SpringMVC —请求源码流程

有道云链接：http://note.youdao.com/noteshare?id=ec3ca523300ad31d7f6f673b9e92bbeb&sub=1D1BF1D55D0148879F53878CB24F8214

前言

Spring官网的MVC模块介绍：

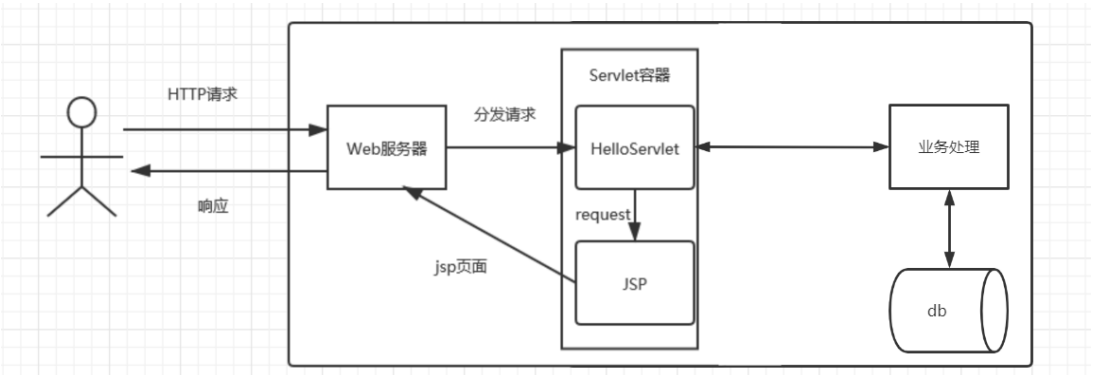
> Spring Web MVC is the original web framework built on the Servlet API and has been included in the Spring Framework from the very beginning. The formal name, “Spring Web MVC,” comes from the name of its source module (spring-webmvc), but it is more commonly known as “Spring MVC”.

Spring Web MVC是基于Servlet API构建的原始Web框架，从一开始就已包含在Spring框架中。正式名称“ Spring Web MVC”来自其源模块的名称（spring-webmvc），但它通常被称为“ Spring MVC”。

从Servlet到SpringMVC

最典型的MVC就是JSP + servlet + javabean的模式。

传统Servlet：



弊端：

1.xml下配置servlet的映射非常麻烦 **开发效率低**

2.必须要继承父类、重写方法 **侵入性强**

2.如果想在一个Servlet中处理同一业务模块的的功能**分发给不同方法进行处理非常麻烦**

3.**参数解析麻烦**:单个参数（转换类型）--->pojo对象 Json文本--->pojo对象

4.**数据响应麻烦:**pojo对象--->json ... Content-type

5.跳转页面麻烦， 对path的控制、 如果使用其他模板也很麻烦 、设置编码麻烦...等等...

> 所以SpringMVC 就是在Servlet的基础上进行了封装，帮我把这些麻烦事都给我们做了。

> Web框架的升级是一个不断偷懒的过程
> 从最开始的Servlet到现在的SpringMVC、SpringBoot等等

SpringMVC

基于xml的实现方式:

1.给Servlet容器配置一个DispatcherServlet（web.xml ）

2.添加SpringMVC的配置信息

- 继承类/实现接口 方式:

```
implements HttpRequestHandler
implements Controller
```

不同的*HandlerMapping*

```
<!--通过设置属性的方式去设置映射路径-->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
 <props>
 <prop key="httpRequest">simpleController</prop>
 </props>
</property>

<!--BeanNameUrlHandlerMapping 一定要为Controller设置一个有效映射地址的名字 如 @Controller("/xxxx")-->
 <bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
```

- 注解方式:

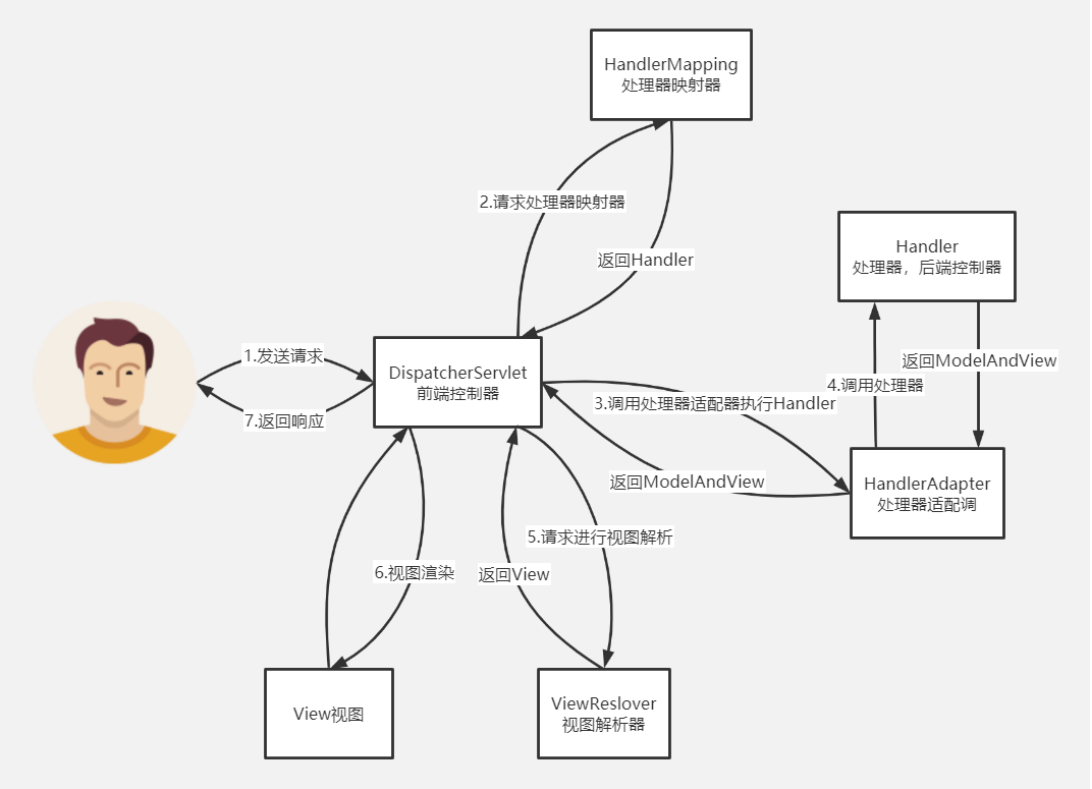
配置控制器@Controller和处理方法的映射—@RequstMapping 即可

> 其实SpringMVC请求原理很简单：说白了就是用一个DispatcherServlet 封装了一个Servlet的调度中心， 由调度中心帮我们调用我们的处理方法：
> 在这个过程中调度中心委托给各个组件执行具体工作 ， 比如帮我们映射方法请求、帮我解析参数、调用处理方法、响应数据和页面 等

> 这就相当于你在家自己做饭和去饭店吃饭的区别了， 在家你买菜、洗菜、蒸饭、炒菜、洗碗都得自己来.
> 饭店都给你做好了， 你只要分服务员说你吃什么、就能得到响应. 殊不知呢， 你只是说了吃什么（请求）， 后厨（DispatcherServlet）就有配菜员你给找到菜单-对应的食材（映射） 、切菜员切菜（解析参数）、 厨师给你炒菜（调用处理方法）、装盘（处理返回值)、 抄完给你端出来（响应）

SpringMVC的具体执行流程：

> Spring MVC 是围绕前端控制器模式设计的，其中：中央 Servlet DispatcherServlet 为请求处理流程提供统一调度，实际工作则交给可配置组件执行。这个模型是灵活的且开放的，我们可以通过自己去定制这些组件从而进行定制自己的工作流。



DispatcherServlet： 前端调度器， 负责将请求拦截下来分发到各控制器方法中
HandlerMapping: 负责根据请求的URL和配置@RequestMapping映射去匹配， 匹配到会返回Handler（具体控制器的方法）
HandlerAdaper: 负责调用Handler-具体的方法- 返回视图的名字 Handler将它封装到ModelAndView(封装视图名，request域的数据)
ViewReslover: 根据ModelAndView里面的视图名地址去找到具体的jsp封装在View对象中
View：进行视图渲染（将jsp转换成html内容 --这是Servlet容器的事情了） 最终response到的客户端

1. 用户发送请求至前端控制器DispatcherServlet

2. DispatcherServlet收到请求调用处理器映射器HandlerMapping。
    a. 处理器映射器根据请求url找到具体的处理器，生成处理器执行链HandlerExecutionChain(包括处理器对象和处理器拦截器)一并返回给DispatcherServlet。
3. DispatcherServlet根据处理器Handler获取处理器适配器HandlerAdapter,执行HandlerAdapter处理一系列的操作，如：参数封装，数据格式转换，数据验证等操作
4. 执行处理器Handler(Controller，也叫页面控制器)。
    a. Handler执行完成返回ModelAndView
    b. HandlerAdapter将Handler执行结果ModelAndView返回到DispatcherServlet
5. DispatcherServlet将ModelAndView传给ViewReslover视图解析器
    a. ViewReslover解析后返回具体View
6. DispatcherServlet对View进行渲染视图（即将模型数据model填充至视图中）。
7. DispatcherServlet响应用户。

整个调用过程其实都在doDispatch中体现了：

1. 用户发送请求至前端控制器DispatcherServlet
    - 由于它是个Servlet会先进入service方法——>doGet/doPost——>processRequestdoService——>doDispatch ↓
    - 这个doDispatch非常重要--体现了整个请求流程

```
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {

  try {

  try {
  // 文件上传相关
  processedRequest = checkMultipart(request);
```

```
 multipartRequestParsed = (processedRequest != request);

 // DispatcherServlet收到请求调用处理器映射器HandlerMapping。
 // 处理器映射器根据请求url找到具体的处理器，生成处理器执行链HandlerExecutionChain
(包括处理器对象和处理器拦截器)一并返回给DispatcherServlet。
 mappedHandler = getHandler(processedRequest);
 if (mappedHandler == null) {
 noHandlerFound(processedRequest, response);
 return;
 }

 4.DispatcherServlet根据处理器Handler获取处理器适配器HandlerAdapter,
 HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());

 // Process last-modified header, if supported by the handler. HTTP缓存相关
 String method = request.getMethod();
 boolean isGet = HttpMethod.GET.matches(method);
 if (isGet || HttpMethod.HEAD.matches(method)) {
 long lastModified = ha.getLastModified(request, mappedHandler.getHandler());
 if (new ServletWebRequest(request, response).checkNotModified(lastModified) &
& isGet) {
 return;
 }
 }
 // 前置拦截器
 if (!mappedHandler.applyPreHandle(processedRequest, response)) {
 // 返回false就不进行后续处理了
 return;
 }

 // 执行HandlerAdapter处理一系列的操作，如：参数封装，数据格式转换，数据验证等操作
 // 执行处理器Handler(Controller，也叫页面控制器)。
 // Handler执行完成返回ModelAndView
 // HandlerAdapter将Handler执行结果ModelAndView返回到DispatcherServlet
 mv = ha.handle(processedRequest, response, mappedHandler.getHandler());

 if (asyncManager.isConcurrentHandlingStarted()) {
 return;
 }
 // 如果没有视图，给你设置默认视图 json忽略
 applyDefaultViewName(processedRequest, mv);
 //后置拦截器
 mappedHandler.applyPostHandle(processedRequest, response, mv);
 }
 catch (Exception ex) {
```

```
51  dispatchException = ex;
52  }
53  catch (Throwable err) {
54  // As of 4.3, we're processing Errors thrown from handler methods as well,
55  // making them available for @ExceptionHandler methods and other scenarios.
56  dispatchException = new NestedServletException("Handler dispatch failed", er
r);
57  }
58  // DispatcherServlet将ModelAndView传给ViewReslover视图解析器
59  // ViewReslover解析后返回具体View
60  // DispatcherServlet对View进行渲染视图（即将模型数据model填充至视图中）。
61  // DispatcherServlet响应用户。
62  processDispatchResult(processedRequest, response, mappedHandler, mv, dispatch
Exception);
63  }
64  catch (Exception ex) {
65  triggerAfterCompletion(processedRequest, response, mappedHandler, ex);
66  }
67  catch (Throwable err) {
68  triggerAfterCompletion(processedRequest, response, mappedHandler,
69  new NestedServletException("Handler processing failed", err));
70  }
71  finally {
72  if (asyncManager.isConcurrentHandlingStarted()) {
73  // Instead of postHandle and afterCompletion
74  if (mappedHandler != null) {
75  mappedHandler.applyAfterConcurrentHandlingStarted(processedRequest, respons
e);
76  }
77  }
78  else {
79  // Clean up any resources used by a multipart request.
80  if (multipartRequestParsed) {
81  cleanupMultipart(processedRequest);
82  }
83  }
84  }
85 }
```

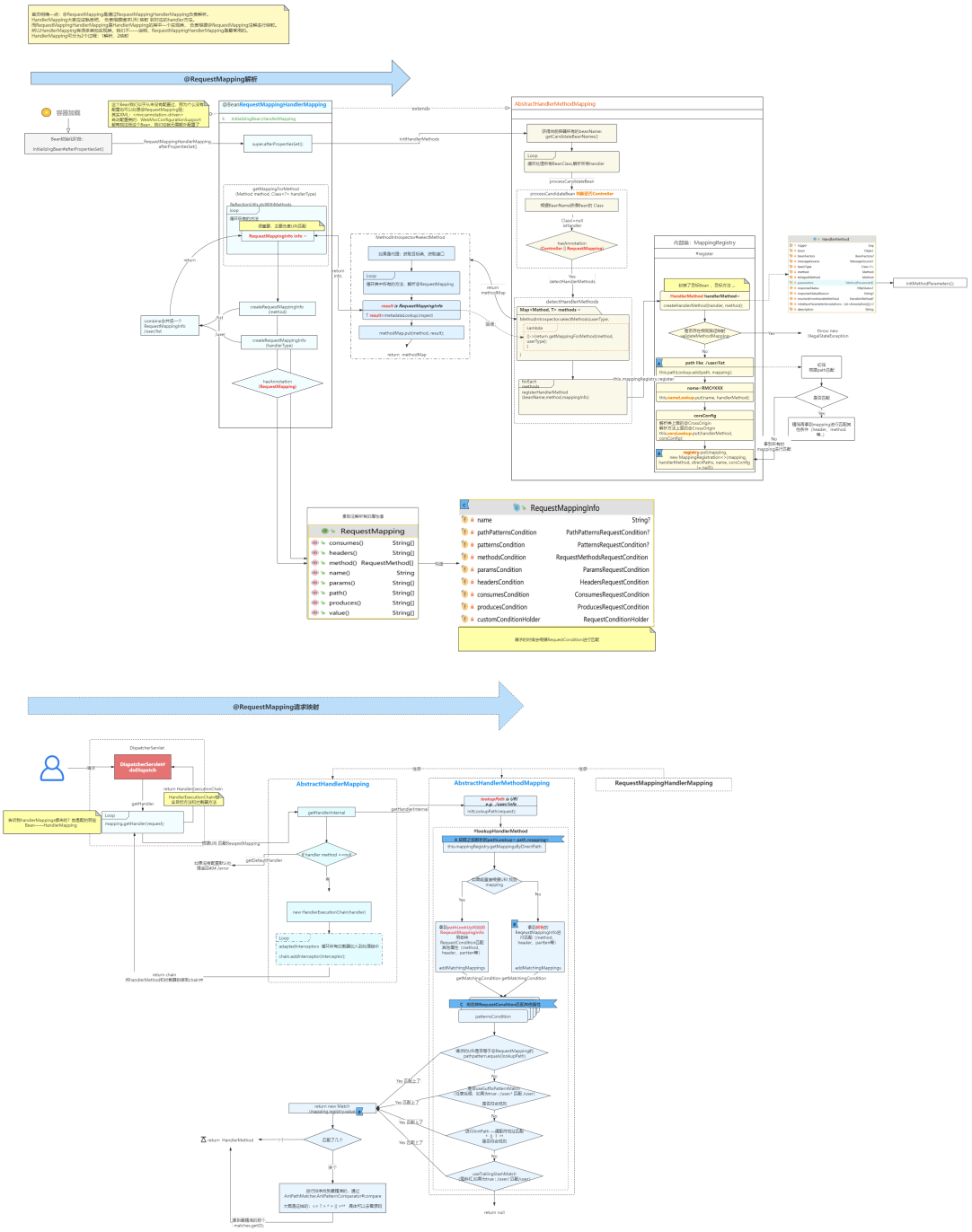
详细过程我们课程中分析....

HandlerMapping

在整个过程中，涉及到非常多的组件，每个组件解析各个环节，其中**HandlerMapping最为重要它是用来映射请求的**，我们就着重介绍下HandlerMapping的解析过程和请求映射过程：

附上流程图：

https://www.processon.com/view/link/615ea79e1efad4070b2d6707



SpringMVC —请求源码流程

SpringMVC —请求源码流程

前言

Spring官网的MVC模块介绍:

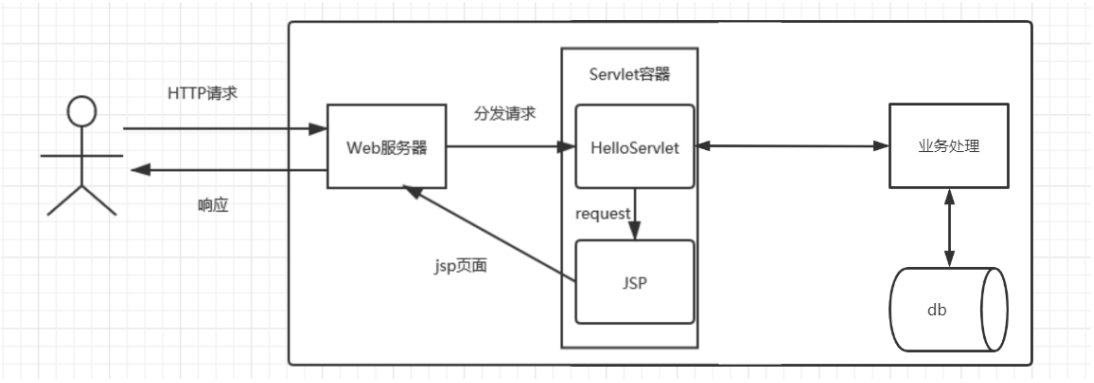
> Spring Web MVC is the original web framework built on the Servlet API and has been included in the Spring Framework from the very beginning. The formal name, “Spring Web MVC,” comes from the name of its source module (spring-webmvc), but it is more commonly known as “Spring MVC”.

Spring Web MVC是基于Servlet API构建的原始Web框架，从一开始就已包含在Spring框架中。正式名称“ Spring Web MVC”来自其源模块的名称（spring-webmvc），但它通常被称为“ Spring MVC”。

从Servlet到SpringMVC

最典型的MVC就是JSP + servlet + javabean的模式。

传统Servlet:



弊端:

1.xml下配置servlet的映射非常麻烦 **开发效率低**

2.必须要继承父类、重写方法 **侵入性强**

2.如果想在一个Servlet中处理同一业务模块的的功能**分发给不同方法进行处理非常麻烦**

3.**参数解析麻烦**:单个参数（转换类型）--->pojo对象 Json文本--->pojo对象

4.**数据响应麻烦:**pojo对象--->json ... Content-type

5.跳转页面麻烦， 对path的控制、 如果使用其他模板也很麻烦 、设置编码麻烦...等等...

> 所以SpringMVC 就是在Servlet的基础上进行了封装，帮我把这些麻烦事都给我们做了。

> Web框架的升级是一个不断偷懒的过程
> 从最开始的Servlet到现在的SpringMVC、SpringBoot等等

SpringMVC

基于xml的实现方式：

1.给Servlet容器配置一个DispatcherServlet（web.xml ）

2.添加SpringMVC的配置信息

- 继承类/实现接口 方式：

```
implements HttpRequestHandler
implements Controller
```

不同的*HandlerMapping*

```
<!--通过设置属性的方式去设置映射路径-->
<bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
<property name="mappings">
 <props>
 <prop key="httpRequest">simpleController</prop>
 </props>
</property>

<!--BeanNameUrlHandlerMapping 一定要为Controller设置一个有效映射地址的名字 如 @Controller("/xxxx")-->
 <bean class="org.springframework.web.servlet.handler.BeanNameUrlHandlerMapping"/>
```

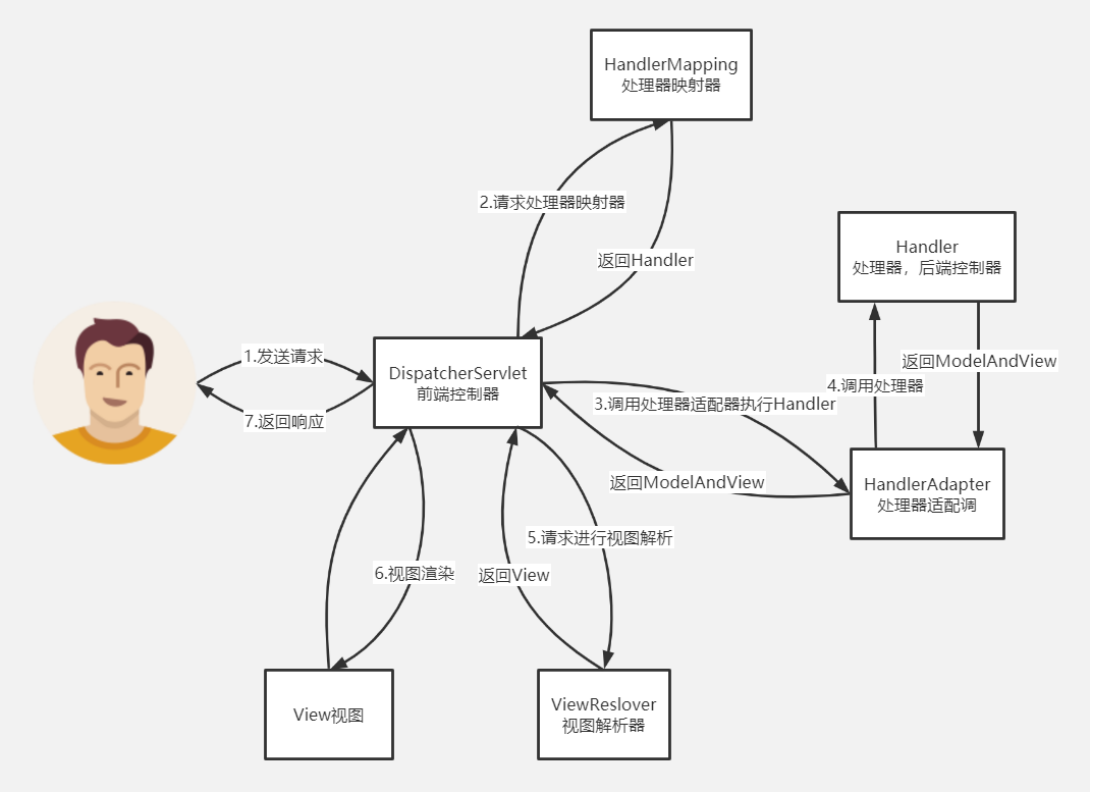
- 注解方式：

配置控制器@Controller和处理方法的映射—@RequstMapping 即可

> 其实SpringMVC请求原理很简单：说白了就是用一个DispatcherServlet 封装了一个Servlet的调度中心， 由调度中心帮我们调用我们的处理方法：
> 在这个过程中调度中心委托给各个组件执行具体工作 ， 比如帮我们映射方法请求、帮我解析参数、调用处理方法、响应数据和页面 等
>
> 这就相当于你在家自己做饭和去饭店吃饭的区别了， 在家你买菜、洗菜、蒸饭、炒菜、洗碗都得自己来.
> 饭店都给你做好了， 你只要分服务员说你吃什么、就能得到响应. 殊不知呢， 你只是说了吃什么（请求）， 后厨（DispatcherServlet）就有配菜员你给找到菜单-对应的食材（映射） 、切菜员切菜（解析参数）、 厨师给你炒菜（调用处理方法）、装盘（处理返回值)、 抄完给你端出来（响应）

SpringMVC的具体执行流程：

> Spring MVC 是围绕前端控制器模式设计的，其中：中央 Servlet DispatcherServlet 为请求处理流程提供统一调度，实际工作则交给可配置组件执行。这个模型是灵活的且开放的，我们可以通过自己去定制这些组件从而进行定制自己的工作流。



DispatcherServlet： 前端调度器， 负责将请求拦截下来分发到各控制器方法中

HandlerMapping: 负责根据请求的URL和配置@RequestMapping映射去匹配， 匹配到会返回Handler（具体控制器的方法）

HandlerAdaper: 负责调用Handler-具体的方法- 返回视图的名字 Handler将它封装到ModelAndView(封装视图名，request域的数据)

ViewReslover: 根据ModelAndView里面的视图名地址去找到具体的jsp封装在View对象中

View：进行视图渲染（将jsp转换成html内容 --这是Servlet容器的事情了） 最终response到的客户端

8. 用户发送请求至前端控制器DispatcherServlet
9. DispatcherServlet收到请求调用处理器映射器HandlerMapping。
    a. 处理器映射器根据请求url找到具体的处理器，生成处理器执行链HandlerExecutionChain(包括处理器对象和处理器拦截器)一并返回给DispatcherServlet。

10. DispatcherServlet根据处理器Handler获取处理器适配器HandlerAdapter,执行HandlerAdapter处理一系列的操作，如：参数封装，数据格式转换，数据验证等操作

11. 执行处理器Handler(Controller，也叫页面控制器)。

    a. Handler执行完成返回ModelAndView

    b. HandlerAdapter将Handler执行结果ModelAndView返回到DispatcherServlet

12. DispatcherServlet将ModelAndView传给ViewReslover视图解析器

    a. ViewReslover解析后返回具体View

13. DispatcherServlet对View进行渲染视图（即将模型数据model填充至视图中）。

14. DispatcherServlet响应用户。

整个调用过程其实都在doDispatch中体现了：

2. 用户发送请求至前端控制器DispatcherServlet

- 由于它是个Servlet会先进入service方法——>doGet/doPost——>processRequestdoService——>doDispatch ↓
- 这个doDispatch非常重要--体现了整个请求流程

```
protected void doDispatch(HttpServletRequest request, HttpServletResponse response) throws Exception {

  try {

  try {
  // 文件上传相关
  processedRequest = checkMultipart(request);
  multipartRequestParsed = (processedRequest != request);

   // DispatcherServlet收到请求调用处理器映射器HandlerMapping。
   // 处理器映射器根据请求url找到具体的处理器，生成处理器执行链HandlerExecutionChain(包括处理器对象和处理器拦截器)一并返回给DispatcherServlet。
   mappedHandler = getHandler(processedRequest);
```

```
13  if (mappedHandler == null) {
14  noHandlerFound(processedRequest, response);
15  return;
16  }
17
18  4.DispatcherServlet根据处理器Handler获取处理器适配器HandlerAdapter,
19  HandlerAdapter ha = getHandlerAdapter(mappedHandler.getHandler());
20
21  // Process last-modified header, if supported by the handler. HTTP缓存相关
22  String method = request.getMethod();
23  boolean isGet = HttpMethod.GET.matches(method);
24  if (isGet || HttpMethod.HEAD.matches(method)) {
25  long lastModified = ha.getLastModified(request, mappedHandler.getHandler());
26  if (new ServletWebRequest(request, response).checkNotModified(lastModified) &
& isGet) {
27  return;
28  }
29  }
30  // 前置拦截器
31  if (!mappedHandler.applyPreHandle(processedRequest, response)) {
32  // 返回false就不进行后续处理了
33  return;
34  }
35
36  // 执行HandlerAdapter处理一系列的操作，如：参数封装，数据格式转换，数据验证等操作
37  // 执行处理器Handler(Controller，也叫页面控制器)。
38  // Handler执行完成返回ModelAndView
39  // HandlerAdapter将Handler执行结果ModelAndView返回到DispatcherServlet
40  mv = ha.handle(processedRequest, response, mappedHandler.getHandler());
41
42  if (asyncManager.isConcurrentHandlingStarted()) {
43  return;
44  }
45  // 如果没有视图，给你设置默认视图 json忽略
46  applyDefaultViewName(processedRequest, mv);
47  //后置拦截器
48  mappedHandler.applyPostHandle(processedRequest, response, mv);
49  }
50  catch (Exception ex) {
51  dispatchException = ex;
52  }
53  catch (Throwable err) {
54  // As of 4.3, we're processing Errors thrown from handler methods as well,
55  // making them available for @ExceptionHandler methods and other scenarios.
```

```
56  dispatchException = new NestedServletException("Handler dispatch failed", er
r);
57  }
58  // DispatcherServlet将ModelAndView传给ViewReslover视图解析器
59  // ViewReslover解析后返回具体View
60  // DispatcherServlet对View进行渲染视图（即将模型数据model填充至视图中）。
61  // DispatcherServlet响应用户。
62  processDispatchResult(processedRequest, response, mappedHandler, mv, dispatch
Exception);
63  }
64  catch (Exception ex) {
65  triggerAfterCompletion(processedRequest, response, mappedHandler, ex);
66  }
67  catch (Throwable err) {
68  triggerAfterCompletion(processedRequest, response, mappedHandler,
69  new NestedServletException("Handler processing failed", err));
70  }
71  finally {
72  if (asyncManager.isConcurrentHandlingStarted()) {
73  // Instead of postHandle and afterCompletion
74  if (mappedHandler != null) {
75  mappedHandler.applyAfterConcurrentHandlingStarted(processedRequest, respons
e);
76  }
77  }
78  else {
79  // Clean up any resources used by a multipart request.
80  if (multipartRequestParsed) {
81  cleanupMultipart(processedRequest);
82  }
83  }
84  }
85 }
```

详细过程我们课程中分析....

HandlerMapping

在整个过程中，涉及到非常多的组件，每个组件解析各个环节，其中**HandlerMapping最为重要它是用来映射请求的**，我们就着重介绍下HandlerMapping的解析过程和请求映射过程：

附上流程图：

https://www.processon.com/view/link/615ea79e1efad4070b2d6707