SpringMVC

1.SpringFramework

Spring框架是Java平台的一个开源的全栈（full-stack）应用程序框架和控制反转容器实现，一般被直接称为spring。该框架的一些核心功能理论上可用于任何Java 应用。

核心功能模块

控制反转容器（依赖注入）

控制反转（IOC，Inverse Of Control），即把创建对象的权利交给框架，也就是指对象的创建，对象的储存，对象的管理交给了Spring容器。Spring容器是Spring中的一个核心模块，用于管理对象，底层可以理解为一个Map集合.

面向切面编程

面向切面编程（Aspect-Oriented Programming,AOP）就是将那些与业务无关，却为业务模块所共同调用的逻辑或责任分开封装起来，便于减少系统的重复代码，降低模块间的耦合度，并有利于未来的可操作性和可维护性

数据访问（Dao层支持）

Spring Data实现了对数据访问接口的统一，支持多种数据库访问框架或组件（如：JDBC，Hibernate，MyBatis（iBatis））作为最终数据访问的实现

2.什么是MVC

MVC是一种软件架构的思想，将软件按照模型，视图，控制器来划分

M：Model，模型层，指工程中的JavaBean，作用是处理数据

JavaBean分为两类：

一类为实体类Bean：专门存储业务数据的，如Student，User等

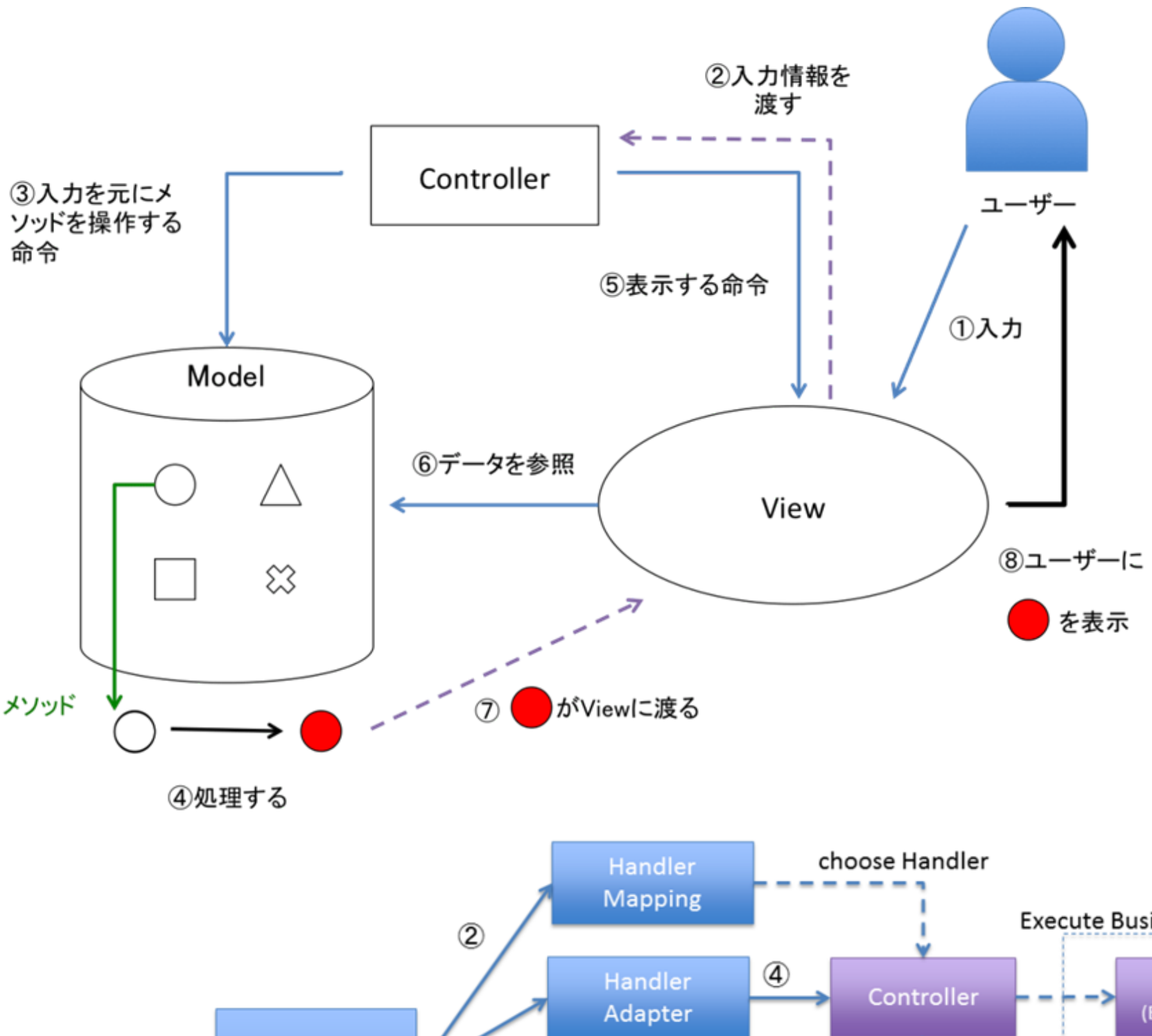
一类称为业务处理Bean：指Service或Dao对象，专门用于处理业务逻辑和数据访问

V：View，视图层，指工程中的html或jsp等页面，作用是与用户进行交互，展示数据

C：Controller, 控制层，指工程中的servlet，作用是接收请求和响应浏览器

MVC的工作流程：

用户通过视图层发送请求到服务器，在服务器中请求被Controller接收，Controller调用相应的Model层处理请求，处理完毕将结果返回到Controller，Controller再根据请求处理的结果找到相应的View视图，渲染数据后最终响应给浏览器



3.什么是SpringMVC

SpringMVC是Spring的一个后续产品，是Spring的一个子项目
SpringMVC是Spring为表述层开发提供的一整套完备的解决方案。表述层框架历经Strust，WebWork，Strust2等诸多产品历代更迭之后，目前业界普遍选择了SpringMVC作为JavaEE项目表述层开发的首选方案
注：三层架构分为表述层（或表示层），业务逻辑层，数据访问层，表述层表示前台页面和后台servlet

4.SpringMVC的特点

Spring家族原生产品，与IOC容器等基础设施无缝对接
基于原生的Servlet，通过了功能强大的前端控制器Dispatcherservlet,对请求和响应经行统一处理
表述层各细分领域需要解决的问题全方位覆盖，提供全面解决方案
代码清晰简洁，大幅度提升开发效率
内部组件化程度高，可插拔式组件即插即用，想要什么功能配置相应组件即可
性能卓著，尤其适合现代大型，超大型互联网项目要求

二 . HelloWorld

1. 开发环境
   idea2019.2
   构建工具 Maven3.5.4
   服务器 tomcat7
   spring版本 5.3.1

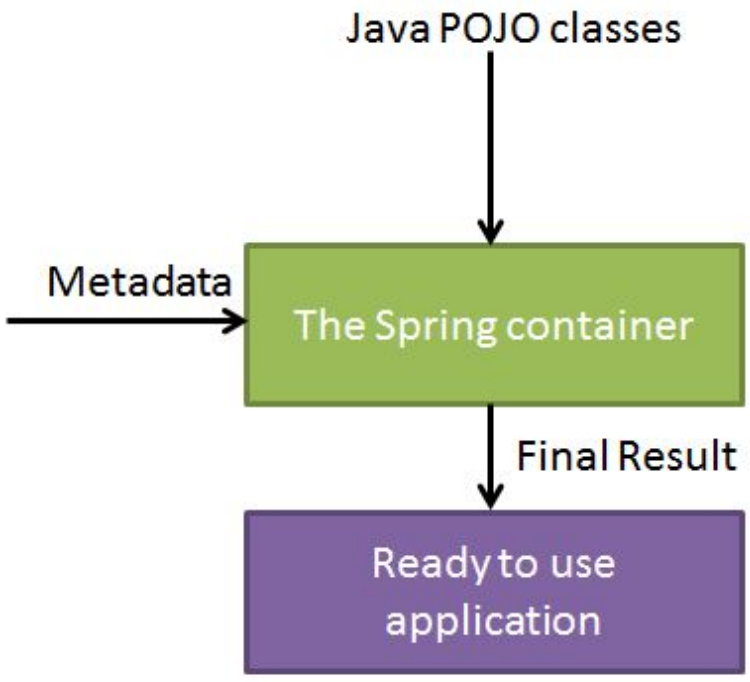
2. 创建maven工程

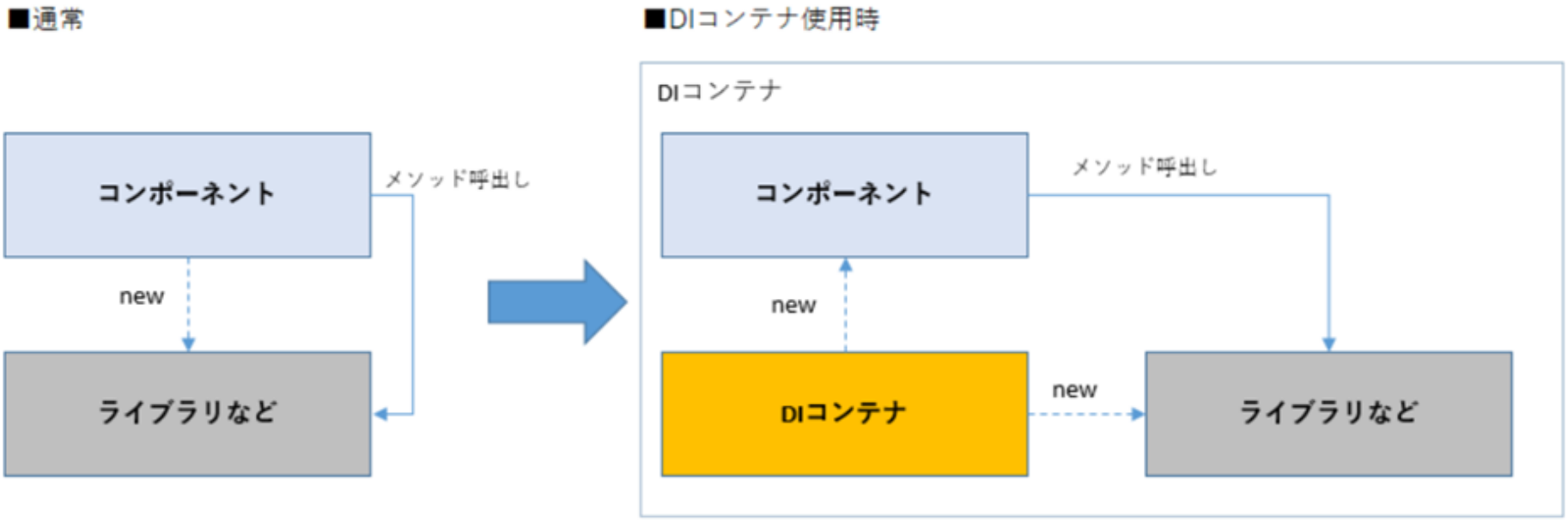
   a.添加web模块
   b.打包方式： war
   c.引入依赖
   web.xml必须要有的东西，放在webapp目录下

3. 配置web.xml
   注册SpringMVC的前端控制器DispatcherServlet
   a>默认配置方式
   b>扩展配置方式
   <servlet>　　配置SpringMVC的前端控制器，对浏览器发送的请求经行统一处理
   <init-param>　　配置springMVC配置文件的位置和名称
   <load-on-startup>　　将控制器DispatcherServlet的初始化时间提前到服务器启动时
   <context:component-scan>　　扫描组件

   配置Thymeleaf视图解析器 ThymeleafviewResolver

4. Spring IoC（Inversion of Control） 容器
   Spring容器是Spring框架的核心。容器将创建对象，把它们连接在一起配置它们，并管理他们的整个生命周期从创建到销毁。Spring容器使用依赖注入（DI）来管理组成一个应用程序的组件。这些对象被称为Spring Beans。
   通过阅读配置元数据提供的指令，容器知道对哪些对象进行实例化，配置和组装。配置元数据可以通过XML，Java注释或Java代码来表示。
   Spring IoC容器利用Java的POJO类和配置元数据来生成完全配置和可执行的系统或应用程序
   IOC容器具有依赖注入功能的容器，它可以创建对象，IOC容器负责实例化，定位，配置应用程序中的对象及建立这些对象间的依赖。通常new一个实例，控制权由程序员控制，而“控制反转”是指new实例工作不由程序员来做而是交给Spring容器来做。在Spring中BeanFactory是IOC容器的实际代表





5.注解后扫描　使spring ioc知道组件的类型

创建的类通过注解进行标识

@Controller　　控制层组件　　返回视图名称

@RequestMapping　　请求映射

作用就是将请求和处理请求的控制器方法关联起来，建立映射关系

SpringMVC接收到指定的请求，就会来找到在映射关系中对应的控制器方法来处理这个请求

@RequestMapping标识一个类：设置映射请求的请求路径的初始信息

@RequestMapping标识一个方法：设置映射请求请求路径的具体信息

value　value属性通过请求的请求地址匹配请求映射

value属性是一个字符串类型的数组，表示该请求映射能够匹配多个请求　　地址所对应的请求

value属性必须设置，至少通过请求地址匹配请求映射

method　通过请求的请求方式（get或post）匹配请求映射

method属性是一个RequestMethod类型的数组，表示该请求映射能够匹配多种请求方式的请求

若当前请求的请求地址满足请求映射的value属性，但是请求方式不满足method属性，则浏览器报错405

1.对于处理指定请求方式的控制器方法，SpringMVC中提供了@RequestMapping的派生注解

处理get请求的映射 —-> @GetMapping

处理post请求的映射 —-> @PostMapping

处理put请求的映射 —-> @PutMapping

处理delete请求的映射 —-> @DeleteMapping

2.常用的请求方式有get，post，put，delete

但是目前浏览器只支持get和post，若在form表单提交时，为method设置了其他请求方式的字符串（put或delete），则按照默认的请求方式get处理

若要发送put和delete请求，ze需要通过spring提供的过滤器HiddenHttpMethodFilter，在restful部分会讲到

3.@RequestMapping注解的params属性（了解）

@RequestMapping注解的params属性通过请求参数匹配请求映射

@RequesMapping注解的params属性是一个字符串类型数组，可以通过四种表达式设置请求参数和请求映射的匹配关系

“param”：要求请求映射所匹配的请求必须携带param请求参数
“！param”：要求请求映射所匹配的请求必须不能携带param请求参数
“param=value”：要求请求映射所匹配的请求必须携带param请求参数且param=value
“param！= value”：要求请求映射所匹配的请求必须携带param请求参数但是param！=value

4.@RequestMapping注解的headers属性（了解）
@RequestMapping注解的headers属性通过请求的请求头信息匹配请求映射
@RequestMapping注解的headers属性是一个字符串类型的数组，可以通过四种表达方式设置请求头信息和请求映射的匹配关系
“header”：要求请求映射所匹配的请求必须携带header请求头信息
“！header”：要求请求映射所匹配的请求必须不携带header请求头信息
“header=value”：要求请求映射所匹配的请求必须携带header请求头且header！=value
若当前请求满足@RequestMapping注解的value和method属性，但是不满足headers属性，此时页面显示404错误，即资源未找到

@Componet 普通组件
@Service 业务型组件
@Repository 持久型组件

6.解决浏览器解析的绝对路径
通过thymeleaf
<a th:href=“@{/target}”

总结
浏览器发送请求，若请求地址符合前端控制器的url-pattern，该请求就会被前端控制器DispatcherServlet处理。
前端控制器会读取SpringMVc的核心配置文件，通过扫描组件找到控制器，将请求地址和控制器中@Request Mapping注解的value属性值进行匹配，若匹配成功，该注解所标识的控制器方法就是处理请求的方法。处理请求的方法需要返回一个字符串类型的视图名称，该试图名称会被试图解析器解析，加上前缀和后缀组成视图的路径，通过Thymeleaf对视图进行渲染，最终转发到视图所对应页面

7.SpringMVC支持ant风格的路径
？：表示任意的单个字符
*：表示任意的0个或多个字符
**：表示任意的一层或多层目录
注意：在使用**时，只能使用/**/xxx的方式。 模糊匹配

8.SpringMVC支持路径中的占位符（重点）
原始的方式：/deleteUser？id = 1
user/SpringMVC/to/controller
rest方式：/deleteUser/1
SpringMVC路径中的占位符常用于restful风格中，当请求路径中将某些数据通过路径的方式传输到服务器中，就可以在相应的@RequestMapping注解的value属性中通过占位符{xxx}表示传输的数据，在通过@PathVariable注解，将占位符所表示的数据赋值给控制器的形参

四，SpringMVC获取请求参数
1.通过servletAPI获取
将HttpServletRequest作为控制器方法的形参，此时HttpServletRequest类型的参数表示封装了当前请求的请求报文的对象

2.通过控制器方法的形参获取请求参数
在控制器方法的形参位置，设置和请求参数同名的形参，当浏览器发送请求，匹配到请求映射时，在DispatcherServlet中就会将请求参数赋值给相应的形参

注：
若请求所传输的请求参数中有多个同名的请求参数，此时可以在控制器方法的形参中设置字符串数组或者字符串类型的形参接收此请求参数
若使用字符串数组类型的形参，此参数的数组中包含了每一个数据
若使用字符串类型的形参，此参数的值为每个数据中间使用逗号拼接的结果

3.@Requestparam
@RequestParam是将请求参数和控制器方法的形参创建映射关系

@RequestParam注解一共有三个属性：
value：指定为形参赋值的请求参数的参数名
required：设置是否必须传输此请求参数，默认值为true
若设置为true时，则当前请求必须传输value所指定的请求参数，所没有传输该请求参数，且没有设置defaultValue属性，则页面报错400：Required String parameter "xxx" is not present; 若设置为false，则当前请求不是必须传输value所指定的请求参数，若没有传输，则注解所标识的形参的值为null
defaultValue：不管required属性值为true或false，当value所指定的请求参数没有传输时，则使用默认值为形参赋值

4.@RequestHeader
@RequestHeader是将请求头信息和控制器方法的形参创建映射关系
@RequestHeader注解一共有三个属性：value，required，defaultValue，用法同@RequestParam

5.@CookieValue
@CookieValue是将cookie数据和控制器方法的形参创建映射关系
@CookieValue注解一共有三个属性：value，required，defaultValue，用法同@RequestParam

6.通过POJO获取请求参数
可以在控制器方法的形参位置设置一个实体类类型的形参，此时若浏览器传输的请求参数名和实体类中的属性名一致，那么请求参数就会为此属性赋值

在tomcat里面最早初始化的是监听器 ——》 filter过滤器 ——》 servlet

7.解决获取请求参数的乱码问题
解决获取请求参数的乱码问题，可以使用springMVC提供的编码过滤器CharacterEncodingFilter，但是必须在web.xml中进行注册

五、域对象共享数据

搭建SpringMVC框架
web.xml
1、配置编码过滤器 <filter>
2、配置springMVC前端控制器DispatcherServlet <servlet>

springMVC.xml
1、扫描组件 <context>
2、配置视图解析器 <bean>

1、使用servletAPI向request域对象共享数据
2、使用ModelAndView向request域对象共享数据
ModelAndView有Model和View的功能
Model主要用于向请求域共享数据
View主要用于设置视图，实现页面跳转
public ModelAndView testModelAndView(){
ModelAndView mav = new ModelAndView();
//处理模型数据，即向请求域request共享数据
mav.addObject(attributename,attributeviewValue);
//设置视图名称
mav.setViewName('success');
return mav;
}

3、使用Model向request域对象共享数据
public String testModel(Model model){
model.addAttribute();
return 'success'
}

4、使用map向request域对象共享数据
public String testMap(Map<String, Object> map){
map.put('testScope', 'hello,map');
return 'success';
}

5、使用ModelMap向request域对象共享数据

```
public String testModelMap(ModelMap modelMap){
    modelMap.addAttribute('testScope', 'Hello, ModelMap');
    return 'success'
}
```

6、Model, ModelMap, Map的关系

Model，ModelMap，Map类型的参数其实本质上都是BindingAwareModelMap类型的

```
public interface Model{ }
public class ModelMap extends LinkedHashMap<String, Object>{ }
public class ExtendedModelMap extends ModelMap implement Model{ }
public class BindingAwareModelMap extends ExtendedModelMap { }
```

7、向session域共享数据

```
@RequestMapping("/testSession")
public String testSession(HttpSession session){
    session.setAttribute("testSessionScope"，"hello,session");
    return "success";
```

8、向application域共享数据

```
@RequestMapping("testApplication")
public String testApplication(HttpSession session){
    ServletContext application = session.getServletContext();
    application.setAttribute("testApplicationScope"，"hello,application");
    return "success"
}
```

六、SpringMVC的视图

SpringMVC中的视图是View接口，视图的作用渲染数据，将模型Model中的数据展示给用户

SpringMVC视图的种类很多，默认有转发视图InternalResourceView和重定向视图RedirectView

当工程引入jstl的依赖，转发视图会自动转换为JstlView

若使用的视图技术为Thymeleaf，在SpringMVC的配置文件中配置了Thymeleaf的视图解析器，由此视图解析器解析之后所得到的是ThyleafView

1、ThymeleafView

当控制器方法中所设置的视图名称没有任何前缀时，此时的视图名称会被SpringMVC配置文件中所配置的视图解析器解析，视图名称拼接视图前缀和视图后缀所得到的最终路径，会通过转发的方式实现跳转

2、转发视图

SpringMVC中默认转发视图是InternalResourceView

SpringMVC中创建转发视图的情况：

当控制器方法中所设置的视图名称以“forward：”为前缀时，创建InternalResourceView视图，此时的视图名称不会被SpringMVC配置文件中所配置的视图解析器解析，而是会将前缀“forward：”去掉，剩余部分作为最终路径通过转发的方式实现跳转

3、重定向视图

SpringMVC中默认的重定向视图是RedirectView

当控制器方法中所设置的视图名称以“redirect：”为前缀，创建RedirectView视图，此时的视图名称不会被SpringMVC配置文件中所配置的试图解析器解析，而是会将前缀“redirect：”去掉，剩余部分作为最终路径通过重定向的方式实现跳转

例如“redirect：/”，“redirect：/employee

4、视图控制器view-controller

当控制器方法中，仅仅用来实现页面跳转，即只需设置视图名称时，可以讲处理器方法使用view-controller标签进行表示

```
<!——
    Path:设置处理的地址
    view-name：设置请求地址所对应的视图名称
——>
<mvc:view-controller path=“/testview” view-name=“success”></mvc:view-controller>
```

注：

当SpringMVC中设置任何一个view-controller时，其他控制器中的请求映射将全部失效，此时需要在SpringMVC的核心配置文件中设置开启MVC注解驱动的标签：

```
<mvc:annotation-driven/>
```

七、RestFul简介

1、REST：Representation State Transfer，表现层资源状态转移。

a>资源

资源是一种看待服务器的方式，即将服务器看作是由很多离散的资源组成。每个资源时服务器上一个可命名的抽象概念。因为资源是一个抽象的概念，所以它不仅仅能代表服务器文件系统中的一个文件、数据库中的一张表等等具体的东西，可以将资源设计的要多抽象有多抽象，只要想象力允许而且客户端应用开发者能够理解。与面向对象设计类似，资源是以名词为核心来组织的，首先关注的是名词。一个资源可以由一个或多个URI来标识。URI即是资源的名称，也是资源在web上的地址。对某个资源感兴趣的客户端应用，可以通过资源的URI与其进行交互。

b>资源的表述

资源的表述是一段对于资源在某个特定时刻的状态的描述。可以在客户端-服务器之间转移（交换）。资源的表述可以有多种格式，例如HTML/XML/JSON？纯文本/图片/视频/音频等等。资源的表述格式可以通过协商机制来确定。请求-响应方向的表述通常使用不同的格式。

c>状态转移

状态转移说的是：在客户端和服务器之间转移（transfer）代表资源状态的表述。通过转移和操作资源的表述，来间接实现操作资源的目的。

2、RESTFul的实现

具体说，就是HTTP协议里面，四个表示操作方式的动词：GET、POST、PUT、DELETE。
它们分别对应四种基本操作：GET用来获取资源，POST用来新建资源，PUT用来更新资源，DELETE用来删除资源。
REST风格提倡URL地址使用统一的风格设计，从前到后各个单词使用斜杠分开，不使用问好键值对方式携带请求参数，而是将要发送给服务器的数据作为URL地址的一部分，以保证整体风格的一致性。

操作	传统方式	REST风格
查询操作	getUserById?id=1	user/1—>get请求方式
保存操作	SaveUser	user—>post请求方式
删除操作	deleteUser?id=1	user/1–>delete请求方式
更新操作	updateUser	user—>put请求方式

3、HiddenHttpMethod Filter

由于浏览器只支持发送get和post方式的请求，那么该如何发送put和delete请求呢?
SpringMVC提供了HiddenHttpMethodFilter帮助我们将post请求转换为delete或put请求
HiddenHttpMethodFilter处理put和delete请求的条件：

请求方式为post
必须传送一个请求参数
<input type=“Hidden” name=“_method” value=“PUT”>

八、HttpMessageConverter

HttpMessageConverter，报文信息转换器，将请求报文转换为Java对象，或将Java对象转换为响应报文
HttpMessageConverter提供了两个注解和两个类型：@RequestBody，@ResponseBody，@RequestEntity，@ResponseEntity

1、@RequestBody

@RequestBody可以获取请求体，需要在控制器方法设置一个形参，使用@RequestBody进行标识，当前请求的请求体就会为当前注解所标识的形参赋值

2、@RequestEntity

RequestEntity封装请求报文的一种类型，需要在控制器方法的形参中设置改类型的形参，当前请求的请求报文就会赋值给改形参，可以通过getHeader()获取请求头信息，通过getBody()获取请求体信息

3、ResponseBody

@ResponseBody用于标识一个控制器方法，可以将该方法的返回值直接作为响应报文的响应体响应到浏览器

4、SpringMVC处理json

@RequestBody处理json的步骤：
a>导入jackson的依赖

```
<dependency>
    <groupId>com.fasterxml，jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.12.1</version>
</dependency>
```

b>在SpringMVC的核心配置文件中开启MVC的注解驱动，此时在HandlerAdaptor中会自动装配一个消息转换器，MappingJackson2HttpMessageConverter，可以将响应到浏览器的Java对象转换为Json格式的字符串
<mvc:annotation-drive/>

c>在处理器方法上使用@ResponseBody注解进行标识

d>将Java对象直接作为控制器方法的返回值返回，就会自动转换为Json格式的字符串

5、SpringMVC处理ajax

a>请求超链接

```
<div id="app">
    <a th:href="@{/testAjax}" @click="testAjax">testAjax</a><br/>
</did>
```

b>通过vue和axios处理点击事件

```
<script type="text/javascript" th:src="@{/static/js/vue.js}"></script>
<script type="text/javascript" th:sec="@{/static/js/axios.min.js}"></script>
<script type="text/javascript">
    var vue = new Vue({
    })
</script>
```

c>控制器方法：

```
@RequestMapping("/testAjax")
@ResponseBody
public String testAjax（String username， String password）{
    System.out.println("username:"+username+" ,password:"+password);
    return "hello,ajax";
}
```

6、@RestController注解是springMVC提供的一个复合注解，标识在控制器的类上，就相当于为类添加了Controller注解，并且为其中的每个方法添加了@ResponseBody注解

7、ResponseEntity

ResponseEntity用于控制器方法的返回值类型，该控制器方法的返回值就是响应到浏览器的响应报文

九、文件上传和下载

1、文件下载

使用ResponseEntity实现下载文件的功能

2、文件上传

文件上传要求form表单的请求方式必须为post，并且添加属性enctype="multipart/form-data"

SpringMVC中将上传的文件封装到MultipartFile对象中，通过此对象可以获取文件相关信息

上传步骤：

a>添加依赖：

b>在SpringMVC配置文件中添加配置：

十、拦截器

1、拦截器的配置

SpringMVC中的拦截器用于拦截控制器方法的执行

SpringMVC中的拦截器需要实现HandlerInterceptor或者继承HandlerInterceptorAdapter类

SpringMVC的拦截器必须在SpringMVC的配置文件进行配置

2、拦截器的单个抽象方法

SpringMVC中的拦截器有三个抽象方法；

preHandle：控制器方法执行之前执行preHandle()，其boolean类型的返回值表示是否拦截或放行，返回true为放行，即调用控制器方法；返回false表示拦截，即不调用控制器方法

postHandle：控制器方法执行之后执行postHandle()

afterComplation：处理完视图和模型数据，渲染视图完毕之后执行afterComplation()

3、多个拦截器的执行顺序

a>若每个拦截器的preHandle()都返回true

此时多个拦截器的执行顺序和拦截器在SpringMVC的配置文件的配置顺序有关：

preHandle()会按照配置的顺序执行，而postHandle()和afterComplation()会照配置的反顺序执行

b>若某个拦截器的preHandle()返回了false

preHandle()返回false和它之前的拦截器的preHandle()都会执行，postHandle()都不执行，返回false的拦截器之前的拦截器的afterComplation()会执行

十一、异常处理

1、基于配置的异常处理

SpringMVC提供了一个处理控制器方法执行过程中所出现的异常的接口：HandlerExceptionResolver
HandlerExceptionResolver接口的实现类有：DefaultHandlerExceptionResolver和SimpleMappingExceptionResolver
SpringMVC提供了自定义的异常处理器SimpleMappingExceptionResolver，使用方式：
。。。。

2、基于注解的异常处理

十二、注解配置SpringMVC
使用配置类和注解代替web.xml和SpringMVC配置文件的功能
1、创建初始类，代替web.xml
在Servlet3.0环境中，容器会在类路径中找实现javax.servlet.ServletContainerInitializer接口的类，如果找到的话就用它来配置Servlet容器。
Spring提供了这个接口的实现，名为SpringServletContainerInitializer，这个类反过来又会查找实现WebApplicationInitializer的类并将配置的任务交给它们来完成。Spring3.2引入了一个便利的WebApplicationInitializer基础实现，名为AbstractAnnotationConfigDispatcherServletInitializer，当我们的类扩展了AbstractAnnotationConfigDispatcherServletInnitializer并将其部署到servlet3.0容器的时候，容器会自动发现它，并用它来配置Servlet上下文
WebInit 代替web.xml
getRootConfigClasses 指定spring的配置类
getServletConfigClasses 指定springMVC的配置类
getServletMappings 指定DispatcherServlet的映射规则，即url-pattern
getServletFilter 注册过滤器

WebConfig 代替SpringMVC
1、扫描组件
2、视图解析器
3、view-controller
4、default-servlet-handler
5、mvc注解驱动
6、文件上传解析器
7、异常处理
8、拦截器

//将当前类标识为一个配置类
@Configuration
扫描组件
@ComponentScan(“com.xxx.xxx.controller”)
mvc注解驱动
@EnableWebMvc
public class WebConfig implements WebMvcConfigurer{
default-servlet-handler
@Override
public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer){
}
拦截器
@Override
public void addInterceptors(InterceptorRegistry registry){

}
view-controller
@Override
public void addViewControllers(ViewControllerRegistry registry){

}

文件上传解析器
@Bean
public MultipartResolver multipartResolver(){

}

@Override

```
public void configureHandlerExceptionResolvers(List<HandlerExceptionResolver> resolvers){

}


配置生成模版解析器
@Bean
public ITemplateResolver template Resolver(){
    return;
}
生成模版引擎并为模版引擎注入模版解析器
@Bean
public SpringTemplateEngine templateEngine(ITemplateResolver templateResolver){
    return;
}
生成视图解析器并为解析器注入模版引擎
@Bean
public ViewResolver viewResolver(SpringTemplateEngine templateEngine){
    return;
}
```

}

十三、SpringMVC执行流程

1、SpringMVC常用组件

DispatcherServlet：前端控制器，不需要工程师开发，由框架提供

作用：统一处理请求和响应，整个流程控制的中心，由它调用其他组件处理用户的请求

HandlerMapping：处理器映射器，不需要工程师开发，由框架提供

作用：根据请求的url、method等信息查找Handler，即控制器方法

Handler：处理器，需要工程师开发

作用：在DispatcherServlet的控制下Handler对具体的用户请求进行处理

HandlerAdapter：处理器适配器，不需要工程师开发，由框架提供

作用：通过HandlerAdapter对处理器（控制器方法）进行执行

ViewResolver：视图解析器，不需要工程师开发，由框架提供

作用：进行视图解析，得到相应的视图，例如：ThymeleafView，IntenalResourceView，RedirectView

View：视图，不需要工程师开发，由框架或视图技术提供

作用：将模型数据通过页面展示给用户

2、DispatcherServlet初始化过程

DispatcherServlet本质上是一个Servlet，所以天然地遵循Servlet的生命周期。所以宏观上是Servlet生命周期来进行调度。

a> 初始化WebApplicationContext

所在类：org.springframework.web.servlet.FrameworkServlet

b>创建WebApplicationContext

所在类：org.springframework.web.servlet.FrameworkServlet

c>DispatcherServlet初始化策略

FrameworkServlet创建WebApplicationContext后，刷新容器，调用onRefresh（wac），此方法在DispatcherServlet中进行了重写，调用了initStrategies（context）方法，初始化策略，即初始化DispatcherServlet的各个组件

所在类:org.springframework.web.servlet.DispatcherServlet

3、DispatcherServlet调用组件处理请求

a>processRequest()

FrameworkServlet重写HttpServlet中的service()和doXxx()，这些方法中调用了processRequest(request，response)

所在类：org.springframework.web.servlet.FrameworkServlet

b>doService()

所在类：org.springframework.web.servlet.DispatcherServlet

c>doDispatch()

所在类：org.springframework.web.servlet.DispatcherServlet

d>processDispatchResult()

4、SpringMVC的执行流程

1）用户向服务器发送请求，请求被SpringMVC前端控制器DispatcherServlet捕获。

2）DispatcherServlet对请求URL进行解析，得到请求资源标示符（URI），判断请求URI对应的映射：

a）不存在

i.再判断是否配置了mvc：default-servlet-handler

ii.如果没有配置，则控制台报映射查询不到，客户端展示404错误

iii.如果有配置，则访问目标资源（一般为静态资源，如：js，css，html），找不到客户端也会展示404错误

b>存在则执行下面流程

3）根据该URI，调用HandlerMApping获得该Handler配置的所有相关的对象（包括Handler对象以及Handler对象对应的拦截器），最后以HandlerExecutionChain执行链对象的形式返回。

4）DispatcherServlet根据获得的Handler，选择一个合适的HandlerAdapter。

5）如果成功获得HandlerAdapter，此时将开始执行拦截器的preHandler(…)方法【正向】

6）提取Request中的模型数据，填充Handler入参，开始执行Handler（Controller）方法，处理请求。在填充Handler的入参过程中，根据你的配置，Spring将帮你做一些额外的工作：

a）HttpMessageConveter：将请求信息（如Json、xml等数据）转换成一个对象，将对象转换为指定的响应信息

b）数据转换：对请求消息进行数据转换。如String转换成Integer、Double等

c）数据格式化：对请求消息进行数据格式化。如将字符串转换成格式化数字或格式化日期等

d）数据验证：验证数据的有效性（长度、格式等），验证结果存储到BindingResult或Error中

7）Handler执行完成后，向DispatcherServlet返回一个ModelAndView对象。

8）此时将开始执行拦截器的postHandle（…）方法【逆向】。

9）根据返回的ModelAndView（此时会判断是否存在异常：如果存在异常，则执行HandlerExceptionResolver进行异常处理）选择一个合适的ViewResolver进行视图解析，根据Model和View，来渲染视图。

10）渲染视图完毕执行拦截器的afterCompletion(…)方法【逆向】。

11）将渲染结果返回给客户端。