SpringMVC

1.概念：专门进行控制器优化的一个框架，提供了自动数据注入，简化了跳转，增加了携带数据的方式

2.优点
- 1.基于mvc架构，分工明确，解耦合
- 2.容易上手，使用简单
- 3.作为Spring框架的子框架，能够使用IOC和AOP
- 4.强化注解使用

3.SpringMVC框架的优化内容

1.数据提交并自动注入

1.单个数据提交并自动注入，保证提交请求中的参数名称与方法的参数名称一致即可
示例：
```
public String one(String uname,int uage){...}
<input name="uname">
```

2.对象封装提交,提交请求中的参数的名称与类中成员变量的名称一致即可
```
public String two(Users users){...}
<input name="uname">
```
Users对象 中的成员变量名称是 uname

3.动态占位符提交（仅限于超链接）
通过@PathVariable注解解析URL地址中的变量的值到对应的方法的参数中
```
@RequestMapping("/three/{myname}/{myage}")
public String three(
        @PathVariable(value = "myname")
        String uname,
        @PathVariable(value = "myage")
        int uage){
    System.out.println("姓名是："+uname+",年龄是："+(uage+100));
    return "main";
}
```
页面上
```
<a href="${pageContext.request.contextPath}/zar/three/dljd/22.action">动态占位符提交 </a>
```

4.请求参数名称与方法参数名称不一致
通过 @RequestParam(value = "name")注解解析请求中的变量名称，对应方法的参数
```
public String four(
        @RequestParam(value = "name")
        String uname,
        @RequestParam(value = "age")
        int uage){
    System.out.println("姓名是："+uname+",年龄是："+(uage+100));
    return "main";
}
```

5.使用HttpServletRequest对象手工处理
```
public String five(HttpServletRequest request){
    String name = request.getParameter("name");
    int age = Integer.parseInt(request.getParameter("age"));
    System.out.println("姓名是："+name+",年龄是："+(age+100));
    return "main";
}
```

2.解决中文乱码
```
<filter>
    <filter-name>encode</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>forceRequestEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
    <init-param>
      <param-name>forceResponseEncoding</param-name>
      <param-value>true</param-value>
    </init-param>
  </filter>
  <filter-mapping>
    <filter-name>encode</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

3.控制器方法的返回值

1.第一种：ModelAndView
返回数据和视图，但是在实际开发中，我们可能只会返回视图，或者只返回数据，所以这种方式返回数据和视图已经很少用了，因为比较繁琐。

2.**第二种：String （常用）**
返回指定视图的名称（ 页面或action ）,其前缀和后缀由视图解析器InternalResourceViewResolver处理。
```
<bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix" value="/admin/" ></property>
    <property name="suffix" value=".jsp" > </property>
</bean>
```
方法中只返回String
```
@RequestMapping("/one")
public String one(String uname,int uage){
    System.out.println("姓名是："+uname+",年龄是："+(uage+100));
    return "main";
}
```

3.第三种： void
服务器不需要返回任何内容，只是进行业务处理，一般用在ajax请求中。
```
public void show1(){...}
```

4.**第四种：Object（常用，用于返回JSON）**
**返回各种对象，主要用于ajax返回JSON数据。**

4.SpringMVC框架支持的方法的默认参数类型
1）HttpServletRequest
2）HttpServletResponse
3）HttpSession
4）Model
5）ModelMap
6）Map<String,Object>

5.SpringMVC的四种跳转方式

1.请求转发页面（默认的跳转方式）
```
return "main";
```
main---->页面的名称，不用写具体的路径，不用写后缀

2.请求转发action
通过关键字forward来取消视图解析器的前缀和后缀拼接。
```
return "forward:/other/show.action"; 指定目标资源的位置
```

3.重定向跳页面
过去：
```
response.sendRedirect(request.getContextPath()+"/admin/main.jsp");
```
现在：
```
return "redirect:/admin/main.jsp";
```
通过关键字redirect来取消视图解析器的前缀和后缀拼接。

4.重定向跳action
```
return "redirect:/other/show.action";
```
//因为我们注册SpringMVC框架时，指定以*.action的方式，只拦截后缀是action的请求

6日期处理

1.在方法的参数上使用注解@DateTimeFormat,切记必须在springmvc.xml的配置文件中添加标签
```
<mvc:annotation-driven/>
```
页页上：
```
<form action="${pageContext.request.contextPath}/mydateShow.action" method="post">
  日期：<input type="date" name="mydate"><br>
  <input type="submit" value="提交">
</form>
```
action中：
```
@RequestMapping("/showDate")
public String showDate(
        @DateTimeFormat(pattern = "yyyy-MM-dd")
        Date mydate){
    System.out.println(mydate);
    return "main";
}
```

2. 使用@InitBinder注解解决本类中全部的日期处理(掌握)
```
@InitBinder
public void initBinder(WebDataBinder dataBinder){
    SimpleDateFormat sf = new SimpleDateFormat("yyyy-MM-dd");
    dataBinder.registerCustomEditor(Date.class,new CustomDateEditor(sf,true));
}
```

7.使用JSTL

1.在pom.xml文件中添加jstl的依赖
```
<dependency>
    <groupId>jstl</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```

2. 要在当前显示日期的页面上导入jstl的标签库
```
<%@taglib prefix="fmt" uri="http://java.sun.com/jsp/jstl/fmt" %>
```

3. 使用标签进行显示
```
<fmt:formatDate value="${mydate}" pattern="yyyy-MM-dd" ></fmt:formatDate>
```

8.携带数据跳转
```
request.setAttribute("requestValue",u);
session.setAttribute("sessionValue",u);
model.addAttribute("modelValue",u);
map.put("mapValue",u);
modelMap.addAttribute("modelMapValue",u);
```
注意：**model map modelmap使用的是请求作用域，必须使用请求转发跳转，如果是重定向跳转，则数据挂失。**

9.静态资源访问问
**最终解决方案，就是在springmvc.xml文件中添加标签**
```
<mvc:resources mapping="/static/**" location="/WEB-INF/static/"></mvc:resources>
<mvc:annotation-driven></mvc:annotation-driven>
```

10.<mvc:annotation-driven/>解析（ 面试点 ）
<mvc:annotation-driven/>会自动注册两个bean，分别为DefaultAnnotationHandlerMapping和AnnotationMethodHandlerAdapter。
1）支持使用 ConversionService 实例对表单参数进行类型转换；
2）支持使用 **@NumberFormat annotation、@DateTimeFormat** ；
3）注解完成数据类型的格式化；
4）支持使用 @RequestBody 和 **@ResponseBody** 注解
5）**支持JSON处理**

4.拦截器

1.概念
过滤器是拦截所有请求进行预处理或后处理，拦截器是SpringMVC框架提供的类似于过滤器功能的一个组件，也是对提交给DispatcherServlet的请求的预处理，后处理以及结束后的清理功能。

2.拦截器的应用场景
1）日志记录
2）权限检查
3）性能监控
...

3.拦截器执行的时机
1）在请求被处理之前
2）在请求处理之后，但还没有进行结果渲染之前，进行处理，这时可以改变响应的结果
3）在请求处理结束后，完成清理，关闭资源，执行一些曾经工作的时机

4.拦截器实现的两种方式
1）继承HandlerInterceptorAdapter父类
2）实现HandlerInterceptor接口，推荐使用这种方式

5.拦截器实现
1）开发拦截器功能
```
@Override
public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
    //完成登录权限验证
    System.out.println("<--------------------");
    //从session中取出登录的用户，如果有，则登录过，放行，如果没有，则打回登录页面
    HttpSession session = request.getSession();
    Users users = (Users) session.getAttribute("users");
    if(users != null){
        //说明登录过，直接放行
        return true;
    }
    //下面的代码为没有登录要执行的代码
    request.setAttribute("msg","您还没有登录，请先去登录！");
    //打回登录页面
    request.getRequestDispatcher("/WEB-INF/jsp/login.jsp").forward(request,response);
    return false;
}
```

2）在springmvc.xml文件中注册拦截器
```
<mvc:interceptors>
    <mvc:interceptor>
<!--        拦截器拦截哪些请求，  /**表示拦截所有请求-->
        <mvc:mapping path="/**"/>
<!--        拦截器放行的请求，例如登录处理的请求，打开登录页面的请求
-->
        <mvc:exclude-mapping path="/showLogin"/>
        <mvc:exclude-mapping path="/login"/>
        <bean class="com.bjpowernode.interceptor.LoginInterceptor">
</bean>
    </mvc:interceptor>
</mvc:interceptors>
```