2、SpringMVC-HelloWorld

笔记本: spring
创建时间: 2022/4/4 17:24
作者: 雷丰阳

1、流程
1）、导包

```
  commons-logging-1.1.3.jar
spring-aop-4.0.0.RELEASE.jar
spring-beans-4.0.0.RELEASE.jar
spring-context-4.0.0.RELEASE.jar
spring-core-4.0.0.RELEASE.jar
spring-expression-4.0.0.RELEASE.jar
spring-web-4.0.0.RELEASE.jar
spring-webmvc-4.0.0.RELEASE.jar
```

2）、写配置；
1）web.xml可能要写什么
配置springmvc的前端控制器，指定springmvc配置文件位置

```
<!-- The front controller of this Spring Web application, responsible for
        handling all application requests -->
    <servlet>
        <servlet-name>springDispatcherServlet</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>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <!-- Map all requests to the DispatcherServlet for handling -->
    <servlet-mapping>
        <servlet-name>springDispatcherServlet</servlet-name>
        <url-pattern>/</url-pattern>
    </servlet-mapping>
```

2）框架自身可能要写什么

```
<context:component-scan base-package="com.atguigu">
</context:component-scan>
    <!-- 视图解析器，可以简化方法的返回值，返回值就是作为目标页面地址，
        只不过视图解析器可以帮我们拼串
     -->
    <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
        <property name="prefix" value="/WEB-INF/pages/"></property>
        <property name="suffix" value=".jsp"></property>
    </bean>
```

3）、测试

HelloWorld细节：
一：运行流程；

```
 *      1）、客户端点击链接会发送 http://localhost:8080/springmvc/hello 请求
 *      2）、来到tomcat服务器；
 *      3）、SpringMVC的前端控制器收到所有请求；
 *      4）、来看请求地址和@RequestMapping标注的哪个匹配，来找到到底使用那个类的哪个方法来处理
 *      5）、前端控制器找到了目标处理器类和目标方法，直接利用返回执行目标方法；
 *      6）、方法执行完成以后会有一个返回值；SpringMVC认为这个返回值就是要去的页面地址
 *      7）、拿到方法返回值以后；用视图解析器进行拼串得到完整的页面地址；
 *      8）、拿到页面地址，前端控制器帮我们转发到页面；
```

一个方法处理一个请求；

```
*2、@RequestMapping；
*       就是告诉SpringMVC；这个方法用来处理什么请求；
*       这个/是可以省略，即使省略了，也是默认从当前项目下开始；
```

```
 *       习惯加上比较好     /hello  /hello
 *       RequestMapping的使用：？
 *3、如果不指定配置文件位置？
 *       /WEB-INF/springDispatcherServlet-servlet.xml
 *       如果不指定也会默认去找一个文件；
 *           /WEB-INF/springDispatcherServlet-servlet.xml
 *   就在web应用的/WEB-INF、下创建一个名叫    前端控制器名-servlet.xml
<!--
        /：拦截所有请求，不拦截jsp页面，*.jsp请求
        /*：拦截所有请求，拦截jsp页面，*.jsp请求
        处理*.jsp是tomcat做的事；所有项目的小web.xml都是继承于大web.xml
        DefaultServlet是Tomcat中处理静态资源的？
            除过jsp，和servlet外剩下的都是静态资源；
            index.html：静态资源，tomcat就会在服务器下找到这个资源并返回；
            我们前端控制器的/禁用了tomcat服务器中的DefaultServlet


        1）服务器的大web.xml中有一个DefaultServlet是url-pattern=/
        2）我们的配置中前端控制器 url-pattern=/
                静态资源会来到DispatcherServlet（前端控制器）看那个方法的RequestMapping是这个index.html
        3）为什么jsp又能访问；因为我们没有覆盖服务器中的JspServlet的配置
        4）  /*  直接就是拦截所有请求；我们写/；也是为了迎合后来Rest风格的URL地址
    -->
```

RequestMapping：

```
/**
     * RequestMapping的其他属性
     * method：限定请求方式、
     *      HTTP协议中的所有请求方式：
     *          【GET】, HEAD, 【POST】, PUT, PATCH, DELETE, OPTIONS, TRACE
     *      GET、POST
     *      method=RequestMethod.POST：只接受这种类型的请求，默认是什么都可以；
     *          不是规定的方式报错：4xx:都是客户端错误
     *              405 - Request method 'GET' not supported
     * params：规定请求参数
     * params 和 headers支持简单的表达式：
     *      param1: 表示请求必须包含名为 param1 的请求参数
     *          eg: params={"username"}:
     *              发送请求的时候必须带上一个名为username的参数；没带都会404
     *
     *      !param1: 表示请求不能包含名为 param1 的请求参数
     *          eg:params={"!username"}
     *              发送请求的时候必须不携带上一个名为username的参数；带了都会404
     *      param1 != value1: 表示请求包含名为 param1 的请求参数，但其值不能为 value1
     *          eg: params={"username!=123"}
     *              发送请求的时候;携带的username值必须不是123(不带username或者username不是123)
     *
     *      {“param1=value1”, “param2”}: 请求必须包含名为 param1 和param2 的两个请求参数，且 param1 参数的值必须为value1
     *          eg:params={"username!=123","pwd","!age"}
     *              请求参数必须满足以上规则；
     *              请求的username不能是123，必须有pwd的值，不能有age
     * headers：规定请求头；也和params一样能写简单的表达式
     *
     *
     *
     * consumes：只接受内容类型是哪种的请求，规定请求头中的Content-Type
     * produces：告诉浏览器返回的内容类型是什么，给响应头中加上Content-Type:text/html;charset=utf-8
     */
    @RequestMapping(value="/handle02",method=RequestMethod.POST)
    public String handle02(){
        System.out.println("handle02...");
        return "success";
    }
```

```
    /**
     *
     * @return
     */
    @RequestMapping(value="/handle03",params={"username!=123","pwd","!age"})
    public String handle03(){
        System.out.println("handle03....");
        return "success";
    }

    /**
     * User-Agent：浏览器信息；
     * 让火狐能访问，让谷歌不能访问
     *
     * 谷歌：
     * User-Agent:Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/58.0.3029.110 Safari/537.36
     * 火狐；
     * User-Agent   Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0) Gecko/20100101 Firefox/34.0
     * @return
     *
     */
    @RequestMapping(value="/handle04",headers={"User-Agent=Mozilla/5.0 (Windows NT 6.3; WOW64; rv:34.0)
Gecko/20100101 Firefox/34.0"})
    public String handle04(){
        System.out.println("handle04....");
        return "success";
    }
```

```
/**
 *  @RequestMapping模糊匹配功能
 *
 *@author lfy
 *
 * URL地址可以写模糊的通配符：
 *  ？：能替代任意一个字符
 *  *：能替代任意多个字符，和一层路径
 *  **：能替代多层路径
 *
 */
@Controller
public class RequestMappingTest {

    @RequestMapping("/antTest01")
    public String antTest01(){
        System.out.println("antTest01...");
        return "success";
    }

    /**
     * ?匹配一个字符,0个多个都不行;
     *       模糊和精确多个匹配情况下，精确优先
     *
     *@return
     */
    @RequestMapping("/antTest0?")
    public String antTest02(){
        System.out.println("antTest02...");
        return "success";
    }

    /**
     *    *匹配任意多个字符
     *@return
     */
    @RequestMapping("/antTest0*")
```

```
    public String antTest03(){
        System.out.println("antTest03...");
        return "success";
    }

    /**
     *  *：匹配一层路径
     * @return
     */
    @RequestMapping("/a/*/antTest01")
    public String antTest04(){
        System.out.println("antTest04...");
        return "success";
    }

    @RequestMapping("/a/**/antTest01")
    public String antTest05(){
        System.out.println("antTest05...");
        return "success";
    }

    //路径上可以有占位符：    占位符 语法就是可以在任意路径的地方写一个{变量名}
    //   /user/admin    /user/leifengyang
    // 路径上的占位符只能占一层路径
    @RequestMapping("/user/{id}")
    public String pathVariableTest(@PathVariable("id")String id){
        System.out.println("路径上的占位符的值"+id);
        return "success";
    }
```
