Springmvc课堂笔记

课程安排：

一天课程

内容较多

学习springmvc的框架原理（掌握）

包括哪些组件：

前端控制器(中心)

处理器映射器(三大组件)

处理器适配器(三大组件)

视图解析器(三大组件)

第一个springmvc程序

强化三大组件学习：

两个处理器映射器

两个处理器适配器

一个视图解析器（jsp+jstl）

命令控制器（过期的，了解）

Springmvc注解开发(重点)

学习springmvc常用的注解

Json数据转换（掌握）

拦截器（了解）

# springmvc的框架原理

## b/s系统mvc原理

mvc是什么？mvc是设计模式。

C （controller）控制器

负责：request和response

1 Request请求

5 Response响应

M（model）模型

包括：

Pojo、service、dao

2 请求模型进行业务处理

V（view）视图

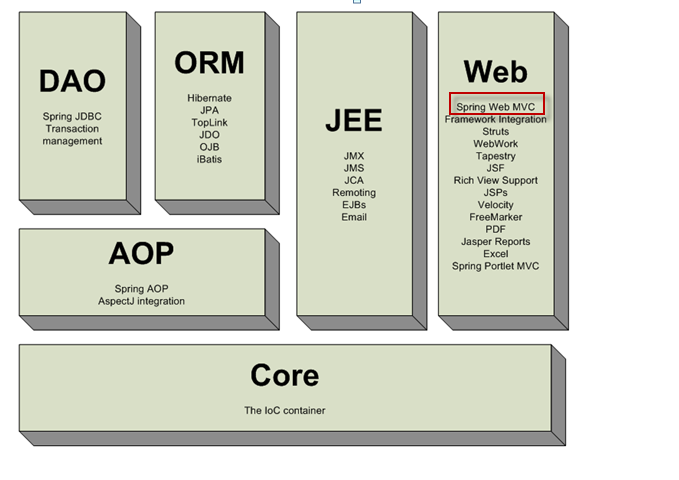
4 视图渲染（将模型数据填充到视图）

b/s系统中模型是无法将数据推送到视图，C/S可以

3、返回结果

## springmvc是什么？

Mvc的框架，是spring框架的一个模块。



## springmvc框架原理

DispatcherServlet前端控制器

负责：request和response

1 Request请求

10Response响应

Handler(处理器)

HandlerMapping处理器映射器

负责：根据url查找Handler

2 请求查找handler

3 返回Handler

HandlerExecutionChain{

HandlerInterceptor

HandlerInterceptor..

Handler

}

HandlerAdapter处理器适配器

负责：执行Handler

4请求执行Handler

5执行

6返回ModelAndView

7返回modelAndView

View resolver视图解析器

负责解析view

View（视图）

8视图解析，解析出view

9视图渲染：

将模型数据填充到view

1. 用户发起请求，请求到DispatcherServlet前端控制器

DispatcherServlet（中央调度），负责request和response，负责调用处理器映射器查找Handler，负责调用处理器适配器执行Handler，有了前端控制器降低了各各组件之间的耦合性，系统扩展性提高。

1. DispatcherServlet前端控制器请求处理器映射器HandlerMapping查找Handler

根据用户请求的url，根据一定的规则去查找（比如：xml配置，注解）

1. HandlerMapping处理器映射器将Handler返回给前端控制器
2. DispatcherServlet前端控制器调用HandlerAdapter处理器适配器执行Handler

程序员编写的Handler是按照适配器要求的规则去执行Handler

1. HandlerAdapter处理器适配器执行Handler

适配器按照一定规则去执行Handler

1. Handler执行完成，返回ModelAndView

ModelAndView是springmvc的封装对象，将model和view封装在一起。

1. HandlerAdapter处理器适配器将ModelAndView返回给前端 控制器
2. 前端控制器调用视图解析器，进行视图解析，解析完成给前端控制器返回View

View是springmvc的封装对象，是一个接口，实现类包括jspview，pdfview。。。。

1. 前端控制器调用view进行视图渲染

将模型数据填充到view（将model数据填充到request）响应给用户

10、前端控制器响应给用户。

小结：

一个中心：（不需要开发）

DispatcherServlet前端控制器

三个组件:（不需要开发）

处理器映射器

处理器适配器

视图解析器

Handler：(需要开发)

处理器，理解成action

View：

需要开发页面：jsp

# Springmvc第一个程序

## 统一开发环境

使用eclipse-3.7-indigo 32位

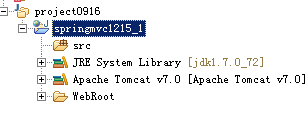
Jdk：1.7.72版本(32位)

Tomcat：apache-tomcat-7.0.53

参考 ：Eclipse开发环境配置-indigo.docx

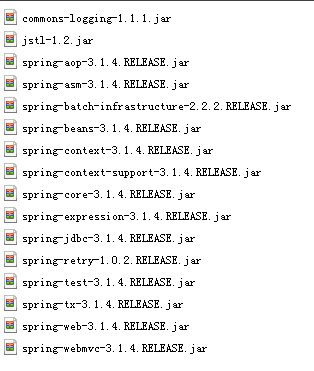
## 第一程序开发

创建一个web 工程



加入 springmvc的jar包

Spring的版本：3.1.4



配置前端控制器

在web.xml中配置servlet。

指定 contextConfigLocation让前端控制器找到springmvc的全局配置文件。

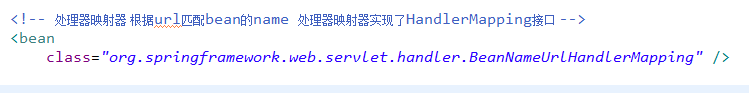


配置三大组件

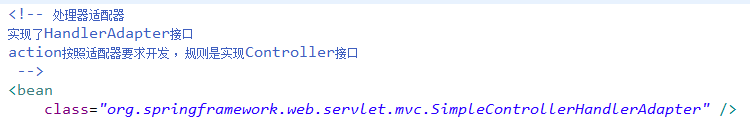
在springmvc.xml中配置：

处理器映射器：

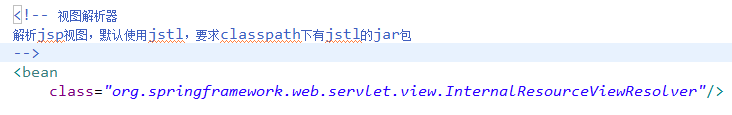
BeanNameUrlHandlerMapping



处理器适配器:



视图解析器：



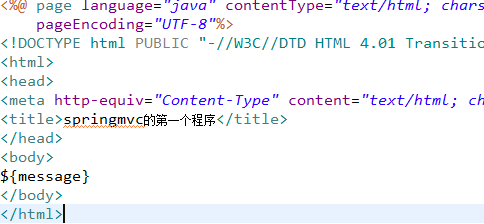
编写Handler

由于使用SimpleControllerHandlerAdapter适配器，要求编写的Handler实现Controller接口。

需求：向页面显示一行信息“helloworld”



编写jsp



配置Handler

由于使用处理器映射器BeanNameUrlHandlerMapping，要求在spring的容器配置Handler，Handler的name就是url。

在springmvc.xml中配置：



部署tomcat，启动

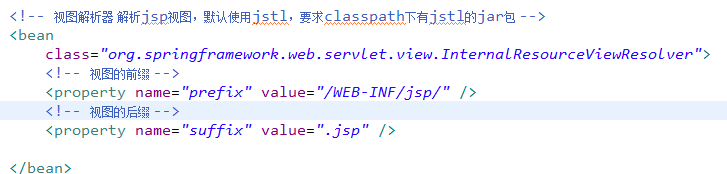
以debug方式运行。

访问路径：http://localhost:8080/springmvc1215\_1/helloworld.action

## 逻辑视图名

调用：modelAndView.setViewName指定逻辑视图名，如果这里指定jsp的完整路径也支持。

在springmvc.xml中修改视图解析器：



修改action：



# 强化三大组件学习

## 处理器映射器

作用：根据url找到Handler.

BeanNameUrlHandlerMapping：根据url匹配bean的name 处理器映射器实现了HandlerMapping接口

SimpleUrlHandlerMapping：将url进行集中配置。

<!-- 简单url映射 -->

<bean class=*"org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"*>

<property name=*"mappings"*>

<props>

<prop key=*"/hello1.action"*>hello\_controller</prop>

<prop key=*"/hello2.action"*>hello\_controller</prop>

</props>

</property>

</bean>

hello\_controller是action这个bean的id。

总结：

多个处理器映射器可以共存

## 处理器适配器

作用：按照一定规则去执行Handler

SimpleControllerHandlerAdapter：规则是Handler要实现Controller接口

HttpRequestHandlerAdapter：规则是Handler要实现HttpRequestHandler接口。

总结：

多个适配器可以共存！

# 命令表单控制器

编写action，继承~~AbstractCommandController。~~

运行方式类似struts的modelDriven。

将页面提交的参数封装在一个pojo对象(命令对象)中。

## 需求

学生信息修改、提交功能。

进入学生信息修改页面，输入信息，提交，将学生的信息提交到action中。

将页面的参数封装到一个pojo对象(命令对象)中。

## 开发

编写两个action类：

修改学生信息页面

修改学生信息提交



注意：

//设置命令对象

**this**.setCommandClass(Student.**class**);

## 问题处理

Post乱码

在web.xml中加入：

<filter>

<filter-name>CharacterEncodingFilter</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>

</filter>

<filter-mapping>

<filter-name>CharacterEncodingFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

对get乱码处理：

对于get请求中文参数出现乱码解决方法有两个：

修改tomcat配置文件添加编码与工程编码一致，如下：

<Connector URIEncoding="utf-8" connectionTimeout="20000" port="8080" protocol="HTTP/1.1" redirectPort="8443"/>

另外一种方法对参数进行重新编码：

String userName new

String(request.getParamter("userName").getBytes("ISO8859-1"),"utf-8")

ISO8859-1是tomcat默认编码，需要将tomcat编码后的内容按utf-8编码

日期型转换

//注册一个属性编辑器

@Override

**protected** **void** initBinder(HttpServletRequest request,

ServletRequestDataBinder binder) **throws** Exception {

//将student类中的birthday属性进行日期转换，类型和birthday一致

binder.registerCustomEditor(Date.**class**, **new** CustomDateEditor(**new** SimpleDateFormat("yyyy-MM-dd"), **true**));

}

小结

使用命令表单控制器也可以开发增、删、改、查。

使用命令表单控制器一定要指定命令对象，但是只能指定一个命令对象。

# 上午课程小结

开发springmvc程序：

第一步：配置前端控制器

在web.xml配置DispathcherServlet前端控制器

第二步：配置springmvc的全局配置文件

配置三大组件：

处理器映射器：

根据url查找 Handler

处理器适配器：

执行Handler

视图解析器：

解析出视图View，根据逻辑视图名解析出真正的视图。

第三步：按照处理器适配器规则开发Handler（action）

第四步：将Handler配置在spring容器中。

第五步：编写视图（jsp+jstl）

# 学生信息修改注解开发

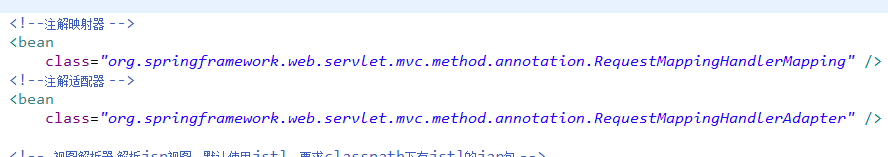
## 创建一个web 工程

## 加入 jar包

## 配置前端控制器

## 全局配置文件springmvc.xml

配置注解处理器映射器和适配器：



## 编写Handler（Action）

// 修改学生信息页面

// @RequestMapping指定url

@RequestMapping("/editstudent")

**public** String editstudent(Model model) **throws** Exception {

// 将学生信息的显示在页面

// 使用静态数据

Student student = **new** Student();

student.setName("张三");

student.setAge(32);

student.setBirthday(**new** Date());

// 需要将学生信息传到页面

model.addAttribute("student", student);

// 返回逻辑视图名

**return** "student/editstudent";

}

// 学生信息修改提交

@RequestMapping("/editstudentsubmit")

**public** String editstudentsubmit(Student student)**throws** Exception{

System.*out*.println(student);

**return** "success";

}

注意：

一个url对应action的一个方法。

## 在spring的容器中配置action

使用@Controller注解：



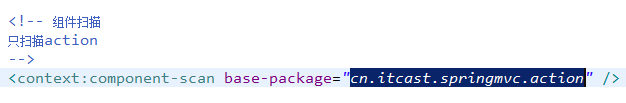
与@Controller注解同类型还有：@Service、@Repository、@Component

@Controller注解作用：

使用spring的组件扫描，自动扫描到action在spring容器中注册。

在springmvc.xml中配置组件扫描：

只扫描action：



## 部署tomcat运行

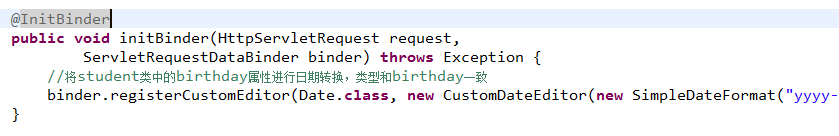
运行地址：

学生信息修改：<http://localhost:8080/springmvc1215_2/editstudent.action>

学生信息修改提交 ：<http://localhost:8080/springmvc1215_2/editstudentsubmit.action>

## 日期型转换

在action类中添加方法：



建议将此方法定义在父类中，让action类继承此父类。

# 与struts的不同

1. springmvc的入口是servlet，struts是filter
2. springmvc是基于方法开发，struts是基于类开发

action类中，一个url对应一个方法，对Handler对象，springmvc框架将一个方法封装到Handler中（只有一个method）。Struts对action类进行实例化。

Springmvc中的action可以使用单例也可以使用多例，建议使用单例（对相同的方法只new一个）。

Struts不能用单例，因为struts传递参数是通过成员变量。

Springmvc通过形参接收参数好处：更符合软件开发的思想，和service接口类似。

1. Struts采用值栈存储请求和响应的数据，通过OGNL存取数据， springmvc通过参数解析器是将request请求内容解析，并给方法形参赋值，将数据和视图封装成ModelAndView对象，最后又将ModelAndView中的模型数据通过reques域传输到页面。Jsp视图解析器默认使用jstl。

参数解析过程：

页面请求数据：key/value

参数解析器：

Spring提供了很多参数解析器，将key/value的数据解析成形参，通过反射解析method中的参数（类型、名称），按照类型和名称规则解析形参

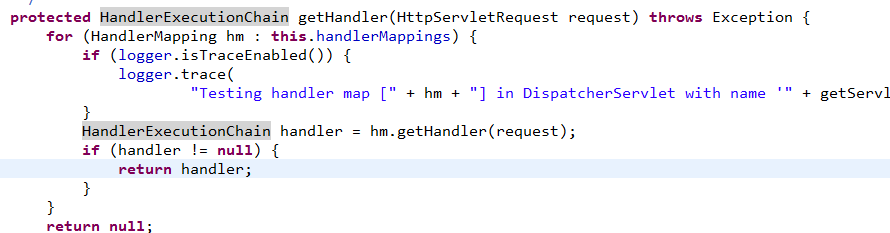
属性编辑器：

将key/value转换成pojo中的属性

Action方法的形参

## 前端控制器代码跟踪

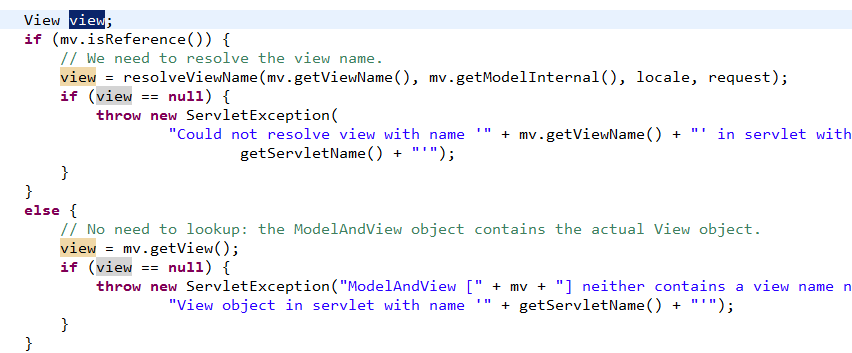
1. 前端控制器调用处理器映射器找Handler



1. 调用处理器适配器执行Handler



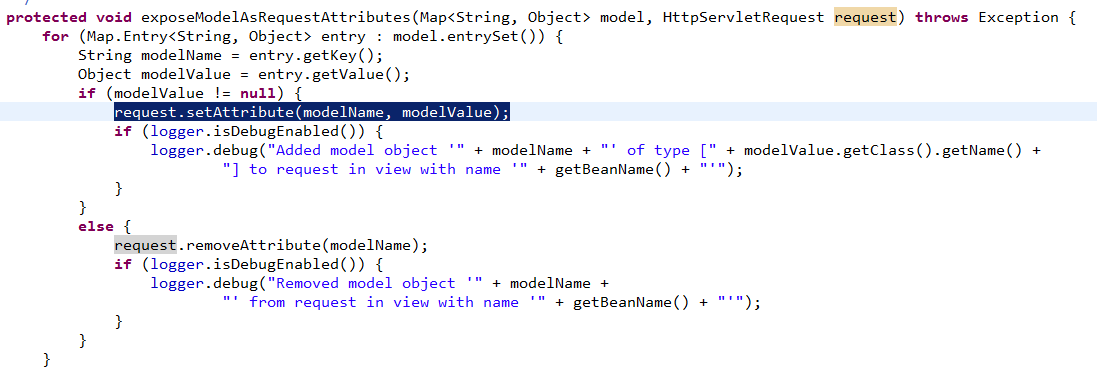
1. 视图解析，最终解析出view



1. 视图渲染

将model的数据放到request域





# 常用注解

## @controller

标识该类为控制器类，@controller、@service、@repository分别对应了web应用三层架构的组件即控制器、服务接口、数据访问接口

## @RequestMapping

进行url映射，一个方法对应一个url，定义方法：在action方法上定义requestMapping

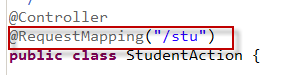
根路径+子路径

需求：为了很好的管理url，为了避免url的冲突，使用根路径+子路径

定义方法：

根路径：

在类名上边定义requestMapping



子路径：

在方法上边定义requestMapping



最终访问路径：根路径+子路径：/stu/editstudent.action

URI 模板模式映射

将参数在url中传递。

定义方法：



修改学生信息列表页面，将修改链接改为：



URI 模板模式映射好处：方便实现restful。

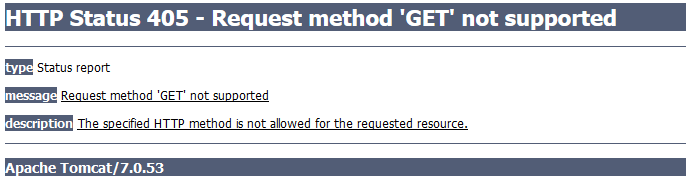
详细参考 ：

请求方法限定

测试，限定为post，



如果不是post请求：



可以限定为get、post、或两者都可以

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="/useredit/{userid}",method=RequestMethod.GET)

限定POST方法

@RequestMapping(method = RequestMethod.*POST*)

如果通过Post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})

# 请求参数绑定(重点)

Springmvc通过参数解析器，将客户端请求的key/value解析成方法形参数，过程：

参数解析过程：

页面请求数据：key/value

参数解析器：

Spring提供了很多参数解析器，将key/value的数据解析成形参，通过反射解析method中的参数（类型、名称），按照类型和名称规则解析形参

属性编辑器：

将key/value转换成pojo中的属性

Action方法的形参

掌握不同参数类型的解析方方法。

## 默认支持的参数类型

HttpServletRequest

通过request对象获取请求信息

HttpServletResponse

通过response处理响应信息

HttpSession

通过session对象得到session中存放的对象

Model

通过model向页面传递数据，如下：

model.addAttribute("user", **new** User("李四"));

model也可以通过modelMap或map将数据传到页面。

## 表单对象-简单类型

Action方法的形参数默认支持：string、int、double等简单类型，建议使用包装类型（可以设置null）。

注意：对于date类型的参数需要注册属性编辑器。

使用方法：

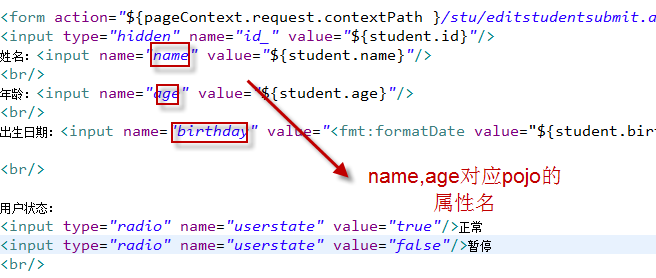
直接在形参上定义

也可以在pojo中通过属性传递简单类型

## 表单对象-pojo

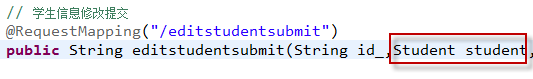
使用pojo属性名传递

页面定义：



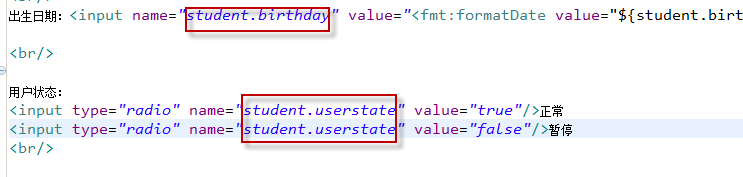
Action方法的定义：

在形参使用pojo接收上边的参数。



使用pojo点属性名传递

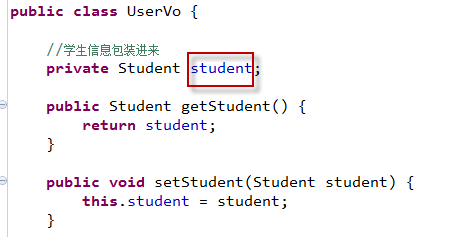
页面定义：



Action方法的定义：

在形参不能直接使用pojo接收上边的参数。

应该使用包装对象接收上边的参数。



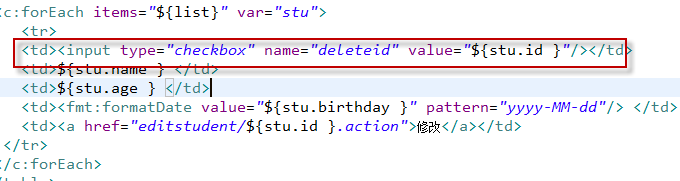
Action形参的定义：



## 字符串数组绑定

需求：在学生列表上多选，删除学生信息。

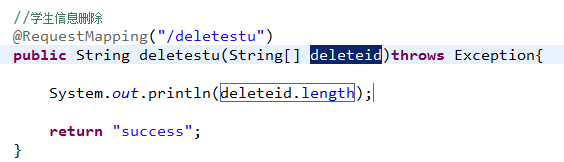
页面定义：



Action方法定义：

使用字符串数组接收。

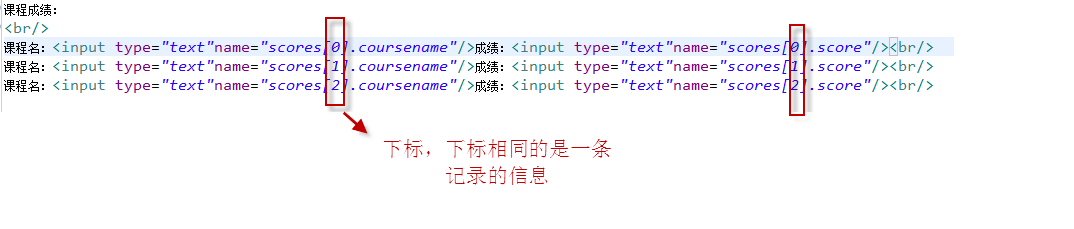
批量删除学生信息方法，接收页面复选框的值（学生id）



## List绑定

页面向action传递复杂的批量数据，比如学生的成绩信息（课程名称、成绩）

页面定义：



Scores：包装对象中list属性的名称

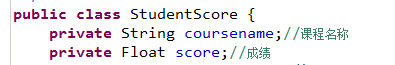
Coursename：包装对象中list中pojo的属性名称。

Score：包装对象中list中pojo的属性名称。

如果上边下标相同的Coursename和Score设置在一个pojo中。

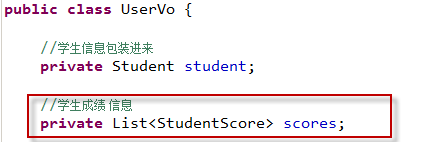
Action方法定义：

使用List<pojo>接收上边的接收，pojo中包括上边课程名称（coursename）和成绩(score)

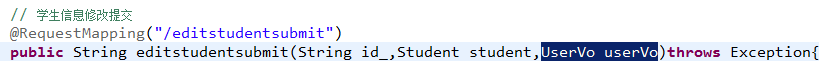


List通过包装对象接收。

在UserVo包装中定义属性：List<>



Action方法形参使用包装对象接收list



## @RequestParam绑定单个请求参数

value：参数名字，即入参的请求参数名字，如value=“studentid”表示请求的参数区中的名字为studentid的参数的值将传入；

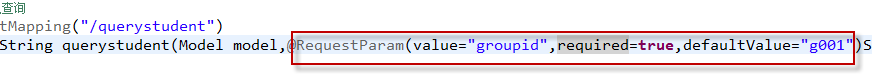
required：是否必须，默认是true，表示请求中一定要有相应的参数，否则将报400错误码；

defaultValue：默认值，表示如果请求中没有同名参数时的默认值

需求 ：

学生查询方法中必须要有一个参数group（学生分组）

实现：



# 结果转发

Redirect：请求重定向

浏览器中地址栏的url通过Redirect变了，重新定义了一个request。

Action方法通过Redirect重定向到另一个方法，方法形参无法带到另一个方法。

定义方法：



Forward：页面转发

浏览器中地址栏的url通过Forward不变，没有重新定义了一个request。

Action方法通过Forward转发到另一个方法，方法形参可以带到另一个方法。

定义方法：



# @RequestBody @ResponseBody实现json数据交互

Json数据在企业中使用好处：

Json在企业开发中已经作为通用的接口参数类型。

Json数据在页面（客户端）解析很方便。

Json定义：

var obj=｛”name”:”张三”,”age”:12｝//key/value格式

Json数据解析：

obj.name

## 需求1：

请求json响应json

页面传入json格式的数据（json串）

Action方法形参：

使用java对象接收json串。使用@RequestBody

Action返回一个json串，将java对象转json

使用@ResponseBody

实现方案：

页面提交数据时，需要将提交的数据传成json进行提交。

**Action方法中将json串转成java对象赋值形参**

可以使用@RequestBody将请求的json串转成java对象。

## 需求2

页面传入普通表单数据：key/value

Action方法接收到key/value数据，返回一个json串

请求的key/value，响应json

实现方案：

Action方法返回的java对象转成 json串。

使用@ResponseBody将action方法返回java对象转成json输出。

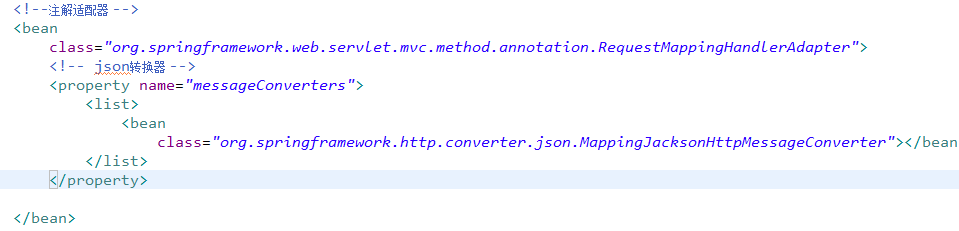
## 实现

Json转换器配置：

1. 将json转换的jar包加入工程

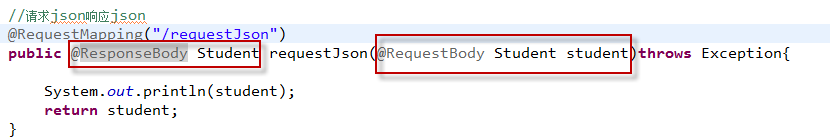


1. 在处理器适配器上配置Json转换器



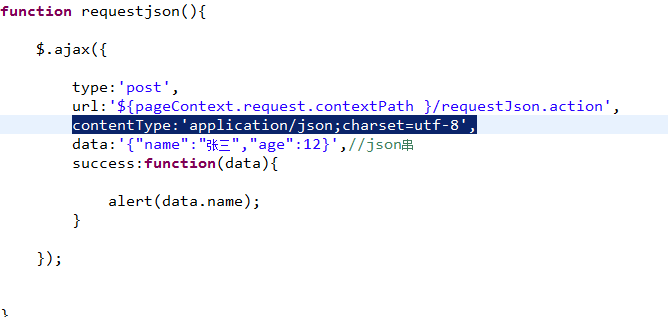
编写需求1的代码：

请求json响应json

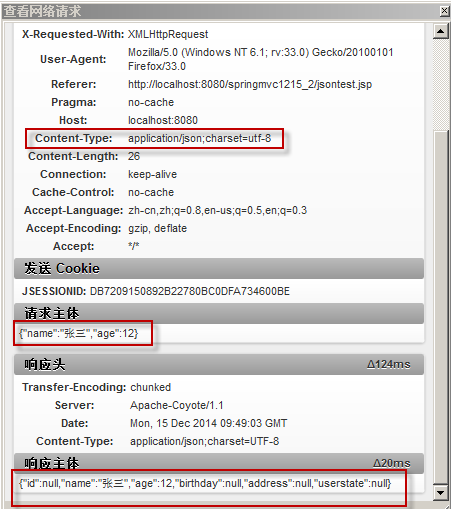


页面代码：

引入 ：jquery-1.4.4.min.js

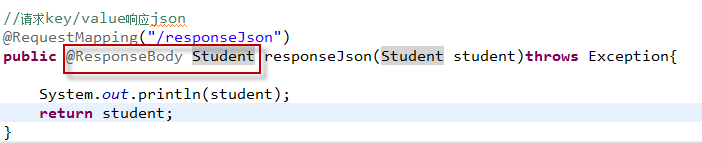


测试：



编写需求2的代码：

请求key/value，响应的json



测试：



小结：

建议使用需求2方法，请求key/value，为了客户端方便解析结果，响应json。

# 拦截器

拦截器是针对handlerMapping的拦截器，由handlerMapping查找Handler后，将拦截器返回给前端控制器。

## 配置拦截器

针对某一个handlerMapping配置拦截器

<bean

class=*"org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"*>

<property name=*"interceptors"*>

<list>

<ref bean=*"handlerInterceptor1"*/>

<ref bean=*"handlerInterceptor2"*/>

</list>

</property>

</bean>

<bean id=*"handlerInterceptor1"* class=*"springmvc.intercapter.HandlerInterceptor1"*/>

<bean id=*"handlerInterceptor2"* class=*"springmvc.intercapter.HandlerInterceptor2"*/>

间接配置全局拦截器：让springmvc框架自动向每个handlerMapping中注册拦截器

<!--拦截器 -->

<mvc:interceptors>

<!--多个拦截器,顺序执行 -->

<mvc:interceptor>

<mvc:mapping path=*"/\*\*"*/>//拦截所有请求

<bean class=*"cn.itcast.springmvc.interceptor.HandlerInterceptor1"*></bean>

</mvc:interceptor>

<mvc:interceptor>

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

<bean class=*"cn.itcast.springmvc.interceptor.HandlerInterceptor2"*></bean>

</mvc:interceptor>

</mvc:interceptors>

## 定义拦截器

实现HandlerInterceptor接口。

**public** **class** HandlerInterceptor1 **implements** HandlerInterceptor {

//handler，springmvc根据url找到Handler(只有一个方法)

//执行时机：进入Handler方法之前执行，如果返回false表示拦截，如果返回true表示放行

//使用场景：用于用户身份校验，用户权限拦截校验

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) **throws** Exception {

System.*out*.println("HandlerInterceptor1..preHandle");

**return** **false**;

}

//执行时机：进入Handler方法之后 ，在返回modelAndView之前

//使用场景：使用modelAndView，向页面传递通用数据，使用统一的view

@Override

**public** **void** postHandle(HttpServletRequest request,

HttpServletResponse response, Object handler,

ModelAndView modelAndView) **throws** Exception {

System.*out*.println("HandlerInterceptor1..postHandle");

}

//执行时机：Handler方法执行完成，（modelAndView已经返回）

//使用场景：统一异常处理，统一记录系统日志，用于action方法执行监控（在preHandle记录一个时间点，在afterCompletion记录执行结束时间点，将结束时间点减去开始执行时间点，得到执行时长）

@Override

**public** **void** afterCompletion(HttpServletRequest request,

HttpServletResponse response, Object handler, Exception ex)

**throws** Exception {

System.*out*.println("HandlerInterceptor1..afterCompletion");

}

## 测试

两个拦截器都放行

HandlerInterceptor1..preHandle

HandlerInterceptor2..preHandle

HandlerInterceptor2..postHandle

HandlerInterceptor1..postHandle

HandlerInterceptor2..afterCompletion

HandlerInterceptor1..afterCompletion

结论：

preHandle是按照拦截器定义顺序执行，

postHandle和afterCompletion是按照拦截器定义逆向执行。

### 第一个放行第二个不放行

HandlerInterceptor1..preHandle

HandlerInterceptor2..preHandle

HandlerInterceptor1..afterCompletion

结论：

只要有一个拦截器不放行，action方法无法完成。

如果拦截器放行，afterCompletion才会执行。

只要有一个拦截器不放行，postHandle不执行。

两个都不放行

HandlerInterceptor1..preHandle

结论：

只要有一个拦截器不放行，action方法无法完成。

只要有一个拦截器不放行，postHandle不执行。

## 拦截器应用

案例：

用户身份认证拦截，用户登陆成功后，系统记录session（用户身份信息），用户去操作url时，拦截器需要校验用户身份是否合法（查看session中是否有用户身份信息，如果没有说明用户身份不合法，不合法重新登陆）

action

@Controller

**public** **class** LoginAction {

//登陆页面

@RequestMapping("/login")

**public** String login(Model model)**throws** Exception{

**return** "login";

}

//登陆提交

//userid：用户账号，pwd：密码

@RequestMapping("/loginsubmit")

**public** String loginsubmit(HttpSession session,String userid,String pwd)**throws** Exception{

//向session记录用户身份信息

session.setAttribute("activeUser", userid);

**return** "redirect:stu/querystudent.action";

}

//退出

**public** String logout(HttpSession session)**throws** Exception{

//session过期

session.invalidate();

**return** "redirect:stu/querystudent.action";

}

}

页面

@Controller

**public** **class** LoginAction {

//登陆页面

@RequestMapping("/login")

**public** String login(Model model)**throws** Exception{

**return** "login";

}

//登陆提交

//userid：用户账号，pwd：密码

@RequestMapping("/loginsubmit")

**public** String loginsubmit(HttpSession session,String userid,String pwd)**throws** Exception{

//向session记录用户身份信息

session.setAttribute("activeUser", userid);

**return** "redirect:stu/querystudent.action";

}

//退出

@RequestMapping("/logout")

**public** String logout(HttpSession session)**throws** Exception{

//session过期

session.invalidate();

**return** "redirect:stu/querystudent.action";

}

}

拦截器

拦截所有url（将公开地址排除（无需登陆即可操作的url）），校验用户身份是否合法（查看session中是否有用户身份信息，如果没有说明用户身份不合法，不合法重新登陆）

@Override

**public** **boolean** preHandle(HttpServletRequest request,

HttpServletResponse response, Object handler) **throws** Exception {

//判断请求的url是否公开 地址(无需登陆即可操作url)

//正常开发时，需要将公开地址配置在配置文件中。

//取出请求的url

String url = request.getRequestURI();

**if**(url.indexOf("loginsubmit.action")>=0){

//说明 公开地址

//放行

**return** **true**;

}

//得到session

HttpSession session = request.getSession();

//从 session取出用户身份信息

String userid = (String) session.getAttribute("activeUser");

**if**(userid!=**null**){

//说明 用户已登陆（用户身份合法）

//放行

**return** **true**;

}

//执行到这里说明 用户身份不合法，拦截，跳转到登陆页面

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

**return** **false**;

}

# <mvc:annotation-driven />

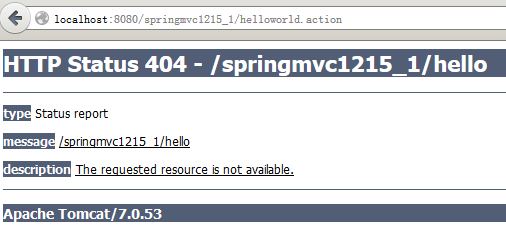


# 与hibernate整合

将spring和hibernate整合好，springmvc作为spring一个模块不用整合。

# 问题

前端控制器调用适配器执行Handler，执行完成后，返回view找不到。



前端控制器调用处理器映射器，找不到Handler

