[今日课程大纲]

自定义拦截器

登录状态验证

[知识点详解]

一.自定义拦截器

1.跟过滤器比较像的技术.

2.发送请求时被拦截器拦截,在控制器的前后添加额外功能.

2.1 跟 AOP 区分开.AOP 在特定方法前后扩充(对 ServiceImpl)

2.2 拦截器,请求的拦截.针对点是控制器方法.(对 Controller)

3.SpringMVC 拦截器和 Filter 的区别

3.1 拦截器只能拦截器 Controller

3.2 Filter 可以拦截任何请求.

4.实现自定义拦截器的步骤:

4.1 新建类实现 HandlerInterceptor

```
public class DemoInterceptor implements
HandlerInterceptor {
    //在进入控制器之前执行
    //如果返回值为 false,阻止进入控制器
    //控制代码
    @Override
```

```
   public boolean preHandle(HttpServletRequest arg0,
HttpServletResponse arg1, Object arg2) throws Exception
{
      System.out.println("arg2:"+arg2);
      System.out.println("preHandle");
      return true;
   }
   //控制器执行完成,进入到 jsp 之前执行.
   //日志记录.
   //敏感词语过滤
   @Override
   public void postHandle(HttpServletRequest arg0,
HttpServletResponse arg1, Object arg2, ModelAndView
arg3)
         throws Exception {

      System.out.println("往"+arg3.getViewName()+"跳转
");
      System.out.println("model 的值
"+arg3.getModel().get("model"));
      String word =
arg3.getModel().get("model").toString();
```

```
        String newWord = word.replace("祖国", "**");
        arg3.getModel().put("model", newWord);
//      arg3.getModel().put("model", "修改后的内容");
        System.out.println("postHandle");
    }
    //jsp 执行完成后执行
    //记录执行过程中出现的异常.
    //可以把异常记录到日志中
    @Override
    public void afterCompletion(HttpServletRequest arg0,
HttpServletResponse arg1, Object arg2, Exception arg3)
            throws Exception {

    System.out.println("afterCompletion"+arg3.getMessag
e());
    }
}
```

4.2 在 springmvc.xml 配置拦截器需要拦截哪些控制器

4.2.1 拦截所有控制器

```
    <mvc:interceptors>
        <bean
class="com.bjsxt.interceptor.DemoInterceptor"></bean>
```

```
    </mvc:interceptors>
```

4.2.2 拦截特定的的 url

```
    <mvc:interceptors>
        <mvc:interceptor>
            <mvc:mapping path="/demo"/>
            <mvc:mapping path="/demo1"/>
            <mvc:mapping path="/demo2"/>
            <bean
class="com.bjsxt.interceptor.DemoInterceptor"></bean>
        </mvc:interceptor>
    </mvc:interceptors>
```

二. 拦截器栈

1. 多个拦截器同时生效时,组成了拦截器栈
2. 顺序:先进后出.
3. 执行顺序和在 springmvc.xml 中配置顺序有关
4. 设置先配置拦截器 A 在配置拦截器 B 执行顺序为

preHandle(A) --> preHandle(B) --> 控制器方法 --> postHandle(B)

--> postHanle(A) --> JSP --> afterCompletion(B) --> afterCompletion(A)

三.SpringMVC 运行原理

1. 文字解释

如果在 web.xml 中设置 DispatcherServlet 的<url-pattern>为/时,当用户发起请求,请求一个控制器,首先会执行 DispatcherServlet. 由 DispatcherServlet 调用 HandlerMapping 的 DefaultAnnotationHandlerMapping 解析 URL, 解析后调用 HandlerAdatper 组件的 AnnotationMethodHandlerAdapter 调用 Controller 中的 HandlerMethod.当 HandlerMethod 执行完成后会返回 View,会被 ViewResovler 进行视图解析,解析后调用 jsp 对应的.class 文件并运行,最终把运行.class 文件的结果响应给客户端.

以上就是 springmvc 运行原理(给面试官说的)

四.SpringMVC 对 Date 类型转换.

1. 在 springmvc.xml 中配置,代码中不需要做任何修改

   1.1 必须额外导入 joda-time.jar

   1.2 时间类型 java.sql.Date

```
<mvc:annotation-driven
conversion-service="conversionService"></mvc:annotati
on-driven>
```

```
<bean id="conversionService"

   class="org.springframework.format.support.FormattingConversionServiceFactoryBean">
      <property name="registerDefaultFormatters" value="false" />
      <property name="formatters">
         <set>
            <bean

   class="org.springframework.format.number.NumberFormatAnnotationFormatterFactory" />
         </set>
      </property>
      <property name="formatterRegistrars">
         <set>
            <bean

   class="org.springframework.format.datetime.joda.JodaTimeFormatterRegistrar">
```

```
                <property name="dateFormatter">
                    <bean


    class="org.springframework.format.datetime.joda.DateTimeFormatterFactoryBean">
                        <property name="pattern"
value="yyyy-MM-dd" />
                    </bean>
                </property>
            </bean>
        </set>
    </property>
</bean>
```

2. 使用注解.在需要转换的参数或实体类属性上添加@DateTimeFormatter(pattern=”表达式”)

2.1 使用 Date 参数接收

```
@RequestMapping("demo")
public String
demo(@DateTimeFormat(pattern="yyyy-MM-dd") Date time){
    System.out.println(time);
    return "abc.jsp";
}
```

2.2 在实体类中

```
@RequestMapping("demo")
public String demo( Demo1 demo){
   System.out.println(demo);
   return "abc.jsp";
}
```

```
public class Demo1 {
   @DateTimeFormat(pattern="yyyy/MM/dd")
   private Date time;
```

2.3 注意地方:

2.3.1 不需要导入额外 jar

2.3.2 Date 是 java.util.Date