**SpringMVC第二天**

**框架课程**

# 课程计划

1. 高级参数绑定
   1. 数组类型的参数绑定
   2. List类型的绑定
2. @RequestMapping注解的使用
3. Controller方法返回值
4. Springmvc中异常处理
5. 图片上传处理
6. Json数据交互 （重点）
7. Springmvc实现RESTful
8. 拦截器

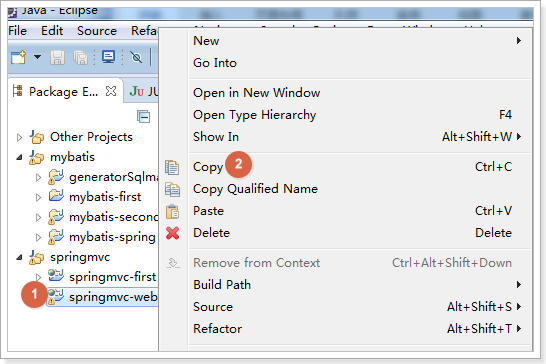
# 高级参数绑定

## 复制工程

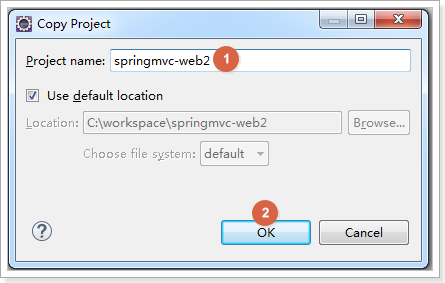
把昨天的springmvc-web工程复制一份，作为今天开发的工程

Annotation: 注释注解

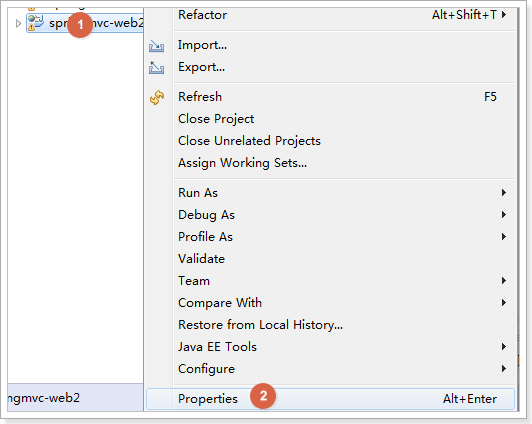
复制工程，如下图：



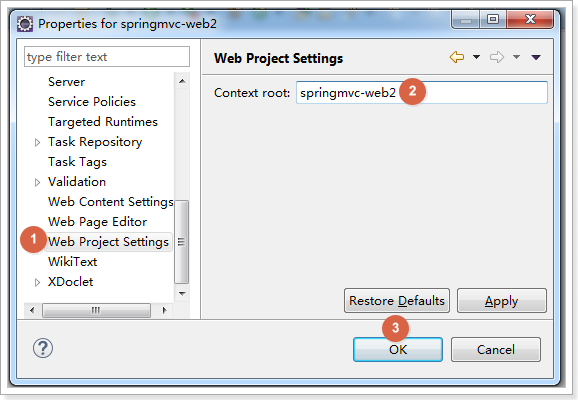
粘贴并修改工程名为web2，如下图：



工程右键点击，如下图：



修改工程名，如下图：



## 绑定数组

### 需求

在商品列表页面选中多个商品，然后删除。

### 需求分析

功能要求商品列表页面中的每个商品前有一个checkbok，选中多个商品后点击删除按钮把商品id传递给Controller，根据商品id删除商品信息。

我们演示可以获取id的数组即可

### Jsp修改

修改itemList.jsp页面,增加多选框，提交url是queryItem.action

<form action=*"*${pageContext.request.contextPath }*/item/queryitem.action"* method=*"post"*>

查询条件：

<table width=*"100%"* border=*1*>

<tr>

<td><input type=*"submit"* value=*"查询"*/></td>

</tr>

</table>

</form>

商品列表：

<form action=*"*${pageContext.request.contextPath }*/queryItem.action"*>

<table width=*"100%"* border=*1*>

<tr>

<td>选择</td>

<td>商品名称</td>

<td>商品价格</td>

<td>生产日期</td>

<td>商品描述</td>

<td>操作</td>

</tr>

<c:forEach items=*"*${itemList }*"* var=*"item"*>

<tr>

<td><input type=*"checkbox"* value=*"*${item.id}*"* name=*"ids"*></td>

<td>${item.name }</td>

<td>${item.price }</td>

<td><fmt:formatDate value=*"*${item.createtime}*"* pattern=*"yyyy-MM-dd HH:mm:ss"*/></td>

<td>${item.detail }</td>

<td><a href=*"*${pageContext.request.contextPath }*/itemEdit.action?id=*${item.id}*"*>修改</a></td>

</tr>

</c:forEach>

</table>

<input type=*"submit"* value=*"提交"*>

</form>

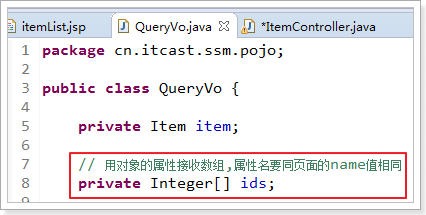
页面选中多个checkbox向controller方法传递

本身属于一个form表单，提交url是queryItem.action

### Controller

Controller方法中可以用String[]接收，或者pojo的String[]属性接收。两种方式任选其一即可。

定义QueryVo，如下图：



ItemController修改queryItem方法：

/\*\*

\* 包装类型 绑定数组类型，可以使用两种方式，pojo的属性接收，和直接接收

\*

\* **@param** queryVo

\* **@return**

\*/

@RequestMapping("queryItem")

**public** String queryItem(QueryVo queryVo, Integer[] ids) {

System.***out***.println(queryVo.getItem().getId());

System.***out***.println(queryVo.getItem().getName());

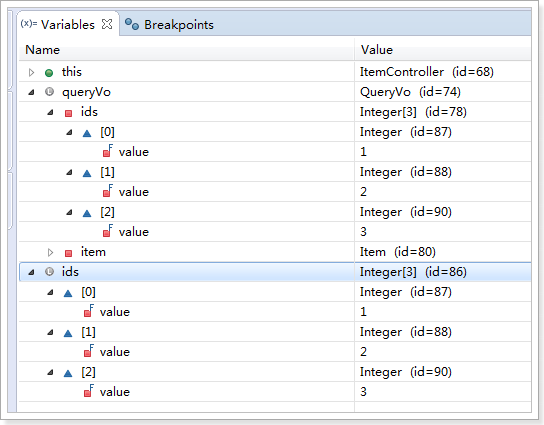
System.***out***.println(queryVo.getIds().length);

System.***out***.println(ids.length);

**return** "success";

}

效果，如下图：



## 将表单的数据绑定到List

### 需求

实现商品数据的批量修改。

### 开发分析

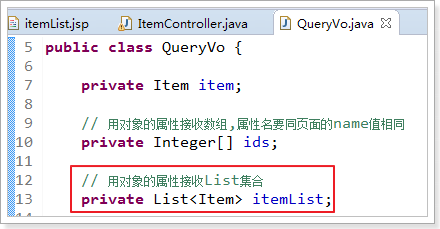
开发分析

1. 在商品列表页面中可以对商品信息进行修改。
2. 可以批量提交修改后的商品数据。

### 定义pojo

List中存放对象，并将定义的List放在包装类QueryVo中

使用包装pojo对象接收，如下图：



### Jsp改造

在jsp页面直接进行修改参数:

<c:forEach items=*"*${itemList }*"* var=*"item"*>

<tr>

<td><input type=*"checkbox"* value=*"*${item.id}*"* name=*"ids"*></td>

<td><input type=*"text"* value=*"*${item.name}*"* name=*"name"*></td>

<td><input type=*"text"* value=*"*${item.price}*"* name=*"price"*></td>

<td><fmt:formatDate value=*"*${item.createtime}*"* pattern=*"yyyy-MM-dd HH:mm:ss"*/></td>

<td>${item.detail }</td>

<td><a href=*"*${pageContext.request.contextPath }*/itemEdit.action?id=*${item.id}*"*>修改</a></td>

</tr>

</c:forEach>

在queryvo中添加属性:

**public** List<Item> getItemList() {

**return** itemList;

}

添加get/set方法.

在jsp页面修改传入数据方式.

<c:forEach items=*"*${itemList }*"* var=*"item"* varStatus=*"s"*> // varStatus变量的地位

<tr>

<td><input type=*"checkbox"* value=*"*${item.id}*"* name=*"ids"*></td>

<td><input type=*"text"* value=*"*${item.name}*"* name=*"itemList[*${s.index}*].name"*></td>

<td><input type=*"text"* value=*"*${item.price}*"* name=*"itemList[*${s.index}*].price"*></td>

<td><fmt:formatDate value=*"*${item.createtime}*"* pattern=*"yyyy-MM-dd HH:mm:ss"*/></td>

<td>${item.detail }</td>

<td><a href=*"*${pageContext.request.contextPath }*/itemEdit.action?id=*${item.id}*"*>修改</a></td>

</tr>

</c:forEach>

测试类:

@RequestMapping(value = "/queryItem.action")

//public ModelAndView queryItem(Integer[] ids){

**public** ModelAndView queryItem(QueryVo vo){

System.***out***.println();

**return** **null**;

}

在class上添加@RequestMapping(url)指定通用请求前缀， 限制此类下的所有方法请求url必须以请求前缀开头

## 请求方法限定

除了可以对url进行设置，还可以限定请求进来的方法

* 限定GET方法

@RequestMapping(method = RequestMethod.***GET***)

如果通过POST访问则报错：

HTTP Status 405 - Request method 'POST' not supported

例如：

@RequestMapping(value = "itemList",method = RequestMethod.***POST***)

* 限定POST方法

@RequestMapping(method = RequestMethod.***POST***)

如果通过GET访问则报错：

HTTP Status 405 - Request method 'GET' not supported

* GET和POST都可以

@RequestMapping(method = {RequestMethod.***GET***,RequestMethod.***POST***})

# Controller方法返回值

## 返回ModelAndView

controller方法中定义ModelAndView对象并返回，对象中可添加model数据、指定view。

参考第一天的内容

## 返回void

在Controller方法形参上可以定义request和response，使用request或response指定响应结果：

1、使用request转发页面，如下：

request.getRequestDispatcher("页面路径").forward(request, response);

request.getRequestDispatcher("/WEB-INF/jsp/success.jsp").forward(request, response);`

2、可以通过response页面重定向：

response.sendRedirect("url")

response.sendRedirect("/springmvc-web2/itemEdit.action");

1. 可以通过response指定响应结果，例如响应json数据如下：

response.getWriter().print("{\"abc\":123}");

## 返回字符串

### 逻辑视图名

controller方法返回字符串可以指定逻辑视图名，通过视图解析器解析为物理视图地址。

//指定逻辑视图名，经过视图解析器解析为jsp物理路径：/WEB-INF/jsp/itemList.jsp

**return** "itemList";

参考第一天内容

### Redirect重定向

Contrller方法返回字符串可以重定向到一个url地址

如下商品修改提交后重定向到商品编辑页面。

/\*\*

\* 更新商品

\*

\* **@param** item

\* **@return**

\*/

@RequestMapping("updateItem")

**public** String updateItemById(Item item) {

// 更新商品

**this**.itemService.updateItemById(item);

// 修改商品成功后，重定向到商品编辑页面

// 重定向后浏览器地址栏变更为重定向的地址，

// 重定向相当于执行了新的request和response，所以之前的请求参数都会丢失

// 如果要指定请求参数，需要在重定向的url后面添加 ?itemId=1 这样的请求参数

**return** "redirect:test/index.action";

Controller方法执行后继续执行另一个Controller方法

如下商品修改提交后转向到商品修改页面，修改商品的id参数可以带到商品修改方法中。

这两个本质一样,自己选择使用.

异常的配置:

创建异常类:

**package** com.itheima.springmvc.exception;

**public** **class** CustomException **extends** Exception{

**private** String msg;

**public** CustomException(String message) {

**super**();

**this**.msg= message;

// **TODO** Auto-generated constructor stub

}

/\*\*

\* **@return** the msg

\*/

**public** String getMsg() {

**return** msg;

}

/\*\*

\* **@param** msg the msg to set

\*/

**public** **void** setMsg(String msg) {

**this**.msg = msg;

}

}

创建异常处理器:

**public** **class** CustomHandleException **implements** HandlerExceptionResolver{

**public** ModelAndView resolveException(HttpServletRequest request, HttpServletResponse response, Object handler,

Exception exception) {

String msg;

**if**(exception **instanceof** CustomException){

msg = exception.getMessage();

}

**else**{

msg="哈哈,你走错了";

}

ModelAndView modelAndView = **new** ModelAndView();

modelAndView.addObject("msg",msg);

modelAndView.setViewName("error");

**return** modelAndView;

}

}

添加error页面

运行报错了 要在springmvc.xml中配置:

<!-- 异常处理类的实例化 -->

<bean class="com.itheima.springmvc.exception.CustomExceptionResolver"></bean>

在editltem.jsp页面打开图片的那段代码.

在item类中添加属性:

**private** String pic;

**public** String getPic() {

**return** pic;

}

**public** **void** setPic(String pic) {

**this**.pic = pic;

}

<body>

<form id=*"itemForm"* action=*"*${pageContext.request.contextPath }*/updateitem.action"* enctype=*"multipart/form-data"* method=*"post"*>

<input type=*"hidden"* name=*"id"* value=*"*${item.id }*"* /> 修改商品信息：

<table width=*"100%"* border=*1*>

<tr>

<td>商品名称</td>

<td><input type=*"text"* name=*"name"* value=*"*${item.name }*"* /></td>

</tr>

<tr>

<td>商品价格</td>

<td><input type=*"text"* name=*"price"* value=*"*${item.price }*"* /></td>

</tr>

<tr>

<td>商品生产日期</td>

<td><input type=*"text"* name=*"createtime"*

value=*"*<fmt:formatDate value="${item.createtime}" pattern=*"yyyy-MM-dd HH:mm:ss"*/>*"* /></td>

</tr>

<tr>

<td>商品图片</td>

<td>

<c:if test=*"*${item.pic !=null}*"*>

<img src=*"/pic/*${item.pic}*"* width=*100* height=*100*/>

<br/>

</c:if>

<input type=*"file"* name=*"pictureFile"*/>

</td>

</tr>

<tr>

<td>商品简介</td>

<td><textarea rows=*"3"* cols=*"30"* name=*"detail"*>${item.detail }</textarea>

</td>

</tr>

<tr>

<td colspan=*"2"* align=*"center"*><input type=*"submit"* value=*"提交"* />

</td>

</tr>

</table>

</form>

</body>

在类中:

@RequestMapping(value = "/updateitem.action")

**public** String updateitem(Item item,MultipartFile pictureFile){

//修改 通过Id对商品信息进行修改

itemService.updateItemById(item);

ModelAndView mav = **new** ModelAndView();

mav.setViewName("success");

**return** "redirect:test/index.action";

}

在配置文件springmvc.xml中:

<!-- 上传图片的实现类 -->id必须是这个名字

<bean id="multipartResolver" class="org.springframework.web.multipart.commons.CommonsMultipartResolver">

将老师资料中的文件上传的两个包导入.

在editItem.jsp中:去掉*/pic/变成 /*

<c:if test=*"*${item.pic !=null}*"*>

<img src=*"/pic/*${item.pic}*"* width=*100* height=*100*/>

<br/>

</c:if>

在控制层:

@RequestMapping(value = "/updateitem.action")

**public** String updateitem(Item item,MultipartFile pictureFile) **throws** IllegalStateException, IOException{

String ext = FilenameUtils.*getExtension*(pictureFile.getOriginalFilename());

//图片名称

String name = UUID.*randomUUID*().toString()+"."+ext;

//保存图片

pictureFile.transferTo(**new** File("D:\\upload\\"+name));

item.setPic(name);

//修改 通过Id对商品信息进行修改

itemService.updateItemById(item);

**return** "redirect:test/index.action";

}

在mybatis的映射配置文件:

<update id=*"updateItemById"* parameterType=*"Item"*>

update items

<set>

<if test=*"name != null and name != ''"*>

name = #{name},

</if>

<if test=*"price != null"*>

price = #{price},

</if>

<if test=*"detail != null and detail != ''"*>

detail = #{detail}

</if>

<if test=*"pic != null and pic != ''"*>

pic = #{pic}

</if>

</set>

<where>

id = #{id}

</where>

</update>

运行后报错 因为xml的映射文件属性后边没有加逗号.

Ajax的异步加载:

导入jq包:

在itemList.jsp中添加ajax:

<script type=*"text/javascript"* src=*"*${pageContext.request.contextPath }*/js/jquery-1.4.4.min.js"*></script>

<script type=*"text/javascript"* >

$(**function**(){

var data = '{"id": 1,"name": "测试商品","price": 99.9,"detail": "测试商品描述","pic": "123456.jpg"}';

$.ajax({

url:"${pageContext.request.contextPath }/test/json.action",

type:post,

data:data,

contentType:"application/json;charset=UTF-8",

dataType:"json",

success:**function**(data){

alert(data.name+":"+data.pic);

}

});

});

</script>

Springmvc对json的支持有三个包要导入:

在类中要添加方法:

@RequestMapping(value = "/test/json.action")

**public** @ResponseBody

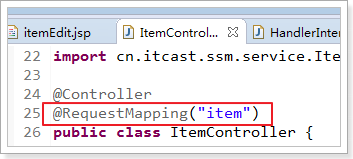
Item json(@RequestBody Item item){

**return** item;

}

### 配置拦截器

只能拦截商品的url，所以需要修改ItemController，让所有的请求都必须以item开头，如下图：



在springmvc.xml配置拦截器

<mvc:interceptor>

<!-- 配置商品被拦截器拦截 -->

<mvc:mapping path=*"/item/\*\*"* />

<!-- 配置具体的拦截器 -->

<bean class=*"cn.itcast.ssm.interceptor.LoginHandlerInterceptor"* />

</mvc:interceptor>