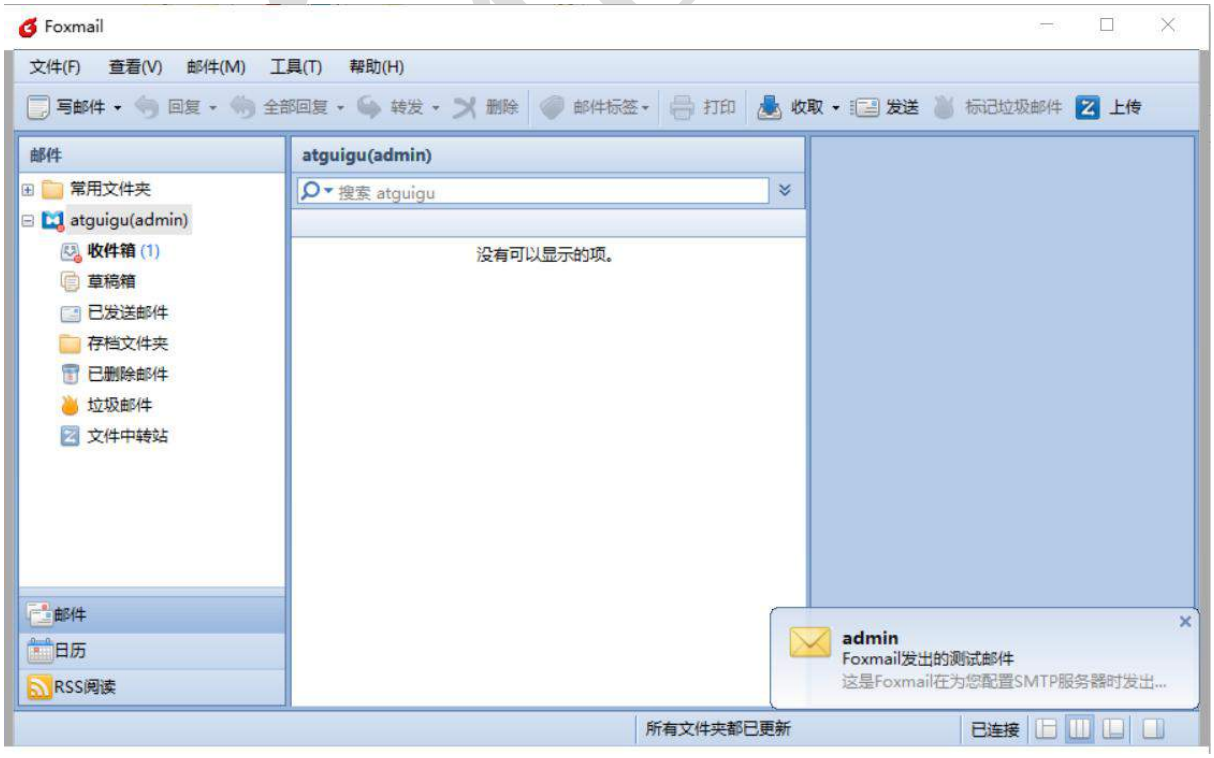
### 3.2.5 创建账号的信息,点击测试按钮进行测试



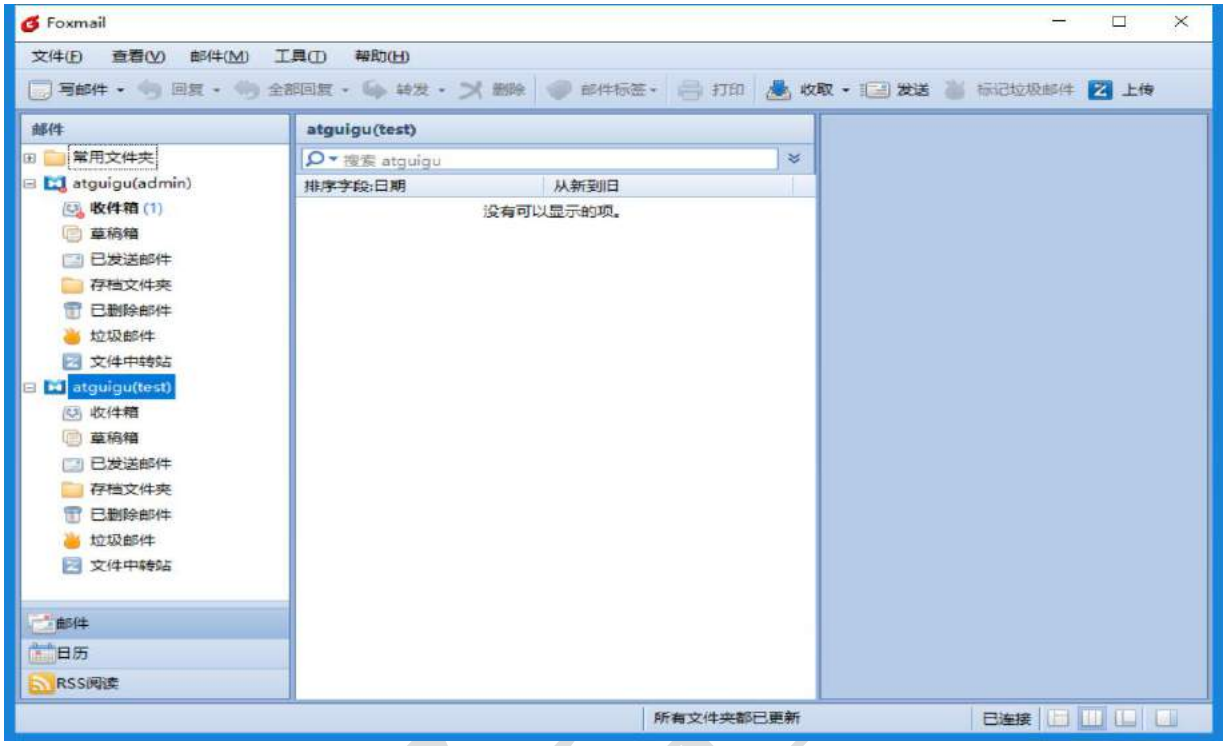
## 3.2.6 测试



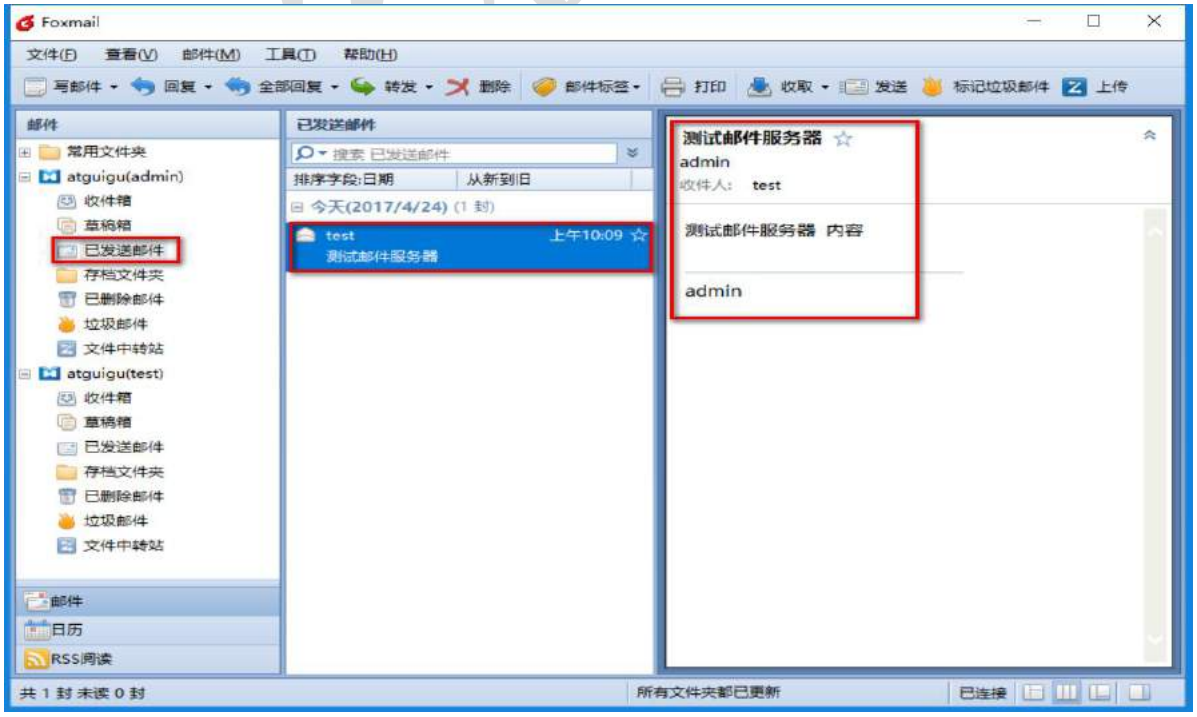

## 3.2.7 配置后的账号



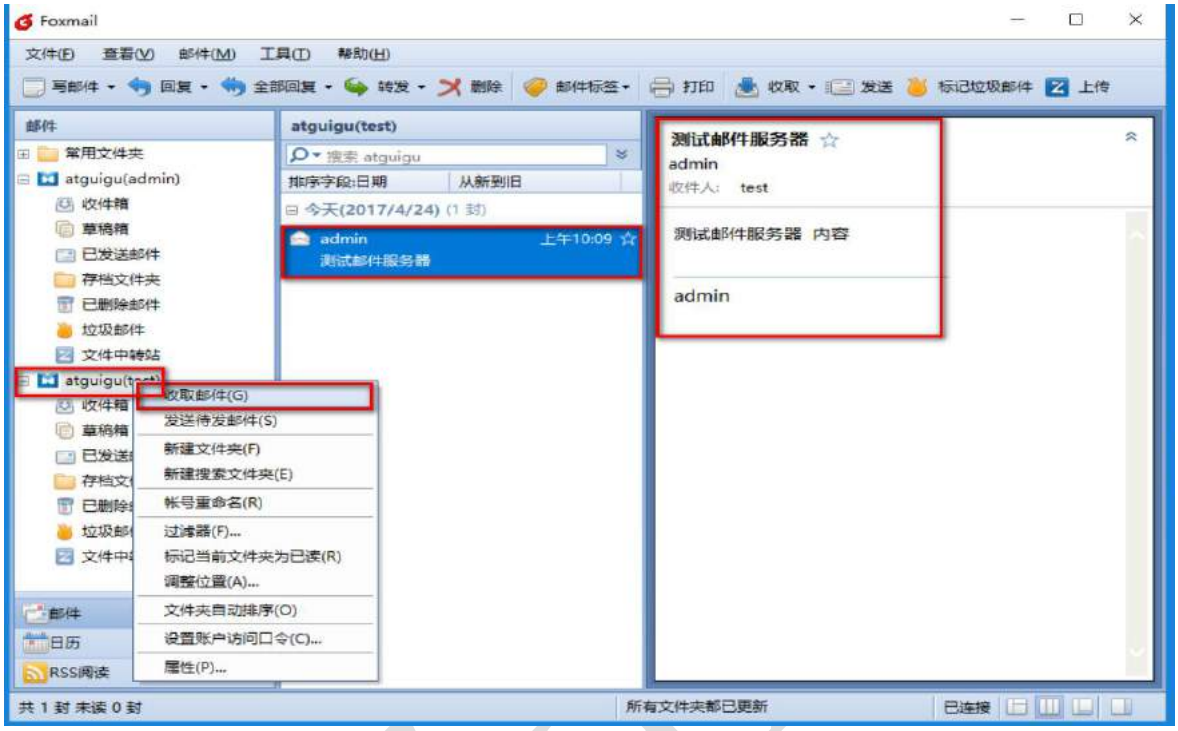
## 3.2.8 配置其他账号




## 3.2.9 发邮件



### 3.2.10 收邮件




# 第 4 章 配置邮件服务器(MySQL)

## 4.1 服务器默认 Derby 数据库

修改 apache-james-3.0-beta4\conf\james-database-template.properties 文件
文件名变为 james-database.properties

```
database.driverClassName=com.mysql.jdbc.Driver
database.url=jdbc:mysql://127.0.0.1:3306/email
database.username=root
database.password=root
vendorAdapter.database=MYSQL
```

将数据库驱动程序 jar 包拷贝到 james/conf/lib 文件夹中

## 4.2 james-database.properties

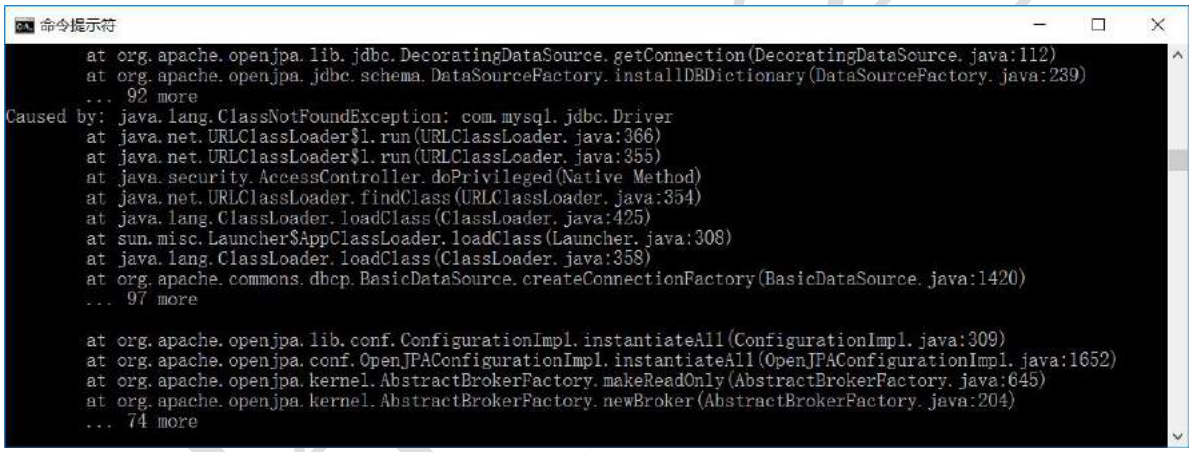
手动创建 email 数据库

james-database.properties

```
22
23  # Use derby as default
24  database.driverClassName=com.mysql.jdbc.Driver
25  database.url=jdbc:mysql://localhost:3306/email
26  database.username=root
27  database.password=root
28
29  # Supported adapters are:
30  # DB2, DERBY, H2, HSQL, INFORMIX, MYSQL, ORACLE,
31  vendorAdapter.database=MYSQL
```

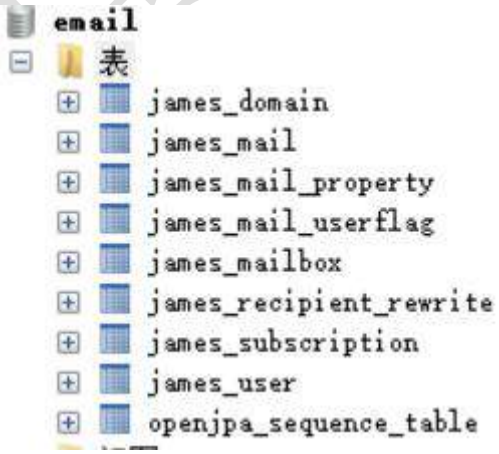
2.启动邮件服务器

①启动，报错




②拷贝 MySQL 驱动 jar 包 mysql-connector-java-5.1.8.jar 到 D:\Server\apache-james-3.0-beta4-app\conf\lib 目录下

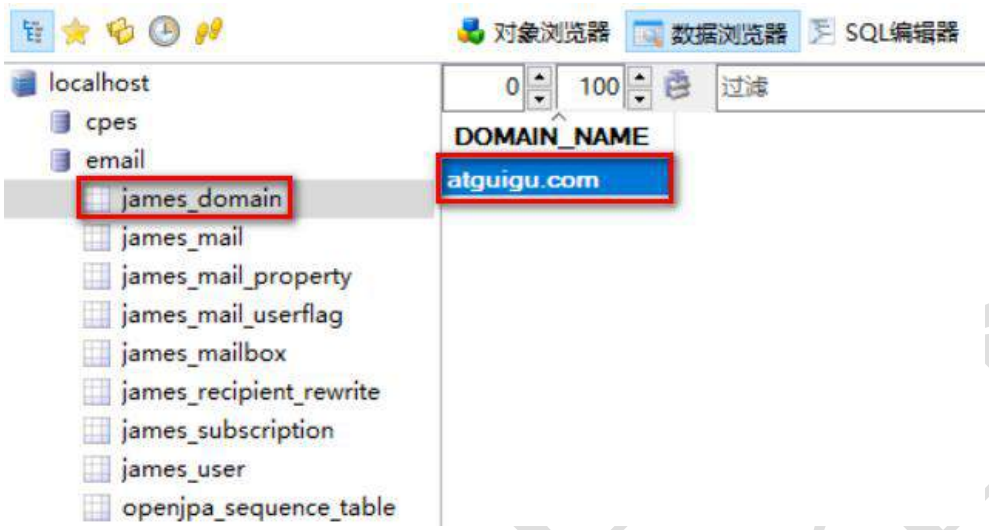
③再次启动服务器，成功创建表




④创建域名

```
D:\Server\apache-james-3.0-beta4-app\bin> james-cli.bat -h localhost -p 9999 adddomain atguigu.com
adddomain command executed sucessfully in 334 ms.
```

james-cli.bat -h localhost -p 9999 adddomain atguigu.com

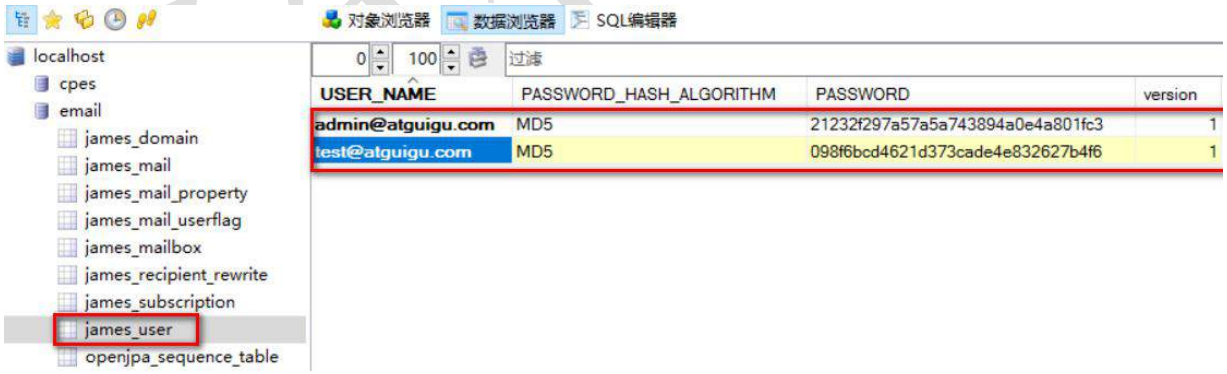



⑤创建邮箱用户

james-cli.bat -h localhost -p 9999 adduser test@atguigu.com test

james-cli.bat -h localhost -p 9999 adduser admin@atguigu.com admin

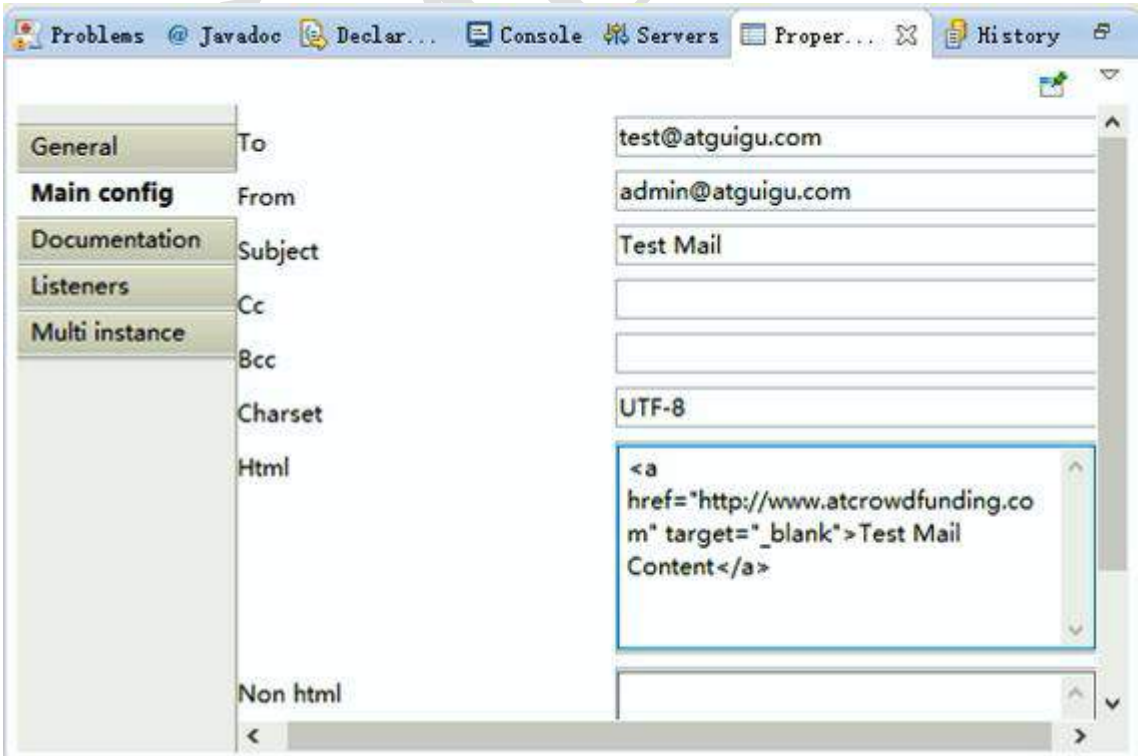
```
D:\Server\apache-james-3.0-beta4-app\bin>james-cli.bat -h localhost -p 9999 adduser test@atguigu.com test
adduser command executed sucessfully in 181 ms.

D:\Server\apache-james-3.0-beta4-app\bin>james-cli.bat -h localhost -p 9999 adduser admin@atguigu.com admin
adduser command executed sucessfully in 98 ms.
```



# 第 5 章 完成自动发送邮件功能

## 5.1 流程

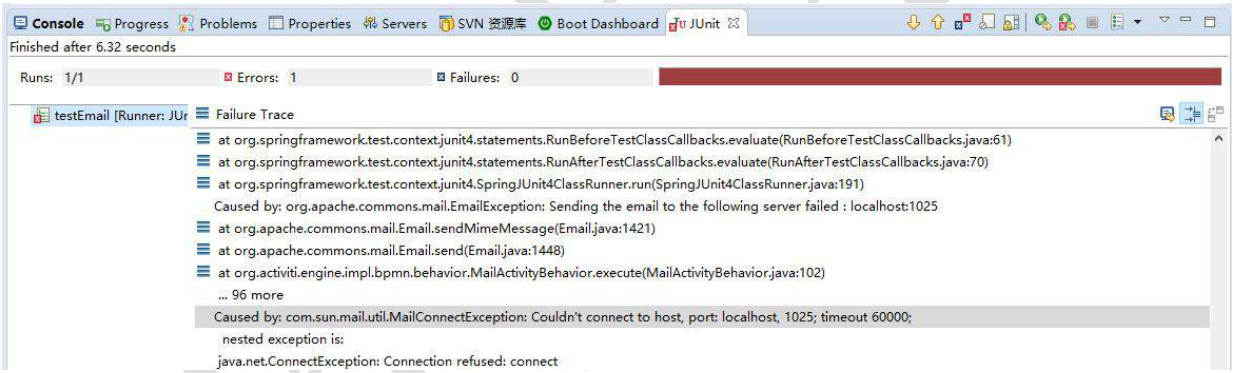

## 5.2 系统自动发送邮件相关设置



## 5.3 部署,启动流程实例

```
@Test
public void sendMail() {
    repositoryService.createDeployment().addClasspathResource("MyProcess8.bpmn").deploy();
    // 获取流程定义 ID
    ProcessDefinition pd =    repositoryService
                    .createProcessDefinitionQuery().processDefinitionKey("myProcess")
                    .latestVersion().singleResult();
    // 启动
    ProcessInstance pi = runtimeService.startProcessInstanceById(pd.getId());
}
```

## 5.4 如果报端口 1025 连接超时错误,需要设置邮件端口.




```
package com.atguigu.atcrowdfunding.member;

import org.activiti.spring.SpringProcessEngineConfiguration;
import org.activiti.spring.boot.ProcessEngineConfigurationConfigurer;
import org.springframework.context.annotation.Configuration;

@Configuration
public class ActivitiConfig implements ProcessEngineConfigurationConfigurer {
    @Override
    public void configure(SpringProcessEngineConfiguration processEngineConfiguration) {
        processEngineConfiguration.setActivityFontName("宋体");
        processEngineConfiguration.setLabelFontName("宋体");
        processEngineConfiguration.setMailServerPort(25);
    }
}
```

# 项目课程系列之 Atcrowdfunding

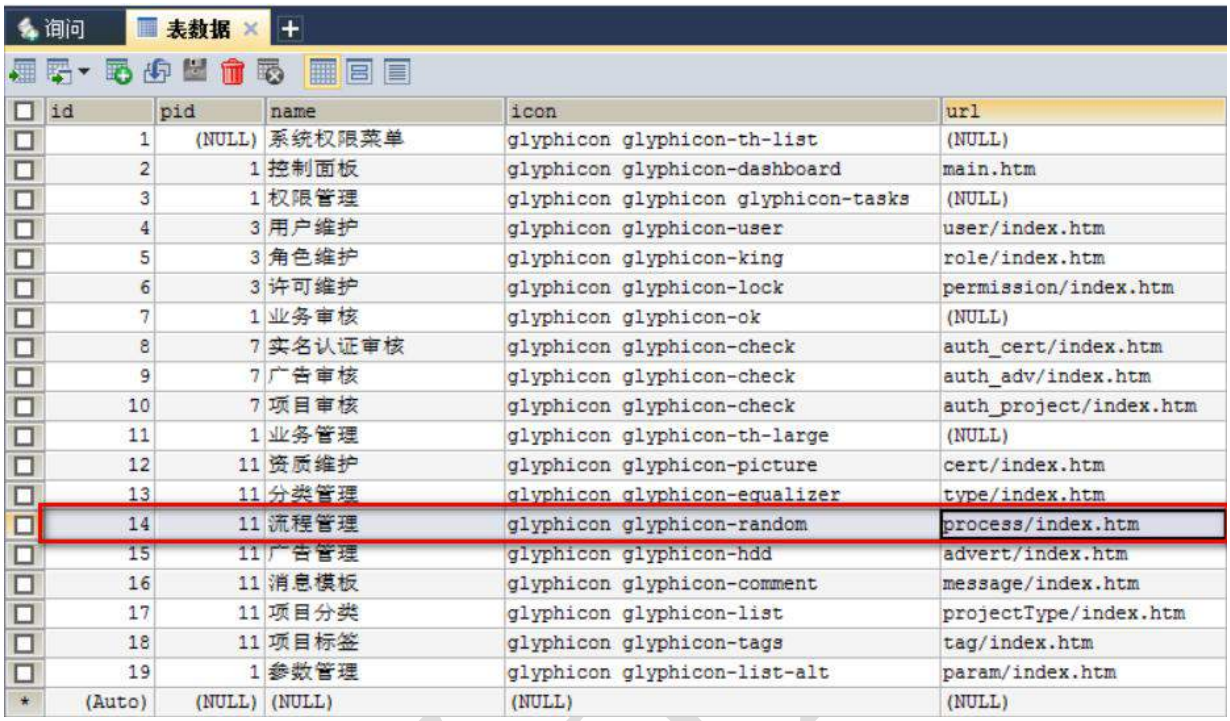
尚硅谷 Java 研究院

版本：V1.0

# 第 1 章 流程管理-流程定义查询

## 1.1 跳转主页面

### 1.1.1 增加链接



| id | pid | name | icon | url |
|---|---|---|---|---|
| 1 | (NULL) | 系统权限菜单 | glyphicon glyphicon-th-list | (NULL) |
| 2 | 1 | 控制面板 | glyphicon glyphicon-dashboard | main.htm |
| 3 | 1 | 权限管理 | glyphicon glyphicon glyphicon-tasks | (NULL) |
| 4 | 3 | 用户维护 | glyphicon glyphicon-user | user/index.htm |
| 5 | 3 | 角色维护 | glyphicon glyphicon-king | role/index.htm |
| 6 | 3 | 许可维护 | glyphicon glyphicon-lock | permission/index.htm |
| 7 | 1 | 业务审核 | glyphicon glyphicon-ok | (NULL) |
| 8 | 7 | 实名认证审核 | glyphicon glyphicon-check | auth_cert/index.htm |
| 9 | 7 | 广告审核 | glyphicon glyphicon-check | auth_adv/index.htm |
| 10 | 7 | 项目审核 | glyphicon glyphicon-check | auth_project/index.htm |
| 11 | 1 | 业务管理 | glyphicon glyphicon-th-large | (NULL) |
| 12 | 11 | 资质维护 | glyphicon glyphicon-picture | cert/index.htm |
| 13 | 11 | 分类管理 | glyphicon glyphicon-equalizer | type/index.htm |
| 14 | 11 | 流程管理 | glyphicon glyphicon-random | process/index.htm |
| 15 | 11 | 广告管理 | glyphicon glyphicon-hdd | advert/index.htm |
| 16 | 11 | 消息模板 | glyphicon glyphicon-comment | message/index.htm |
| 17 | 11 | 项目分类 | glyphicon glyphicon-list | projectType/index.htm |
| 18 | 11 | 项目标签 | glyphicon glyphicon-tags | tag/index.htm |
| 19 | 1 | 参数管理 | glyphicon glyphicon-list-alt | param/index.htm |
| (Auto) | (NULL) | (NULL) | (NULL) | (NULL) |

### 1.1.2 atcrowdfunding-boot-manager 项目中增加相关配置

与 atcrowdfunding-boot-potal 相同,只需要修改端口即可

### 1.1.3 atcrowdfunding-boot-manager 增加处理器方法

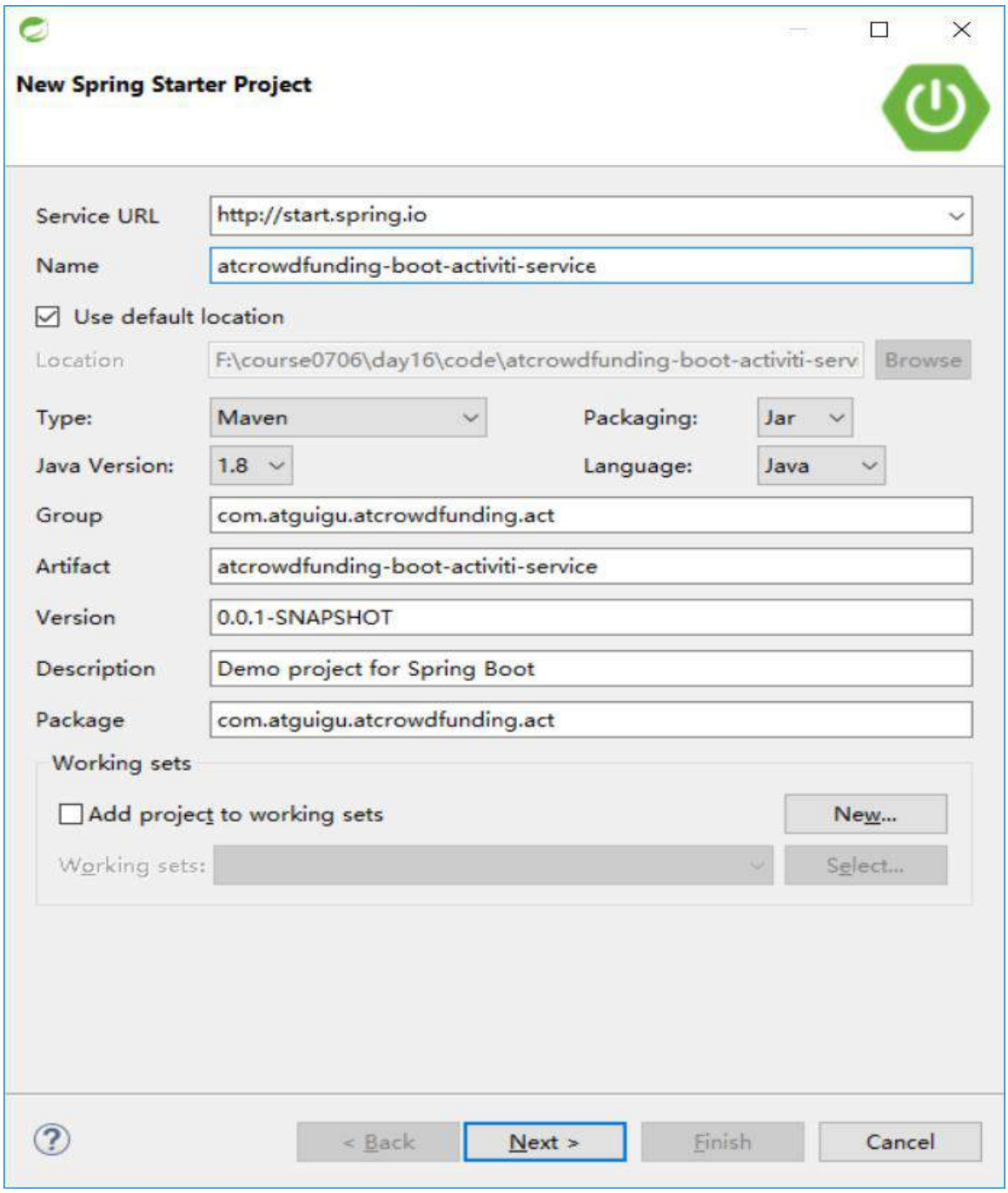
```
package com.atguigu.atcrowdfunding.manager.controller;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import com.atguigu.atcrowdfunding.common.BaseController;

@Controller
@RequestMapping("/process")
public class ProcessController extends BaseController {
	@RequestMapping("/index")
	public String index() {
		return "process/index";
	}
}
```

## 1.1.4 atcrowdfunding-boot-manager 增加主页面

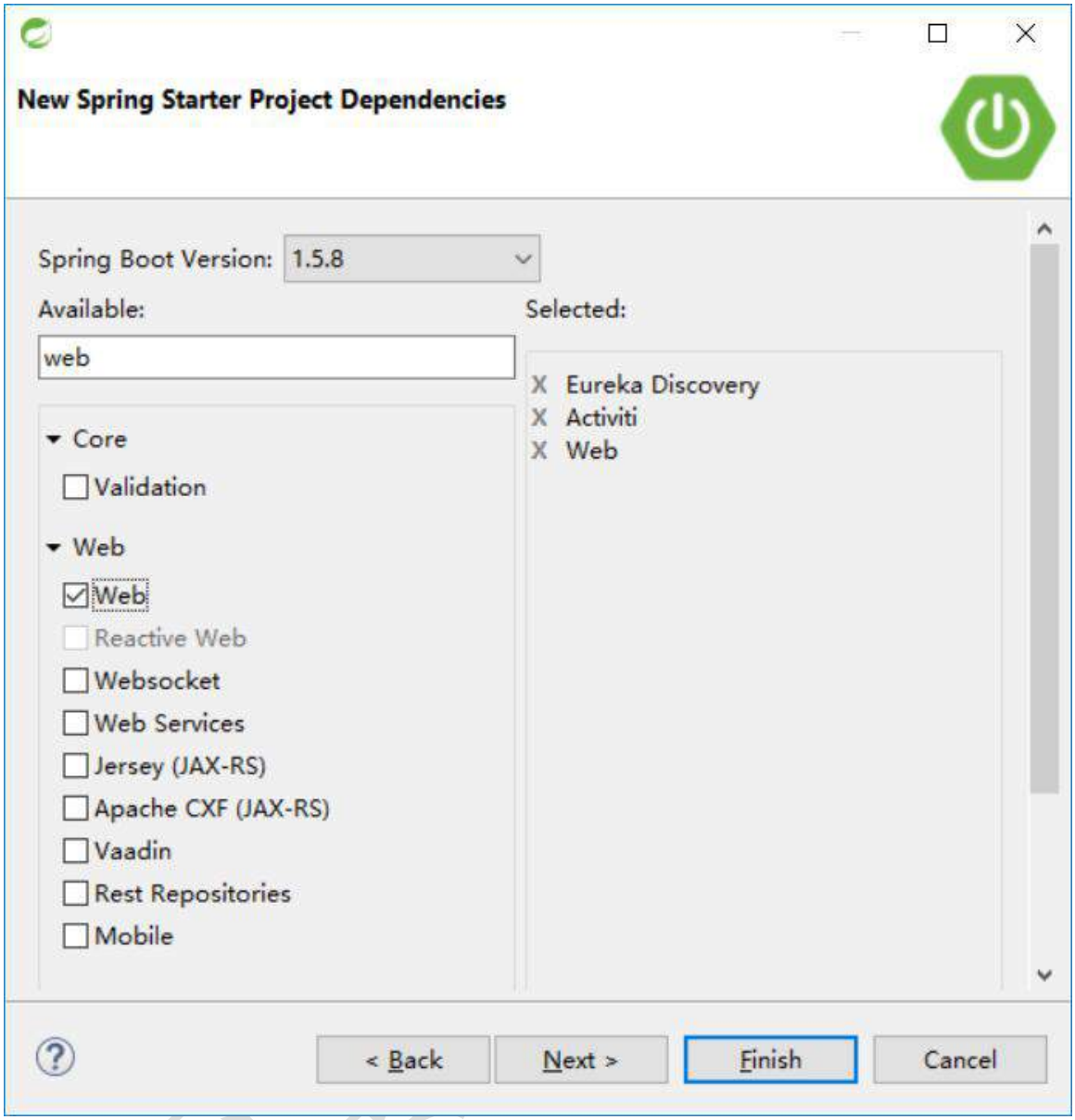
- process/index.ftl
- 并增加相关静态资源.

## 1.2 流程管理-流程定义-查询显示列表数据

### 1.2.1 创建服务项目 atcrowdfunding-boot-activiti-service




- 增加 Discovery 会自动增加 web 模块.




- 主类上,增加@EnableDiscoveryClient
- application.properties

```
spring.application.name=atcrowdfunding-activiti-service
server.port=3001
eureka.client.serviceUrl.defaultZone=http://127.0.0.1:1001/eureka/
```

- 增加配置配置,并放置在它(activiti)之前.mybatis-3.4.0.jar

```
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
</dependency>
<!-- 数据库连接池 -->
<dependency>
    <groupId>com.alibaba</groupId>
```

```
        <artifactId>druid</artifactId>
        <version>1.0.5</version>
</dependency>
<dependency>
        <groupId>org.mybatis.spring.boot</groupId>
        <artifactId>mybatis-spring-boot-starter</artifactId>
        <version>1.1.1</version>
</dependency>
```

```
<dependency>
        <groupId>org.activiti</groupId>
        <artifactId>activiti-spring-boot-starter-basic</artifactId>
        <version>5.21.0</version>
</dependency>
```

- 增加配置 application.yml

```
---
spring:
   datasource:
      name: mydb
      type: com.alibaba.druid.pool.DruidDataSource
      url: jdbc:mysql://127.0.0.1:3306/atcrowdfunding
      username: root
      password: root
      driver-class-name: com.mysql.jdbc.Driver
mybatis:
   mapperLocations: classpath*:mybatis/mapper-*.xml
   typeAliasesPackage: com.atguigu.**.bean
```

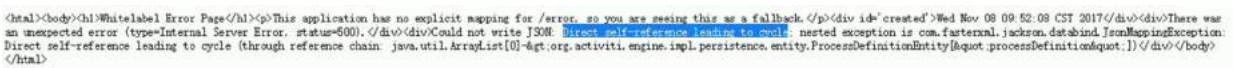
- src/main/java 路径下增加 processes 文件夹.
- 依赖于 common 项目

## 1.2.2 atcrowdfunding-boot-activiti-service 项目中增加 Controller

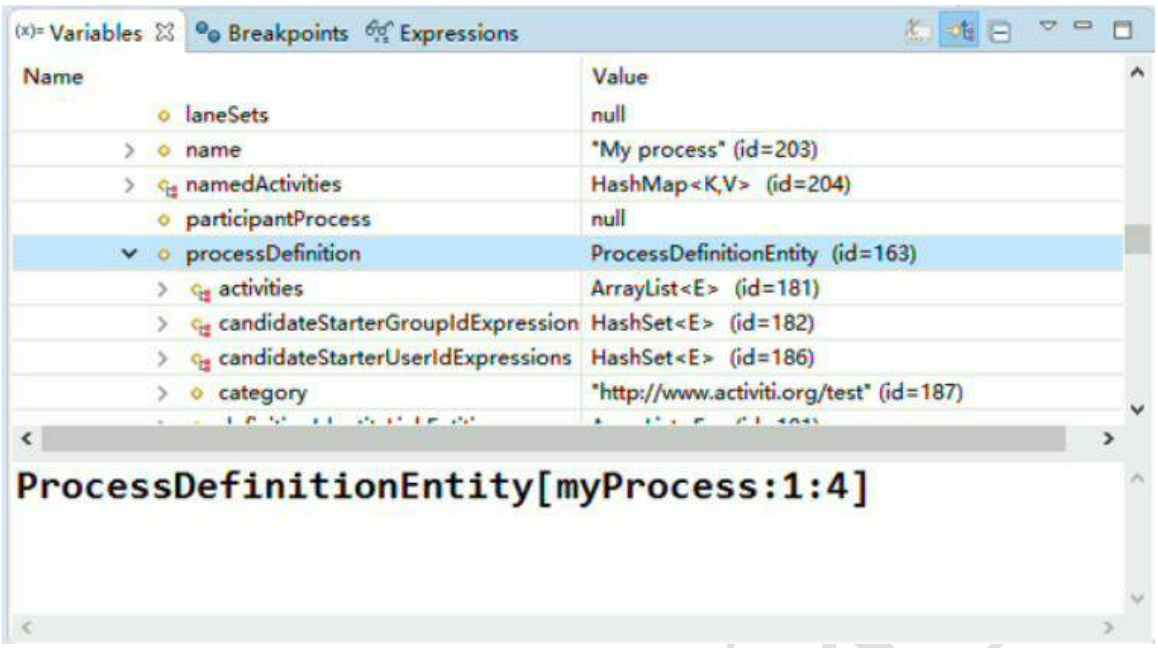
```
package com.atguigu.atcrowdfunding.act.controller;
@RestController
@RequestMapping("/act")
public class ActivitiController extends BaseController {
        @Autowired
        private RepositoryService repositoryService;

        @RequestMapping("/queryPageCount")
        public Object queryPageCount() {
```

```
        ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();
        int count = (int)query.count();
        return count;
    }

    /**
     * 采用 feign 方式实现 service 方法调用时，传递参数需要注意类型
     * 1）如果简单类型（String, Integer），可以通过路径直接传递
     * 2）如果引用类型（Object）,那么需要通过请求体传递参数，所以需要增加注解：@RequestBody
    */
    @RequestMapping("/queryPageData")
    public List<Map<String, Object>> queryPageData( @RequestBody Map<String, Object> varMap ) {

        int start = (int)varMap.get("start");
        int size = (int)varMap.get("size");

        ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();
        // 查询的对象如果存在内部自关联，那么在转换 JSON 时，会出现映射异常。
        // [{}, {}, {}]
        List<ProcessDefinition> list = query.listPage(start, size);
        List<Map<String, Object>> pdMapList = new ArrayList<Map<String,Object>>();

        for ( ProcessDefinition pd : list ) {
            Map<String, Object> pdMap = new HashMap<String, Object>();
            pdMap.put("id", pd.getId());
            pdMap.put("key", pd.getKey());
            pdMap.put("name", pd.getName());
            pdMap.put("version", pd.getVersion());
            pdMap.put("deployid", pd.getDeploymentId());
            pdMapList.add(pdMap);
        }

        return pdMapList;
    }
}
```

- 需要做集合转换,否则,转换为 json 时报死循环.




- Debug 调试,会发现 ProcessDefinition 对象中存在自关联.




- 解决问题：

将 List 中数据使用 Map 存储，然后进行 JSON 数据格式化

## 1.2.3 atcrowdfunding-boot-manager 在 process/index.ftl 主页面上增加异步请求

<link rel="stylesheet" href="/css/pagination.css">
<button id="uploadBtn" type="button" class="btn btn-primary" style="float:right;" > 上传流程定义文件</button>

## 1.2.4 参考代码

```
<thead>
    <tr >
      <th width="30">#</th>
      <th width="30"><input type="checkbox" onclick="checkAll(this)"></th>
      <th>Key</th>
      <th>名称</th>
      <th>版本号</th>
      <th width="100">操作</th>
    </tr>
</thead>
<tbody id="userTbody">
</tbody>
```

```
<tfoot>
    <tr >
        <td colspan="6" align="center">
            <div id="Pagination" class="pagination" />
        </td>
    </tr>
</tfoot>
```

```
<script src="/jquery/jquery-2.1.1.min.js"></script>
<script src="/bootstrap/js/bootstrap.min.js"></script>
<script src="/script/docs.min.js"></script>
<script src="/layer/layer.js"></script>
<script src="/jquery/jquery.pagination.js"></script>
<script src="/jquery/jquery-form.min.js"></script>
    <script type="text/javascript">
        $(function () {
            pageQuery(0);
        });

        var pagesize = 10;
        function pageQuery(pageIndex) {

            var pageno = pageIndex + 1;

            var loadingIndex = -1;

            var jsonData = {
                "pageno" : pageIndex+1,
                "pagesize" : pagesize
            };

            $.ajax({
                type : "POST",
                url    : "/process/loadPageData",
                data : jsonData,
                beforeSend : function() {
                    loadingIndex = layer.load(2, {time: 10*1000});
                },
                success :    function(result) {
                    layer.close(loadingIndex);
                    // 获取用户集合数据
                    var pageObj = result.data;
                    var users = pageObj.datas;
```

```
                    var content = "";
                    $.each(users, function(index, pd){
                        content += '<tr>';
                        content += '    <td>'+(index+1)+'</td>';
                        content += '    <td><input type="checkbox" value="'+pd.id+'"></td>';
                        content += '    <td>'+pd.key+'</td>';
                        content += '    <td>'+pd.name+'</td>';
                        content += '    <td>'+pd.version+'</td>';
                        content += '    <td>';
                        content += '        <button type="button" onclick="window.location.href=\'/process/view?id='+pd.id+'\'" class="btn btn-success btn-xs"><i class="glyphicon glyphicon-check"></i></button>';
                        content += '        <button type="button" onclick="deleteProcDef('+pd.deployid+', \''+pd.name+'\')" class="btn btn-danger btn-xs"><i class="glyphicon glyphicon-remove"></i></button>';
                        content += '    </td>';
                        content += '</tr>';
                    });

                    $("#userTbody").html(content);

                    $("#Pagination").pagination(pageObj.totalsize, {
                        num_edge_entries: 1, //边缘页数
                        num_display_entries: 4, //主体页数
                        callback: pageQuery,
                        prev_text:"上一页",
                        next_text:"下一页",
                        current_page:pageIndex,
                        items_per_page:pagesize
                    });
                }

            });
        }

    </script>
</body>
</html>
```

## 1.2.5 注意

页面分页死循环问题.修改 jquery.pagination.js 文件,注释掉最后一行代码.

## 1.2.6 atcrowdfunding-boot-manager 项目中增加代码

### 1 ProcessController

```
package com.atguigu.atcrowdfunding.manager.controller;

...

@Controller
@RequestMapping("/process")
public class ProcessController extends BaseController {
	@Autowired
	private ActivitiService activitiService;

	@RequestMapping("/index")
	public String index() {
		return "process/index";
	}

	@ResponseBody
	@RequestMapping("/loadPageData")
	public Object loadPageData( Integer pageno, Integer pagesize ) {
		start();

		try {

			Map<String, Object> varMap = new HashMap<String, Object>();
			varMap.put("start", (pageno-1)*pagesize);
			varMap.put("size", pagesize);
			List<Map<String, Object>> data = activitiService.queryPageData(varMap);
			Integer count = (Integer)activitiService.queryPageCount();

			Page<Map<String, Object>> page = new Page<Map<String, Object>>();

			page.setDatas(data);
```

```
                page.setTotalsize(count);
                data(page);
                success(true);
            } catch ( Exception e ) {
                e.printStackTrace();
                success(false);
            }

            return end();
        }
}
```

## 2 ActivitiService

```
package com.atguigu.atcrowdfunding.manager.service;

...

@FeignClient("atcrowdfunding-activiti-service")
public interface ActivitiService {
    @RequestMapping("/act/queryPageCount")
    public Object queryPageCount();

    @RequestMapping("/act/queryPageData")
    public List<Map<String, Object>> queryPageData( @RequestBody Map<String, Object> varMap );

}
```

**注意:**

- 如果不采用 Map 集合传参,可以不使用**@RequestBody 注解；直接传递简单值即可**

```
@RequestMapping("/act/queryPageData/{startindex}/{pagesize}")
public List<Map<String, Object>> queryPageData(@PathVariable("startindex") Integer startindex,
@PathVariable("pagesize") Integer pagesize);
```

# 第 2 章 流程管理-流程定义部署


## 2.1 上传流程定义文件按钮

```
<button id="uploadBtn" type="button" class="btn btn-primary" style="float:right;" ><i class="glyphicon glyphicon-upload"></i> 上传流程定义文件</button>
```

## 2.2 增加上传的表单

```
<form id="procDefForm" style="display:none;" action="/process/upload" method="post"
        enctype="multipart/form-data">
            <input type="file" id="procDefFile" name="procDefFile">
</form>
```

## 2.3 给文件上传表单增加点击事件

```
$("#uploadBtn").click(function(){
            $("#procDefFile").click();
});
$("#procDefFile").change(function(){

                        var loadingIndex = -1 ;
                        var options = {
                                resetForm: true,
                          url:"${APP_PATH}/process/upload",
                        beforeSubmit : function(){
                              loadingIndex = layer.msg('数据正在保存中', {icon: 6});
```

```
                    return true ;
                },
                success : function(result){
                        layer.close(loadingIndex);
                        if(result.success){
                            layer.msg("部署成功", {time:1000, icon:6});
                            pageQuery(0);
                        }else{
                            layer.msg("部署失败", {time:1000, icon:5, shift:6});
                        }
                 }
            };
            $("#procDefForm").ajaxSubmit(options);
});
```

## 2.4 atcrowdfunding-boot-manager 项目中处理文件上传

### 2.4.1 在主类中增加配置

```
@Bean
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

### 2.4.2 增加上传文件处理

```
@Autowired
private RestTemplate restTemplate;

@ResponseBody
@RequestMapping("/upload")
public Object upload( HttpServletRequest req ) {
    start();

    try {
        MultipartHttpServletRequest request =(MultipartHttpServletRequest)req;

        MultipartFile file = request.getFile("procDefFile");
```

```
            // 获取的是表单中文件域的 name 属性值
            System.out.println(file.getName());
            // 获取的是传递的文件名称
            System.out.println(file.getOriginalFilename());

            String uuid = UUID.randomUUID().toString();
            String fileName = file.getOriginalFilename();
            final File tempFile = File.createTempFile(uuid, fileName.substring(fileName.lastIndexOf(".")));

            file.transferTo(tempFile);

            FileSystemResource resource = new FileSystemResource(tempFile);
                    MultiValueMap<String, Object> param = new LinkedMultiValueMap<String, Object>();
                    param.add("pdfile", resource);
                    String s =
restTemplate.postForObject("http://atcrowdfunding-activiti-service/act/deploy",param,String.class);
            System.out.println("result string = " + s);
                    tempFile.delete();
            success(true);
        } catch ( Exception e ) {
            e.printStackTrace();
            success(false);
        }

        return end();
}
```

## 2.5 部署流程定义

在 atcrowdfunding-boot-activiti-service 项目的 com.atguigu.atcrowdfunding.act.controller.ActivitiController 控制器中部署流程定义

```
@RequestMapping("/act/deploy")
public Object depolyProcDef( @RequestParam("pdfile") MultipartFile file ) {
    start();

    try {
        // 部署流程定义图形
        repositoryService
            .createDeployment()
            .addInputStream(file.getOriginalFilename(), file.getInputStream())
            //.addClasspathResource(file.getOriginalFilename())
```

```
                .deploy();

            success(true);
        } catch ( Exception e ) {
            e.printStackTrace();
            false();
        }

        return end();
}
```

## 2.6 测试过程可能出现异常

### 2.6.1 在使用 RestTemplate 进行远程连接服务器时,需要增加特殊的注解配置.否则报错

```
Description:

Field restTemplate in com.atguigu.atcrowdfunding.manager.controller.ProcessController required a bean of type 'org.springframework.web.client.RestTemplate' that could not be found.

Action:
Consider defining a bean of type 'org.springframework.web.client.RestTemplate' in your configuration.
```

### 2.6.2 @LoadBalanced

```
@Bean
@LoadBalanced    //采用负载均衡方式连接远程服务器
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

# 第 3 章 流程管理-流程定义图片展示

## 3.1 增加显示流程定义图片的链接

```
<button type="button" onclick="window.location.href=\'/process/view?id='+pd.id+\'" class="btn btn-success btn-xs"><i class=" glyphicon glyphicon-check"></i></button>
```

## 3.2 处理请求

```
@RequestMapping("/view")
public String view( String id, Model model ) { //流程实例 id 是字符串类型.
    model.addAttribute("id", id);
    return "process/view";
}
```

## 3.3 显示页面

```
<img id="procDefImg" src="/ztree/img/loading.gif">
```

① 拷贝图片：.\layer-v3.0.1\layer\skin\default\loading.gif 到 /ztree/img/loading.gif

## 3.4 发起请求加载流程图片

```
<script type="text/javascript">
    $(function () {
        $("#procDefImg").attr("src", "/process/loadImg?id=${id}");
    });
</script>
```

## 3.5 处理请求

```
@RequestMapping("/loadImg")
public void loadImg(String id, HttpServletResponse resp) throws Exception { //流程实例 id 是字符串类型.
    // 通过响应对象返回图形信息
    HttpHeaders headers = new HttpHeaders();
    headers.setContentType(MediaType.IMAGE_PNG);
```

```
    String url = "http://atcrowdfunding-activiti-service/act/loadImgById/"+id;
    ResponseEntity<byte[]> response =
    restTemplate.exchange( url, HttpMethod.POST,   new HttpEntity<byte[]>(headers), byte[].class);
    byte[] result = response.getBody();

    InputStream in = new ByteArrayInputStream(result);
    OutputStream out = resp.getOutputStream();

    int i = -1;
    while ( (i = in.read()) != -1 ) {
        out.write(i);
    }
}
```

## 3.6 远程服务调用

```
@RequestMapping("/loadImgById/{id}")
public byte[] loadImgById(@PathVariable("id")String id) {
    // 部署 ID ==>  流程定义 ID
    // 从数据库中读取流程定义的图片
    //根据流程部署 id 和部署资源名称获取部署图片的输入流。
    ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();
    ProcessDefinition pd = query.processDefinitionId(id).singleResult();
    InputStream in =
        repositoryService.getResourceAsStream(pd.getDeploymentId(),pd.getDiagramResourceName());

    ByteArrayOutputStream swapStream = new ByteArrayOutputStream();
    byte[] buff = new byte[100]; //buff 用于存放循环读取的临时数据
    int rc = 0;
    try {
        while ((rc = in.read(buff, 0, 100)) > 0) {
            swapStream.write(buff, 0, rc);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    byte[] in_b = swapStream.toByteArray(); //in_b 为转换之后的结果

    return in_b;
}
```

# 第 4 章 流程管理-流程定义删除

删除流程定义（act_re_procdef）（级联删除部署表数据（act_re_deployment ）和部署的图片数据（act_ge_bytearray））

## 4.1 增加链接

```
<button type="button" onclick="deleteProcDef('+pd.deployid+', \"+pd.name+\')" class="btn btn-danger btn-xs"><i class=" glyphicon glyphicon-remove"></i></button>
```

## 4.2 增加函数

```
function deleteProcDef(id, name) {
    layer.confirm("删除流程定义信息["+name+"]，是否继续？", {icon: 3, title:'提示'}, function(cindex){
        $.ajax({
            type : "POST",
            url  : "${APP_PATH}/process/delete",
            data : {
                "id" : id  //流程部署 id.
            },
            success : function(result) {
                if ( result.success ) {
                    layer.msg("流程定义信息删除成功", {time:1500, icon:6}, function(){
                        pageQuery(0);
                    });
                } else {
                    layer.msg("流程定义信息删除失败", {time:1500, icon:5, shift:6});
                }
            }
        });
        layer.close(cindex);
    }, function(cindex){
        layer.close(cindex);
    });
}
```

## 4.3 增加处理请求

```
@ResponseBody
@RequestMapping("/delete")
public Object delete( String id ) { //流程部署 id.
    start();
    try {
        activitiService.delete(id);
        success(true);
    } catch ( Exception e ) {
        e.printStackTrace();
        success(false);
    }

    return end();
}
```

```
package com.atguigu.atcrowdfunding.manager.service;

import java.util.List;
import java.util.Map;

import org.springframework.cloud.netflix.feign.FeignClient;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;

@FeignClient("atcrowdfunding-activiti-service")
public interface ActivitiService {
    @RequestMapping("/act/delete/{id}")
    public Object delete(@PathVariable("id")String id); //流程部署 id.
}
```

## 4.4 远程服务处理

```
@RequestMapping("/delete/{id}")
public Object delete(@PathVariable("id")String id) {
    start();

    try {
```

```
            /*
            ProcessDefinitionQuery query = repositoryService.createProcessDefinitionQuery();
            ProcessDefinition pd = query.processDefinitionId(id).singleResult();
            */
            repositoryService.deleteDeployment(id, true); //流程部署 id.    级联删除.
            success(true);
        } catch ( Exception e ) {
            e.printStackTrace();
            success(false);
        }

        return end();
}
```

# 附录 1 Spring Cloud 文件上传

## 1.1 Spring Cloud 文件上传

在 Spring Cloud 项目中，应用和服务间数据的交互一般采用 feign 的方式。基本使用没有问题，而且也比较方便，但是进行文件数据的交互就比较麻烦了，所以需要采用其他的方式进行处理。
我们一般创建一个 web 应用后，需要通过 Web 浏览器对这个应用中的功能进行访问，然后在浏览器中显示处理的结果（HTML）或进行 JS 逻辑处理（JSON），但是在某些时候，我们希望在 JAVA 程序代码中也可以访问这个 web 系统中的功能，那么就可以采用 Spring 3.0 框架中的 **RestTemplate** 来完成。
RestTemplate 简化了发起 HTTP 请求以及处理响应的过程，简单来说，就是操作 web 服务更加的容易了。代码操作如下：

- 文件上传（发文件）

在 Application 类中增加 RestTemplate 配置：

```
@Bean
public RestTemplate restTemplate() {
    return new RestTemplate();
}
```

在 Controller 中，自动配置对应的属性

```
@Autowired
private RestTemplate restTemplate;
```

在逻辑方法中，调用 restTemplate 对象，将文件上传给远程 web 服务

```
MultipartFile file =      XXXXXX;
HttpHeaders headers = new HttpHeaders();
```

```
headers.setContentType(MediaType.APPLICATION_OCTET_STREAM);
String uuid = UUID.randomUUID().toString();
String fileName = file.getOriginalFilename();
final File tempFile = File.createTempFile(uuid, fileName.substring(fileName.lastIndexOf(".")));
file.transferTo(tempFile);
FileSystemResource resource = new FileSystemResource(tempFile);
MultiValueMap<String, Object> param = new LinkedMultiValueMap<>();
param.add("pdfile", resource);
restTemplate.postForObject("http://service-name/url",param,String.class);
```

- 文件上传（接收文件）

```
在 web 服务的处理方法中增加对应的请求参数接收文件即可
public String depolyProcDef( @RequestParam("pdfile") MultipartFile file )
```

## 1.2 HTML5 文件上传

```
浏览器客户端判断上传文件的大小。

$("#文件域对象 ID").change(function(event){
    // 获取当前选择的文件 event.target.files
    var files = event.target.files, file;
    if (files && files.length > 0) {
        file = files[0];
    }
    // 判断上传文件的大小 file.size ， 单位字节 Byte
    // 判断上传文件的类型 file.type
    // 本地生成上传文件后的临时文件地址
    var URL = window.URL || window.webkitURL;
    var imgURL = URL.createObjectURL(file);
});
```

# 附录 2 文件下载

## 2.1 文件下载（发文件）

```
在 Web 服务的处理方法中返回字节数组。逻辑中将文件转换为流后再转换为字节数组即可
public byte[] loadImgById(String id) {
    InputStream in = XXXXX;
    ByteArrayOutputStream swapStream = new ByteArrayOutputStream();
```

```
    byte[] buff = new byte[100]; //buff 用于存放循环读取的临时数据
    int rc = 0;
    try {
        while ((rc = in.read(buff, 0, 100)) > 0) {
            swapStream.write(buff, 0, rc);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
    byte[] in_b = swapStream.toByteArray(); //in_b 为转换之后的结果
    return in_b;
}
```

## 2.2 文件下载（接收文件）

```
//采用 RestTemplate 在逻辑方法中接收返回的字节数组再转换为流生成文件即可
HttpHeaders headers = new HttpHeaders();
headers.setContentType(MediaType.IMAGE_PNG);

String url = "http://service-name/url";

ResponseEntity<byte[]> response = restTemplate.exchange(url, HttpMethod.POST, new
HttpEntity<byte[]>(headers),byte[].class);
byte[] result = response.getBody();


InputStream in = new ByteArrayInputStream(result);
OutputStream out = resp.getOutputStream();
int i = -1;
while ( (i = in.read()) != -1 ) {
    out.write(i);
}
```

# 作业

- 资质管理
- 分类管理

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 项目计划

- 需求
- 流程图
- 流程审批单
- 会员实名认证申请
  - 选择账户类型
  - 填写基本信息
  - 检查邮箱,发送验证码
  - 检查验证码,申请完成
- 会员实名认证审核
  - 后台审核

# 第 1 章 会员实名认证-需求

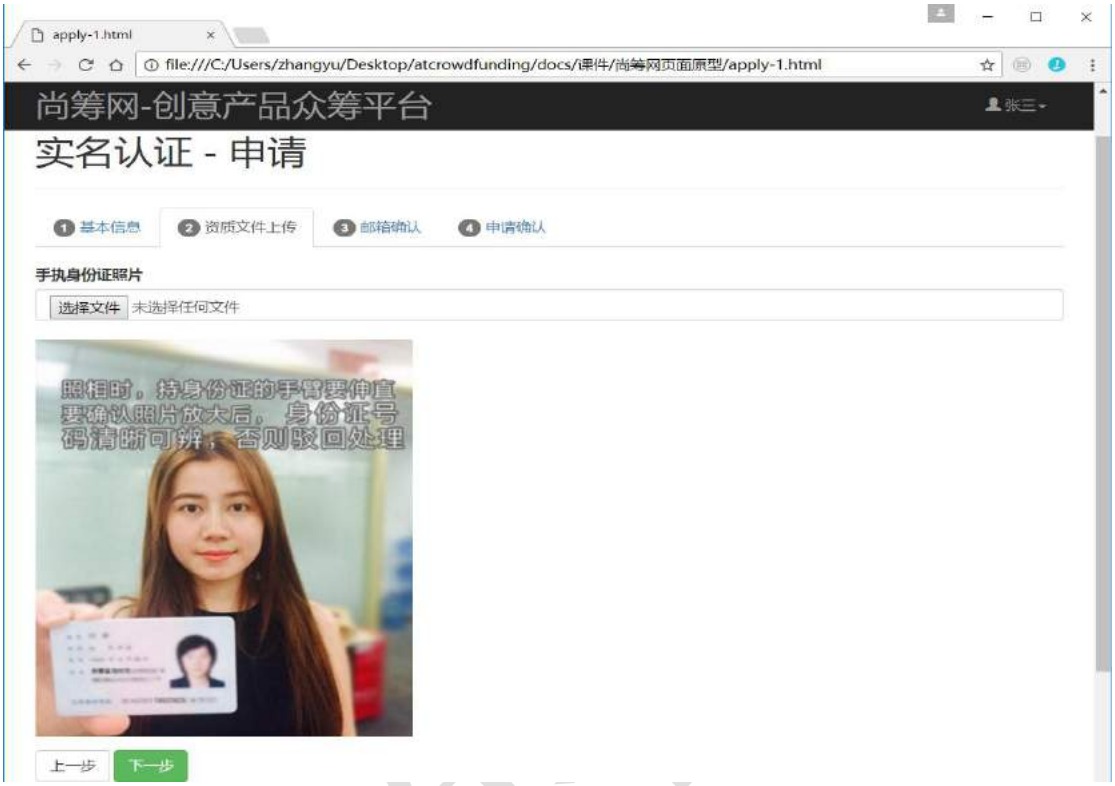
## 1.1 会员登录

- 点击[未实名认证],进行会员实名认证申请.
  - 会员实名认证状态:
    - 未实名认证
    - 实名认证审核中
    - 已实名认证

## 1.2 选择账户类型,点击[认证申请]按钮


## 1.3 填写实名认证申请的基本信息.

## 1.4 选择资质认证相关图片




## 1.5 设置认证的邮箱地址

- 设置好邮箱地址后,点击[下一步],此时系统启动认证审核流程.
- 系统自动发送验证码给会员邮箱
- 会员通过登录邮箱获得验证码,在下一个页面中输入验证码进行验证.


## 1.6 输入邮件验证码,申请完成

用户通过邮箱获取验证码,将验证码填写到输入框并点击[申请完成],此时系统执行[审核验证码]流程任务.


## 1.7 用户提交[申请完成]

此时用户状态为[认证审核中]状态.等待后台管理员系统人工审核.审核通过用户状态为[已实名认证].

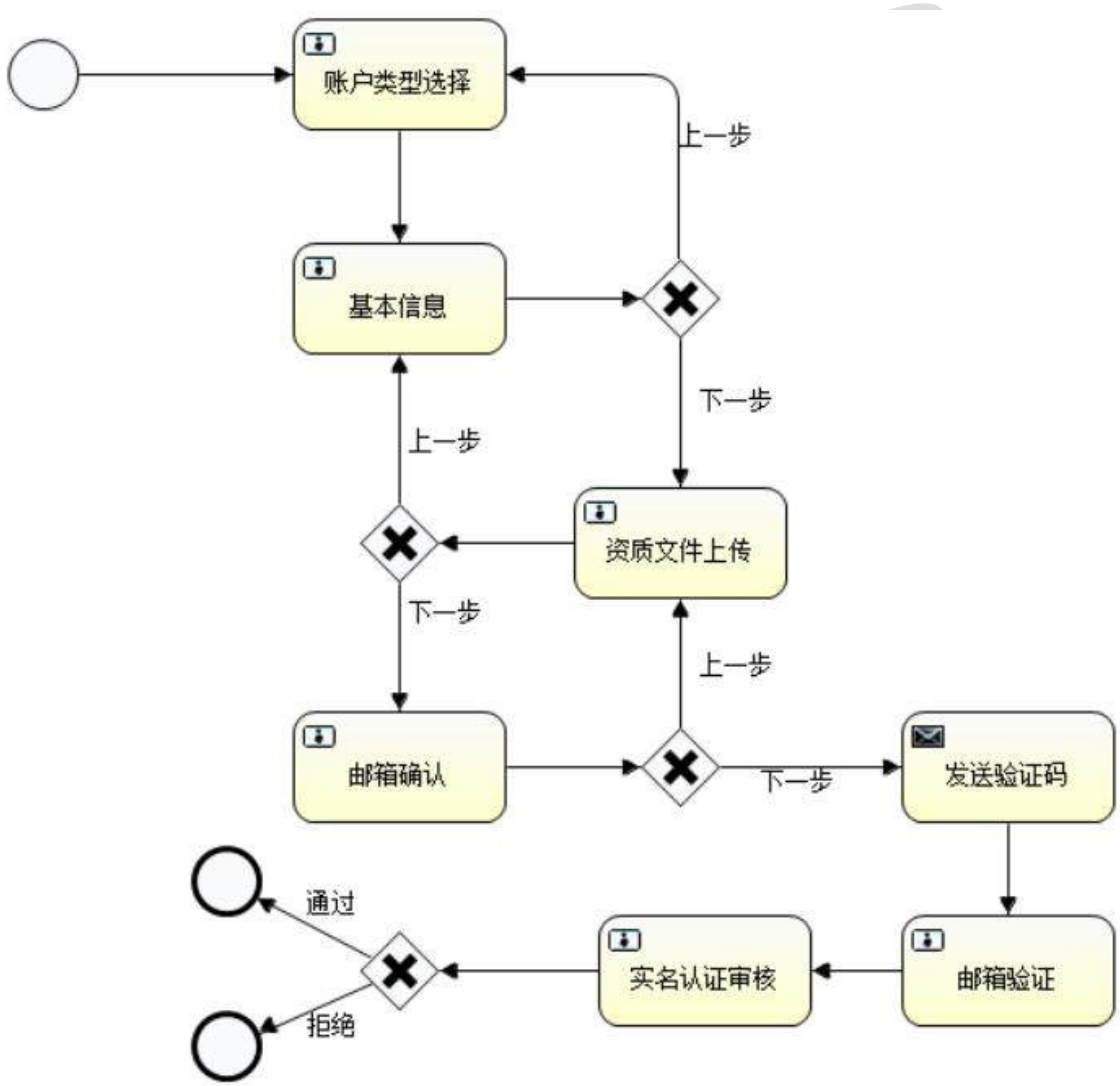
## 1.8 登录后台管理员系统

- 点击实名认证审核,查询待审核申请列表


- 点击某个实名认证申请进行审核:
  - 审核通过
  - 审核拒绝

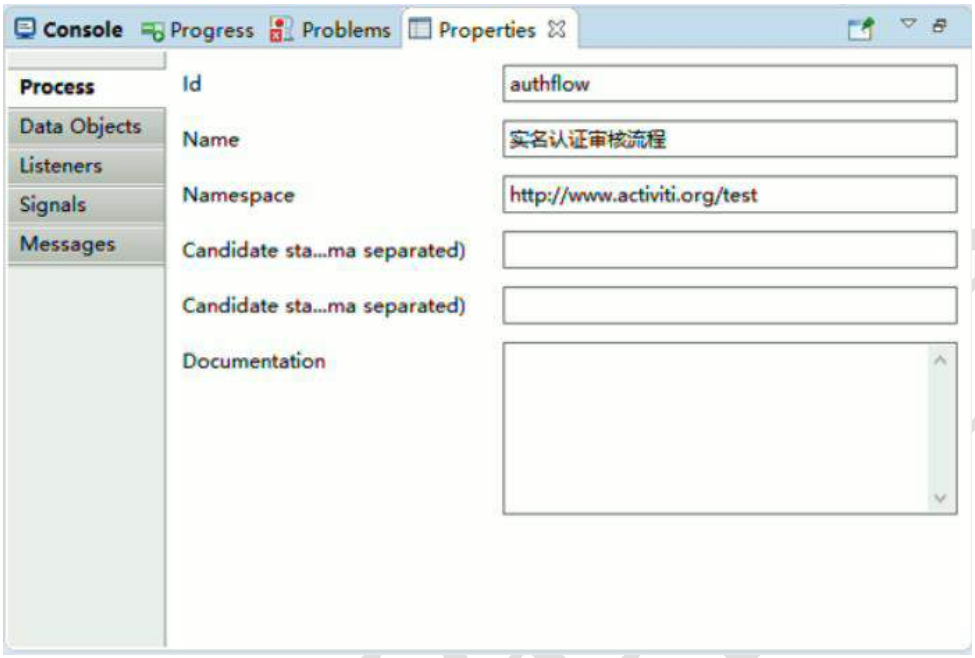
# 第 2 章 流程管理-流程定义-实名认证审核流程图

## 2.1 流程图



## 2.2 流程设置

### 2.2.1 设置流程信息




### 2.2.2 账户类型选择

设置委托人 : ${ loginacct }

### 2.2.3 基本信息

设置委托人 : ${ loginacct }

上一步(账户类型选择): ${flag == false}

下一步(资质文件上传): ${flag == true}

### 2.2.4 资质文件上传

设置委托人 : ${ loginacct }

上一步(基本信息): ${flag == false}

下一步(邮箱确认): ${flag == true}

## 2.2.5 邮箱确认

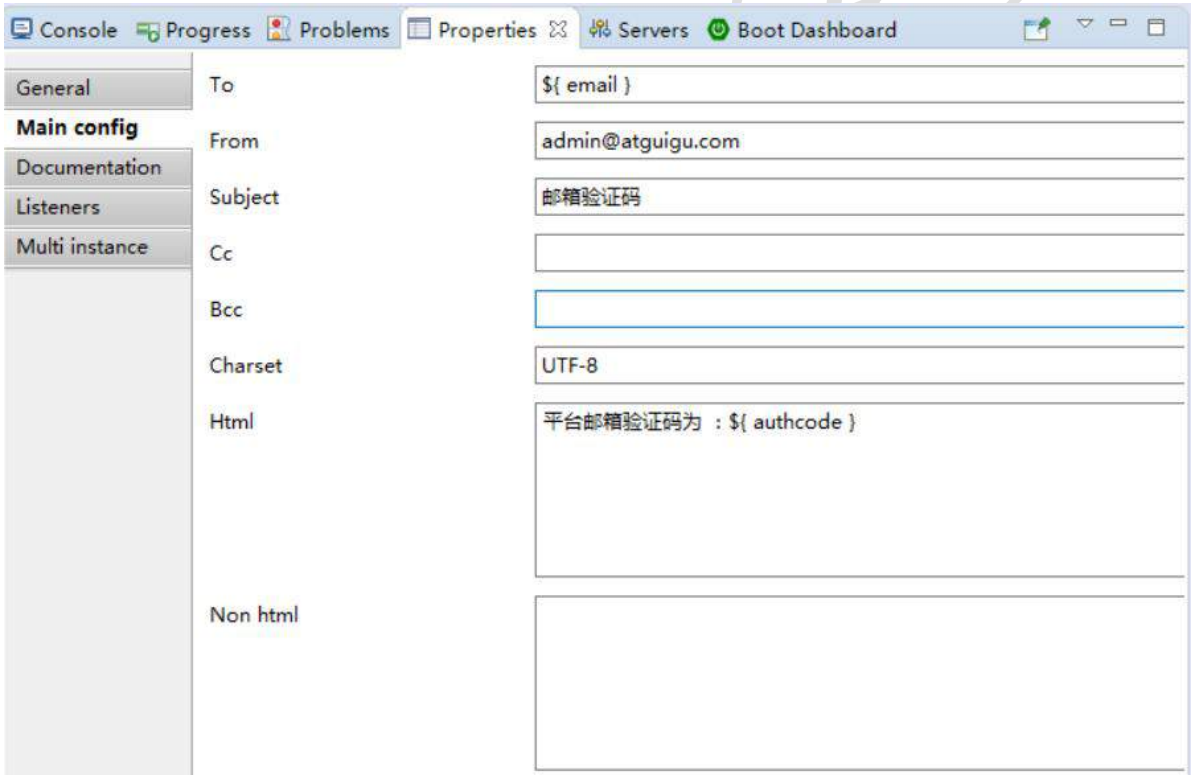
设置委托人 : ${ loginacct }

上一步(资质文件上传): ${flag == false}

下一步(发送验证码): ${flag == true}

## 2.2.6 发送验证码

系统自动发送邮件,不需要设置委托人.




## 2.2.7 邮箱验证

- 设置委托人 : ${ loginacct }

## 2.2.8 实名认证审核

- 设置委托组: backcheck

## 2.2.9 审批(atcrowdfunding-boot-activiti-service)

- 通过:${flag == true}

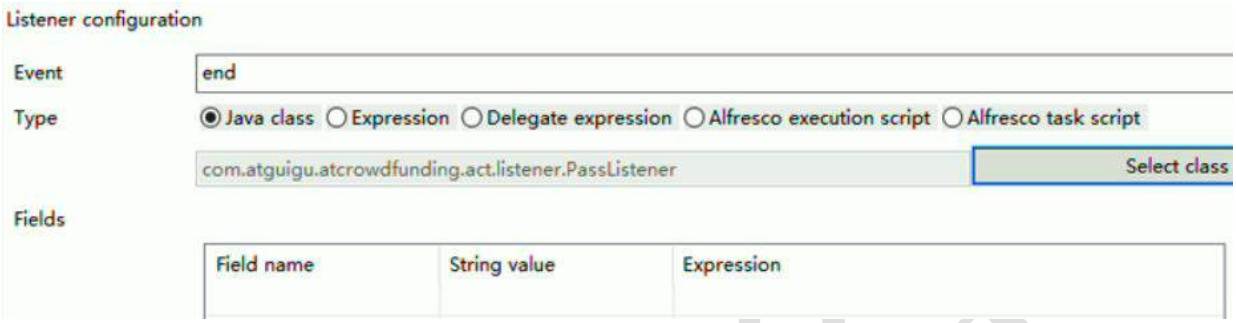
```
package com.atguigu.atcrowdfunding.act.listener;


import org.activiti.engine.delegate.DelegateExecution;
import org.activiti.engine.delegate.ExecutionListener;


public class PassListener implements ExecutionListener {


    @Override
    public void notify(DelegateExecution execution) throws Exception {
    }
}
```

- 拒绝:${flag == false}

```
package com.atguigu.atcrowdfunding.act.listener;


import org.activiti.engine.delegate.DelegateExecution;
import org.activiti.engine.delegate.ExecutionListener;


public class RefuseListener implements ExecutionListener {


    @Override
    public void notify(DelegateExecution execution) throws Exception {
```

```
    }
}
```

- 设置流程监听器(注意:在结束节点上进行设置)




- 设置字体配置

```
package com.atguigu.atcrowdfunding.act.config;


import org.activiti.spring.SpringProcessEngineConfiguration;
import org.activiti.spring.boot.ProcessEngineConfigurationConfigurer;
import org.springframework.context.annotation.Configuration;


@Configuration
public class ActivitiConfig implements ProcessEngineConfigurationConfigurer {
    @Override
    public void configure(SpringProcessEngineConfiguration processEngineConfiguration) {
        processEngineConfiguration.setActivityFontName("宋体");
        processEngineConfiguration.setLabelFontName("宋体");
        processEngineConfiguration.setMailServerPort(25);
    }
}
```

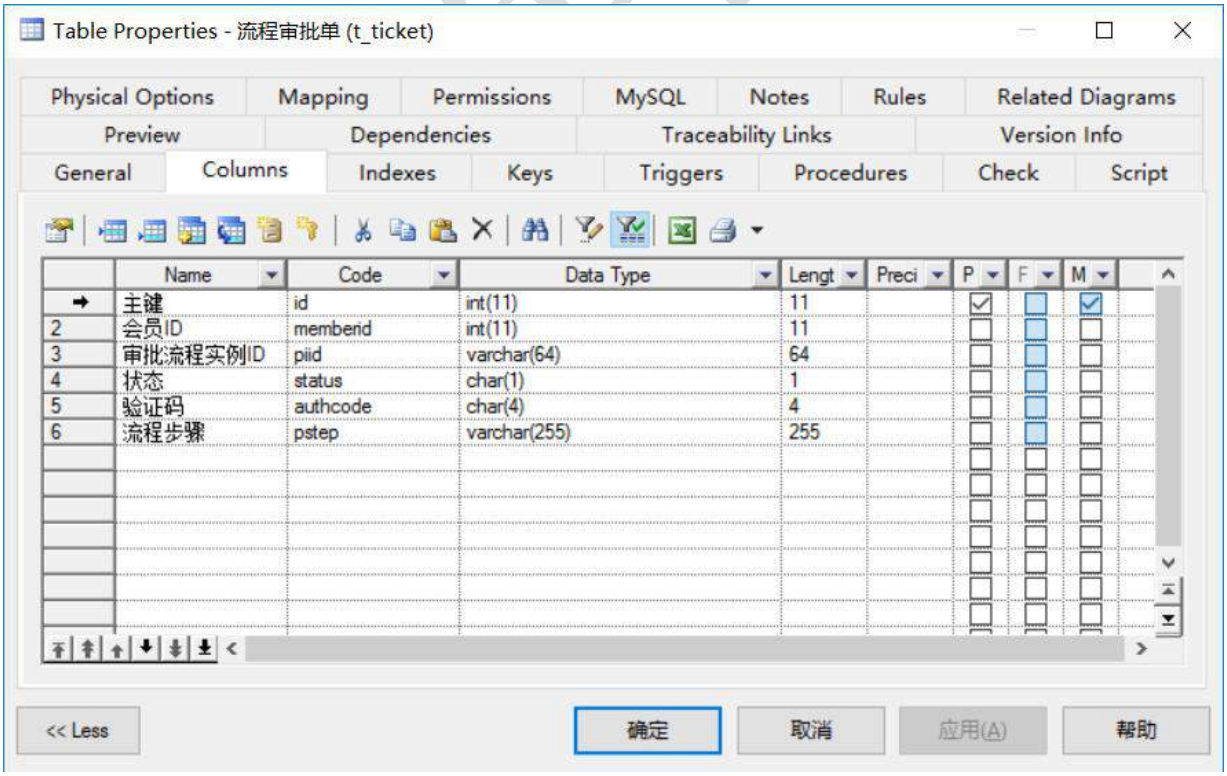
# 第 3 章 流程审批单

- 需求:

避免会员申请流程进行一半突然停止,当再次申请时不需要重头开始申请,之前申请的步骤进行保留,从中间步骤开始继续开始即可.

## 3.1 PDM 设计流程审批单

### 3.1.1 t_ticket (流程审批单)

### 3.1.2 表 SQL

```
create table t_ticket
(
    id                  int(11) not null auto_increment,
    memberid            int(11),
    piid                varchar(64),
    status              char(1) comment '0 - 申请中， 1 - 申请完毕',
    authcode            char(4),
    pstep               varchar(255) comment 'accttype-账户类型，basicinfo-基本信息，uploadfile-资质文件上传，checkemail-邮箱确认',
    primary key (id)
);
```

# 第 4 章 会员实名认证-实名认证账户的状态

## 4.1 实名认证账户的状态

- 数据库 t_member 表 authstatus 字段的状态:
  - 0:未实名认证
  - 1:实名认证审核中
  - 2:已实名认证



- 账户类型分为：
  - 0 - 企业
  - 1 - 个体工商户
  - 2 - 个人
  - 3 - 政府

## 4.2 会员登录主页面

- member.ftl

```
<#if loginMember.authstatus == '1'>
    <span class="label label-info" style="cursor:pointer;" >实名认证申请中</span>
<#elseif loginMember.authstatus == '2'>
    <span class="label label-success" style="cursor:pointer;" >已实名认证</span>
```

```
<#else>
    <span class="label label-danger" style="cursor:pointer;"
    onclick="window.location.href='${APP_PATH}/member/apply'">未实名认证</span>
</#if>
```

# 第 5 章 会员实名认证-开始申请

- 说明

1.点击"未实名认证"按钮,开始申请

2.根据登录会员 id,查询流程单

- 如果流程单为 null
  - 启动流程
  - 保存流程单
  - 跳转到账户类型选择页面
- 如果流程单不为 null
  - 根据流程单"pstep"判断,跳转到相关页面.

## 5.1 实名认证开始申请,启动流程实例并跳转到账户类型选择页面

## 5.2 增加相关代码

- 前台:
- **Ticket　流程单实体类,用于封装流程的流转信息.**

```
public class Ticket {
    private Integer id,memberid;
    private String piid,status,authcode,pstep;
    …
}
```

- **MemberController 控制流程跳转的步骤,具有记忆功能.**

```
/**
 * 实名认证申请
 * 1）查询流程审批单
 * 2）根据流程审批单跳转页面
 */
@RequestMapping("/apply")
public String apply( HttpSession session ) {
    // 获取会员信息
    Member loginMember = (Member)session.getAttribute("loginMember");

    // 查询流程审批单
    Ticket ticket = memberService.queryTicketByMemberid(loginMember.getId());

    if ( ticket == null ) {

        // 创建流程实例
        String piid = activitiService.startProcess(loginMember.getLoginacct());// 流程实例 ID
```

```
            // 新增流程审批单
            ticket = new Ticket();
            ticket.setMemberid(loginMember.getId());
            ticket.setPstep("accttype");
            ticket.setStatus("0");// 申请中
            ticket.setPiid(piid);

            memberService.insertTicket(ticket);
            return "member/accttype";
        } else {
            String pstep = ticket.getPstep();
            if ( "accttype".equals(pstep) ) {
                return "member/accttype";
            } else if ( "basicinfo".equals(pstep) ) {
                return "member/basicinfo";
            } else if ( "certupload".equals(pstep) ) {
                return "member/certupload";
            } else if ( "checkemail".equals(pstep) ) {
                return "member/checkemail";
            } else if ( "checkauthcode".equals(pstep) ) {
                return "member/checkauthcode";
            }
        }
        return "member/apply";
    }
```

- 在 atcrowdfunding-boot-portal 项目中增加 ActivitiService,用于调用后台服务项目

```
package com.atguigu.atcrowdfunding.portal.service;
...
@FeignClient("atcrowdfunding-activiti-service")
public interface ActivitiService {
    @RequestMapping("/act/startProcess/{loginacct}")
    public String startProcess(@PathVariable("loginacct")String loginacct);
}
```

- MemberService

```
@FeignClient("atcrowdfunding-member-service")
public interface MemberService {
    @RequestMapping("/member/insertTicket")
    public void insertTicket(@RequestBody Ticket ticket);

    @RequestMapping("/member/queryTicketByMemberid/{id}")
    public Ticket queryTicketByMemberid(@PathVariable("id")Integer id);
}
```

**注:业务方法进行增,删,改操作需要增加事务,不能是只读操作.**

## 5.2.1 后台

- com.atguigu.atcrowdfunding.member.controller.MemberController

```
@RequestMapping("/member/insertTicket")
public void insertTicket(@RequestBody Ticket ticket) {
    memberService.insertTicket(ticket);
}
@RequestMapping("/member/queryTicketByMemberid/{id}")
public Ticket queryTicketByMemberid(@PathVariable("id")Integer id) {
```

```
    return memberService.queryTicketByMemberid(id);
}
```

- 执行 sql

```
@Ins
ert("insert into t_ticket (memberid, status, piid, pstep) values ( #{memberid}, #{status}, #{piid},
#{pstep}  )")
void insertTicket(Ticket ticket);

@Select("select * from t_ticket where memberid = #{memberid} and status = '0'")
Ticket queryTicketByMemberid(Integer id);
```

## 5.2.2 后台

- 启动流程

```
@RequestMapping("/startProcess/{loginacct}")
public String startProcess( @PathVariable("loginacct")String loginacct ) {
    // 查询实名认证审批流程定义
    ProcessDefinition pd =
    repositoryService
        .createProcessDefinitionQuery()
        .processDefinitionKey("authflow")
        .latestVersion()
        .singleResult();
    // 启动实名认证流程实例
    Map<String, Object> varMap = new HashMap<String, Object>();
    varMap.put("loginacct", loginacct);
    ProcessInstance pi = runtimeService.startProcessInstanceById(pd.getId(), varMap);
```

```
    return pi.getId();
}
```

- 增加页面：member/accttype.ftl
- 解决页面图片路径问题


- 通过 jquery 给所有 img 标签的 src 属性增加上下文路径.

```
    $.each($(".thumbnail img"), function(i, n){
        $(this).attr("src", "${APP_PATH}/" + $(this).attr("src"));
    });
```

- 页面图片选择效果


```
<h2>商业公司</h2>
            <a href="#" accttype="0" class="thumbnail">
                <img alt="100%x180" src="img/services-box1.jpg" data-holder-rendered="true" style="height:
180px; width: 100%; display: block;">
            </a>
```

```
</div>
```

```
$(".thumbnail").click(function(){
        $('.seltype').remove();
        $(this).prepend('<div class="glyphicon glyphicon-ok seltype"></div>');
});
```

# 第 6 章 会员实名认证-账户类型选择

说明:

1.选择账户类型,点击申请按钮,提交所选择的账户类型

2.更新账户类型

- 将账户类型封装到当前会员对象,根据会员对象,更新账户类型
- 根据会员 id 查询流程单,更新流程单"pstep"步骤
- 根据 piid 和 loginacct 完成流程任务,执行下一步(涉及服务间调用问题)

## 6.1.账户类型选择

### 6.1.1 增加按钮 id 属性,便于增加事件处理

```
<button type="button" class="btn btn-danger btn-lg btn-block" onclick="doApply()">认证申请</button>
```

### 6.1.2 页面账户类型设置

```
<div class="container theme-showcase" role="main">
  <div class="page-header">
    <h1>实名认证 - 账户类型选择</h1>
  </div>
```

```
  <div style="padding-top:10px;">
<div class="row">
        <div class="col-xs-6 col-md-3">


        <h2>商业公司</h2>
          <a href="#" accttype="0" class="thumbnail">


            <img alt="100%x180" src="img/services-box1.jpg" data-holder-rendered="true" style="height:
180px; width: 100%; display: block;">
          </a>
        </div>
        <div class="col-xs-6 col-md-3">
          <h2>个体工商户</h2>
          <a href="#" accttype="1" class="thumbnail">
            <img alt="100%x180" src="img/services-box2.jpg" data-holder-rendered="true" style="height:
180px; width: 100%; display: block;">
          </a>
        </div>
        <div class="col-xs-6 col-md-3">
          <h2>个人经营</h2>
          <a href="#" accttype="2" class="thumbnail">
            <img alt="100%x180" src="img/services-box3.jpg" data-holder-rendered="true" style="height:
180px; width: 100%; display: block;">
          </a>
        </div>
        <div class="col-xs-6 col-md-3">
          <h2>政府及非营利组织</h2>
```

```
            <a href="#" accttype="3" class="thumbnail">
                <img alt="100%x180" src="img/services-box4.jpg" data-holder-rendered="true" style="height: 180px; width: 100%; display: block;">
            </a>
        </div>
    </div>
```

### 6.1.3 增加事件处理

```
var accttype = null;

$(".thumbnail").click(function(){
    $('.seltype').remove();
    $(this).prepend('<div class="glyphicon glyphicon-ok seltype"></div>');

    accttype = $(this).attr("accttype");

});
function doApply() {
    // 判断是否选择账户类型
    if ( accttype == null ) {
        // 提示错误信息
        layer.msg("请选择账户类型继续申请", {time:1500, icon:5, shift:6});
    } else {
        // 更新会员数据
        $.ajax({
            type : "POST",
            url  : "${APP_PATH}/member/updateAccttype",
```

```
                data : {
                    "accttype" : accttype
                },
                success : function( result ) {
                    if ( result.success ) {
                        // 跳转到下一步
                        window.location.href = "${APP_PATH}/member/apply";
                    } else {
                        layer.msg("账户类型更新失败", {time:1500, icon:5, shift:6});
                    }
                }
            });
        }
    }
```

### 6.1.4 更新账户类型

- 前台:

```
@ResponseBody
@RequestMapping("/updateAccttype")
public Object updateAccttype( HttpSession session, String accttype ) {
    start();
    try {
        Member loginMember = (Member)session.getAttribute("loginMember");
        loginMember.setAccttype(accttype);
        session.setAttribute("loginMember", loginMember); //Session 共享,需要同步 redis 缓存数据.
        int count = memberService.updateAccttype(loginMember);
        success(count == 1);
```

```
    } catch ( Exception e ) {
        e.printStackTrace();
        message(e.getMessage());
    }

    return end();
}
```

- 后台:

```
@RequestMapping("/member/updateAccttype")
public int updateAccttype(@RequestBody Member loginMember);
```

```
@RequestMapping("/member/updateAccttype")
public int updateAccttype(@RequestBody Member loginMember) {
    // 更新会员账户类型
    int cnt = memberService.updateAccttype(loginMember);
    // 更新流程审批单的流程步骤
    Ticket ticket = memberService.queryTicketByMemberid(loginMember.getId());
    ticket.setPstep("basicinfo");
    memberService.updateTicketPstep(ticket);
    // 让流程继续执行（taskService.complete(taskId)）
    Map<String, Object> varMap = new HashMap<String, Object>();
    varMap.put("piid", ticket.getPiid());
    varMap.put("loginacct", loginMember.getLoginacct());
    activitiService.completeTask(varMap); //服务间调用
    return cnt;
}
```

```
@Update("update t_member set accttype = #{accttype} where id = #{id}")
```

```
int updateAccttype(Member loginMember);
```

```
@Select("select * from t_ticket where memberid = #{memberid} and status = '0'")
Ticket queryTicketByMemberid(Integer id);
```

```
@Update("update t_ticket set pstep = #{pstep} where id = #{id}")
void updateTicketPstep(Ticket ticket);
```

- @EnableFeignClients

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-feign</artifactId>
</dependency>
```

```
@FeignClient("atcrowdfunding-activiti-service")
public interface ActivitiService {
    @RequestMapping("/act/completeTask")
    public void completeTask(@RequestBody Map<String, Object> varMap);
}
```

- package com.atguigu.atcrowdfunding.act.controller.ActivitiController

```
@RequestMapping("/completeTask")
public void completeTask(@RequestBody Map<String, Object> varMap) {
    TaskQuery query = taskService.createTaskQuery();
    Task task = query
                    .processInstanceId("" + varMap.get("piid"))
                    .taskAssignee("" + varMap.get("loginacct"))
                    .singleResult();
```

```
    taskService.complete(task.getId(), varMap);
}
```

- 问题 1:
  - 更新账户的类型,数据库的数据发生了变化,session 中的数据也应该发生变化,我们是否需要重新将 Member 数据存放到 session 中?
  - Session 共享,需要同步 redis 缓存数据.
- 问题 2:
  - 会员服务项目,调用流程服务项目;这里涉及两个服务项目之间的调用

### 6.1.5 数据库 accttype 字段值发生变化

- 增加 basicinfo.ftl 页面
- 测试
- 观察数据库数据变化,没有问题,继续 GoGoGo

# 第 7 章 会员实名认证-填写基本信息

说明：

- 完成基本信息提交（realname,cardno,tel），跳转到资质文件上传的页面：
- 注意:
  - 在会员表增加 tel 字段
- 更新会员信息
- 更新流程单步骤"pstep"="certupload"
- 完成当前任务节点
  - 设置流程变量${ flag == true}

## 7.1 实名认证申请的基本页面 basicinfo.ftl


## 7.2 会员实名认证-基本信息

①member/basicinfo.ftl

```
<form role="form" style="margin-top:20px;">
  <div class="form-group">
    <label for="exampleInputEmail1">真实名称</label>
    <input type="text" class="form-control" id="realname" placeholder="请输入真实名称">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">身份证号码</label>
    <input type="text" class="form-control" id="cardnum" placeholder="请输入身份证号码">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">电话号码</label>
```

```
            <input type="text" class="form-control" id="tel" placeholder="请输入电话号码">
        </div>
        <button type="button" onclick="window.location.href='accttype.html'" class="btn btn-default">上一步</button>
        <button type="button" onclick="nextStep()"  class="btn btn-success">下一步</button>
    </form>
```

```
function nextStep() {
    // 更新会员基本信息
    $.ajax({
        type : "POST",
        url  : "${APP_PATH}/member/updateBasicinfo",
        data : {
            "realname" : $("#realname").val(),
            "cardnum"  : $("#cardnum").val(),
            "tel"      : $("#tel").val()
        },
        success : function(result) {
            if ( result.success ) {
                // 跳转到下一步
                window.location.href = "${APP_PATH}/member/apply";
            } else {
                layer.msg("基本信息更新失败", {time:1500, icon:5, shift:6});
            }
        }
    });
}
```

```
@ResponseBody
@RequestMapping("/updateBasicinfo")
public Object updateBasicinfo( HttpSession session, Member member ) {
start();

try {
Member loginMember = (Member)session.getAttribute("loginMember");
loginMember.setRealname(member.getRealname());
loginMember.setCardnum(member.getCardnum());
loginMember.setTel(member.getTel());
// 使用 Session 共享组件，必须在更新数据时，通知 redis 组件同时进行更新
session.setAttribute("loginMember", loginMember);
int cnt = memberService.updateBasicinfo(loginMember);
success(cnt == 1);
} catch ( Exception e ) {
e.printStackTrace();
fail();
}

return end();
}
```

```
@RequestMapping("/member/updateBasicinfo")
public int updateBasicinfo(@RequestBody Member loginMember);
```

```
@RequestMapping("/member/updateBasicinfo")
public int updateBasicinfo(@RequestBody Member loginMember) {
// 更新会员基本信息
int cnt = memberService.updateBasicinfo(loginMember);
```

```
// 更新流程审批单的流程步骤
Ticket ticket = memberService.queryTicketByMemberid(loginMember.getId());
ticket.setPstep("certupload");
memberService.updateTicketPstep(ticket);
// 让流程继续执行（taskService.complete(taskId)）
Map<String, Object> varMap = new HashMap<String, Object>();
varMap.put("piid", ticket.getPiid());
varMap.put("loginacct", loginMember.getLoginacct());
varMap.put("flag", true);
activitiService.completeTask(varMap);
return cnt;
}
```

```
@Transactional
public int updateBasicinfo(Member loginMember) {
return memberDao.updateBasicinfo(loginMember);
}
```

```
@Update("update t_member set realname = #{realname}, cardnum = #{cardnum}, tel = #{tel} where id
= #{id}")
int updateBasicinfo(Member loginMember);
```

```
@FeignClient("atcrowdfunding-activiti-service")
public interface ActivitiService {
@RequestMapping("/act/completeTask")
public void completeTask(@RequestBody Map<String, Object> varMap);
}
```

- 增加/member/certupload.ftl
- 测试
- 观察数据库数据变化,没有问题,继续 GoGoGo

# 第 8 章 会员实名认证-资质文件上传(显示上传页面)

- 需求：

资质文件上传页面，迭代显示当前用户类型该上传的资质图片，并提交上传功能

- 注意:

需要增加图片预览功能.

静态资源分离

多条数据提交

提前完成"资质管理模块"和"资质与账户类型的分类模块".(可手动增加模拟数据代替功能开发,略过…)


## 8.1 准备模拟数据:t_account_type_cert

```
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('4','0','1');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('5','0','2');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('6','0','3');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('7','0','5');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('8','1','1');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('9','1','6');
```

```
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('10','2','6');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('11','2','7');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('12','3','5');
INSERT INTO `t_account_type_cert` (`id`, `accttype`, `certid`) VALUES('13','3','4');
```

## 8.2 根据当前会员的账户类型查询需要上传的资质信息

①增加 Cert 实体类

```
public class Cert implements Serializable {
    private Integer id;
    private String name;
    ...
}
```

②在跳转到 certupload.ftl 页面时需要查询用户的类型对应的上传的资质信息。

```
// 查询当前用户需要传递的资质文件
List<Cert> certs = memberService.queryCertsByAccttype(loginMember);
model.addAttribute("certs", certs);
```

③查询当前用户账户类型所对应的资质文件集合

```
@RequestMapping("/queryCertsByAccttype")
public List<Cert> queryCertsByAccttype(@RequestBody Member loginMember);
```

```
@RequestMapping("/queryCertsByAccttype")
public List<Cert> queryCertsByAccttype(@RequestBody Member loginMember) {
return memberService.queryCertsByAccttype(loginMember.getAccttype());
}
```

```
List<Cert> queryCertsByAccttype(String accttype);
```

④atcrowdfunding-boot-member-service/src/main/resources/mybatis/**mapper-member.xml**

```
<mapper namespace="com.atguigu.atcrowdfunding.member.dao.MemberDao">
    <select id="queryCertsByAccttype" resultType="com.atguigu.atcrowdfunding.common.bean.Cert">
        select
            *
        from t_cert
        where id in (
            select
                certid
            from t_account_type_cert
            where accttype = #{accttype}
        )
    </select>
</mapper>
```

## 8.3 迭代数据,显示表单页面/member/certupload.ft

```
        <form id="uploadForm" action=""    role="form" style="margin-top:20px;">
            <#list certs as cert>
            <div class="form-group">
              <label for="exampleInputEmail1">${cert.name}</label>
               <input type="file"    class="form-control" >
            </div>
             </#list>
            <button type="button" onclick="window.location.href='apply.html'" class="btn btn-default">上一步</button>
            <button type="button"    class="btn btn-success">下一步</button>
        </form>
```

# 第 9 章 会员实名认证-资质文件上传(预览)

说明:

文件预览通过 HTML5 新特效来进行实现.

## 9.1 表单页面，选择图片后的预览功能

①增加<img>标签，用于图片预览

```
<form id="uploadForm" action=""    role="form" style="margin-top:20px;">
    <#list certs as cert>
   <div class="form-group">
      <label for="exampleInputEmail1">${cert.name}</label>
      <input type="file"    class="form-control" >
      <img src="" style="display:none;">
   </div>
    </#list>
   <button  type="button"  onclick="window.location.href='apply.html'"  class="btn  btn-default"> 上 一 步
</button>
   <button type="button" onclick="nextStep()"    class="btn btn-success">下一步</button>
</form>
```

②事件处理（选择图片后进行预览）

```
$(":file").change(function(event){
      var files = event.target.files, file;
      if (files && files.length > 0) {
            file = files[0];
      }
      var URL = window.URL || window.webkitURL;
```

```
    var imgURL = URL.createObjectURL(file);


    var imgobj = $(this).next();
    imgobj.attr("src", imgURL);
    imgobj.show();
});
```

# 第 10 章 会员实名认证-资质文件上传(上传)

## 10.1 表单页面，选择图片后的预览功能

①增加<img>标签，用于图片预览

```
<form id="uploadForm" action="${APP_PATH}/member/uploadCerts" method="post"
 enctype="multipart/form-data" style="margin-top:20px;">
            <#list certs as cert>
            <div class="form-group">
              <label for="exampleInputEmail1">${cert.name}</label>
              <input type="hidden" name="mcs[${cert_index}].certid" value="${cert.id}">
              <input type="file" name="mcs[${cert_index}].certfile" class="form-control" >
              <img src="" style="display:none;">
            </div>
            <#list>
            <button type="button" onclick="window.location.href='apply.html'" class="btn btn-default">上一步</button>
            <button type="button" onclick="nextStep()"   class="btn btn-success">下一步</button>
</form>
```

## 10.2 提交表单，保存上传图片(多数据提交)

①定义实体类

```
public class MemberCert{
    private Integer memberid;
    private Integer certid;
    private String iconpath;
    private MultipartFile certfile;
}
```

```
public class Datas {
    private List<User> datas;
    private List<Integer> ids;
    private List<MemberCert> mcs;
}
```

②保存数据到 t_member_cert 表

```
CREATE TABLE t_member_cert
(
    ID                  INT(11) NOT NULL AUTO_INCREMENT,
    certid              INT(11) NOT NULL,
    memberid            INT(11) NOT NULL,
    iconpath            VARCHAR(255) NOT NULL,
    PRIMARY KEY (id)
);
```

①提交文件上传表单：member/certupload.ftl

<script src="${APP_PATH}/jquery/jquery-form.min.js"></script>

```
function nextStep() {
    $("#uploadForm").ajaxSubmit({
        success : function(result) {
            window.location.href = "${APP_PATH}/member/apply";
        }
    });
}
```

②处理请求：

- 前台:
- MemberController

```
@ResponseBody
@RequestMapping("/uploadCerts")
public Object uploadCerts( HttpSession session, Datas ds ) {
    start();

    try {
        Member loginMember = (Member)session.getAttribute("loginMember");
        // 保存文件
        // 增加数据
        List<MemberCert> mcs = ds.getMcs();
        //不往后台传 MultipartFile 对象,在前台直接进行上传文件保存操作.这里需要将图片保存特定路径下,采用静态资源分离加载图片.
        List<MemberCertFile> mcfs = new ArrayList<MemberCertFile>();
        for ( MemberCert mc : mcs ) {
            MemberCertFile mcf = new MemberCertFile();
            mcf.setMemberid(loginMember.getId());
            mcf.setCertid(mc.getCertid());
            String iconpath = "";
            // 文件处理 start
            MultipartFile file = mc.getCertfile();
            String fileName = file.getOriginalFilename();
            String suffix = fileName.substring(fileName.lastIndexOf("."));
            String uuid = UUID.randomUUID().toString();
            iconpath = uuid + suffix;//xxxx.jpg
```

```
                String path = "D:/resources/atcrowdfunding/img/cert/" + iconpath;
                //System.out.println("path = " + path);
                file.transferTo(new File(path));
                // 文件处理 end
                mcf.setIconpath(iconpath);
                mcfs.add(mcf);
            }
            memberService.insertMemberCerts(mcfs);
            success();
        } catch ( Exception e ) {
            e.printStackTrace();
            fail();
        }

        return end();
}
```

- MemberCertFile

```
public class MemberCertFile{
    private Integer memberid;
    private Integer certid;
    private String iconpath;
}
```

- MemberService

```
@RequestMapping("/member/insertMemberCerts")
public void insertMemberCerts(@RequestBody List<MemberCertFile> mcfs);
```

后台:

MemberController

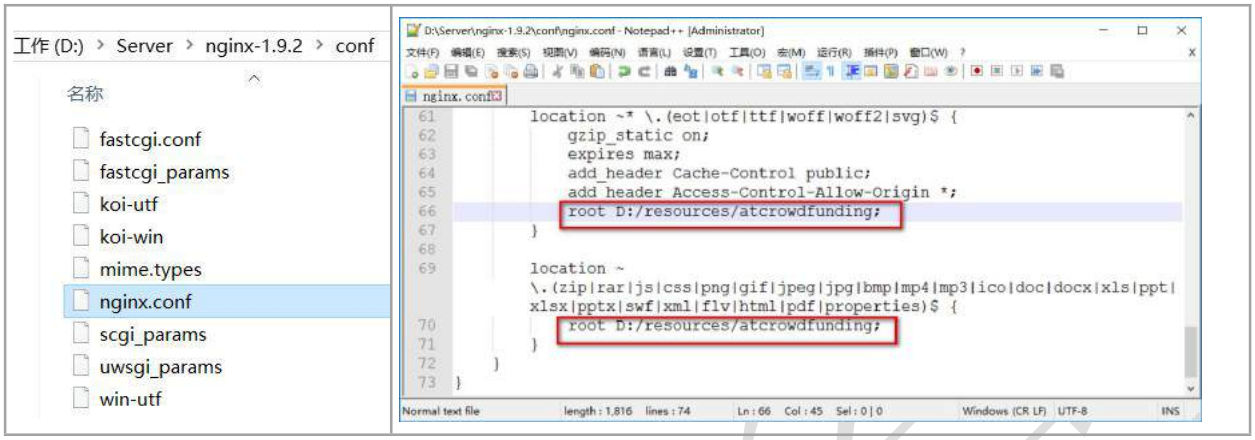
```
@RequestMapping("/member/insertMemberCerts")
public void insertMemberCerts(@RequestBody List<MemberCertFile> mcfs) {
    memberService.insertMemberCerts(mcfs);
    // 更新流程审批单的流程步骤
    Integer memberid = mcfs.get(0).getMemberid();
    Member loginMember = memberService.queryMemberById(memberid);
    Ticket ticket = memberService.queryTicketByMemberid(loginMember.getId());
    ticket.setPstep("checkemail");
    memberService.updateTicketPstep(ticket);
    // 让流程继续执行（taskService.complete(taskId)）
    Map<String, Object> varMap = new HashMap<String, Object>();
    varMap.put("piid", ticket.getPiid());
    varMap.put("loginacct", loginMember.getLoginacct());
    varMap.put("flag", true);
    activitiService.completeTask(varMap);
}
```

```
<insert id="insertMemberCerts">
    insert into t_member_cert(
        memberid, certid, iconpath
    ) values
    <foreach collection="mcs" item="mc" separator=",">
    ( #{mc.memberid}, #{mc.certid}, #{mc.iconpath} )
    </foreach>
</insert>
```

```
@Select("select * from t_member where id = #{id}")
Member queryMemberById(Integer memberid);
```

## 10.3 静态资源路径

- 在 nginx 配置文件中增加,设置静态资源路径




- D:\resources\atcrowdfunding\img\cert 用于存放上传资质图片.

# 第 11 章 会员实名认证-发送邮件验证码



## 11.1 认证邮箱页面:checkemail.ftl

```
<form role="form" style="margin-top:20px;">
  <div class="form-group">
    <label for="exampleInputEmail1">邮箱地址</label>
```

```
                <input type="text" class="form-control" id="email" value="${loginMember.email}"
placeholder="请输入用于接收验证码的邮箱地址">
            </div>
            <button type="button" onclick="window.location.href='apply-1.html'" class="btn btn-default">
上一步</button>
            <button type="button" onclick="nextStep()" class="btn btn-success">下一步</button>
        </form>
```

```
        function nextStep() {
            $.ajax({
                type : "POST",
                url  : "${APP_PATH}/member/sendAuthcode",
                data : {
                    "email" : $("#email").val()
                },
                success : function() {
                    window.location.href = "${APP_PATH}/member/apply";
                }
            });
        }
```

## 11.2 处理请求

- 前台:
- MemberController

```
@ResponseBody
@RequestMapping("/sendAuthcode")
public Object sendAuthcode(HttpSession session, String email) {
```

```
    start();

    try {
        Member loginMember = (Member)session.getAttribute("loginMember");
        loginMember.setEmail(email);
        session.setAttribute("loginMember", loginMember);
        memberService.updateEmail(loginMember);
        success(true);
    } catch ( Exception e ) {
        e.printStackTrace();
        success(false);
    }

    return end();
}
```

```
@RequestMapping("/member/updateEmail")
public void updateEmail(@RequestBody Member loginMember);
```

- 后台:

```
@RequestMapping("/member/updateEmail")
public void updateEmail(@RequestBody Member loginMember) {
    memberService.updateEmail(loginMember);
    Ticket ticket = memberService.queryTicketByMemberid(loginMember.getId());
    ticket.setPstep("checkauthcode");

    Map<String, Object> varMap = new HashMap<String, Object>();
    varMap.put("piid", ticket.getPiid());
```

```
        varMap.put("loginacct", loginMember.getLoginacct());
        varMap.put("flag", true);
        varMap.put("email", loginMember.getEmail());
        StringBuilder authcode = new StringBuilder();
        for ( int i =0; i< 4; i++ ) {
            authcode.append(new Random().nextInt(10));
        }
        ticket.setAuthcode(authcode.toString());
        memberService.updateTicket(ticket);

        varMap.put("authcode", authcode.toString());

        activitiService.completeTask(varMap);
}
```

- MemberDao

```
@Update("update t_member set email = #{email} where id = #{id}")
void updateEmail(Member loginMember);

@Select("select * from t_ticket where memberid = #{memberid} and status = '0'")
Ticket queryTicketByMemberid(Integer id);

@Update("update t_ticket set authcode = #{authcode}, pstep = #{pstep} where id = #{id}")
void updateTicket(Ticket ticket);
```

测试:系统自动发送验证码,验证用户邮箱地址有效性.

①注意启动邮箱服务器,否则,无法发送邮件.

②启动流程后,系统自动发送邮件,数据库数据变化

| id | memberid | piid | sta... | authc... | pstep |
|---|---|---|---|---|---|
| 1 | 1 | 101 | 0 | 3419 | checkemail |

③用户接收到验证码


④用户输入验证码,继续申请实名认证.


# 第 12 章 会员实名认证-验证邮件验证码

说明:

- 输入邮箱验证码进行验证
  - 如果验证码一致,
    - 修改用户的申请状态为:实名认证申请中
- 当用户提交验证码,点击[申请完成]

## 12.1 确认验证码

### 12.1.1 页面:checkapply.ftl

```
<form role="form" style="margin-top:20px;">
  <div class="form-group">
    <label for="exampleInputEmail1">验证码</label>
    <input type="text" class="form-control" id="authcode" placeholder="请输入你邮箱中接收到的验证码">
  </div>
  <button type="button" onclick="javascript:;" class="btn btn-primary">重新发送验证码</button>
  <button type="button" onclick="completeApply()" class="btn btn-success">申请完成</button>
</form>
```

```
function completeApply() {
    $.ajax({
        type : "POST",
        url  : "${APP_PATH}/member/completeApply",
        data : {
            "authcode" : $("#authcode").val()
        },
        success : function(result) {
            if (result.success) {
            layer.msg("实名认证申请结束，请等待审核", {time:1500, icon:6}, function(){
                window.location.href = "${APP_PATH}/member";
            });
```

```
                    } else {
                    layer.msg("验证码不正确，请重新输入", {time:1500, icon:5, shift:6}, function(){
                    });
                    }
                }
            });
        }
```

## 12.1.2 处理请求

```
@ResponseBody
@RequestMapping("/completeApply")
public Object completeApply( HttpSession session, String authcode ) {
    start();

    try {
        Member loginMember = (Member)session.getAttribute("loginMember");

        Ticket ticket = memberService.queryTicketByMemberid(loginMember.getId());
        if ( authcode.equals(ticket.getAuthcode()) ) {
            loginMember.setAuthstatus("1");

            memberService.completeApply(loginMember);
            session.setAttribute("loginMember", loginMember);
            success(true);
        } else {
            success(false);
        }
```

```
        } catch ( Exception e ) {
            e.printStackTrace();
            success(false);
        }
        return end();
}
```

```
@RequestMapping("/member/completeApply")
public void completeApply(@RequestBody Member loginMember);
```

```
@RequestMapping("/member/completeApply")
public void completeApply(@RequestBody Member loginMember) {
        memberService.updateAuthstatus(loginMember);

        Ticket ticket = memberService.queryTicketByMemberid(loginMember.getId());
        ticket.setStatus("1");
        memberService.updateTicketStatus(ticket);

        Map<String, Object> varMap = new HashMap<String, Object>();
        varMap.put("piid", ticket.getPiid());
        varMap.put("loginacct", loginMember.getLoginacct());
        varMap.put("flag", true);
        activitiService.completeTask(varMap);
}
```

执行更新会员申请状态

```
@Update("update t_member set authstatus = #{authstatus} where id = #{id}")
void updateAuthstatus(Member loginMember);
```

```
@Update("update t_ticket set status = #{status} where id = #{id}")
void updateTicketStatus(Ticket ticket);
```

## 12. 2 申请完成

### 12.2.1 用户申请状态为:[实名认证申请中]

# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

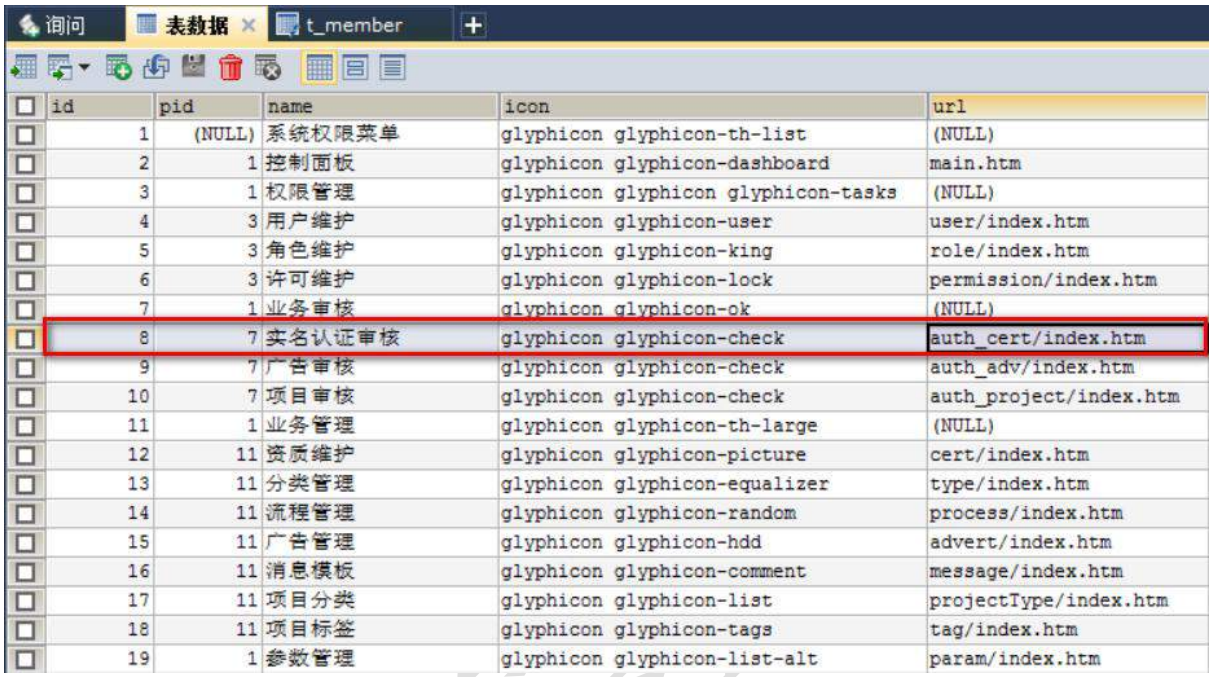
# 项目计划

- 实名认证审核

# 第 1 章 会员实名认证-后台审核-列表查询

需求:查询数据效果

## 1.1 增加链接:实名认证审核



询问 | 表数据 | t_member

| id | pid | name | icon | url |
|---|---|---|---|---|
| 1 | (NULL) | 系统权限菜单 | glyphicon glyphicon-th-list | (NULL) |
| 2 | 1 | 控制面板 | glyphicon glyphicon-dashboard | main.htm |
| 3 | 1 | 权限管理 | glyphicon glyphicon glyphicon-tasks | (NULL) |
| 4 | 3 | 用户维护 | glyphicon glyphicon-user | user/index.htm |
| 5 | 3 | 角色维护 | glyphicon glyphicon-king | role/index.htm |
| 6 | 3 | 许可维护 | glyphicon glyphicon-lock | permission/index.htm |
| 7 | 1 | 业务审核 | glyphicon glyphicon-ok | (NULL) |
| 8 | 7 | 实名认证审核 | glyphicon glyphicon-check | auth_cert/index.htm |
| 9 | 7 | 广告审核 | glyphicon glyphicon-check | auth_adv/index.htm |
| 10 | 7 | 项目审核 | glyphicon glyphicon-check | auth_project/index.htm |
| 11 | 1 | 业务管理 | glyphicon glyphicon-th-large | (NULL) |
| 12 | 11 | 资质维护 | glyphicon glyphicon-picture | cert/index.htm |
| 13 | 11 | 分类管理 | glyphicon glyphicon-equalizer | type/index.htm |
| 14 | 11 | 流程管理 | glyphicon glyphicon-random | process/index.htm |
| 15 | 11 | 广告管理 | glyphicon glyphicon-hdd | advert/index.htm |
| 16 | 11 | 消息模板 | glyphicon glyphicon-comment | message/index.htm |
| 17 | 11 | 项目分类 | glyphicon glyphicon-list | projectType/index.htm |
| 18 | 11 | 项目标签 | glyphicon glyphicon-tags | tag/index.htm |
| 19 | 1 | 参数管理 | glyphicon glyphicon-list-alt | param/index.htm |

## 1.2 点击链接【实名认证审核】跳转页面

atcrowdfunding-boot-manager

com.atguigu.atcrowdfunding.manager.controller.AuthController

```
package com.atguigu.atcrowdfunding.manager.controller;

...
@Controller
@RequestMapping("/auth")
public class AuthController extends BaseController {
	@RequestMapping("/index")
	public String index() {
		return "auth/index";
	}
}
```

## 1.3 增加跳转页面：index.ftl

```
<table class="table    table-bordered">
  <thead>
    <tr >
      <th width="30">#</th>
      <th>流程名称</th>
      <th>流程版本号</th>
      <th>任务名称</th>
      <th>申请会员</th>
      <th width="100">操作</th>
    </tr>
  </thead>
  <tbody id="userTbody">
  </tbody>
  <tfoot>
     <tr >
        <td colspan="6" align="center">
           <div id="Pagination" class="pagination" />
        </td>
     </tr>
  </tfoot>
</table>
```

```
<script type="text/javascript">
    $(function () {
        pageQuery(0);
    });

    function pageQuery(pageno) {
        var loadingIndex = -1;

        var obj = {
                "pageno" : pageno+1,
                "pagesize" : 10
        };

        $.ajax({
            url : "${APP_PATH}/auth/loadPageData",
            type : "POST",
            data : obj,
            beforeSend : function() {
```

```
                    loadingIndex = layer.msg('数据查询中', {icon: 16});
                    return true;
                },
                success : function( result ) {
                    layer.close(loadingIndex);
                    if ( result.success ) {
                        // 将查询结果循环遍历，将数据展现出来
                        var page = result.data;
                        var taskList = page.datas;

                        var content = "";
                        $.each(taskList, function(i, task){
            content += '<tr>';
            content += '    <td>'+(i+1)+'</td>';
            content += '    <td>'+task.pdname+'</td>';
            content += '    <td>'+task.pdversion+'</td>';
            content += '    <td>'+task.taskname+'</td>';
            content += '    <td>'+task.membername+'</td>';
            content += '    <td>';
            content += '<button type="button" onclick="window.location.href=\'/auth/detail?memberid='+task.memberid+'&taskid='+task.taskid+'\'" class="btn btn-success btn-xs"><i class=" glyphicon glyphicon-check"></i></button>';
            content += '    </td>';
            content += '</tr>';
                        });
                        $("tbody").html(content);

                        // 创建分页
                        var num_entries = page.totalsize ;
                        $("#Pagination").pagination(num_entries, {
                            num_edge_entries: 2, //边缘页数
                            num_display_entries: 4, //主体页数
                            callback: pageQuery, //查询当前页的数据.
                            items_per_page:page.pagesize, //每页显示 1 项
                            current_page:(page.pageno-1), //当前页,索引从 0 开始
                            prev_text:"上一页",
                            next_text:"下一页"
                        });
                    } else {
                        layer.msg("审核任务分页查询失败", {time:1000, icon:5, shift:6});
                    }
                }
```

```
                });
            }
        </script>
```

## 1.4 加载列表数据

前台:
atcrowdfunding-boot-manager
com.atguigu.atcrowdfunding.manager.controller.AuthController

```
@ResponseBody
@RequestMapping("/loadPageData")
public Object loadPageData( Integer pageno, Integer pagesize ) {
    start();

    try {

        Map<String, Object> varMap = new HashMap<String, Object>();
        varMap.put("start", (pageno-1)*pagesize);
        varMap.put("size", pagesize);
        List<Map<String, Object>> data = activitiService.queryTaskPageData(varMap);
        Integer count = (Integer)activitiService.queryTaskPageCount();

        Page<Map<String, Object>> page = new Page<Map<String, Object>>(pageno,pagesize);

        page.setDatas(data);
        page.setTotalsize(count);
        data(page);
        success();
    } catch ( Exception e ) {
        e.printStackTrace();
        fail();
    }

    return end();
}
```

```
@RequestMapping("/act/queryTaskPageCount")
public int queryTaskPageCount();
```

```
@RequestMapping("/act/queryTaskPageData")
public List<Map<String, Object>> queryTaskPageData( @RequestBody Map<String, Object> varMap );
```

后台:

- atcrowdfunding-boot-activiti-service
- com.atguigu.atcrowdfunding.act.controller.ActivitiController

```
@RequestMapping("/act/queryTaskPageCount")
public int queryTaskPageCount() {
    TaskQuery query = taskService.createTaskQuery();
    return (int)query.taskCandidateGroup("backcheck").count();
}

@RequestMapping("/act/queryTaskPageData")
public List<Map<String, Object>> queryTaskPageData( @RequestBody Map<String, Object> varMap ) {
    int start = (int)varMap.get("start");
    int size = (int)varMap.get("size");

    TaskQuery query = taskService.createTaskQuery();
    List<Task> tasks = query.taskCandidateGroup("backcheck").listPage(start, size);

    List<Map<String, Object>> taskMapList = new ArrayList<Map<String, Object>>();

    for ( Task task : tasks ) {
        Map<String, Object> taskMap = new HashMap<String, Object>();

        taskMap.put("taskid", task.getId());
        taskMap.put("taskname", task.getName());
        // task ==> pi ==> ticket ==> member
        // 会员名称
        String piid = task.getProcessInstanceId();
        Ticket ticket = memberService.queryTicketByPiid(piid);
        Member member = memberService.queryMemberById(ticket.getMemberid());
        taskMap.put("membername", member.getUsername());
        taskMap.put("memberid", member.getId());
        // task ==> pd
        // 流程定义名称
        String pdid = task.getProcessDefinitionId();
        ProcessDefinition pd =
            repositoryService
                .createProcessDefinitionQuery()
                .processDefinitionId(pdid).singleResult();
        taskMap.put("pdname", pd.getName());
        // 流程定义版本
```

```
            taskMap.put("pdversion", pd.getVersion());

            taskMapList.add(taskMap);
        }

        return taskMapList;
}
```

服务之间调用.

```
package com.atguigu.atcrowdfunding.act.service;
…
@FeignClient("atcrowdfunding-member-service")
public interface MemberService {
        @RequestMapping("/member/queryTicketByPiid/{piid}")
        Ticket queryTicketByPiid(@PathVariable("piid")String piid);

        @RequestMapping("/member/queryMemberById/{memberid}")
        Member queryMemberById(@PathVariable("memberid")Integer memberid);
}
```

```
@RequestMapping("/queryTicketByPiid/{piid}")
public Ticket queryTicketByPiid(@PathVariable("piid")String piid) {
        return memberService.queryTicketByPiid(piid);
}

@RequestMapping("/member/queryMemberById/{memberid}")
public Member queryMemberById(@PathVariable("memberid")Integer memberid) {
        return memberService.queryMemberById(memberid);
}
```

# 第 2 章 会员实名认证-后台审核-查看资质信息

## 2.1 增加链接

```
<button                                                                    type="button"
onclick="window.location.href=\'/auth/detail?memberid='+task.memberid+'&taskid='+task.taskid+'\'"
class="btn btn-success btn-xs"><i class=" glyphicon glyphicon-check"></i></button>
```

## 2.2 查看审核的资质

```
@RequestMapping("/detail")
public String detail(Integer memberid, String taskid, Model model) {
    // 将数据保存到请求范围中，可以在 freemarker 页面中直接通过${}访问，也可以通过${RequestParameters["xxx"]}访问
    model.addAttribute("memberid", memberid);
    model.addAttribute("taskid", taskid);

    // 查询申请会员提交的资质文件数据
    List<MemberCert> mcfs = memberService.queryMemberCertsByMemberId(memberid);
    model.addAttribute("mcfs", mcfs);
    return "auth/detail";
}
```

```
package com.atguigu.atcrowdfunding.manager.service;
…
@FeignClient("atcrowdfunding-member-service")
public interface MemberService {
    @RequestMapping("/member/queryMemberCertsByMemberId/{id}")
    List<MemberCert> queryMemberCertsByMemberId(@PathVariable("id")Integer memberid);
}
```

com.atguigu.atcrowdfunding.member.controller.MemberController

```
@RequestMapping("/member/queryMemberCertsByMemberId/{id}")
List<MemberCert> queryMemberCertsByMemberId(@PathVariable("id")Integer memberid) {
    return memberService.queryMemberCertsByMemberId(memberid);
}
```

```
    <select id="queryMemberCertsByMemberId"
      resultType="com.atguigu.atcrowdfunding.common.bean.MemberCert">
        select
            mc.*,
            c.name
        from t_member_cert mc
        join t_cert c on mc.certid = c.id
        where mc.memberid = #{memberid}
    </select>
```

3.后台:审核资质展示:member/detail.ftl

```
<div class="col-sm-9 col-sm-offset-3 col-md-10 col-md-offset-2 main">
    <div class="panel panel-default">
      <div class="panel-heading">
        <h3 class="panel-title"><i class="glyphicon glyphicon-th"></i> 申请详细信息</h3>
      </div>
      <div class="panel-body">
          <#list mcfs as mcf>
              <h1>${mcf.certname}</h1>
              <img src="${APP_PATH}/img/cert/${mcf.iconpath}">
          </#list>

        <button type="button" onclick="passApply()" class="btn btn-default btn-danger">
          <span class="glyphicon glyphicon-ok"></span> 通过
        </button>
        <button type="button" class="btn btn-default btn-danger">
          <span class="glyphicon glyphicon-remove"></span> 拒绝
        </button>
      </div>
    </div>
</div>
```

# 第 3 章 会员实名认证-后台审核-审核完成

## 3.1 审核按钮: "通过"

```
function passApply() {
    layer.confirm("审核通过实名认证申请，是否继续？",  {icon: 3, title:'提示'}, function(cindex) {
        $.ajax({
            type : "POST",
            url  : "${APP_PATH}/auth/ok",
            data : {
                "taskid" : "${taskid}",
                "memberid" : "${memberid}"
            },
            success : function(result) {
                if ( result.success ) {
                    layer.msg("实名认证申请审核通过", {time:1500, icon:6}, function(){
                        window.location.href = "${APP_PATH}/auth/index";
                    });
```

```
                    } else {
                        layer.msg("实名认证申请审核失败", {time:1500, icon:5, shift:6});
                    }
                }
            });
            layer.close(cindex);
        }, function(cindex){
            layer.close(cindex);
        });
}
```

## 3.2 审核处理

```
@ResponseBody
@RequestMapping("/ok")
public Object ok( String taskid, Integer memberid ) {
    start();

    try {
        // 通过认证申请
        Map<String, Object> varMap = new HashMap<String, Object>();
        varMap.put("taskid", taskid);
        varMap.put("memberid", memberid);

        activitiService.passApply(varMap);
        success();
    } catch ( Exception e ) {
        e.printStackTrace();
        fail();
    }

    return end();
}
```

```
@RequestMapping("/act/passApply")
public void passApply(@RequestBody Map<String, Object> varMap);
```

```
@RequestMapping("/act/passApply")
```

```
public void passApply(@RequestBody Map<String, Object> varMap) {
    // 完成任务
    String taskId = (String)varMap.get("taskid");
    varMap.put("flag", true);
    taskService.complete(taskId, varMap);
}
```

## 3.3 监听器

- atcrowdfunding-boot-activiti-service

```
package com.atguigu.atcrowdfunding.act.config;

import org.springframework.beans.BeansException;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ApplicationContextAware;
import org.springframework.stereotype.Component;

@Component
public class ApplicationContextUtils implements ApplicationContextAware {//接口注入.
    public static ApplicationContext applicationContext;

    public void setApplicationContext(ApplicationContext applicationContext)
throws BeansException {
        System.out.println( "applicationContext = " + applicationContext );
        ApplicationContextUtils.applicationContext = applicationContext;
    }
}
```

```
package com.atguigu.atcrowdfunding.act.listener;

import org.activiti.engine.delegate.DelegateExecution;
import org.activiti.engine.delegate.ExecutionListener;
import org.springframework.stereotype.Component;

import com.atguigu.atcrowdfunding.act.config.ApplicationContextConfig;
import com.atguigu.atcrowdfunding.act.service.MemberService;

@Component
public class PassListener implements ExecutionListener {
```

```
    public void notify(DelegateExecution execution) throws Exception {
        Integer id = (Integer)execution.getVariable("memberid");
        MemberService memberService =
 ApplicationContextUtils.applicationContext.getBean(MemberService.class);
        memberService.updateMemberAuthStatus(id);
    }
}
```

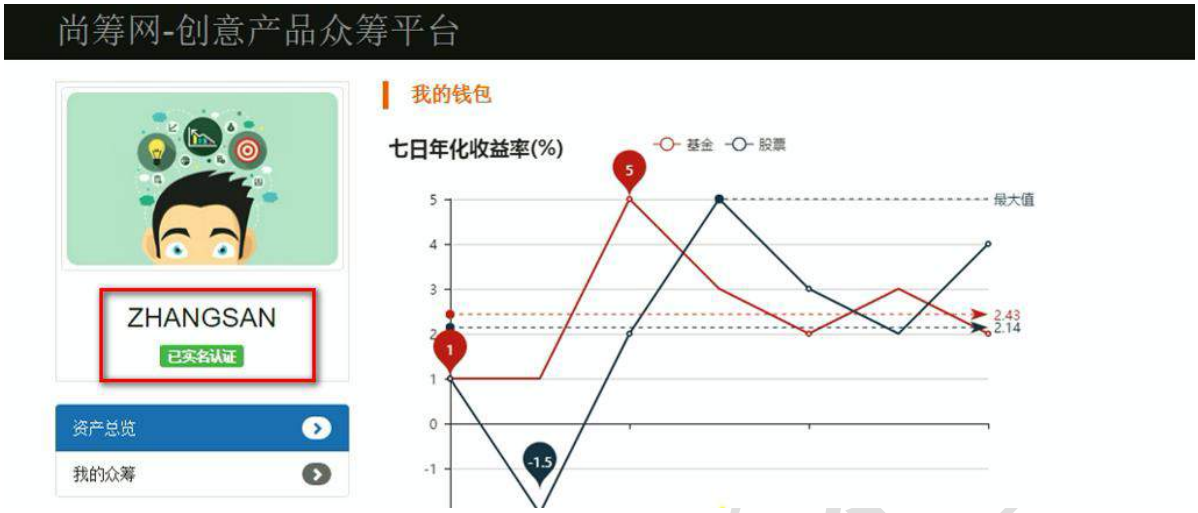
- atcrowdfunding-boot-activiti-service

```
package com.atguigu.atcrowdfunding.act.service;

...
@FeignClient("atcrowdfunding-member-service")
public interface MemberService {

@RequestMapping("/member/updateMemberAuthStatus/{id}")
void updateMemberAuthStatus(@PathVariable("id")int id);

}
```

```
@RequestMapping("/updateMemberAuthStatus/{id}")
public void updateMemberAuthStatus(@PathVariable("id")int id) {
memberService.updateMemberAuthStatus(id);
}
```

```
@Update("update t_member set authstatus = '2' where id = #{id}")
void updateMemberAuthStatus(int id);
```

## 3.4 会员重新登录，查看认证结果



# 项目课程系列之 Atcrowdfunding

尚硅谷 Java 研究院

版本：V1.0

# 第 1 章 项目总结

AtCrowdFunding 尚筹网

## 1.1 项目背景

### 1.1.1 互联网金融

### 1.1.2 互联网众筹平台

### 1.1.3 尚筹网

## 1.2 项目管理

### 2.2.1 软件工程( 6 个主要阶段 )

| 阶段 | 含义 |
| --- | --- |
| 制定计划 | 列出工程项目要做的主要工作和任务清单，要回答“工程项目做什么” |
| 需求分析 | 分析，了解，采集软件用户的想法和要求 |
| 软件设计 | 通过图形，文档的方式对软件的范围，功能，应用进行说明和描述 |

| | |
|---|---|
| 程序编写 | 采用编程或脚本语言实现用户的软件需求 |
| 软件测试 | 识别程序编写中的错误而执行的过程 |
| 运行维护 | 对软件产品运行所做的综合性管理过程 |

### 2.2.2 软件开发周期

基础时间 6~8 个月
如果拥有好的开发团队，和成熟的系统模板，时间可缩短为 2~3 个月

## 1.3 项目介绍

### 3.3.1 项目描述

互联网金融（ITFIN）是指传统金融机构与互联网企业利用互联网技术和信息通信技术实现资金融通、支付、投资和信息中介服务的新型金融业务模式。

众筹系统，是互联网金融的一种业务模式，可以在互联网上面向公众进行项目的融资，对于年轻的创业者而言，这是一个契机、一个机遇、一个开创自己事业的平台

正是因为这些利好，让更多的人愿意去运用众筹系统，作为其项目发展孵化的平台

尚筹网项目可以将创新项目发布到的平台上，并通过互联网的注册会员进行融资，达到项目孵化的目的

### 3.3.2 开发运行环境

| 分类 | 名称 |
|---|---|
| **操作系统** | Win8（10）(CentOS) |
| **JDK** | 1.7（1.8） |
| **Web 服务器** | Tomcat7（8），Nginx |
| **缓存服务器** | Memcached（Redis） |
| **数据库服务器** | MySQL5.5 |
| **项目管理工具** | Maven， SVN/Git |

| IDE | Eclipse（IDEA）/IDEA/STS |
|---|---|

## 3.3.3 软件架构

| 分类 | 名称 |
|---|---|
| 表现层框架 | SpringMVC4 |
| 业务（逻辑）层框架 | Spring4 |
| 持久层框架 | Mybatis3 |
| 权限模型 | RBAC |
| JS 框架 | JQuery2.X |
| HTML5（前端）框架 | Bootstrap3 |
| 流程框架 | Activiti5 |
| 分布式框架 | SpringBoot,SpringCloud |

## 3.3.4 系统功能模块


- 前台网站子系统

| 系统模块 | 说明 |
|---|---|

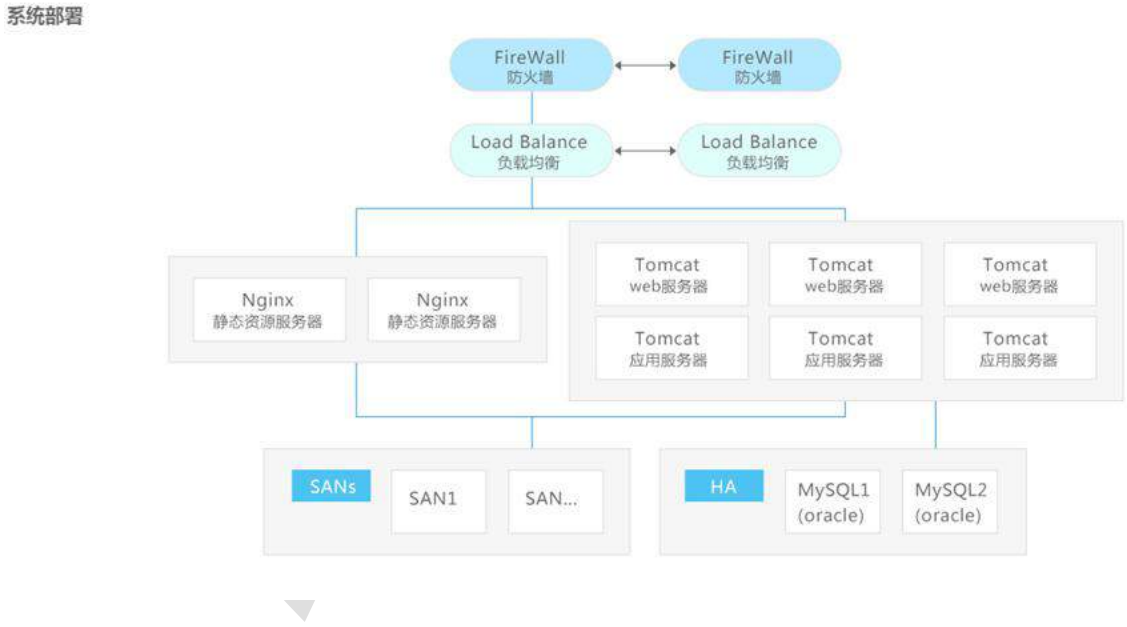
| | |
|---|---|
| **网站首页** | |
| **会员注册登录** | 基本的网站功能，含找回密码，认证激活等 |
| **会员中心** | 包含下列子模块：项目管理，我的项目，个人资料，消息中心，资金管理 |
| **创新创业项目** | 展示关注度比较高的项目信息 |
| **投资人** | 会员可查询当前网站认证的投资人，如果网站有好的投资人，就可以吸引好的项目和好的创业团队进驻网站 |
| **项目动态** | |
| **投资资讯** | |
| **关于我们** | |
| **帮助中心** | 会员可根据帮助，逐步进行项目的投资操作 |

- 后台管理子系统

| 系统模块 | 说明 |
|---|---|
| **创业者管理** | |
| **投资人管理** | |
| **项目管理** | 众筹项目的信息审核 |
| **投后管理** | |
| **财务管理** | |
| **信息管理** | 网站信息的发布，撤回 |

| | |
|---|---|
| **统计管理** | 系统数据统计分析，采用图表方式展示 |
| **权限管理** | 通过权限模型对用户功能进行限定 |
| **系统管理** | 系统配置参数管理 |
| **认证管理** | 实名认证资质管理以及资质审核 |
| **订单管理** | 项目订单的信息维护 |
| **流程管理** | 系统平台的流程维护及管理 |
| **回报管理** | 众筹项目的回报信息维护 |

- 系统部署架构




## 1.4 我的工作

- 创新项目的创建审核
- 项目投资
- 项目回报
- 项目订单
- 订单支付

- 实名认证申请及审核
- 流程配置
- 权限设定
- 广告

## 1.5 项目亮点

- 使用 nginx 作为服务器前端负载均衡
- 使用 memcached 实现服务器 session 数据共享
- 使用 memcached 作为数据库前置，缓存热点数据
- 使用 nginx 服务器进行动静资源分离
- 基于 Activiti5 的互联网众筹实名认证系统的设计与实现
- 热门互联网应用的设计与实现（全文检索，RPC 服务）
- 互联网分布式集群设计和应用（架构）
- 基于 SpringMVC + Spring + MyBatis 成熟框架的应用
- 互联网支付功能的实现
- 互联网短信登录验证的实现

## 1.6 技术清单

| 技术 | 说明 |
| --- | --- |
| Java | 编程语言 |
| Spring | Java 集成框架 |
| SpringMVC | Web 应用框架 |
| Mybatis | 数据库持久层框架 |
| Activiti5 | 工作流程框架 |
| SVN | 资源版本控制器 |
| maven | 软件项目管理工具 |
| Bootstrap | 前端框架 |

| | |
|---|---|
| JQuery | JS 脚本框架 |
| JQuery 插件 | |
| HTML5 | 前端网页编程语言 |
| James | 邮件服务器 |
| RBAC 权限模型 | |
| NGINX | web 负载均衡服务器 |
| Memcached(Redis) | 分布式内存对象缓存系统 |
| ztree | 前端树形结构控件 |
| UML | 统一建模工具（业务） |
| PowerDesigner | 数据库建模工具 |
| MySQL | 数据库 |

# 第 2 章 自我介绍&项目介绍