**第二讲：几种 处理 程序中的异常+ 框架中拦截器的使用（自定义拦截器）；**

**局部异常：针对某个action**

异常处理标签：<exception-mapping result=”名称” exception=”异常类型”></<exception-mapping>

<result name=”名称”>异常同一处理的界面</result>

**全局异常：程序中的所有，继承了该包 的action类**

<global-result>

<result name = “error”>/error.jsp</resutl>

</global-result>

**范围更广的异常类型处理**

在 web.xml中配置，状态码异常，它的优先级 高于其他任何异常；

**知识点：拦截器的使用**

**回顾：过滤器 监听器**

**1. 什么是拦截器？ 为什么使用拦截器？**

拦截请求的目标方法，在执行action之前，进行拦截操作。

action有很多的默认拦截器，比如：action给属性和方法赋值，如何实现该功能?

使用了struts2框架中的预处理拦截器:调用set方法给参数赋值；

**2.拦截器 过滤器区别？**

**3.如何使用拦截器？**

说明拦截器需要继承父类？

methodfilterInterceptor 和 abstractInterceptor 拦截器父类

methodfilterInterceptor 底层继承了abstractInterceptor 类，同时：methodfilterInterceptor 里面定义了两个参数，分别是：excludeMethods 和 includeMethods;

abstractInterceptor:类不能指定拦截方法，但是可以通过初始化参数 实现放行/拦截 操作；

**实现步骤：**

1. 定义一个拦截器类（必须继承父类：methodfilterInterceptor 和 abstractInterceptor）

2. 从写父类的方法 intercept()方法。（在执行action目标方法之前执行）

3. 配置当前拦截器,在struts2中启用[自定义拦截器后，默认拦截器 不会被执行，在拦截器栈中的第一行必须加入，struts2框架的默认拦截器];

4. 拦截器 的拦截业务逻辑；[自己写]

第二天讲的内容：

1.methodfilterInterceptor 拦截类的使用

2. 阻止表单重复提交

3. 数据类型转换+自定义数据转换

4. 后台数据验证：方法验证+配置文件验证

**2. 阻止表单重复提交**

**实现过程：**

**1.在from表单中，使用s标签会有一组UUID值;**

**2.在struts.xml中配置拦截器，引入拦截类;**

**3.在jsp 页面引入struts标签，UUID存放页面的表单中;**

**3. 数据类型转换+自定义数据转换**

**1. 自动类型转换？**

**.............**

**2. 手动类型转换？**

**例如：将数据【1001，小白，20，大学，未婚... 】转换为 用户对象**

**3. 如何实现手动类型转换功能？[局部使用的转换器，全局的]**

**步骤：**

1.定义转换器类;[继承默认转换器DefaultTypeConverter]

2.重写方法converterValue(); 如何实现转换？

在执行action方法之前，调用自定义转换器中的转换方法，将数组转换成自定类型；

3.返回目标函数值类型

4.配置转换器 启动

注意：必须在action所在目录，创建配置文件；

文件命名格式: Action类名-conversion.properties;

全局：放在src目录下

统配：xwork-conversion.properties;

**案例实现：**

**4. 后台数据验证：方法验证+配置文件验证**

**验证：前台js验证 + 后台数据验证（struts框架验证）**

**如何实现：**

**1.请求类的必须继承制 actionSupport**

**2.在执行 方法之前 调用方法validateXxxx();**

**validateLoginUser();**

**判断是否调用 addfiledError() 方法；**

**3.如果没有通过，在struts.xml配置反回值类型<result name=”input”/> 指向跳转的页面；**

**通过配置文件，验证数据的合法性？**

**写对应类的配置文件，Action类-validation.xml，对action中使用的参数进行配置；**

**案例：登录功能数据验证**