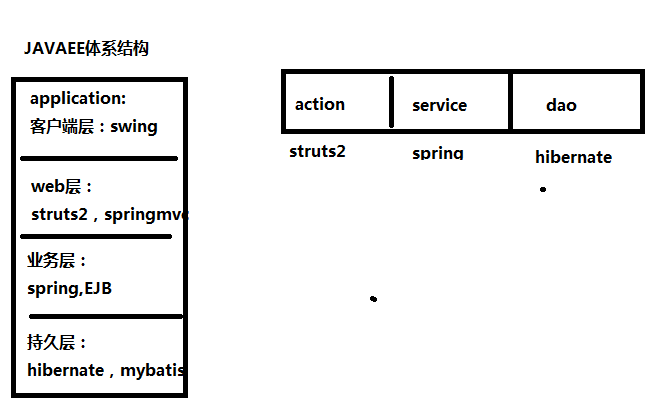
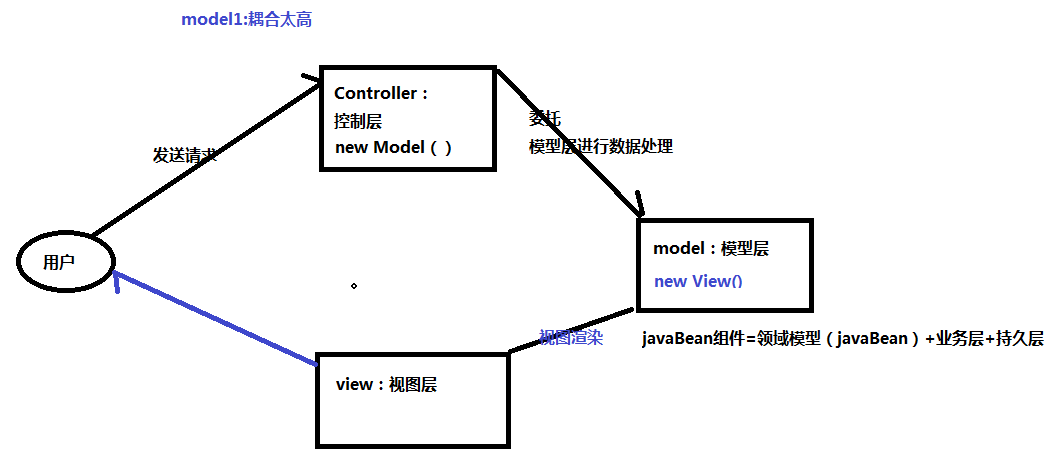
# Springmvc

# JAVAEE体系结构

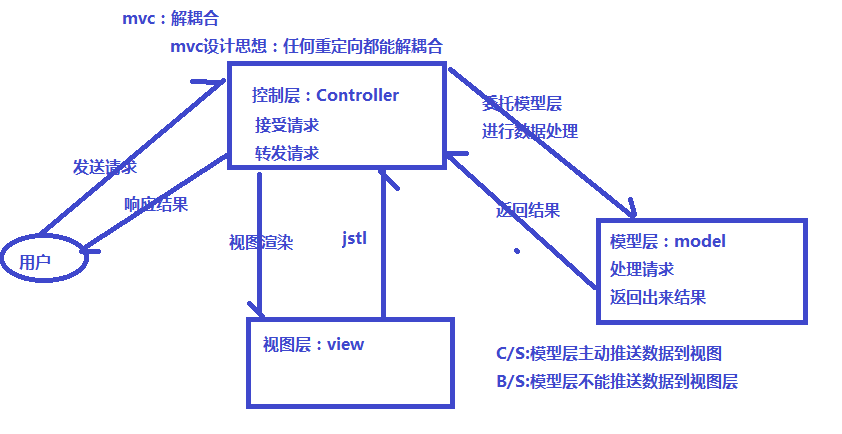


# 什么是mvc？

Model1：



Model2：



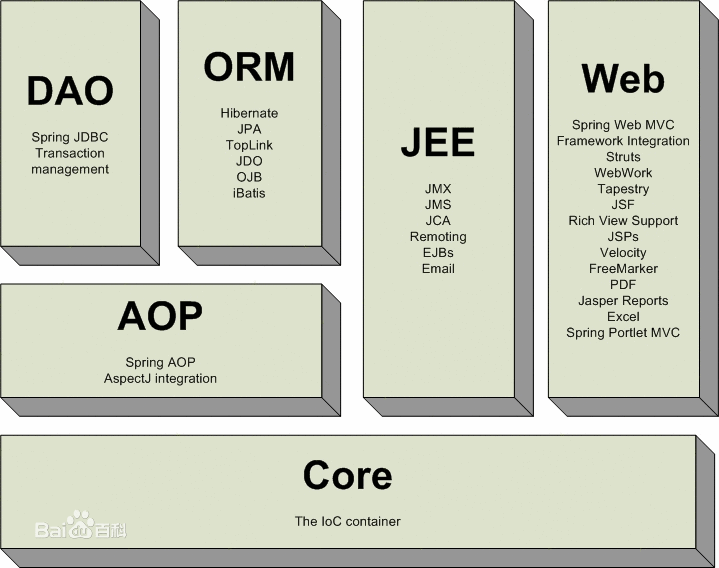
Cs可以主动推送数据

而bs不能主动推送数据到客户端

# Springmvc是什么?

Springmvc是一个web层mvc框架，类似struts2.

# Springmvc和spring？



Springmvc是spring的一部分。

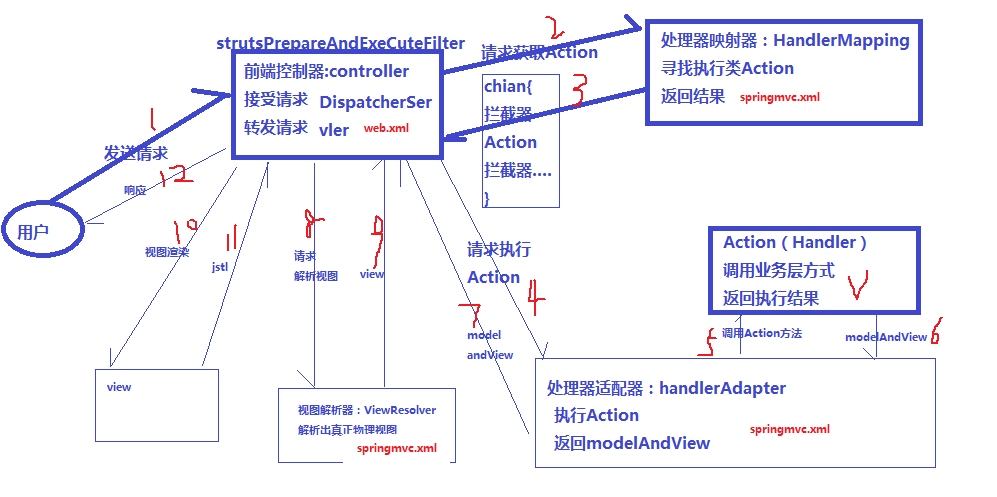
# Springmvc执行流程

Struts2执行流程：

* strutsPrepareAndExcuteFilter拦截请求（控制层），拦截请求，转发请求
* 寻找Action执行
* ActionProxy：strutsActionProxy extends defaultActionProxy
* ActionMapping去寻找执行类Action

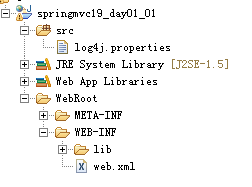
根据mvc设计模式：

自己来设计springmvc？

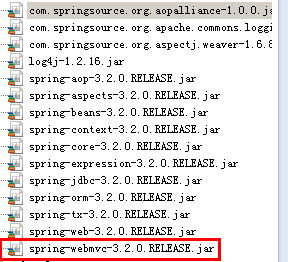


# Springmvc入门程序

## 创建一个web工程



## 导入jar



Spring-core spring-aop spring-context spring-jdbc spring-orm spring-beans spring-aspects spring-webmvc

## 配置web.xml

在web.xml配置前端控制器：DispatcherServlet



## 配置springmvc.xml

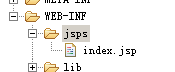


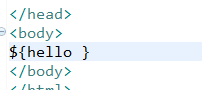
## 自定义Controller



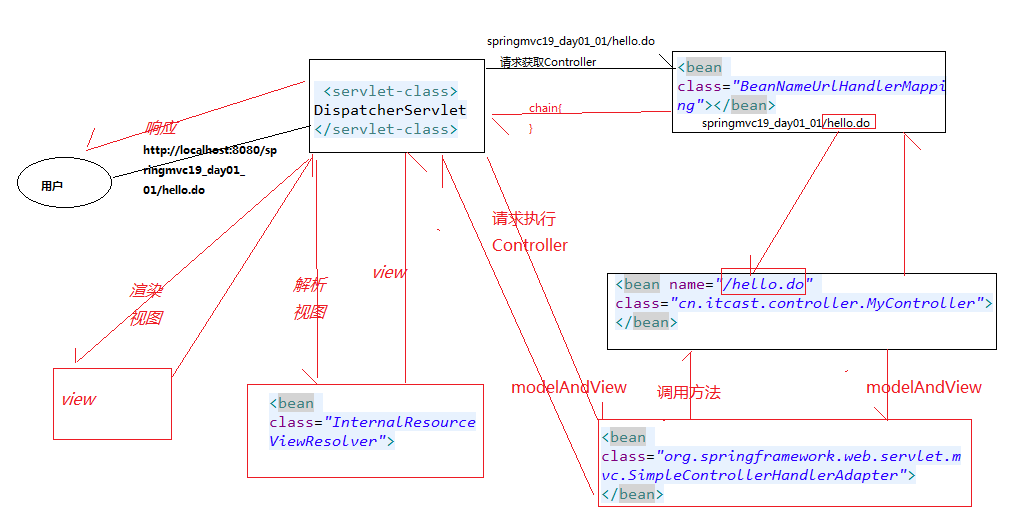
## 定义视图页面

根据视图解析路径：WEB-INF/jsps/index.jsp



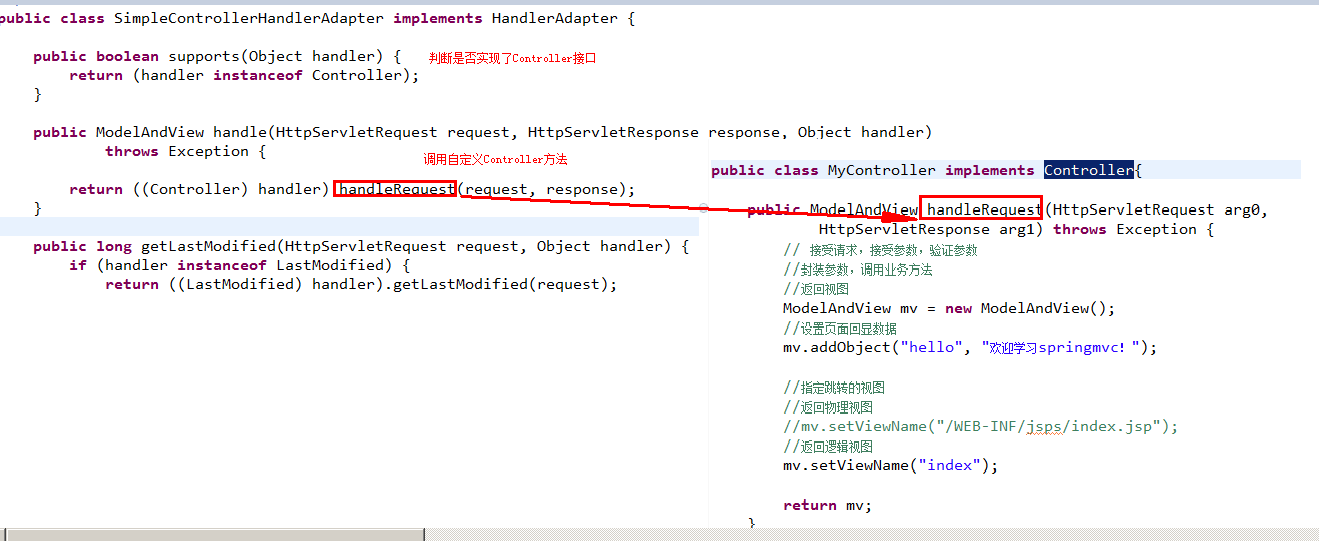


## 根据代码分析springmvc执行流程



Dispatcherservlet（初始化）🡪doDispatch（Dispatcherservlet类，父类重写了service方法调用了doService,然后调用doDispatch）🡪handlerAdapter（handle）🡪Controller(handleRequest)🡪simplecontrollerhandleradapter(handleRequestInternal)🡪controller

## 适配器源码

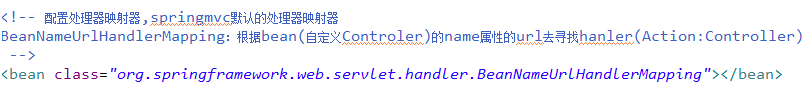


# 处理器映射器

## BeanNameUrlHandlerMapping

功能：寻找Controller

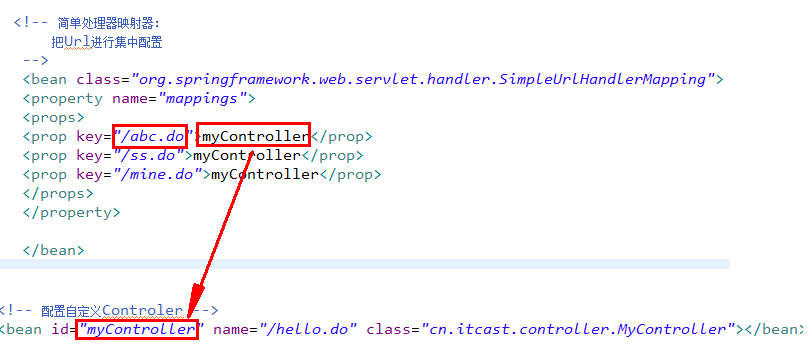
根据url请求去匹配bean的name属性url，从而获取Controller



## SimpleUrlHandlerMaping

功能：寻找Controller

根据浏览器url匹配简单url的key，key由Controller的id找到Controller



## ControllerClassNameHandlerMapping

功能：寻找Controller

根据类名（MyController）类名.do来访问,类名首字母小写



## 映射器之间能不能共存？

3个处理器映射器可以共存。

# 处理器适配器

## SimpleControllerHandlerAdapter

功能：执行controller

调用controller里面方法，返回modelAndView。



## HttpRequestHandlerAdapter

功能：执行controller



## 2个处理器适配器能共存？

可以共存

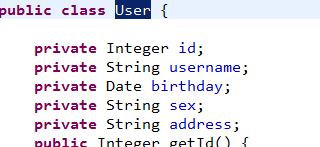
# 命令控制器

Springmvc通过命令设计模式接受页面参数。

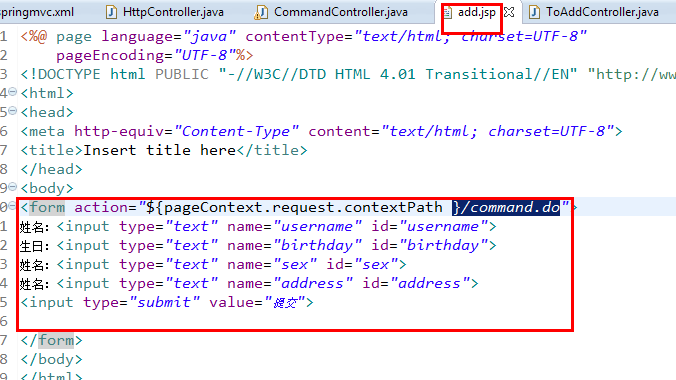
## 自定义命令控制器



## 定义javaBean



## 封装参数页面

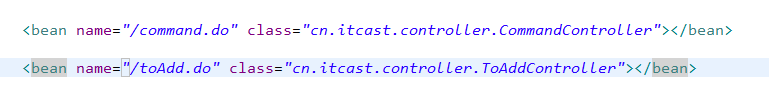


## 跳转到add页面

由于add页面在WEB-INF下面不能直接访问，需要通过Controller来访问（me：web-inf不能被外部直接访问）。

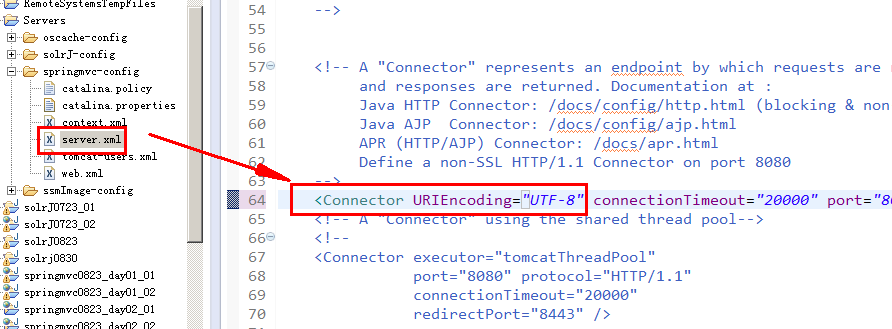


## 在springmvc配置bean



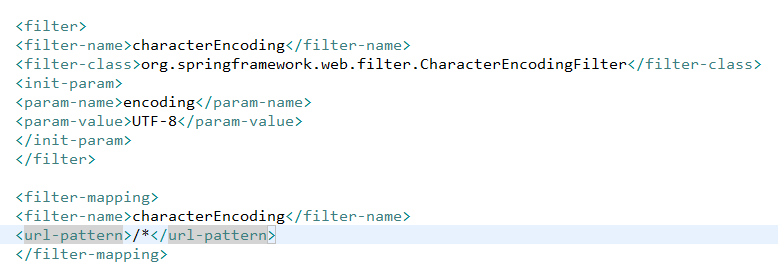
# 中文乱码解决

## Get请求乱码

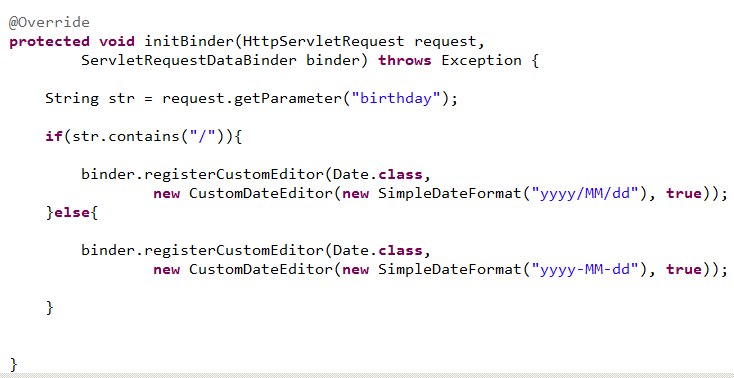


## Post乱码

Spring编码过滤器：在web.xml配置

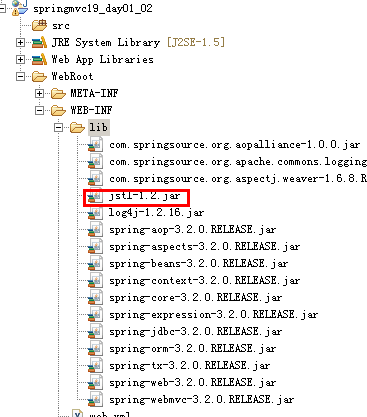


# 时间类型转换



# 注解开发

## 创建一个web工程，并导入jar



## 配置web.xml

<filter>

<filter-name>characterEncoding</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>characterEncoding</filter-name>

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

</filter-mapping>

<servlet>

<servlet-name>springmvc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<!-- 默认加载方式

默认加载必须规范：

\* 文件命名：servlet-name-servlet.xml====springmvc-servlet.xml

\* 路径规范：必须在WEB-INF目录下面

-->

<init-param>

<param-name>contextConfigLocation</param-name>

<param-value>classpath:springmvc.xml</param-value>

</init-param>

</servlet>

<servlet-mapping>

<servlet-name>springmvc</servlet-name>

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

</servlet-mapping>

## 配置springmvc配置文件

<context:component-scanbase-package=*"cn.itcast"*></context:component-scan>

<!-- 配置注解处理器映射器

功能：寻找执行类Controller

-->

<beanclass=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerMapping"*></bean>

<!-- 配置注解处理器适配器

功能：调用controller方法，执行controller

-->

<beanclass=*"org.springframework.web.servlet.mvc.method.annotation.RequestMappingHandlerAdapter"*></bean>

<!-- 配置sprigmvc视图解析器：解析逻辑视图

后台返回逻辑试图：index

视图解析器解析出真正物理视图：前缀+逻辑试图+后缀====/WEB-INF/jsps/index.jsp

-->

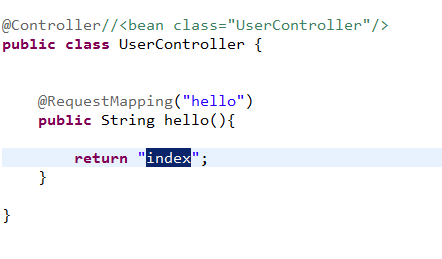
<bean class=*"org.springframework.web.servlet.view.InternalResourceViewResolver"*>

<propertyname=*"prefix"*value=*"/WEB-INF/jsps/"*></property>

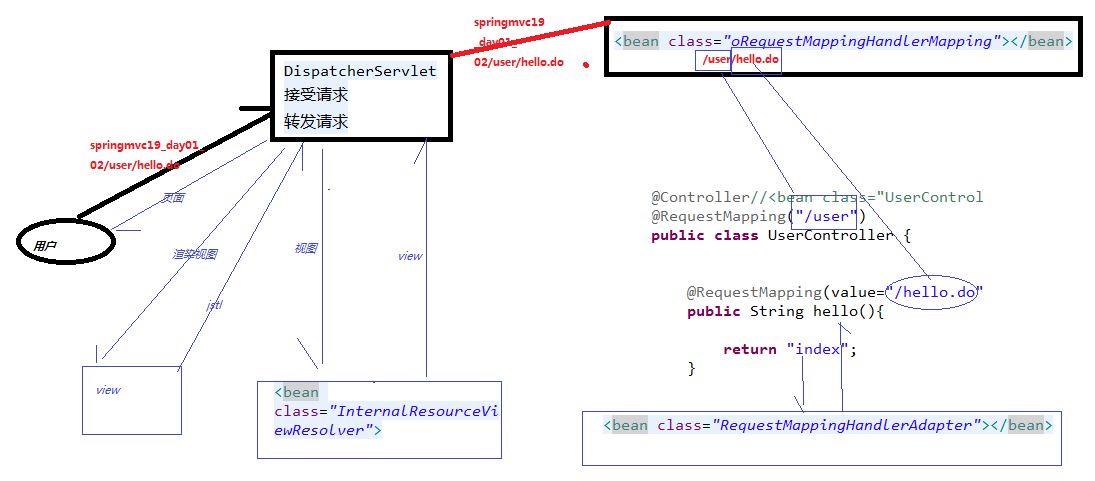
<propertyname=*"suffix"*value=*".jsp"*></property>

</bean>

## 自定义Controller类



## 注解开发流程

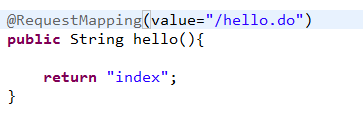


# RequestMapping

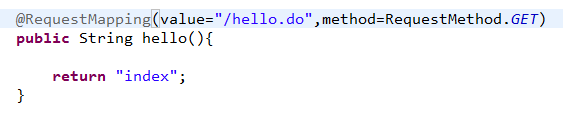
**requestMapping(“hello”)**

requestMapping(“/hello.do”)

requestMapping(value=”/hello.do”)



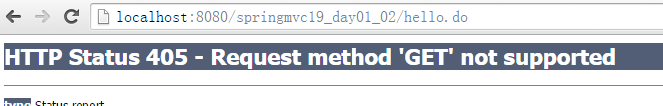
requestMapping(value=”/hello.do”,method=RequestMethod.GET)

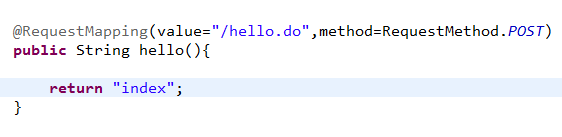


requestMapping(value=”/hello.do”,method=RequestMethod.POST)

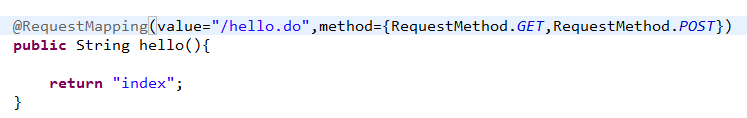
浏览器直接访问,a标签都是get请求

表单提交(指定post)，ajax指定post提交，post提交。





requestMapping(value=”/hello.do”,method={RequestMethod.POST,RequestMethod.GET})



# RequestMaping根路径

@RequestMapping（”/user”）

UserController｛

requestMapping(“save”)

Save()

requestMapping(“update”)

Update{}

requestMapping(“find”)

Fiind()

｝

项目名/user/save.do

@RequestMapping（”/items”）

ItemsController｛

requestMapping(“save”)

Save()

requestMapping(“update”)

Update{}

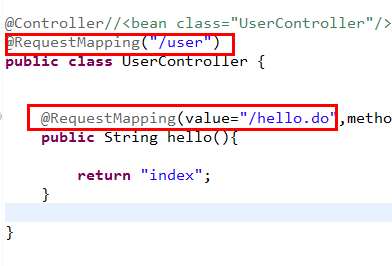
requestMapping(“find”)

Fiind()

｝

项目名/items/save.do

## 自定义根路径



# 封装参数

分析接受参数类型：

基本类型，int，String等等基本类型。

Pojo类型

包装类型

Springmvc默认支持类型：

HttpSession，HttpRequstServlet，Model等等。

Struts2参数：基于属性封装。

Springmvc参数封装：基于方法进行封装。

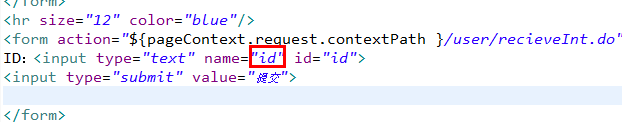
## 基本类型

### 需求

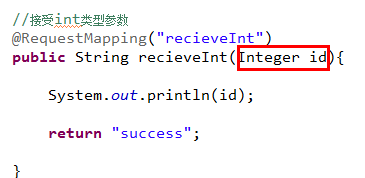
封装int类型参数

### 页面

页面传递参数都是字符串。

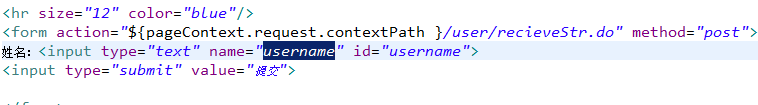


### 接受参数方法

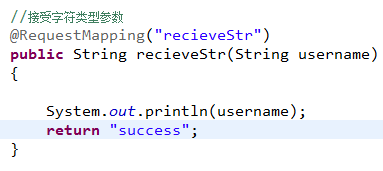


## 接受字符串类型

### 页面



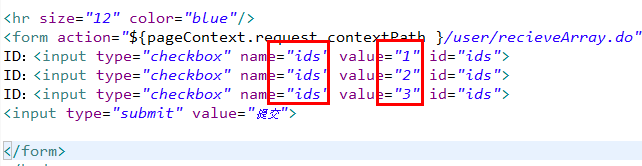
### 代码



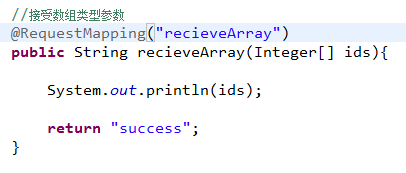
## 接受数组

分析：批量删除：checkbox复选框。Value必须有值。

### 页面

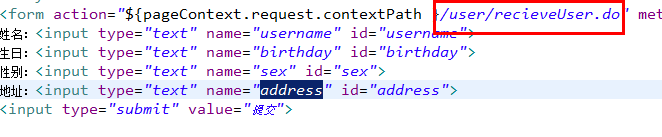


### 代码

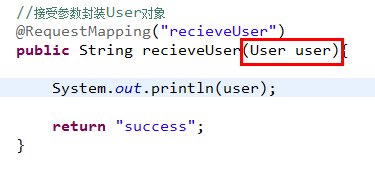


## 接受Pojo

### 页面



### 代码



## 接受包装类型参数

userCustom｛

private user user；

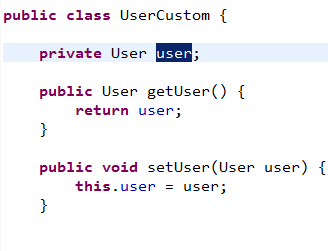
private List<User> userList;

private Map<K,V> maps;

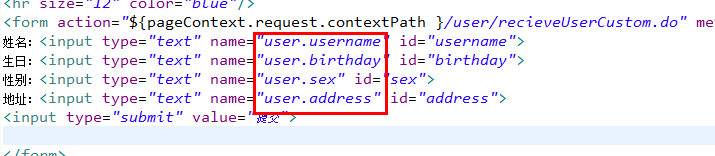
private items items;

｝

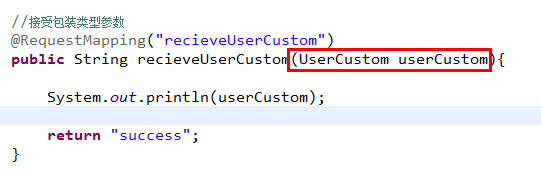
### 定义UserCustom



### 页面

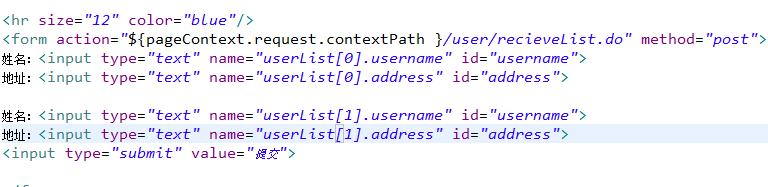


### 代码

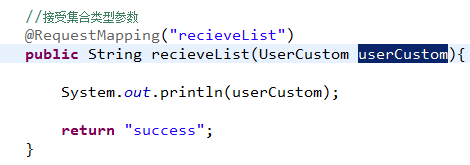


## 接受集合类型参数

### 接受list集合

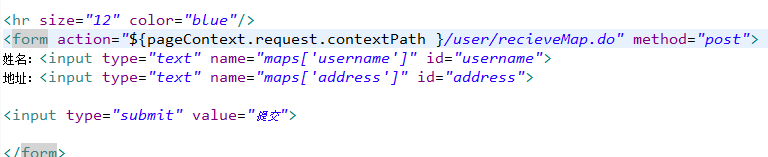


代码：

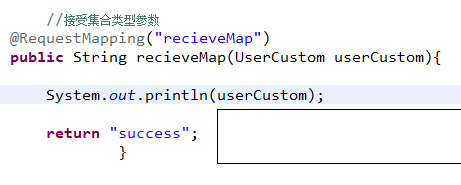


### 接受map

#### 页面



#### 代码



## 有了struts2，为什么还需要sprigmvc？

实现机制：

Struts2是基于过滤器实现的。

Springmvc基于servlet实现。Servlet比过滤器快。

运行速度：

Struts2是多列

请求来了以后，struts2创建多少个对象：

ActionContext，valuestack，UserAction，ActionSuport，ModelDriven

userAction里面属性：User对象，userlist集合等

Springmvc是单列。

参数封装来分析：

Struts基于属性进行封装。

Springmvc基于方法封装。

# 页面回显

## 查询所有

@RequestMapping("list")

**public** String list(Model model){

//model 相当于application域对象

List<User> userList = **new** ArrayList<User>();

User user1 = **new** User();

user1.setId(1);

user1.setSex("男");

user1.setUsername("张山峰");

user1.setAddress("武当山");

user1.setBirthday(**new** Date());

User user2 = **new** User();

user2.setId(2);

user2.setSex("男2");

user2.setUsername("张山峰222");

user2.setAddress("武当山222");

user2.setBirthday(**new** Date());

User user3 = **new** User();

user3.setId(3);

user3.setSex("男3");

user3.setUsername("张山峰333");

user3.setAddress("武当山333");

user3.setBirthday(**new** Date());

userList.add(user1);

userList.add(user2);

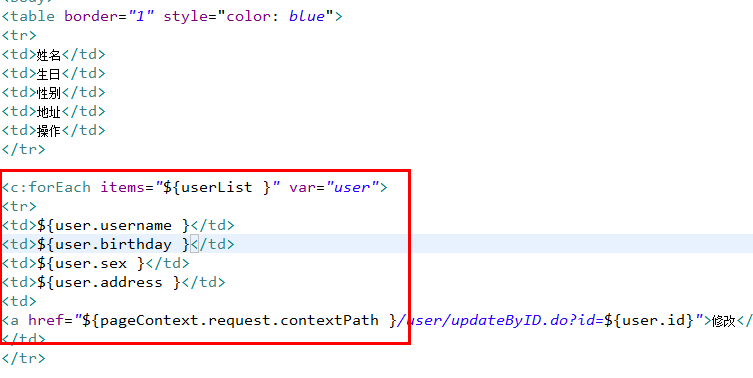
userList.add(user3);

model.addAttribute("userList", userList);

**return**"list";

}

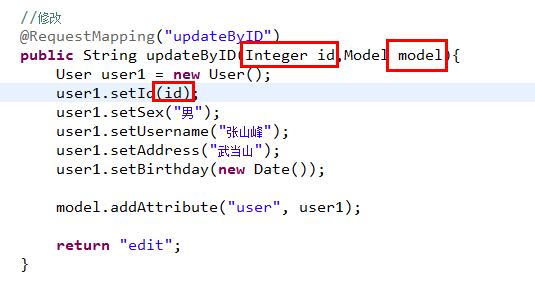
## 页面获取



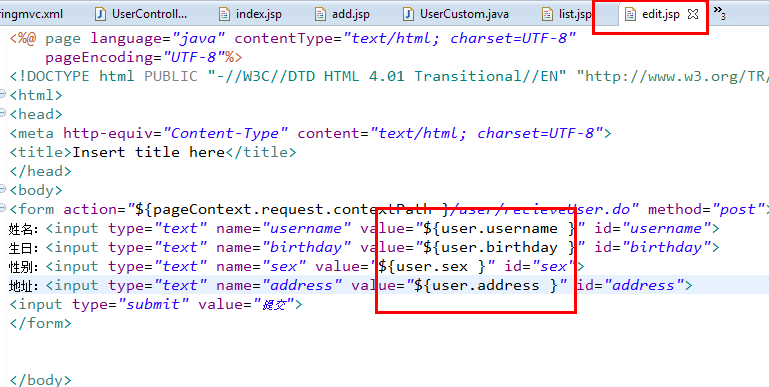
## 修改



## 修改代码



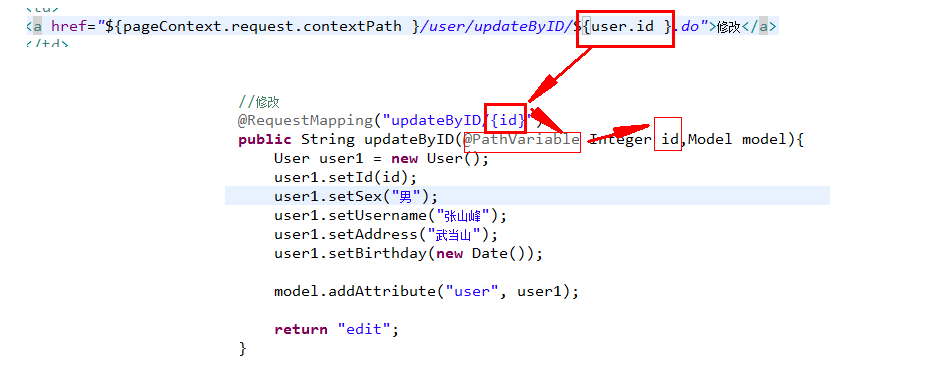
## 回显



# URL模版映射

url模版映射可以restfull软件架构。

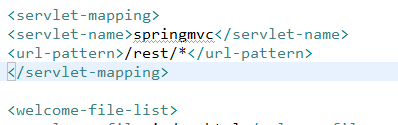
## url模版映射过程



## Restfull风格设计

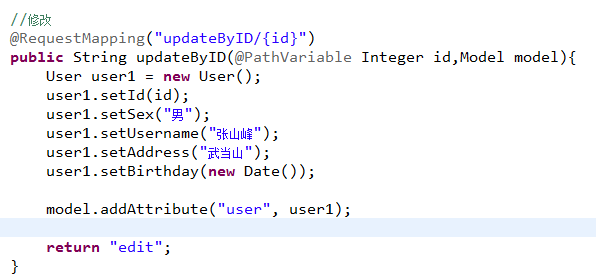


Web.xml拦截方式：在rest目录下所有请求都被拦截，servlet可以拦截目录。



{}:匹配接受页面Url路径参数

@Pathariable：{}里面参数注入后面参数里面



# 转发和重定向

## 转发

关键字：forward

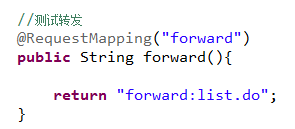
本类进行转发：

本类方法与方法之间进行forward

转发方式：

方式一：return ”forward：list.do“；

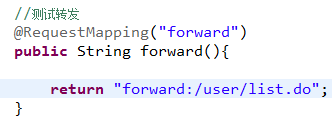
代码：



测试方式：在list方法打断点，如果断点能成功，证明转发成功。



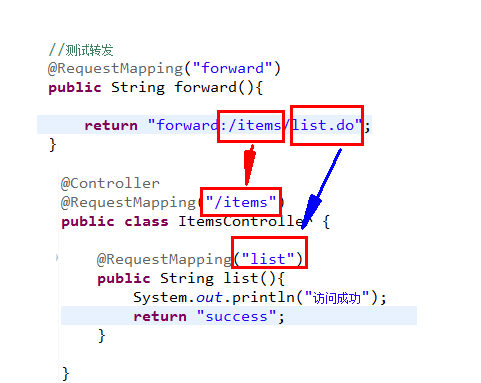
方式二：return ”forward：/user/list.do“；



注意：user根路径前面必须有/.

跨类进行转发：

转发方式：return ”forward：/items/list.do“；



## 重定向

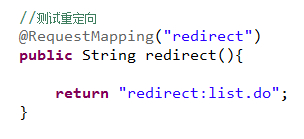
关键字：redirect

本类进行重定向：

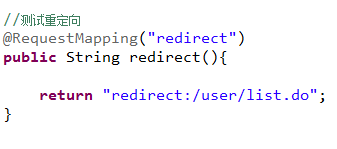
本类方法与方法之间进行redirect

重定向方式：

方式一：return ”redirect：list.do“；



方式二：return ”redirect：/user/list.do“；



跨类进行重定向：

转发方式：return ”redirect：/items/list.do“；

