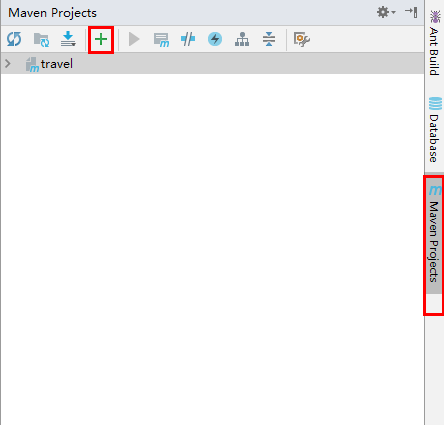
《黑马旅游网》综合案例

# 前言

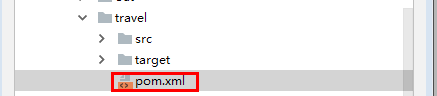
为了巩固web基础知识，提升综合运用能力，故而讲解此案例。要求，每位同学能够独立完成此案例。

# 项目导入

点击绿色＋按钮

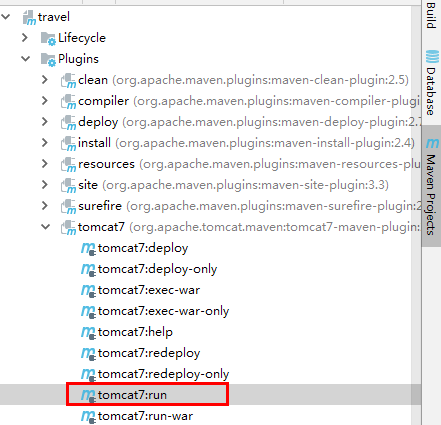


选择travel项目的pom.xml文件，点击ok，完成项目导入。需要等待一小会，项目初始化完成。

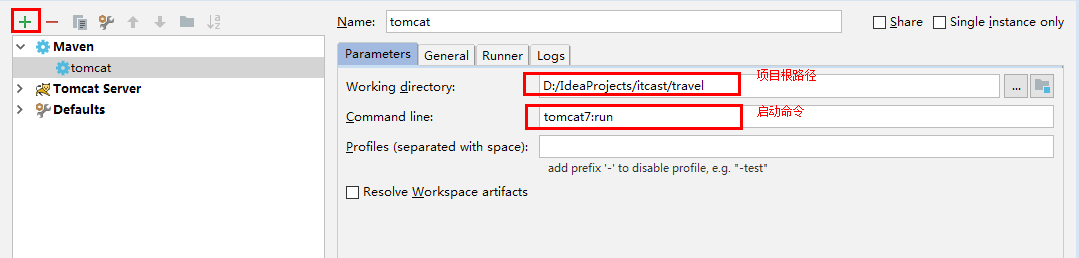


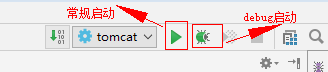
# 启动项目

## 方式一：



## 方式二：配置maven快捷启动





# 技术选型

## Web层

* 1. Servlet：前端控制器
  2. html：视图
  3. Filter：过滤器
  4. BeanUtils：数据封装
  5. Jackson：json序列化工具

## Service层

* 1. Javamail：java发送邮件工具
  2. Redis：nosql内存数据库
  3. Jedis：java的redis客户端

## Dao层

* 1. Mysql：数据库
  2. Druid：数据库连接池
  3. JdbcTemplate：jdbc的工具

# 创建数据库

-- 创建数据库

CREATE DATABASE travel;

-- 使用数据库

USE travel;

--创建表

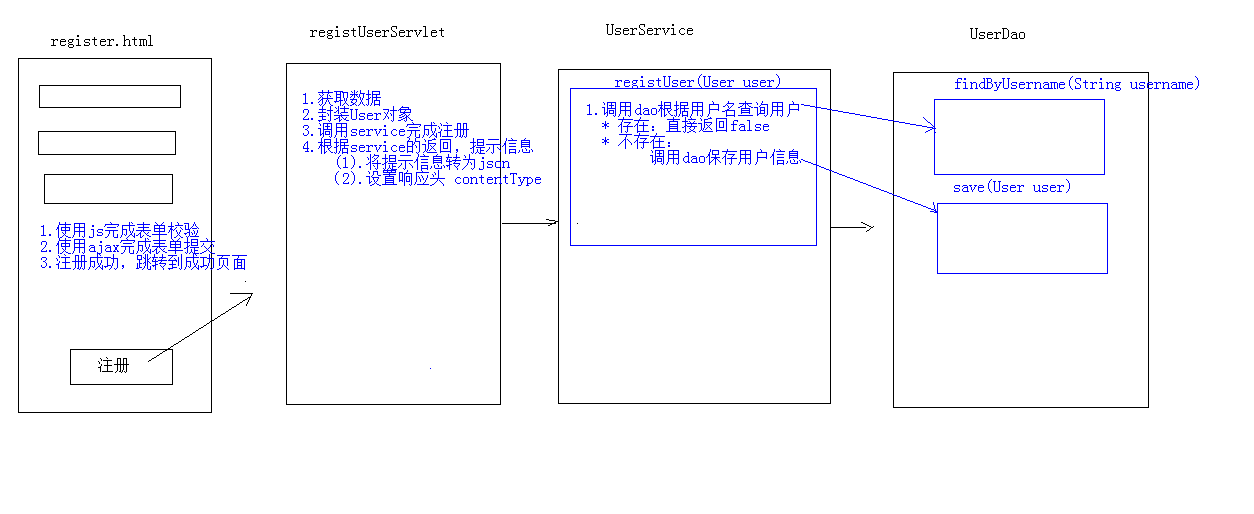
复制提供好的sql

# 注册功能

## 页面效果



## 功能分析



## 代码实现

### 前台代码实现

### 表单校验

提升用户体验，并减轻服务器压力。

//校验用户名  
//单词字符，长度8到20位  
**function** *checkUsername*() {  
 //1.获取用户名值  
 **var** username = **$**(**"#username"**).val();  
 //2.定义正则  
 **var** reg\_username = /^\w{8,20}$/;  
   
 //3.判断，给出提示信息  
 **var** flag = reg\_username.test(username);  
 **if**(flag){  
 //用户名合法  
 **$**(**"#username"**).css(**"border"**,**""**);  
 }**else**{  
 //用户名非法,加一个红色边框  
 **$**(**"#username"**).css(**"border"**,**"1px solid red"**);  
 }  
   
 **return** flag;  
 }  
  
 //校验密码  
 **function** *checkPassword*() {  
 //1.获取密码值  
 **var** password = **$**(**"#password"**).val();  
 //2.定义正则  
 **var** reg\_password = /^\w{8,20}$/;  
  
 //3.判断，给出提示信息  
 **var** flag = reg\_password.test(password);  
 **if**(flag){  
 //密码合法  
 **$**(**"#password"**).css(**"border"**,**""**);  
 }**else**{  
 //密码非法,加一个红色边框  
 **$**(**"#password"**).css(**"border"**,**"1px solid red"**);  
 }  
  
 **return** flag;  
 }  
  
 //校验邮箱  
**function** *checkEmail*(){  
 //1.获取邮箱  
 **var** email = **$**(**"#email"**).val();  
 //2.定义正则 itcast@163.com  
 **var** reg\_email = /^\w+@\w+\.\w+$/;  
  
 //3.判断  
 **var** flag = reg\_email.test(email);  
 **if**(flag){  
 **$**(**"#email"**).css(**"border"**,**""**);  
 }**else**{  
 **$**(**"#email"**).css(**"border"**,**"1px solid red"**);  
 }  
  
 **return** flag;  
}  
  
**$**(**function** () {  
 //当表单提交时，调用所有的校验方法  
 **$**(**"#registerForm"**).submit(**function**(){  
  
 **return** *checkUsername*() && *checkPassword*() && *checkEmail*();  
 //如果这个方法没有返回值，或者返回为true，则表单提交，如果返回为false，则表单不提交  
 });  
  
 //当某一个组件失去焦点是，调用对应的校验方法  
 **$**(**"#username"**).blur(*checkUsername*);  
 **$**(**"#password"**).blur(*checkPassword*);  
 **$**(**"#email"**).blur(*checkEmail*);  
  
  
 });

### 异步(ajax)提交表单

在此使用异步提交表单是为了获取服务器响应的数据。因为我们前台使用的是html作为视图层，不能够直接从servlet相关的域对象获取值，只能通过ajax获取响应数据



### 后台代码实现

### 编写RegistUserServlet

@WebServlet(**"/registUserServlet"**)  
**public class** RegistUserServlet **extends** HttpServlet {  
 **protected void** doPost(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
  
  
 //验证校验  
 String check = request.getParameter(**"check"**);  
 //从sesion中获取验证码  
 HttpSession session = request.getSession();  
 String checkcode\_server = (String) session.getAttribute(**"CHECKCODE\_SERVER"**);  
 session.removeAttribute(**"CHECKCODE\_SERVER"**);//为了保证验证码只能使用一次  
 //比较  
 **if**(checkcode\_server == **null** || !checkcode\_server.equalsIgnoreCase(check)){  
 //验证码错误  
 ResultInfo info = **new** ResultInfo();  
 //注册失败  
 info.setFlag(**false**);  
 info.setErrorMsg(**"验证码错误"**);  
 //将info对象序列化为json  
 ObjectMapper mapper = **new** ObjectMapper();  
 String json = mapper.writeValueAsString(info);  
 response.setContentType(**"application/json;charset=utf-8"**);  
 response.getWriter().write(json);  
 **return**;  
 }  
  
 //1.获取数据  
 Map<String, String[]> map = request.getParameterMap();  
  
 //2.封装对象  
 User user = **new** User();  
 **try** {  
 BeanUtils.*populate*(user,map);  
 } **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
 }  
  
 //3.调用service完成注册  
 UserService service = **new** UserServiceImpl();  
 **boolean** flag = service.regist(user);  
 ResultInfo info = **new** ResultInfo();  
 //4.响应结果  
 **if**(flag){  
 //注册成功  
 info.setFlag(**true**);  
 }**else**{  
 //注册失败  
 info.setFlag(**false**);  
 info.setErrorMsg(**"注册失败!"**);  
 }  
  
 //将info对象序列化为json  
 ObjectMapper mapper = **new** ObjectMapper();  
 String json = mapper.writeValueAsString(info);  
  
 //将json数据写回客户端  
 //设置content-type  
 response.setContentType(**"application/json;charset=utf-8"**);  
 response.getWriter().write(json);  
  
  
 }  
  
 **protected void** doGet(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 **this**.doPost(request, response);  
 }  
}

### 编写UserService以及UserServiceImpl

**public class** UserServiceImpl **implements** UserService {  
  
 **private** UserDao **userDao** = **new** UserDaoImpl();  
 /\*\*  
 \* 注册用户  
 \* **@param *user*** \* **@return** \*/  
 @Override  
 **public boolean** regist(User user) {  
 //1.根据用户名查询用户对象  
 User u = **userDao**.findByUsername(user.getUsername());  
 //判断u是否为null  
 **if**(u != **null**){  
 //用户名存在，注册失败  
 **return false**;  
 }  
 //2.保存用户信息  
 **userDao**.save(user);  
 **return true**;  
 }  
}

### 编写UserDao以及UserDaoImpl

**public class** UserDaoImpl **implements** UserDao {  
  
 **private** JdbcTemplate **template** = **new** JdbcTemplate(JDBCUtils.*getDataSource*());  
  
 @Override  
 **public** User findByUsername(String username) {  
 User user = **null**;  
 **try** {  
 //1.定义sql  
 String sql = **"select** *\** **from tab\_user where username = ?"**;  
 //2.执行sql  
 user = **template**.queryForObject(sql, **new** BeanPropertyRowMapper<User>(User.**class**), username);  
 } **catch** (Exception e) {  
  
 }  
  
 **return** user;  
 }  
  
 @Override  
 **public void** save(User user) {  
 //1.定义sql  
 String sql = **"insert into tab\_user(username,password,name,birthday,sex,telephone,email) values(?,?,?,?,?,?,?)"**;  
 //2.执行sql  
  
 **template**.update(sql,user.getUsername(),  
 user.getPassword(),  
 user.getName(),  
 user.getBirthday(),  
 user.getSex(),  
 user.getTelephone(),  
 user.getEmail());  
 }  
}

### 邮件激活

为什么要进行邮件激活？为了保证用户填写的邮箱是正确的。将来可以推广一些宣传信息，到用户邮箱中。

### 发送邮件

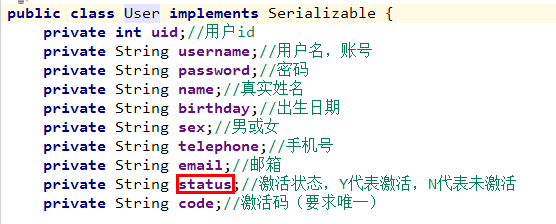
1. 申请邮箱
2. 开启授权码
3. 在MailUtils中设置自己的邮箱账号和密码(授权码)



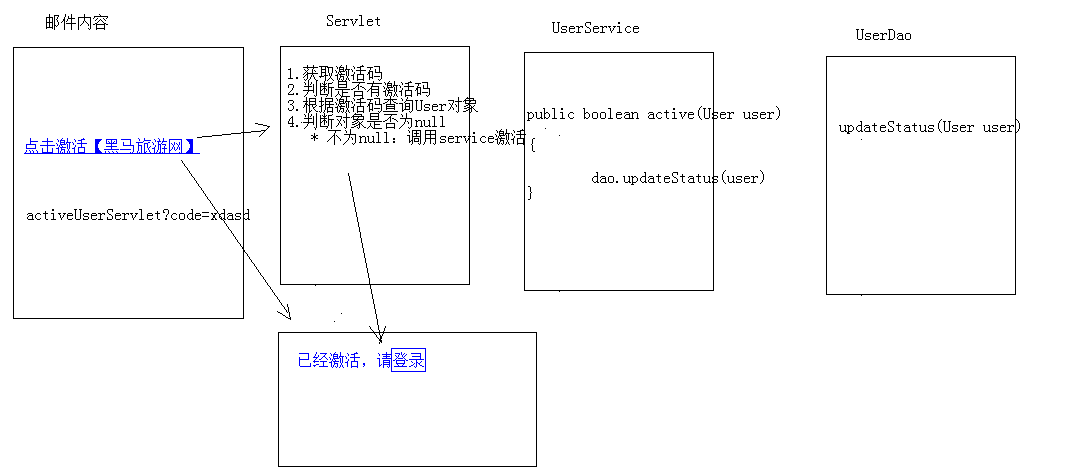
邮件工具类：MailUtils，调用其中sendMail方法可以完成邮件发送

### 用户点击邮件激活

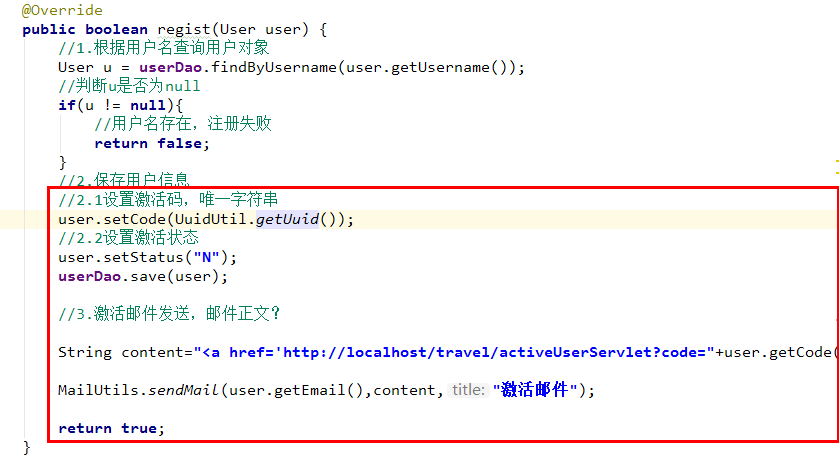
经过分析，发现，用户激活其实就是修改用户表中的status为‘Y’



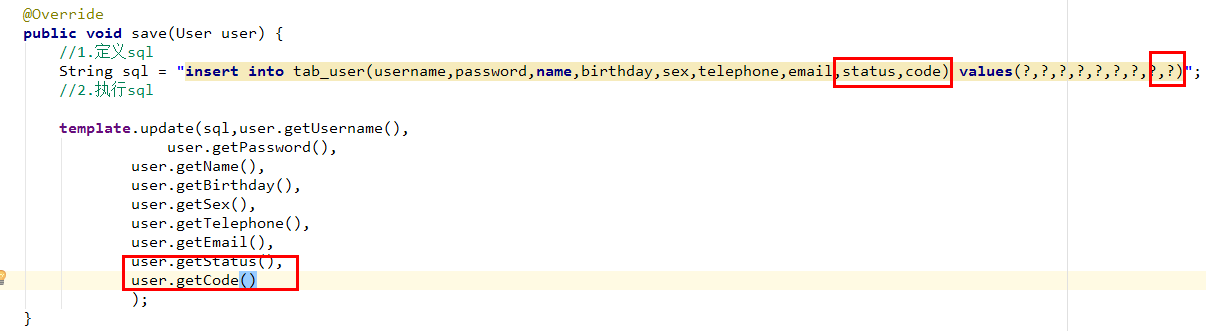
分析：



发送邮件代码：



修改保存Dao代码，加上存储status和code 的代码逻辑



激活代码实现：

ActiveUserServlet

//1.获取激活码  
String code = request.getParameter(**"code"**);  
**if**(code != **null**){  
 //2.调用service完成激活  
 UserService service = **new** UserServiceImpl();  
 **boolean** flag = service.active(code);  
  
 //3.判断标记  
 String msg = **null**;  
 **if**(flag){  
 //激活成功  
 msg = **"激活成功，请<a href='login.html'>登录</a>"**;  
 }**else**{  
 //激活失败  
 msg = **"激活失败，请联系管理员!"**;  
 }  
 response.setContentType(**"text/html;charset=utf-8"**);  
 response.getWriter().write(msg);

UserService：active

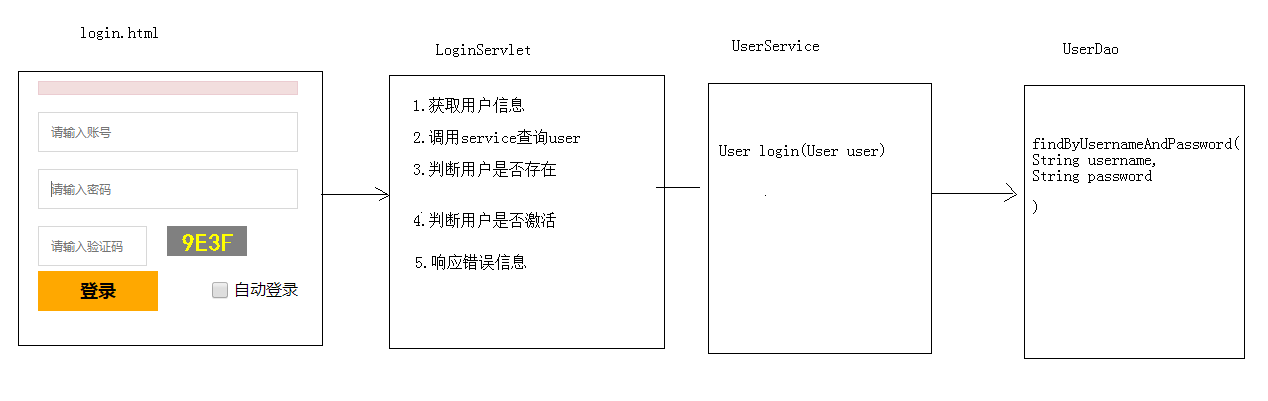
@Override  
**public boolean** active(String code) {  
 //1.根据激活码查询用户对象  
 User user = **userDao**.findByCode(code);  
 **if**(user != **null**){  
 //2.调用dao的修改激活状态的方法  
 **userDao**.updateStatus(user);  
 **return true**;  
 }**else**{  
 **return false**;  
 }  
  
}

UserDao：findByCode updateStatus

/\*\*  
 \* 根据激活码查询用户对象  
 \* **@param *code*** \* **@return** \*/  
@Override  
**public** User findByCode(String code) {  
 User user = **null**;  
 **try** {  
 String sql = **"select** *\** **from tab\_user where code = ?"**;  
  
 user = **template**.queryForObject(sql,**new** BeanPropertyRowMapper<User>(User.**class**),code);  
 } **catch** (DataAccessException e) {  
 e.printStackTrace();  
 }  
  
 **return** user;  
}  
  
/\*\*  
 \* 修改指定用户激活状态  
 \* **@param *user*** \*/  
@Override  
**public void** updateStatus(User user) {  
 String sql = **" update tab\_user set status = 'Y' where uid=?"**;  
 **template**.update(sql,user.getUid());  
}

# 登录

## 分析



## 代码实现

### 前台代码



### 后台代码

LoginServlet

//1.获取用户名和密码数据  
Map<String, String[]> map = request.getParameterMap();  
//2.封装User对象  
User user = **new** User();  
**try** {  
 BeanUtils.*populate*(user,map);  
} **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
} **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
}  
  
//3.调用Service查询  
UserService service = **new** UserServiceImpl();  
User u = service.login(user);  
  
ResultInfo info = **new** ResultInfo();  
  
//4.判断用户对象是否为null  
**if**(u == **null**){  
 //用户名密码或错误  
 info.setFlag(**false**);  
 info.setErrorMsg(**"用户名密码或错误"**);  
}  
//5.判断用户是否激活  
**if**(u != **null** && !**"Y"**.equals(u.getStatus())){  
 //用户尚未激活  
 info.setFlag(**false**);  
 info.setErrorMsg(**"您尚未激活，请激活"**);  
}  
//6.判断登录成功  
**if**(u != **null** && **"Y"**.equals(u.getStatus())){  
 //登录成功  
 info.setFlag(**true**);  
}  
  
//响应数据  
ObjectMapper mapper = **new** ObjectMapper();  
  
response.setContentType(**"application/json;charset=utf-8"**);  
mapper.writeValue(response.getOutputStream(),info);

UserService

**public** User login(User user) {  
 **return userDao**.findByUsernameAndPassword(user.getUsername(),user.getPassword());  
}

UserDao

**public** User findByUsernameAndPassword(String username, String password) {  
 User user = **null**;  
 **try** {  
 //1.定义sql  
 String sql = **"select** *\** **from tab\_user where username = ? and password = ?"**;  
 //2.执行sql  
 user = **template**.queryForObject(sql, **new** BeanPropertyRowMapper<User>(User.**class**), username,password);  
 } **catch** (Exception e) {  
  
 }  
  
 **return** user;  
}

### index页面中用户姓名的提示信息功能

效果：



header.html代码



Servlet代码

//从session中获取登录用户  
Object user = request.getSession().getAttribute(**"user"**);  
//将user写回客户端  
  
ObjectMapper mapper = **new** ObjectMapper();  
response.setContentType(**"application/json;charset=utf-8"**);  
mapper.writeValue(response.getOutputStream(),user);

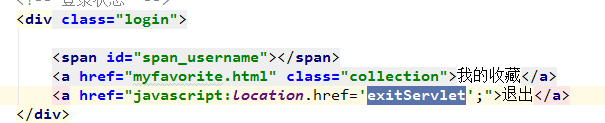
# 退出

什么叫做登录了？session中有user对象。

实现步骤：

* + 1. 访问servlet，将session销毁
    2. 跳转到登录页面

代码实现：

Header.html

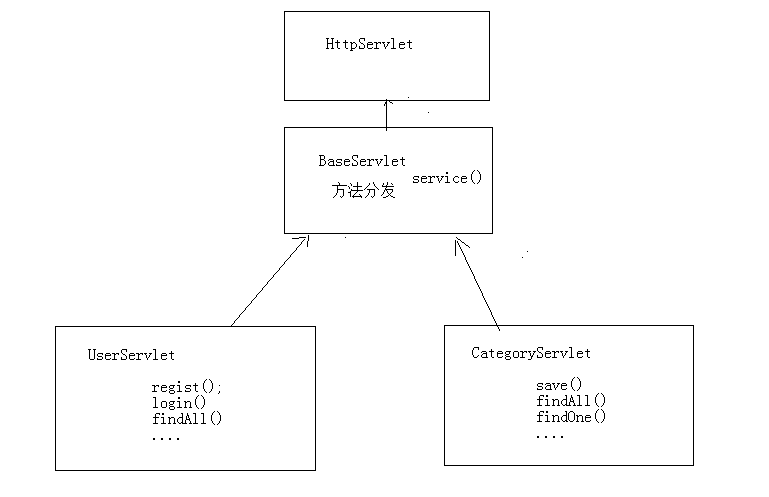
Servlet:

//1.销毁session  
request.getSession().invalidate();  
  
//2.跳转登录页面  
response.sendRedirect(request.getContextPath()+**"/login.html"**);

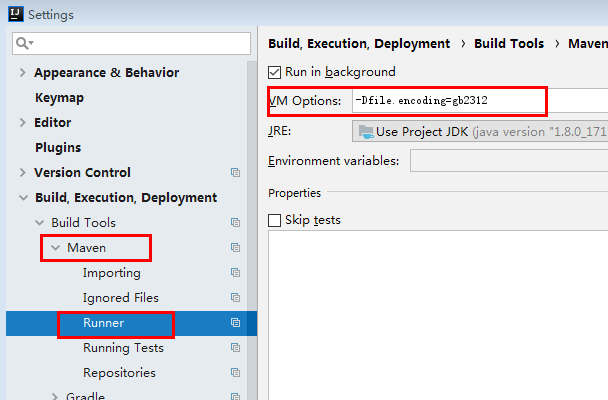
# 优化Servlet

## 目的

减少Servlet的数量，现在是一个功能一个Servlet，将其优化为一个模块一个Servlet，相当于在数据库中一张表对应一个Servlet，在Servlet中提供不同的方法，完成用户的请求。



Idea控制台中文乱码解决：-Dfile.encoding=gb2312



## BaseServlet编写：

**public class** BaseServlet **extends** HttpServlet {  
  
  
 @Override  
 **protected void** service(HttpServletRequest req, HttpServletResponse resp) **throws** ServletException, IOException {  
 //System.out.println("baseServlet的service方法被执行了...");  
  
 //完成方法分发  
 //1.获取请求路径  
 String uri = req.getRequestURI(); // /travel/user/add  
 System.***out***.println(**"请求uri:"**+uri);// /travel/user/add  
 //2.获取方法名称  
 String methodName = uri.substring(uri.lastIndexOf(**'/'**) + 1);  
 System.***out***.println(**"方法名称："**+methodName);  
 //3.获取方法对象Method  
 //谁调用我？我代表谁  
 System.***out***.println(**this**);//UserServlet的对象cn.itcast.travel.web.servlet.UserServlet@4903d97e  
 **try** {  
 //获取方法  
 Method method = **this**.getClass().getMethod(methodName, HttpServletRequest.**class**, HttpServletResponse.**class**);  
 //4.执行方法  
 //暴力反射  
 //method.setAccessible(true);  
 method.invoke(**this**,req,resp);  
 } **catch** (NoSuchMethodException e) {  
 e.printStackTrace();  
 } **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
 }  
  
  
 }  
}

## UserServlet改写

将之前的Servlet实现的功能，抽取到UserServlet中的不同方法中实现，并且将UserService创建抽取到成员变量位置

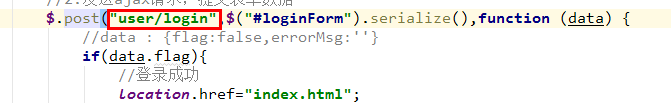
@WebServlet(**"/user/\*"**) // /user/add /user/find  
**public class** UserServlet **extends** BaseServlet {  
  
 //声明UserService业务对象  
 **private** UserService **service** = **new** UserServiceImpl();  
  
 /\*\*  
 \* 注册功能  
 \* **@param *request*** \* **@param *response*** \* **@throws** ServletException  
 \* **@throws** IOException  
 \*/  
 **public void** regist(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
  
 //验证校验  
 String check = request.getParameter(**"check"**);  
 //从sesion中获取验证码  
 HttpSession session = request.getSession();  
 String checkcode\_server = (String) session.getAttribute(**"CHECKCODE\_SERVER"**);  
 session.removeAttribute(**"CHECKCODE\_SERVER"**);//为了保证验证码只能使用一次  
 //比较  
 **if**(checkcode\_server == **null** || !checkcode\_server.equalsIgnoreCase(check)){  
 //验证码错误  
 ResultInfo info = **new** ResultInfo();  
 //注册失败  
 info.setFlag(**false**);  
 info.setErrorMsg(**"验证码错误"**);  
 //将info对象序列化为json  
 ObjectMapper mapper = **new** ObjectMapper();  
 String json = mapper.writeValueAsString(info);  
 response.setContentType(**"application/json;charset=utf-8"**);  
 response.getWriter().write(json);  
 **return**;  
 }  
  
 //1.获取数据  
 Map<String, String[]> map = request.getParameterMap();  
  
 //2.封装对象  
 User user = **new** User();  
 **try** {  
 BeanUtils.*populate*(user,map);  
 } **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
 }  
  
 //3.调用service完成注册  
 //UserService service = new UserServiceImpl();  
 **boolean** flag = **service**.regist(user);  
 ResultInfo info = **new** ResultInfo();  
 //4.响应结果  
 **if**(flag){  
 //注册成功  
 info.setFlag(**true**);  
 }**else**{  
 //注册失败  
 info.setFlag(**false**);  
 info.setErrorMsg(**"注册失败!"**);  
 }  
  
 //将info对象序列化为json  
 ObjectMapper mapper = **new** ObjectMapper();  
 String json = mapper.writeValueAsString(info);  
  
 //将json数据写回客户端  
 //设置content-type  
 response.setContentType(**"application/json;charset=utf-8"**);  
 response.getWriter().write(json);  
  
 }  
  
 /\*\*  
 \* 登录功能  
 \* **@param *request*** \* **@param *response*** \* **@throws** ServletException  
 \* **@throws** IOException  
 \*/  
 **public void** login(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 //1.获取用户名和密码数据  
 Map<String, String[]> map = request.getParameterMap();  
 //2.封装User对象  
 User user = **new** User();  
 **try** {  
 BeanUtils.*populate*(user,map);  
 } **catch** (IllegalAccessException e) {  
 e.printStackTrace();  
 } **catch** (InvocationTargetException e) {  
 e.printStackTrace();  
 }  
  
 //3.调用Service查询  
 // UserService service = new UserServiceImpl();  
 User u = **service**.login(user);  
  
 ResultInfo info = **new** ResultInfo();  
  
 //4.判断用户对象是否为null  
 **if**(u == **null**){  
 //用户名密码或错误  
 info.setFlag(**false**);  
 info.setErrorMsg(**"用户名密码或错误"**);  
 }  
 //5.判断用户是否激活  
 **if**(u != **null** && !**"Y"**.equals(u.getStatus())){  
 //用户尚未激活  
 info.setFlag(**false**);  
 info.setErrorMsg(**"您尚未激活，请激活"**);  
 }  
 //6.判断登录成功  
 **if**(u != **null** && **"Y"**.equals(u.getStatus())){  
 request.getSession().setAttribute(**"user"**,u);//登录成功标记  
  
 //登录成功  
 info.setFlag(**true**);  
 }  
  
 //响应数据  
 ObjectMapper mapper = **new** ObjectMapper();  
  
 response.setContentType(**"application/json;charset=utf-8"**);  
 mapper.writeValue(response.getOutputStream(),info);  
 }  
  
 /\*\*  
 \* 查询单个对象  
 \* **@param *request*** \* **@param *response*** \* **@throws** ServletException  
 \* **@throws** IOException  
 \*/  
 **public void** findOne(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 //从session中获取登录用户  
 Object user = request.getSession().getAttribute(**"user"**);  
 //将user写回客户端  
  
 ObjectMapper mapper = **new** ObjectMapper();  
 response.setContentType(**"application/json;charset=utf-8"**);  
 mapper.writeValue(response.getOutputStream(),user);  
 }  
  
 /\*\*  
 \* 退出功能  
 \* **@param *request*** \* **@param *response*** \* **@throws** ServletException  
 \* **@throws** IOException  
 \*/  
 **public void** exit(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 //1.销毁session  
 request.getSession().invalidate();  
  
 //2.跳转登录页面  
 response.sendRedirect(request.getContextPath()+**"/login.html"**);  
 }  
  
 /\*\*  
 \* 激活功能  
 \* **@param *request*** \* **@param *response*** \* **@throws** ServletException  
 \* **@throws** IOException  
 \*/  
 **public void** active(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 //1.获取激活码  
 String code = request.getParameter(**"code"**);  
 **if**(code != **null**){  
 //2.调用service完成激活  
 //UserService service = new UserServiceImpl();  
 **boolean** flag = **service**.active(code);  
  
 //3.判断标记  
 String msg = **null**;  
 **if**(flag){  
 //激活成功  
 msg = **"激活成功，请<a href='login.html'>登录</a>"**;  
 }**else**{  
 //激活失败  
 msg = **"激活失败，请联系管理员!"**;  
 }  
 response.setContentType(**"text/html;charset=utf-8"**);  
 response.getWriter().write(msg);  
 }  
 }  
}

## 页面路径改写

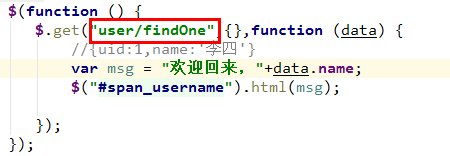
register.html



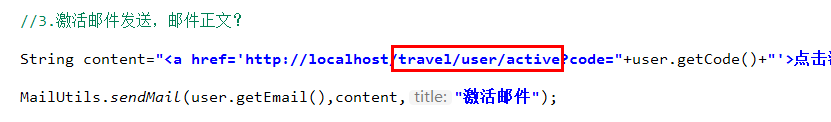
login.html



header.html



UserServiceImpl发送邮件

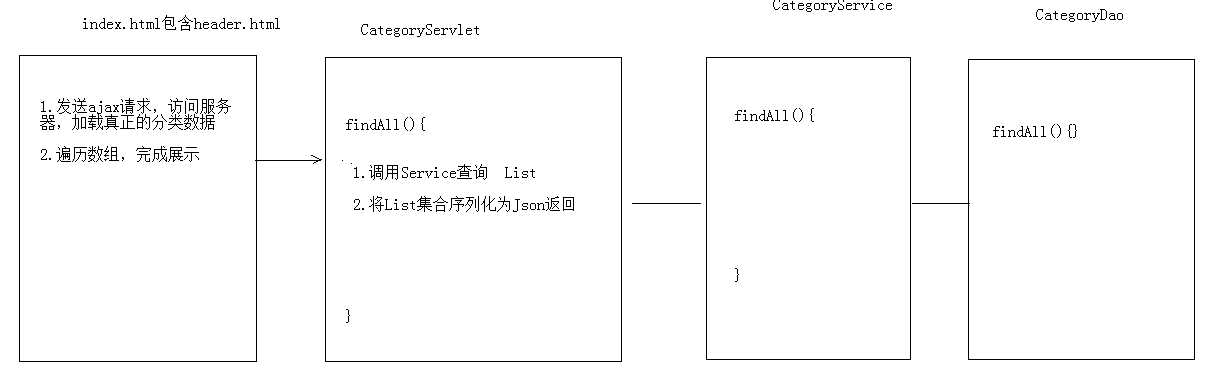


# 分类数据展示

## 效果：



## 分析：



## 代码实现：

### 后台代码

CategoryServlet

@WebServlet(**"/category/\*"**)  
**public class** CategoryServlet **extends** BaseServlet {  
  
 **private** CategoryService **service** = **new** CategoryServiceImpl();  
  
 /\*\*  
 \* 查询所有  
 \* **@param *request*** \* **@param *response*** \* **@throws** ServletException  
 \* **@throws** IOException  
 \*/  
 **public void** findAll(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 //1.调用service查询所有  
 List<Category> cs = **service**.findAll();  
 //2.序列化json返回  
 /\* ObjectMapper mapper = new ObjectMapper();  
 response.setContentType("application/json;charset=utf-8");  
 mapper.writeValue(response.getOutputStream(),cs);\*/  
 writeValue(cs,response);  
  
 }  
  
}

CategoryService

**public class** CategoryServiceImpl **implements** CategoryService {  
  
 **private** CategoryDao **categoryDao** = **new** CategoryDaoImpl();  
  
 @Override  
 **public** List<Category> findAll() {  
 **return categoryDao**.findAll();  
 }  
}

CategoryDao

**public class** CategoryDaoImpl **implements** CategoryDao {  
  
 **private** JdbcTemplate **template** = **new** JdbcTemplate(JDBCUtils.*getDataSource*());  
  
 @Override  
 **public** List<Category> findAll() {  
 String sql = **"select** *\** **from tab\_category "**;  
 **return template**.query(sql,**new** BeanPropertyRowMapper<Category>(Category.**class**));  
 }  
}

在BaseServlet中封装了序列化json的方法

/\*\*  
 \* 直接将传入的对象序列化为json，并且写回客户端  
 \* **@param *obj*** \*/  
**public void** writeValue(Object obj,HttpServletResponse response) **throws** IOException {  
 ObjectMapper mapper = **new** ObjectMapper();  
 response.setContentType(**"application/json;charset=utf-8"**);  
 mapper.writeValue(response.getOutputStream(),obj);  
}  
  
/\*\*  
 \* 将传入的对象序列化为json，返回  
 \* **@param *obj*** \* **@return** \*/  
**public** String writeValueAsString(Object obj) **throws** JsonProcessingException {  
 ObjectMapper mapper = **new** ObjectMapper();  
 **return** mapper.writeValueAsString(obj);  
}

### 前台代码

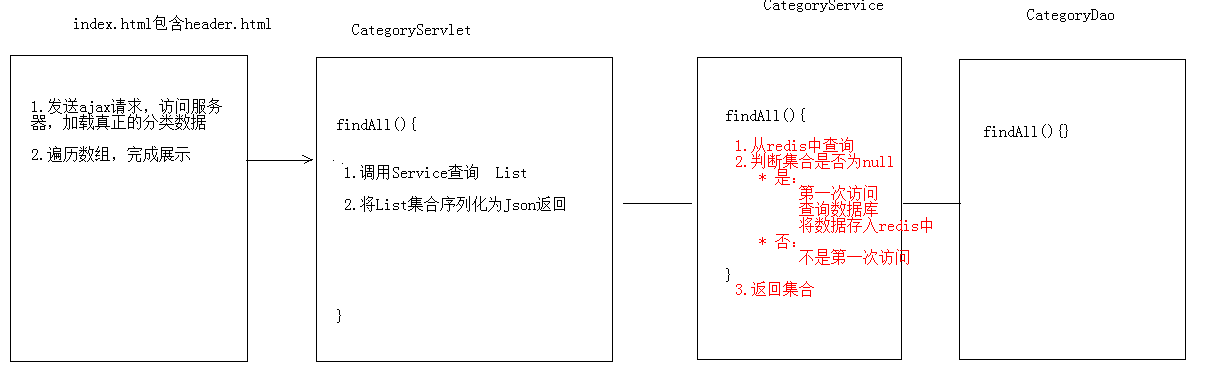
hader.html加载后，发送ajax请求，请求category/findAll

//查询分类数据  
 **$**.get(**"category/findAll"**,{},**function** (data) {  
 //[{cid:1,cname:国内游},{},{}]  
 **var** lis = **'<li class="nav-active"><a href="index.html">首页</a></li>'**;  
 //遍历数组,拼接字符串(<li>)  
 **for** (**var** i = 0; i < data.**length**; i++) {  
 **var** li = **'<li><a href="route\_list.html">'**+data[i].cname+**'</a></li>'**;  
  
 lis += li;  
   
 }  
 //拼接收藏排行榜的li,<li><a href="favoriterank.html">收藏排行榜</a></li>  
   
 lis+= **'<li><a href="favoriterank.html">收藏排行榜</a></li>'**;  
  
 //将lis字符串，设置到ul的html内容中  
 **$**(**"#category"**).html(lis);  
 });

## 对分类数据进行缓存优化

分析发现，分类的数据在每一次页面加载后都会重新请求数据库来加载，对数据库的压力比较大，而且分类的数据不会经常产生变化，所有可以使用redis来缓存这个数据。

分析：



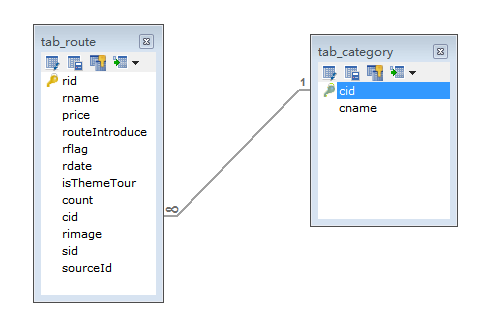
## 优化代码实现

期望数据中存储的顺序就是将来展示的顺序，使用redis的sortedset

@Override  
**public** List<Category> findAll() {  
 //1.从redis中查询  
 //1.1获取jedis客户端  
 Jedis jedis = JedisUtil.*getJedis*();  
 //1.2可使用sortedset排序查询  
 Set<String> categorys = jedis.zrange(**"category"**, 0, -1);  
 List<Category> cs = **null**;  
 //2.判断查询的集合是否为空  
 **if** (categorys == **null** || categorys.size() == 0) {  
  
 System.***out***.println(**"从数据库查询...."**);  
 //3.如果为空,需要从数据库查询,在将数据存入redis  
 //3.1 从数据库查询  
 cs = **categoryDao**.findAll();  
 //3.2 将集合数据存储到redis中的 category的key  
 **for** (**int** i = 0; i < cs.size(); i++) {  
  
 jedis.zadd(**"category"**, cs.get(i).getCid(), cs.get(i).getCname());  
 }  
 } **else** {  
 System.***out***.println(**"从redis中查询....."**);  
  
 //4.如果不为空,将set的数据存入list  
 cs = **new** ArrayList<Category>();  
 **for** (String name : categorys) {  
 Category category = **new** Category();  
 category.setCname(name);  
 cs.add(category);  
  
 }  
 }  
  
  
 **return** cs;  
}

# 旅游线路的分页展示

点击了不同的分类后，将来看到的旅游线路不一样的。通过分析数据库表结构，发现，旅游线路表和分类表时一个多对一的关系



查询不同分类的旅游线路sql

Select \* from tab\_route where cid = ?;

## 类别id的传递

Redis中查询score（cid）

**public class** CategoryServiceImpl **implements** CategoryService {  
  
 **private** CategoryDao **categoryDao** = **new** CategoryDaoImpl();  
  
 @Override  
 **public** List<Category> findAll() {  
 //1.从redis中查询  
 //1.1获取jedis客户端  
 Jedis jedis = JedisUtil.*getJedis*();  
 //1.2可使用sortedset排序查询  
 //Set<String> categorys = jedis.zrange("category", 0, -1);  
 //1.3查询sortedset中的分数(cid)和值(cname)  
 Set<Tuple> categorys = jedis.zrangeWithScores(**"category"**, 0, -1);  
  
 List<Category> cs = **null**;  
 //2.判断查询的集合是否为空  
 **if** (categorys == **null** || categorys.size() == 0) {  
  
 System.***out***.println(**"从数据库查询...."**);  
 //3.如果为空,需要从数据库查询,在将数据存入redis  
 //3.1 从数据库查询  
 cs = **categoryDao**.findAll();  
 //3.2 将集合数据存储到redis中的 category的key  
 **for** (**int** i = 0; i < cs.size(); i++) {  
  
 jedis.zadd(**"category"**, cs.get(i).getCid(), cs.get(i).getCname());  
 }  
 } **else** {  
 System.***out***.println(**"从redis中查询....."**);  
  
 //4.如果不为空,将set的数据存入list  
 cs = **new** ArrayList<Category>();  
 **for** (Tuple tuple : categorys) {  
 Category category = **new** Category();  
 category.setCname(tuple.getElement());  
 category.setCid((**int**)tuple.getScore());  
 cs.add(category);  
  
 }  
 }  
  
  
 **return** cs;  
 }  
}

页面传递cid

header.html传递cid

**var** li = **'<li><a href="route\_list.html?cid='**+data[i].cid+**'">'**+data[i].cname+**'</a></li>'**;

获取cid

**$**(**function** () {  
 **var** search = ***location***.search;  
 //alert(search);//?id=5  
 // 切割字符串，拿到第二个值  
 **var** cid = search.split(**"="**)[1];  
});

## 根据id查询不同类别的旅游线路数据

分页展示旅游线路数据：

### 分析



### 编码

1. 客户端代码编写

**$**(**function** () {  
 **var** search = ***location***.search;  
 // 切割字符串，拿到第二个值  
 **var** cid = search.split(**"="**)[1];  
  
 //当页码加载完成后，调用load方法，发送ajax请求加载数据  
 *load*(cid);  
});  
  
**function** *load*(cid ,currentPage){  
 //发送ajax请求，请求route/pageQuery,传递cid  
 **$**.get(**"route/pageQuery"**,{**cid**:cid,**currentPage**:currentPage},**function** (pb) {  
 //解析pagebean数据，展示到页面上  
  
 //1.分页工具条数据展示  
 //1.1 展示总页码和总记录数  
 **$**(**"#totalPage"**).html(pb.totalPage);  
 **$**(**"#totalCount"**).html(pb.totalCount);  
  
 **var** lis = **""**;  
  
 **var** fristPage = **'<li onclick="javascipt:***load***('**+cid+**')"><a href="javascript:void(0)">首页</a></li>'**;  
  
 //计算上一页的页码  
 **var** beforeNum = pb.**currentPage** - 1;  
 **if**(beforeNum <= 0){  
 beforeNum = 1;  
 }  
  
 **var** beforePage = **'<li onclick="javascipt:***load***('**+cid+**','**+beforeNum+**')" class="threeword"><a href="javascript:void(0)">上一页</a></li>'**;  
  
 lis += fristPage;  
 lis += beforePage;  
 //1.2 展示分页页码  
 /\*  
 1.一共展示10个页码，能够达到前5后4的效果  
 2.如果前边不够5个，后边补齐10个  
 3.如果后边不足4个，前边补齐10个  
 \*/  
  
 // 定义开始位置begin,结束位置 end  
 **var** begin; // 开始位置  
 **var** end ; // 结束位置  
  
  
 //1.要显示10个页码  
 **if**(pb.totalPage < 10){  
 //总页码不够10页  
  
 begin = 1;  
 end = pb.totalPage;  
 }**else**{  
 //总页码超过10页  
  
 begin = pb.**currentPage** - 5 ;  
 end = pb.**currentPage** + 4 ;  
  
 //2.如果前边不够5个，后边补齐10个  
 **if**(begin < 1){  
 begin = 1;  
 end = begin + 9;  
 }  
  
 //3.如果后边不足4个，前边补齐10个  
 **if**(end > pb.totalPage){  
 end = pb.totalPage;  
 begin = end - 9 ;  
 }  
 }  
  
  
 **for** (**var** i = begin; i <= end ; i++) {  
 **var** li;  
 //判断当前页码是否等于i  
 **if**(pb.**currentPage** == i){  
  
 li = **'<li class="curPage" onclick="javascipt:***load***('**+cid+**','**+i+**')"><a href="javascript:void(0)">'**+i+**'</a></li>'**;  
  
 }**else**{  
 //创建页码的li  
 li = **'<li onclick="javascipt:***load***('**+cid+**','**+i+**')"><a href="javascript:void(0)">'**+i+**'</a></li>'**;  
 }  
 //拼接字符串  
 lis += li;  
 }  
  
  
  
  
  
 **var** lastPage = **'<li class="threeword"><a href="javascript:;">末页</a></li>'**;  
 **var** nextPage = **'<li class="threeword"><a href="javascript:;">下一页</a></li>'**;  
  
 lis += nextPage;  
 lis += lastPage;  
  
  
 //将lis内容设置到 ul  
 **$**(**"#pageNum"**).html(lis);  
  
  
  
   
 //2.列表数据展示  
 **var** route\_lis = **""**;  
  
 **for** (**var** i = 0; i < pb.list.**length**; i++) {  
 //获取{rid:1,rname:"xxx"}  
 **var** route = pb.list[i];  
  
 **var** li = **'<li>\n'** +  
 **' <div class="img"><img src="'**+route.rimage+**'" style="width: 299px;"></div>\n'** +  
 **' <div class="text1">\n'** +  
 **' <p>'**+route.rname+**'</p>\n'** +  
 **' <br/>\n'** +  
 **' <p>'**+route.routeIntroduce+**'</p>\n'** +  
 **' </div>\n'** +  
 **' <div class="price">\n'** +  
 **' <p class="price\_num">\n'** +  
 **' <span>&yen;</span>\n'** +  
 **' <span>'**+route.price+**'</span>\n'** +  
 **' <span>起</span>\n'** +  
 **' </p>\n'** +  
 **' <p><a href="route\_detail.html">查看详情</a></p>\n'** +  
 **' </div>\n'** +  
 **' </li>'**;  
 route\_lis += li;  
 }  
 **$**(**"#route"**).html(route\_lis);  
  
 //定位到页面顶部  
 ***window***.scrollTo(0,0);  
 });  
  
}

1. 服务器端代码编写
   1. 创建PageBean对象

**public class** PageBean<T> {  
  
 **private int totalCount**;//总记录数  
 **private int totalPage**;//总页数  
 **private int currentPage**;//当前页码  
 **private int pageSize**;//每页显示的条数  
  
 **private** List<T> **list**;//每页显示的数据集合  
  
 **public int** getTotalCount() {  
 **return totalCount**;  
 }  
  
 **public void** setTotalCount(**int** totalCount) {  
 **this**.**totalCount** = totalCount;  
 }  
  
 **public int** getTotalPage() {  
 **return totalPage**;  
 }  
  
 **public void** setTotalPage(**int** totalPage) {  
 **this**.**totalPage** = totalPage;  
 }  
  
 **public int** getCurrentPage() {  
 **return currentPage**;  
 }  
  
 **public void** setCurrentPage(**int** currentPage) {  
 **this**.**currentPage** = currentPage;  
 }  
  
 **public int** getPageSize() {  
 **return pageSize**;  
 }  
  
 **public void** setPageSize(**int** pageSize) {  
 **this**.**pageSize** = pageSize;  
 }  
  
 **public** List<T> getList() {  
 **return list**;  
 }  
  
 **public void** setList(List<T> list) {  
 **this**.**list** = list;  
 }  
}

* 1. RouteServlet

@WebServlet(**"/route/\*"**)  
**public class** RouteServlet **extends** BaseServlet {  
  
 **private** RouteService **routeService** = **new** RouteServiceImpl();  
  
 /\*\*  
 \* 分页查询  
 \* **@param *request*** \* **@param *response*** \* **@throws** ServletException  
 \* **@throws** IOException  
 \*/  
 **public void** pageQuery(HttpServletRequest request, HttpServletResponse response) **throws** ServletException, IOException {  
 //1.接受参数  
 String currentPageStr = request.getParameter(**"currentPage"**);  
 String pageSizeStr = request.getParameter(**"pageSize"**);  
 String cidStr = request.getParameter(**"cid"**);  
  
 **int** cid = 0;//类别id  
 //2.处理参数  
 **if**(cidStr != **null** && cidStr.length() > 0){  
 cid = Integer.*parseInt*(cidStr);  
 }  
 **int** currentPage = 0;//当前页码，如果不传递，则默认为第一页  
 **if**(currentPageStr != **null** && currentPageStr.length() > 0){  
 currentPage = Integer.*parseInt*(currentPageStr);  
 }**else**{  
 currentPage = 1;  
 }  
  
 **int** pageSize = 0;//每页显示条数，如果不传递，默认每页显示5条记录  
 **if**(pageSizeStr != **null** && pageSizeStr.length() > 0){  
 pageSize = Integer.*parseInt*(pageSizeStr);  
 }**else**{  
 pageSize = 5;  
 }  
  
 //3. 调用service查询PageBean对象  
 PageBean<Route> pb = **routeService**.pageQuery(cid, currentPage, pageSize);  
  
 //4. 将pageBean对象序列化为json，返回  
 writeValue(pb,response);  
  
 }  
  
}

* 1. RouteService

**public class** RouteServiceImpl **implements** RouteService {  
 **private** RouteDao **routeDao** = **new** RouteDaoImpl();  
 @Override  
 **public** PageBean<Route> pageQuery(**int** cid, **int** currentPage, **int** pageSize) {  
 //封装PageBean  
 PageBean<Route> pb = **new** PageBean<Route>();  
 //设置当前页码  
 pb.setCurrentPage(currentPage);  
 //设置每页显示条数  
 pb.setPageSize(pageSize);  
   
 //设置总记录数  
 **int** totalCount = **routeDao**.findTotalCount(cid);  
 pb.setTotalCount(totalCount);  
 //设置当前页显示的数据集合  
 **int** start = (currentPage - 1) \* pageSize;//开始的记录数  
 List<Route> list = **routeDao**.findByPage(cid,start,pageSize);  
 pb.setList(list);  
  
 //设置总页数 = 总记录数/每页显示条数  
 **int** totalPage = totalCount % pageSize == 0 ? totalCount / pageSize :(totalCount / pageSize) + 1 ;  
 pb.setTotalPage(totalPage);  
  
  
 **return** pb;  
 }  
}

* 1. RouteDao

**public class** RouteDaoImpl **implements** RouteDao {  
 **private** JdbcTemplate **template** = **new** JdbcTemplate(JDBCUtils.*getDataSource*());  
  
 @Override  
 **public int** findTotalCount(**int** cid) {  
 String sql = **"select** *count***(***\****) from tab\_route where cid = ?"**;  
 **return template**.queryForObject(sql,Integer.**class**,cid);  
 }  
  
 @Override  
 **public** List<Route> findByPage(**int** cid, **int** start, **int** pageSize) {  
 String sql = **"select** *\** **from tab\_route where cid = ? limit ? , ?"**;  
  
 **return template**.query(sql,**new** BeanPropertyRowMapper<Route>(Route.**class**),cid,start,pageSize);  
 }  
}