**1．目录介绍：**

Apps： 简单项目案例

Docs：学习文档

Lib：存放所需jar包

Src：存放的是struts2的源码

**2.MVC框架完成的事情。**

Servlet做哪些:

处理用户提交的数据。

调用业务方法

处理业务结果

控制视图显示

用户请求映射到一个java类。

Mvc框架做的事情：

将用户请求映射到一个java类。

获取用户提交的数据

渲染数据 将数据封装到前台显示（request）

控制试图跳转

**步骤**

1.导入jar包

2.配置Struts2的核心控制器 web.xml文件

<filter>

<filter-name> struts2 </filter-name>

<filter-class> org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>struts2</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

3.在src下添加Struts2 的配置文件 Struts.Xml 文件名不能改

Xml文件的头文件 拷贝一个Struts.Xml文件，或者到core包下的Struts-default.xml文件拷贝

**注：到这里Struts2的框架搭建完成。**

4.编写Action类--------helloAction

**package** cn.sxt.action;

**public** **class** HelloAction {

**public** String execute()

{

System.*out*.println("hello");

**return** "success";

}

}

注：在servlet中默认执行service方法

在Struts2中默认执行execute方法

在servlet中service方法为HttpServletRequest , HttpServletResponse无返回值。

在Struts2中方法都是public返回值都是String 类型而且方法都是没有参数的。

配置Action类 在Struts.xml文件中配置helloAction

SStruts2是基于包管理的。

<struts>

<!--extends必须写 直接或间接的继承自 struts-default name自定义 -->

<package name=*"hello"* extends=*"struts-default"*>

<!-- name请求名称 不要写/; class对应action的完全限定名 = 包名+类名 -->

<action name=*"hello"* class=*"cn.sxt.action.HelloAction"*>

<!-- result结果集 name和对应action中的方法的返回值匹配默认是success -->

<result name=*"success"*>/index.jsp</result>

</action>

</package>

</struts>

配置文件

Constant 常量配置

在/org/apache/struts2/default.properties中找到对应的配置

1. 乱码解决

<!-- 解决乱码 -->

<constant name=*"struts.i18n.encoding"* value=*"UTF-8"*></constant>

1. 设置自定义扩展名

<!--设置对应的扩展名-->

<constant name=*"struts.action.extensionaction,,"* value=*"action,,mcr"*></constant>

1. 友好的提示信息

<!-- 显示更友好的提示信息 -->

<constant name=*"struts.devMode"* value=*"false"*></constant>

4.设置配置文件修改后自动加载（推荐在开发中使用）

<constant name="struts.i18n.reload" value="true"></constant>

或者在src目录下新建 strust. Properties文件

写入 xxxx = xxx 的形式

团队协作开发配置

通过include添加不同人员的配置文件

<include file="config/sxt/strust/user.xml"></include>

Package的配置

包的名称自定义，可以配置多个包；namespace命名空间：不同模块可以指定不同的空间

Extends值是直接或者间接继承strust-default

Action的配置

<!--

name是url请求名，不需要加后缀（.action）

class是处理url请求对应的java类，class要求包名+类名

method配置处理请求类的方法，默认为execute 方法要满足是公共的，返回值类型是string ,无参的

method 和name无关

-->

<action name=*"login"* class=*"cn.mcr.action.loginaction"*>

result的配置

<!-- result-结果集

name：匹配请求处理方法的 返回值;默认是success

type:结果处理类型，默认是dispatcher转发

chain:指action链：链接下一个Action执行actionA以后直接执行ActionB后直接执行ActionC，用的较少，地址栏执行第一个action

dispatcher：转发；和servlet一致，如果request中有数据要到试图显示，那么使用 dispatcher

redirect：重定向；如果是重定向到jsp页面，可以直接重定向，如果是重定向到另一个Action，需注意是否配置了action的后缀名，如果要求有后缀名，那么重定向的action一定要加上后缀名

redirectAction:重定向到另一个action；不用加action的后缀名，会将前一个action的后缀名自动加上

stream：以流的形式显示---文件下载

-->

显示传值：${requestScope.username}

全局配置

<global-results>

<result></result>

</global-results>

在action的配置中如果不去配置class属性，将会由默认的的action来执行，默认的action是Actionsuppot类。

<!-- 配置默认的action 当所请求的action不存在时执行默认的action-->

<default-action-ref name=*""*></default-action-ref>

<!-- 配置默认执行的class -->

<default-class-ref class=*""*></default-class-ref>

通配符的配置

使用通配符来配置action，可以减少action的配置\*表示匹配所有。占位符用{1}表示第一个\*所代表的内容

Action的实现方式

1. 定义一个pojo类

好处：定义一个普通的java类即可 不具有侵入性

**public** **class** helloAction {

**public** String execute()

{

System.*out*.println("ok ok");

**return** "success";

}

}

1. 实现action接口

好处：使得我们编写的代码更加规范

**public** **class** hello1action **implements** Action{

**public** String execute() **throws** Exception {

System.*out*.println("ok 2 ok");

**return** *SUCCESS*;

}

}

1. 继承Actionsupport 类

好处：可以继承一些Actionsupport实现的功能；如验证

**public** **class** hello2action **extends** ActionSupport{

**public** String execute() **throws** Exception {

System.*out*.println("ok 3 ok");

**return** *SUCCESS*;

}

}

以对象的方式处理表达数据

1. 属性驱动

Jsp页面

<form action=*"zc.action"* method=*"post"*>

用户名：<input type=*"text"* name=*"u1.name"*>

密码： <input type=*"password"* name=*"u1.pwd"*>

性别：<input type=*"text"* name=*"u1.sex"*>

邮箱：<input type=*"text"* name=*"u1.email"*>

<input type=*"submit"* value=*"注册"*>

</form

Action代码

**public** **class** hello1action {

**private** user u1;

**public** String zc()

{

System.*out*.println(u1);

**return** Action.*SUCCESS*;

}

**public** user getU1() {

**return** u1;

}

**public** **void** setU1(user u1) {

**this**.u1 = u1;

}

}

Struts.xml文件

<struts>

<package name=*"stu"* extends=*"struts-default"*>

<action name=*"zc"* class=*"cn.mcr.action.hello1action"* method=*"zc"*>

<result name=*"success"*>/show.jsp</result>

</action>

</package>

</struts>

User类型

**public** **class** user {

**private** String name;

**private** String pwd;

**private** String sex;

**private** String email;

**public** String toString() {

**return** "user [name=" + name + ", pwd=" + pwd + ", sex=" + sex + ", email="

+ email + "]";

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getPwd() {

**return** pwd;

}

**public** **void** setPwd(String pwd) {

**this**.pwd = pwd;

}

**public** String getSex() {

**return** sex;

}

**public** **void** setSex(String sex) {

**this**.sex = sex;

}

**public** String getEmail() {

**return** email;

}

**public** **void** setEmail(String email) {

**this**.email = email;

}

}

1. 模型驱动（ModelDriven）

Jsp页面

<form action=*"showzc.action"* method=*"post"*>

用户名：<input type=*"text"* name=*"name"*>

密码： <input type=*"password"* name=*"pwd"*>

性别：<input type=*"text"* name=*"sex"*>

邮箱：<input type=*"text"* name=*"email"*>

<input type=*"submit"* value=*"注册"*>

</form>

USerAction

**public** **class** USerAction **implements** ModelDriven<user> {

user u2=**new** user();

**public** String zc1()

{

System.*out*.println(u2);

**return** Action.*SUCCESS*;

}

**public** user getModel() {

**return** u2;

}

}

属性驱动和模型驱动自选，如果属性较多时可以使用模型驱动。

第九讲 获取servletAPI

1. Struts2有两种方式去获取servletAPI 一种是解耦，一种是 耦合

解耦使得使用struts2来进行测试的时候不需要启动服务器。在一定程度上提高了开发效率。

1. 使用解耦的方式来获取servlapi通过Actioncontext对象获取。

获取session

ActionContext.getContext().getSession().put("u2",name);

获取request

Map request=(Map)ActionContext.*getContext*().get("request");

获取Application

Map Application=(Map)ActionContext.getContext().getApplication();

1. 通过耦合的方式来获取

Actioncontext

什么是actioncontext：

Actioncontext是一个map结构的容器，actioncontext是action的上下文，存放action执行过程中的一些信息。 Actioncontext存放action的数据，actioninvocation.Request的数据，session的数据，application的数据，locale的数据等。每次请求时会为当前线程创建一个新的actioncontext。

而actioncontext采用了tradlocal的方式来存放actioncontext。所以actioncontext是线程安全的。

1. 获取actioncontext

Actioncontext.getcontext()获取。由于actioncontext是线程安全的，并且是通过静态方法获取的，所以在本线程中的非action类中也可以直接访问。

注意点：actioncontext是基于请求创建的，所以在非请求的线程中是不能使用actioncontext对象的。如filter的init（）方法。

**tradlocal模式**

tradlocal存放线程局部变量的容器

存放在tradlocal中的变量是线程安全的

**Actioncontext中包含的六大对象**

**Application**

**Session**

**Request**

**Parameters**

**Attr (page🡪request🡪session🡪application)**

**Valuestack 值（特殊的）**

值桟和ognl

1. 值桟**Valuestack**是Actioncontext中的一个对象。值桟是桟结构；struts2中值桟存放的数据是action对象。
2. Ognl表达式简介

表达式—el、re、ognl—用简洁的表达式完成比较复杂的功能。

Ognl(对象图形导航语言)，相对于EL语言，除了保持EL语言的优点外，他的其他优点入下：

能够访问对象的普通方法

能够访问类的静态属性和动态属性

强大的操作集合类对象的能力

支持赋值操作和表达式串联

访问ognl上下文和actioncontext

1. **使用ognl**

**public** **static** **void** main(String[] args) **throws** OgnlException {

//原则：数据分为2类--常用的和不常用的 常用的一般为小数据 不常用的额一般为大数据

//表达式常用的数据直接取，不常用的数据加#取

Map<String,Object> map=**new** HashMap<String,Object>();

map.put("name", "xdc");

map.put("age", 125);

map.put("sex", "nan");

user u1=**new** user();

u1.setName("mcr");

Object obj=Ognl.*getValue*("#name", map, u1);

System.*out*.println(obj);

}

}

Struts2中使用ognl表达式是通过struts2的标签来取值的

在jsp页面中导入struts2的标签库

<%@ taglib prefix=*"s"* uri=*"/struts-tags"*%>

注意：要使用struts2的标签，那么要通过struts2过滤器来启用。如果过滤器的配置为\*.Action结尾时,不能直接访问jsp页面。需要 通过action跳转。如果过滤器配置为/\*时，可以直接访问jsp页面。

Struts推荐不直接访问jsp页面，推荐使用action来控制。

结论：使用ognl表达式访问action的属性时，可以直接访问，访问actioncontext中的属性时需要加#

类型转换

1在servlet中如果表单提交非字符串数据时，需要进行类型转化：如提交age

2在struts2中，常见数据类型struts2已经自动进行了类型转换。无需程序员进行手动转换。

3在某些情况下，有自定义的类型时，struts2不能完成类型转换，那么需要手动转换，如果自定义类型使用的频率较高时，手动转化重复代码会增多—使用struts2提供的类型转换器来进行类型转换。

案例：比如坐标点（x,y）

**不进行类型转换的处理方式：**

Jsp页面：

<form action=*"point.action"* method=*"post"*>

x:<input type=*"text"* name=*"point.x"*><br>

y:<input type=*"text"* name=*"point.y"*><br>

<input type=*"submit"* value=*"提交"*>

</form>

Action代码：

**public** **class** pointaction {

**private** Point point;

**public** String execute()

{

System.*out*.println(point.getX()+"-------"+point.getY());

**return** Action.*SUCCESS*;

}

**public** Point getPoint() {

**return** point;

}

**public** **void** setPoint(Point point) {

**this**.point = point;

}

}

Struts.xml文件的配置：

<package name=*"point"* extends=*"struts-default"*>

<action name=*"point"* class=*"cn.mcr.action.pointaction"*>

<result>/index.jsp</result>

</action>

</package>

这种方式不符合数学的表达形式；采用类型转换 的处理

**使用类型转化的步骤：**

1. 编写转换器—继承strutstypeconverter类
2. 编写xwork-conversion.properties的配置文件，放于src下内容为要转换的类型=类型转换器

**使用类型转换完成上述案例**

类型转换器

**public** **class** pty **extends** StrutsTypeConverter{

//将表单提交的数据转换为自定义数据类型

//arg0 是actioncontxt

//arg1要进行类型转换的字符串数组

//arg2 被转换的类型

**public** Object convertFromString(Map arg0, String[] arg1, Class arg2) {

String value=arg1[0];

System.*out*.println(value);

Point point =**new** Point();

String x=value.substring(1,value.indexOf(","));

String y=value.substring(value.indexOf(",")+1, value.length()-1);

point.setX(Integer.*parseInt*(x));

point.setY(Integer.*parseInt*(y));

System.*out*.println(point.getX());

System.*out*.println(point.getY());

**return** point;

}

//将自定义类型转换为字符串在前台显示 ---通过ognl表达式将会使用该方法进行转换

// arg0 是actioncontxt

//arg1 被转换的类型

**public** String convertToString(Map arg0, Object arg1) {

Point point=(Point)arg1;

**return** "("+point.getX()+","+point.getY()+")";

}

}

xwork-conversion.properties配置

cn.mcr.lei.Point=cn.mcr.pointtype.pty

action代码不变，struts2配置不变

jsp页面：

<body>

x:<s:property value=*"point.x"*/><br>

y:<s:property value=*"point.y"*/><br>

点:<s:property value=*"point"*/><br>

${point }

</body>

Struts2的验证机制

1. 服务端验证：如果action类继承actionsupport类，那么该action类将会继承actionsupp的相关功能：如：验证功能。

**public** **class** ZcAction **extends** ActionSupport{

**private** String name;

**private** **int** age;

**private** String sex;

**private** Date sr;

**public** String execute()

{

**return** Action.*SUCCESS*;

}

@Override

**public** **void** validate( ){

**if**(age>100 || age<0)

{

**this**.addActionError("1111");

}

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getSex() {

**return** sex;

}

**public** **void** setSex(String sex) {

**this**.sex = sex;

}

**public** Date getSr() {

**return** sr;

}

**public** **void** setSr(Date sr) {

**this**.sr = sr;

}

}

Jsp 页面

<%@taglib prefix="m" uri="/struts-tags" %>

<m:actionerror/>

**Struts2验证框架**

使用验证框架步骤：在对应的action的包下添加一个验证框架的配置文件该文件名为actionName-validation.xml

**案例：**  
**jsp页面**

<%@taglib prefix=*"m"* uri=*"/struts-tags"* %>

<m:fielderror></m:fielderror>

**Action**

**public** **class** ValidateAction **extends** ActionSupport{

**private** String name;

**private** **int** age;

**private** String sex;

**private** Date sr;

**public** String execute()

{

**return** Action.*SUCCESS*;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** **int** getAge() {

**return** age;

}

**public** **void** setAge(**int** age) {

**this**.age = age;

}

**public** String getSex() {

**return** sex;

}

**public** **void** setSex(String sex) {

**this**.sex = sex;

}

**public** Date getSr() {

**return** sr;

}

**public** **void** setSr(Date sr) {

**this**.sr = sr;

}

}

**Struts.Xml**

<struts>

<package name=*"point1"* extends=*"struts-default"* namespace=*"/"*>

<action name=*"zc"* class=*"cn.mcr.Action.ValidateAction"*>

<result name=*"success"*>/index.jsp</result>

<result name=*"input"*>/zc.jsp</result>

</action>

</package>

</struts>

**ValidateAction-validation.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE validators PUBLIC

"-//Apache Struts//XWork Validator 1.0.3//EN"

"http://struts.apache.org/dtds/xwork-validator-1.0.3.dtd">

<validators>

<field name=*"name"*>

<field-validator type=*"requiredstring"*>

<param name=*"trim"*>true</param>

<message>用户名必填</message>

</field-validator>

<field-validator type=*"stringlength"*>

<param name=*"trim"*>true</param>

<param name=*"maxlength"*>8</param>

<param name=*"minlength"*>4</param>

<message>去掉空格后的长度应在${minlength}到${maxlength}之间</message>

</field-validator>

</field>

<field name=*"age"*>

<field-validator type=*"int"*>

<param name=*"max"*>150</param>

<param name=*"min"*>1</param>

<message>年龄的范围${max}到${min}</message>

</field-validator>

</field>

</validators>

**第十四讲、拦截器**

1.拦截器：拦截器和过滤器很相似。在actio执行的前后执行。Struts2的核心功能都是通过拦截器来实现。

2.拦截器桟：由多个拦截器

3.作用：对于action的一些公共处理代码可以放到拦截器中来实现。如：权限控制，日志等等。

4.多个拦截器之间的执行是采用责任链设计模式来实现。

5.拦截器的实现：

**6.步骤**

1. 编写拦截器（实现interceptor接口或者继承abstractinterceptor）
2. 在struts.Xml中配置拦截器
3. 在action中引用拦截器

7.案例：

定义拦截器

**public** **class** Timeinterceptor **extends** AbstractInterceptor {

**public** String intercept(ActionInvocation arg0) **throws** Exception {

**long** strat=System.*currentTimeMillis*();

String result=arg0.invoke();

**long** end=System.*currentTimeMillis*();

System.*out*.println("执行时间为"+(end-strat)+"ms");

**return** result;

}

}

在struts.xml中配置拦截器并引用拦截器

<package name=*"hello"* extends=*"struts-default"* namespace=*"/"*>

<!-- 配置拦截器 -->

<interceptors>

<interceptor name=*"time"* class=*"cn.mcrr.interceptor.Timeinterceptor"*></interceptor>

</interceptors>

<action name=*"hello"* class=*"cn.mcrr.action.HelloAction"*>

<result name=*"success"*>/index.jsp</result>

<!-- 应用拦截器-->

<interceptor-ref name=*"time"*></interceptor-ref>

</action>

</package>

8.拦截器的配置详解

a)当引用了自定义拦截器时，默认的拦截器将不起作用

b)默认拦截器：在struts-default.Xml中，配置了默认拦截器。当配置默认拦截器

以后，如果不引用拦截器，那么默认的拦截器将起作用。

<default-interceptor-ref name="defaultStack"/>

c)当引用自定义拦截器后，又想使用struts2提供的拦截器功能，那么需要手动引用

<action name=*"hello"* class=*"cn.mcrr.action.HelloAction"*>

<result name=*"success"*>/index.jsp</result>

<!-- 引用拦截器-->

<interceptor-ref name=*"time"*></interceptor-ref>

<!-- 引用默认的拦截器桟-->

<interceptor-ref name=*"defaultStack"*></interceptor-ref>

</action>

1. 当action应用的拦截器个数比较多时，可以将多个拦截器放入一个拦截器桟中。

<!-- 配置拦截器 -->

<interceptors>

<interceptor name=*"time"* class=*"cn.mcrr.interceptor.Timeinterceptor"*></interceptor>

<!-- 定义拦截器桟 一个拦截器桟中可以包含多个拦截器的引用 -->

<interceptor-stack name=*"mystack"*>

<interceptor-ref name=*"time"*></interceptor-ref>

<interceptor-ref name=*"defaultstack"*></interceptor-ref>

</interceptor-stack>

</interceptors>

1. 当自定义拦截器桟在这个包下的所有action都使用时，可以定义为默认的拦截器桟,或默认的拦截器

</interceptors>

<default-interceptor-ref name=*"mystack"*></default-interceptor-ref>

<action name=*"hello"* class=*"cn.mcrr.action.HelloAction"*>

9.拦截器的应用案例：

对于登录权限的控制：

获取请求的名称：String actionname=invocation.getproxy().getActionNAme();

获取存入的对象：Object obj=invocation.getInvocationContext().getSession.get(“user”);

继续执行其他过滤器：return invocation.nvoke();

10．方法拦截器：方法拦截器时比action拦截器更加细粒度的控制，主体实现和action拦截器一致。但是方法拦截器是继承methodfilterceptor类，重写dointercept（）方法。

配置被拦截规则：

<interceptor-ref name=*"methodfilterceptor"*>

<!-- 配置被拦截的方法 -->

<param name=*"includeMethods"*>list,add</param>

<!-- 配置不被拦截的方法 -->

<param name=*"excludeMethods"*>login</param>

</interceptor-ref>

文件上传

1. struts2的文件上传功能使用的是apache下的commons-fileupload来实现。
2. jsp的表单要求是post提交，并且enctype=”multipart/form-data

<form action=*"upload.action"* method=*"post"* enctype=*"multipart/form-data"*>

文件：<input type=*"file"* name=*"file"*/>

<input type=*"submit"* name=*"上传"*/>

</form>

1. UploadAction代码：在action中需要提供3个属性，一个file，名称是表单域名，其他两个分别是表单域名+FileName，表单域名+ContentType；

**public** **class** UploadAction **extends** ActionSupport{

**private** File file;

**private** String fileFileName;

**private** String fileContentType;

@SuppressWarnings("deprecation")

**public** String upload() **throws** IOException

{

HttpServletRequest request =ServletActionContext.*getRequest*();

String path=request.~~getRealPath~~("/upload");

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

System.*out*.println(fileFileName);

System.*out*.println(fileContentType);

System.*out*.println(path);

System.*out*.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

InputStream is=**new** FileInputStream(file);

OutputStream os=**new** FileOutputStream(**new** File(path,fileFileName));

**byte**[] buffer=**new** **byte**[200];

**int** len=0;

**while**((len=is.read(buffer))!=-1)

{

os.write(buffer, 0, len);

}

is.close();

os.close();

**return** Action.*SUCCESS*;

}

**public** File getFile() {

**return** file;

}

**public** **void** setFile(File file) {

**this**.file = file;

}

**public** String getFileFileName() {

**return** fileFileName;

}

**public** **void** setFileFileName(String fileFileName) {

**this**.fileFileName = fileFileName;

}

**public** String getFileContentType() {

**return** fileContentType;

}

**public** **void** setFileContentType(String fileContentType) {

**this**.fileContentType = fileContentType;

}

}

1. struts.xml配置文件的编写，主要是上传文件大小的控制。需要配置拦截器的maximumsize属性和strust2的静态属性strust2.multipart.maxsize
2. maxsize要大于等于maximumsize

文件下载

1. 在servlet中的文件下载是通过流来实现的
2. 在struts2中是不是可以通过流来下载 ? YES

实现：使用httpservletresponse来实现，和以前select中文件下载的方式一致。注意在action执行方法中 返回null；

Action代码

**public** **class** DownloadAction {

**public** String execute() **throws** IOException

{

HttpServletRequest req= ServletActionContext.*getRequest*();

HttpServletResponse res=ServletActionContext.*getResponse*();

String path=req.~~getRealPath~~("/download");

File file =**new** File(path, "12.txt");

res.setContentLength((**int**)file.length());

res.setCharacterEncoding("utf-8");

res.setContentType("application/octet-stream");

res.setHeader("Content-Disposition", "attachment;filename=12.txt");

**byte**[] buffer =**new** **byte**[400];

**int** len =0;

InputStream is=**new** FileInputStream(file);

OutputStream os=res.getOutputStream();

**while**((len=is.read(buffer))!=-1)

{

os.write(buffer,0,len);

}

os.close();

is.close();

**return** **null**;

}

}

Struts.xml配置

<struts>

<package name=*"defualt"* extends=*"struts-default"*>

<!-- 当action的处理方法返回空时 不用配置结果集 -->

<action name=*"download"* class=*"cn.mcr.action.DownloadAction"*>

</action>

</package>

</struts>

Jsp页面

<body>

<a href=*"download.action"*>下载12.txt </a>

</body>

1. 使用struts2本身的结果集进行文件下载

Action代码

**public** **class** StreamDownloadAction {

**private** String fileName;

**public** String execute()

{

**return** Action.*SUCCESS*;

}

**public** InputStream getInputStream() **throws** FileNotFoundException

{

HttpServletRequest req=ServletActionContext.*getRequest*();

String path =req.~~getRealPath~~("/download");

**return** **new** FileInputStream (**new** File(path,fileName));

}

**public** String getFileName() {

**return** fileName;

}

**public** **void** setFileName(String fileName) {

**this**.fileName = fileName;

}

}

Struts.xml配置文件

<struts>

<package name="defualt" extends="struts-default">

<!-- 当action的处理方法返回空时 不用配置结果集 -->

<action name="download" class="cn.mcr.action.DownloadAction">

</action>

<action name="streamdownload" class="cn.mcr.action.StreamDownloadAction">

<result name="success" type="stream">

<param name="contentDisposition">attachment;filename=${fileName}</param>

</result>

Jsp页面

<body>

<a href=*"streamdownload.action?fileName=12.txt"*>下载12.txt</a>

<a href=*"streamdownload.action?fileName=t.txt"*>t.txt </a>

</body>

批量提交

1. 数组和集合的处理：

Jsp中代码：

<form action=*"user.action"* method=*"post"*>

用户名：<input type=*"text"* name=*"name"*><br>

爱好的水果： <input type=*"checkbox"* name=*"like"* value=*"苹果"*>苹果

<input type=*"checkbox"* name=*"like"* value=*"香蕉"*>香蕉

<input type=*"checkbox"* name=*"like"* value=*"橘子"*>橘子

<input type=*"checkbox"* name=*"like"* value=*"西瓜"*>西瓜

<input type=*"checkbox"* name=*"like"* value=*"菠萝"*>菠萝

爱好的游戏：<input type=*"checkbox"* name=*"like1"* value=*"菠萝"*>菠萝

<input type=*"checkbox"* name=*"like1"* value=*"lol"*>lol

<input type=*"checkbox"* name=*"like1"* value=*"cf"*>cf

<input type=*"checkbox"* name=*"like1"* value=*"dota"*>dota

<input type=*"checkbox"* name=*"like1"* value=*"dnf"*>dnf

<input type=*"checkbox"* name=*"like1"* value=*"流星蝴蝶剑"*>流星蝴蝶剑

<input type=*"submit"* value=*"ok"*>

</form>

Action处理

**public** **class** LikeAction {

**private** String name;

**private** String like[];

**private** List<String> like1[];

**public** String execute()

{

System.*out*.println("name===="+name);

System.*out*.print("like====");

**for** (**int** i=0; i<like.length;i++)

{

System.*out*.print(like[i]);

}

System.*out*.println();

System.*out*.println("gamee="+like1);

**return** Action.*SUCCESS*;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String[] getLike() {

**return** like;

}

**public** **void** setLike(String like[]) {

**this**.like = like;

}

**public** List<String>[] getLike1() {

**return** like1;

}

**public** **void** setLike1(List<String> like1[]) {

**this**.like1 = like1;

}

}

1. 集合对象的处理：

需要自定义拦截器来实现。

Jsp页面

<body>

<form action=*"batch.action"* method=*"post"*>

用户名： <input type=*"text"* name=*"us.name"*> 年龄：<input type=*"text"* name=*"us.age"*><br>

用户名： <input type=*"text"* name=*"us.name"*> 年龄：<input type=*"text"* name=*"us.age"*><br>

用户名： <input type=*"text"* name=*"us.name"*> 年龄：<input type=*"text"* name=*"us.age"*><br>

<input type=*"submit"* value=*"提交"*>

</form>

</body>

Action

public class BatchAction {

private List<user> us;

public String execute()

{

for(user u:us)

{

System.out.println(u);

}

return "success";

}

public List<user> getUs() {

return us;

}

public void setUs(List<user> us) {

this.us = us;

}

}

注解的实现

1. 通过struts2-convention-plugin-2.3.4.jar插件来实现

导入插件

Action需要继承actionsupper类

Action要发放在以.Action结尾的包中