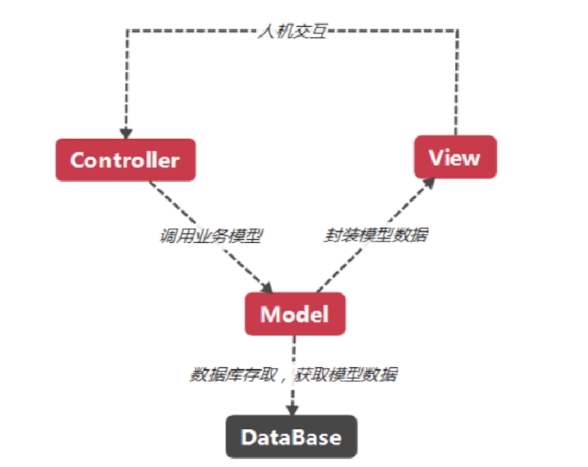
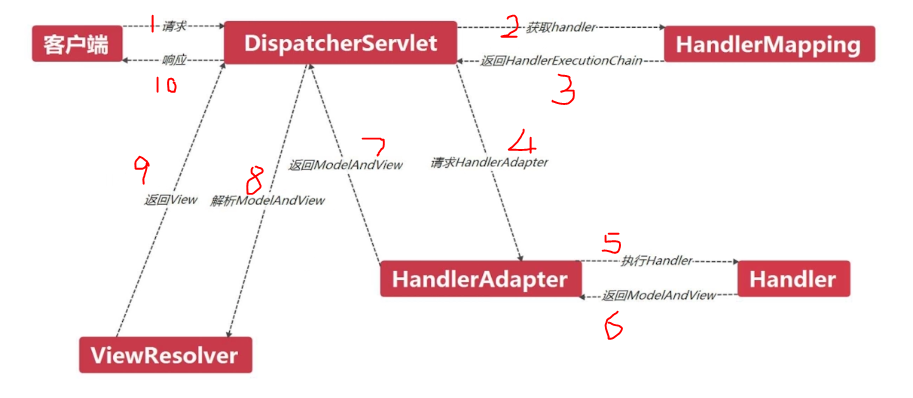
SpringMVC

MVC示意图：



SpringMVC核心组件：

1. DispatcherServlet 前置控制器
2. Handler 处理器，完成业务逻辑
3. HandlerMapping 将请求映射到Handler
4. HandlerInterceptor 处理器拦截器
5. HandlerExecutionChain 处理器执行链
6. HandlerAdapter 处理器适配器
7. ModelAndView 装载模型数据和视图信息
8. ViewResolver 视图解析器



基于XML配置的实现方法：

1. 在web.xml文件中配置DispatcherServlet

<**servlet**>  
 <**servlet-name**>SpringMVC</**servlet-name**>  
 <**servlet-class**>org.springframework.web.servlet.DispatcherServlet</**servlet-class**>  
 <**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**>/</**url-pattern**>  
</**servlet-mapping**>

1. 配置Handler和Mapping

<**bean id="handlerMapping" class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping"**>  
 <**property name="mappings"**>  
 <**props**>  
 *<!--为test请求配置Handler-->* <**prop key="/test"**>testHandler</**prop**>  
 </**props**>  
 </**property**>  
</**bean**>  
  
*<!--配置handler-->*<**bean id="testHandler" class="com.example.handler.MyHandler"**></**bean**>

1. 配置视图解析器

<**bean class="org.springframework.web.servlet.view.InternalResourceViewResolver"**>  
 *<!--前缀-->* <**property name="prefix" value="/"**></**property**>  
 *<!--后缀-->* <**property name="suffix" value=".jsp"**></**property**>  
</**bean**>

基于注解的实现方法：

区别去XML方式的第二步，采用注解配置Handler和映射关系。

在XML文件中开启自动扫描

在Handler类上添加注解@Controller

在包装方法上添加注解@RequestMapping(“/test”)

问题：

1 静态资源不能加载。原因是静态资源请求也被DispatcherServlet拦截，需要设置

<**servlet-mapping**>  
 <**servlet-name**>default</**servlet-name**>  
 <**url-pattern**>\*.css</**url-pattern**>  
</**servlet-mapping**>

在springmvc.xml文件中设置

<**mvc:default-servlet-handler**/>

2 编码过滤器，SpringMVC框架中提供了配置方法

<**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**>  
 <**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**>

**数据绑定**，将Http请求中的参数绑定到Handler业务方法的形参。

传统方式，需要通过HttpRequest获取参数，然后封装为对象。SpringMVC会自动完成数据的封装操作。

常用的数据绑定类型

基本数据类型、包装类、数组、对象、集合、JSON

@ResponseBody注解，执行完Handler方法，直接将结果返回到前台，不跳转。

1 绑定基本类型、包装类时，用@RequestParam(value=”id”)注解放在形参前面，来指定访问的参数名称；

2 绑定对象时，在jsp界面的name属性要和对象类的属性名一一对应，如果有级联对象，如要获取tutor对象下的name属性，jsp中tutor.name

3 绑定对象List集合时，**需要新建一个类**，有属性List<Course> courses，在jsp界面中name属性使用下标，如courses[0].name

4 绑定对象Map集合时，需要新建一个类，有属性Map<String, Course> courseMap，在jsp界面中name属性使用键值，如courses[‘one’].name

5 绑定json数据，需要添加依赖包jackson-databind，并且在XML中配置

<**mvc:annotation-driven**>  
 <**mvc:message-converters**>  
 <**bean class="org.springframework.http.converter.json.MappingJackson2HttpMessageConverter"**></**bean**>  
 </**mvc:message-converters**>  
</**mvc:annotation-driven**>

Handler方法中，@RequestBody注解放在参数上，表示接收json并转化为实体类对象。

RESTful

REST Representational State Transfer 表述性状态转移

符合REST的约束条件和原则的架构，称为RESTful架构。

资源的表述：资源在客户端和服务端之间传送；

状态转移：客户端获取资源后，做出修改

RESTful架构特点：

1. 统一了客户端访问资源的接口；
2. url更加简洁，易于理解，便于扩展；
3. 有利于不同系统之间的资源共享

RESTful具体来讲，就是HTTP协议的四种形式表示四种基本操作

GET：获取资源

POST：新建资源

PUT：修改资源

DELETE：删除资源

首先要在web.xml文件中，配置一个filter，filter-class为org.springframework.web.filter.HiddenHttpMethodFilter。配置完成后，form表单可以支持DELETE和PUT请求。

编码时，将RequestMapping换成GetMapping PostMapping PutMapping DeleteMapping，访问的参数不再使用?，而是通过/连接到url后，如@GetMapping(value=”/getById/{id}”)，形参前添加注解@PathVariable(value=”id”)，jsp界面中如下：

<form action=”${pageContext.request.contextPath}/getById/${course.id}” method=”get”>

怎样将PUT请求转化为POST请求，利用隐藏的input标签，如下：

<input type=”hidden” name=”\_method” value=”PUT”/>

拦截器和过滤器

拦截器使用JDK动态代理实现的，拦截的是对应的调用方法。

过滤器使用Filter实现的，拦截的是request对象。

拦截器配置方法

在XML文件中引入mvc的命名空间和schemaLocation

注册拦截器方法如下

<**mvc:interceptors**>  
 <**mvc:interceptor**><**mvc:mapping path="/user/\*\*"**></**mvc:mapping**>  
 *<!--exclude-mapping在所有拦截中进行排除，一般在通配符会有意义。-->* <**mvc:exclude-mapping path="/user/updatepwd"**></**mvc:exclude-mapping**>  
 <**mvc:exclude-mapping path="/user/updatebackground/\*"**></**mvc:exclude-mapping**>  
 <**bean class="com.imooc.core.LoginInterceptor"**></**bean**>  
 </**mvc:interceptor**>  
</**mvc:interceptors**>

Interceptor中的三个方法：

preHandle() 在业务处理器处理请求之前被调用，如果返回false，不再继续执行；

postHandle() 在业务处理器处理完请求之后，DispatcherServlet向客户端返回响应前被调用；

afterCompletion() 在DispatcherServlet完全处理完请求之后被调用，可以在该方法中进行一些资源清理的操作。

多个拦截器的执行过程

