第 3 章　Struts 2 的类型转换

学习的目的

本章主要学习 Struts 2 内置的类型转换器和自定义类型转换器，理解类型转换器的原理，掌握类型转换器的配置。

本章主要内容

- 类型转换的意义；
- Struts 2 内置的类型转换器；
- 自定义类型转换器。

在 MVC 框架中，需要收集用户请求参数，并将请求参数传递给应用的控制器组件。此时存在一个问题，所有的请求参数类型只能是字符串数据类型，但 Java 是强类型语言，所以 MVC 框架必须将这些字符串请求参数转换成相应的数据类型。

Struts 2 不仅提供了强大的类型转换机制，而且开发者还可以方便地开发自己的类型转换器，完成字符串和各种数据类型之间的转换。

3.1　类型转换的意义

本节通过一个简单应用(JSP ＋ Servlet)为例来介绍类型转换的意义。如图 3.1 所示，添加商品页面用于收集用户输入的商品信息。商品信息包括商品名称(字符串类型 String)、商品价格(双精度浮点类型 double)、商品数量(整数类型 int)。

图 3.1　添加商品信息的收集页面

addGoods.jsp 页面的代码如下：

```
<body>
    <form action="addGoods" method="post">
        商品名称：<input type="text" name="goodsname"/><br>
        商品价格：<input type="text" name="goodsprice"/><br>
        商品数量：<input type="text" name="goodsnumber"/><br>
        <input type="submit" value="提交"/>
    </form>
</body>
```

希望页面收集到的数据提交到 addGoods 的 Servlet(AddGoodsServlet 类)，该 Servlet 将这些请求信息封装成一个 Goods 类的值对象。

Goods 类的代码如下：

```
package model;
```

```
public class Goods {
    private String goodsname;
    private double goodsprice;
    private int goodsnumber;
    //无参数的构造方法
    public Goods(){}
    //有参数的构造方法
    public Goods(String goodsname, double goodsprice, int goodsnumber) {
        super();
        this.goodsname=goodsname;
        this.goodsprice=goodsprice;
        this.goodsnumber=goodsnumber;
    }
    //此处省略了 setter 和 getter 方法
    ⋮
}
```

AddGoodsServlet 类的代码如下：

```
package servlet;
import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import model.Goods;
public class AddGoodsServlet extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        doPost(request, response);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        //设置编码,防止乱码
        request.setCharacterEncoding("utf-8");
        //获取参数值
        String goodsname=request.getParameter("goodsname");
        String goodsprice=request.getParameter("goodsprice");
        String goodsnumber=request.getParameter("goodsnumber");
        //下面进行类型转换
        double newgoodsprice=Double.parseDouble(goodsprice);
        int newgoodsnumber=Integer.parseInt(goodsnumber);
        //将转换后的数据封装成 goods 值对象
        Goods goods=new Goods(goodsname, newgoodsprice, newgoodsnumber);
        //将 goods 值对象传递给数据访问层,进行添加操作,代码省略
```

```
        ⋮
    }
}
```

对于上面这个应用而言，开发者需要自己在 Servlet 中进行类型转换，并将其封装成值对象。这些类型转换操作全部手工完成，异常繁琐。

对于 MVC 框架而言，必须将请求参数转换成值对象类里各属性对应的数据类型——这就是类型转换的意义。下面介绍使用 Struts 2 来完成类型转换。

3.2 Struts 2 内置的类型转换器

在 Struts 2 框架中，对于常用的数据类型，开发者无须创建自己的类型转换器，因为 Struts 2 有许多内建的类型转换器完成常用的类型转换。Struts 2 提供的内置类型转换，包括如下几种类型。

- boolean 和 Boolean：完成 String 和布尔型之间的转换。
- char 和 Character：完成 String 和字符型之间的转换。
- int 和 Integer：完成 String 和整型之间的转换。
- long 和 Long：完成 String 和长整型之间的转换。
- float 和 Float：完成 String 和单精度浮点型之间的转换。
- double 和 Double：完成 String 和双精度浮点型之间的转换。
- Date：完成 String 和日期类型之间的转换，日期格式为用户请求本地的 SHORT 格式(如 yy-mm-dd)。
- 数组(arrays)：该类型在数据转换时，必须满足需要转换的数据中每一个元素都能转换成数组的类型。
- 集合(collections)：在使用集合类型转换器时，如果集合中的数据无法确定，可以先将其封装到一个 String 类型的集合中，然后在用到某个元素时再进行手动转换。

类型转换是在页面与 Action 相互传递数据时发生的。Struts2 对于基本类型如 int、long、float、double、boolean 以及 char 等(包括 Date)，已经做好了基本类型转换。如果 Action 中包含基本类型属性(一定要有 getter 和 setter 方法)，在页面上只要包含对应此属性的名称，Struts 2 会自动进行类型转换。例如，有这样一个 Action，代码如下：

```
public class TestAction extends ActionSupport{
    //用来封装用户请求中用户名的信息
    private String uname;
    //用来封装用户请求中年龄的信息
    private int uage;
    public String getUname() {
        return uname;
    }
    public void setUname(String uname) {
        this.uname=uname;
    }
```

```
    public int getUage() {
        return uage;
    }
    public void setUage(int uage) {
        this.uage=uage;
    }
    public String execute(){
        System.out.println(uage);
        return SUCCESS;
    }
}
```

另外,还需要提供一个 JSP 页面,代码如下:

```
<body>
    <form action="input.action" method="post">
    <h3>信息提交</h3>
        用户名: <input type="text" name="uname"><br>
        年龄: <input type="text" name="uage"><br>
        <input type="submit" value="提交">
    </form>
</body>
```

当 JSP 页面的 form 表单提交到 Action 时,Struts 2 使用内置的类型转换器自动将表单中的参数 uage 转换成 int 类型。上述的 Action(TestAction)代码可以编写如下:

```
package action;
import model.Information;
import com.opensymphony.xwork2.ActionSupport;
import com.opensymphony.xwork2.ModelDriven;
public class ReceiveAction extends ActionSupport implements ModelDriven<Information>{
    //定义 info 对象,需要创建该对象
    private Information info=new Information();
    /**
     * 处理用户请求的方法
     */
    public String execute(){
        System.out.println("uname:"+info.getUname());
        System.out.println("uage:"+info.getUage());
        return SUCCESS;
    }
    @Override
    public Information getModel() {
        return info;
    }
}
```

上面 Action 中的 Model 类 Information 的代码如下：

```
package model;
public class Information {
    //用来封装用户请求中用户名的信息
    private String uname;
    //用来封装用户请求中年龄的信息
    private int uage;
    public String getUname() {
        return uname;
    }
    public void setUname(String uname) {
        this.uname=uname;
    }
    public int getUage() {
        return uage;
    }
    public void setUage(int uage) {
        this.uage=uage;
    }
}
```

3.3 自定义类型转换器

当 Struts 2 内置的类型转换器不能满足需求时，开发者可以开发自己的类型转换器。例如，有个应用 ch3 希望用户在页面上的表单输入信息来创建商品信息。当输入“苹果，10.58，200”时，表示在程序中自动创建一个 new Goods，并将“苹果”自动赋值给 goodsname 属性，将 10.58 自动赋值给 goodsprice 属性，将 200 自动赋值给 goodsnumber 属性。

想实现上述应用功能需要做以下 4 件事：

- 需要一个 POJO 类 Goods；
- 创建一个 Action；
- 创建自定义类型转换器；
- 配置类型转换器。

其他剩下的配置与普通 Action 并无区别。现在按照上述步骤采用自定义类型转换器完成需求。

1. 编写 POJO 类 Goods

创建名为 Goods.java 的类文件，代码如下：

```
package model;
public class Goods {
    private String goodsname;
    private double goodsprice;
    private int goodsnumber;
```

```
    public String getGoodsname() {
        return goodsname;
    }
    public void setGoodsname(String goodsname) {
        this.goodsname=goodsname;
    }
    public double getGoodsprice() {
        return goodsprice;
    }
    public void setGoodsprice(double goodsprice) {
        this.goodsprice=goodsprice;
    }
    public int getGoodsnumber() {
        return goodsnumber;
    }
    public void setGoodsnumber(int goodsnumber) {
        this.goodsnumber=goodsnumber;
    }
}
```

2. 编写 Action 类

创建名为 GoodsConvertAction.java 的 Action 文件，代码如下：

```
package action;
import model.Goods;
import com.opensymphony.xwork2.ActionSupport;
public class GoodsConvertAction extends ActionSupport{
    //Struts 2 的转换器会自动将请求过来的值转换成 Goods 类型
    private Goods goods;
    public Goods getGoods() {
        return goods;
    }
    public void setGoods(Goods goods) {
        this.goods=goods;
    }
    public String execute(){
        return SUCCESS;
    }
}
```

3. 编写自定义类型转换器类

编写自定义类型转换器见 3.3.1 节。

4. 注册自定义类型转换器

实现了自定义类型转换器之后，将该类型转换器注册在 Web 应用中，Struts 2 框架才可以正常使用该类型转换器。注册自定义类型转换器见 3.3.2 节。

5. 修改配置文件

根据业务流程，修改 struts.xml 配置文件，具体代码如下：

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts Configuration
2.1//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
    <package name="converter" namespace="/" extends="struts-default">
        <action name="convert" class="action.GoodsConvertAction">
            <result>/showGoods.jsp</result>
        </action>
    </package>
</struts>
```

6. 新建相关页面

提供输入和输出的 JSP 页面。注意，与以往不同的是，输入页面的 form 表单的 input 元素的 name 属性设置为要转换后的目标类型对象 goods。

输入页面 input.jsp 的代码如下：

```
<form action="convert.action" method="post">
    请输入商品信息(格式为：苹果,10.58,200)：
    <input type="text" name="goods"/><br>
    <input type="submit" value="提交"/>
</form>
```

输出页面并未有任何变化，可以使用 EL 表达式将 goods 对象的属性输出到页面，也可以使用 Struts 2 的标签(后续章节讲解)输出到页面。输出页面 showGoods.jsp 的代码如下：

```
<body>
    您创建的商品信息如下：<br>
    <!--使用 EL 表达式取出 Action 类的属性 goods 的值 -->
    商品名为：${goods.goodsname },
    商品价格为：${goods.goodsprice },
    商品数量为：${goods.goodsnumber }。
</body>
```

程序发布后，打开 IE，输入 http://localhost:8080/ch3/input.jsp，显示的页面如图 3.2 所示。

图 3.2　自定义类型转换器示例的首页面

输入“桃子,10.88,800”，提交表单查看结果，输出如图 3.3 所示。

图 3.3 自定义类型转换器示例的结果页面

至此，Struts 2 的自定义类型转换器就完成了。

3.3.1 实现类型转换器

创建自定义类型转换器类，有三种方法：

- 实现 ognl.TypeConverter 接口；
- 继承 DefaultTypeConverter 类；
- 继承 StrutsTypeConverter 类。

1. 实现 ognl.TypeConverter 接口

TypeConverter 接口有一个接口方法：

```
public Object convertValue(Map context, Object target, Member member,
    String propertyName,Object value, Class toType);
```

实现 ognl.TypeConverter 接口创建自定义类型转换器，必须实现上述接口方法，不过该接口方法过于复杂，所以 OGNL 项目还提供一个 TypeConverter 接口的实现类：DefaultTypeConverter。一般通过继承该类实现自己的类型转换器。

2. 继承 DefaultTypeConverter 类

DefaultTypeConverter 类实现了 TypeConverter 接口，并提供了一个简化的 convertValue 方法。

```
public Object convertValue(Map context, Object value, Class toType)
```

convertValue 方法负责完成类型的转换，这种转换是双向的：当需要把字符串转换成 Goods 实例时，是通过该方法实现的；当需要把 Goods 实例转换成字符串时，也是通过该方法实现的。应用 ch3 的自定义类型转换器类 GoodsConverter1.java 的代码如下：

```
package converter;
import java.util.Map;
import model.Goods;
import ognl.DefaultTypeConverter;
//通过继承 DefaultTypeConverter 实现自定义类型转换器
public class GoodsConverter1 extends DefaultTypeConverter{
    //类型转换器必须重写该方法,实现双向转换
    public Object convertValue(Map context, Object value, Class toType) {
        //当需要字符串向 Goods 类型转换时
        if(toType==Goods.class){
            //系统的请求参数是一个字符串数组
            String params[]=(String[])value;
            //创建一个 Goods 实例
```

```
            Goods goods=new Goods();
            //只处理请求参数数组的第一个元素(因为页面只有一个请求参数),并以","分隔
            String stringValues[]=params[0].split(",");
            //为 Goods 实例赋值
            goods.setGoodsname(stringValues[0]);
            goods.setGoodsprice(Double.parseDouble(stringValues[1]));
            goods.setGoodsnumber(Integer.parseInt(stringValues[2]));
            return goods;
        }
        //当需要将 Goods 实例向字符串转换时
        else if(toType==String.class){
            Goods goods= (Goods)value;
            return "[" +goods.getGoodsname() +","
                       +goods.getGoodsprice() +","
                       +goods.getGoodsnumber() +"]";
        }
        return null;
    }
}
```

3. 继承 StrutsTypeConverter 类

Struts 2 提供了一个 TypeConverter 接口的默认实现类 StrutsTypeConverter。该实现类有两个抽象方法：public Object convertFromString(Map context, String[] values, Class toClass)和 public String convertToString(Map context, Object obj),在定义类型转换器必须被实现。

convertFromString 方法的功能是将一个或多个字符串值转换为指定的类型。参数 context 是表示 Action 上下文的 Map 对象,参数 values 是要转换的字符串值,参数 toClass 是要转换的目标类型。

convertToString 方法的功能是将指定的对象转换为字符串。参数 context 是表示 Action 上下文的 Map 对象,参数 obj 是要转换的对象。

应用 ch3 的自定义类型转换器类 GoodsConverter2.java 的代码如下：

```
package converter;
import java.util.Map;
import model.Goods;
import org.apache.struts2.util.StrutsTypeConverter;
public class GoodsConverter2 extends StrutsTypeConverter{
    //将字符串值转换为指定的类型
    @Override
    public Object convertFromString(Map context, String[] values, Class toClass){
        if(values.length >0){
            //创建一个 Goods 实例
            Goods goods=new Goods();
            //只处理请求参数数组的第一个元素(因为页面只有一个请求参数),并以","分隔
```

```
            String stringValues[]=values[0].split(",");
            //为 Goods 实例赋值
            goods.setGoodsname(stringValues[0]);
            goods.setGoodsprice(Double.parseDouble(stringValues[1]));
            goods.setGoodsnumber(Integer.parseInt(stringValues[2]));
            return goods;
        }else{
            return null;
        }
    }
    //将指定的对象转换为字符串
    @Override
    public String convertToString(Map context, Object obj) {
        if(obj instanceof Goods){
            Goods goods=(Goods)obj;
            return "[" +goods.getGoodsname() +","
                       +goods.getGoodsprice() +","
                       +goods.getGoodsnumber() +"]";
        }
        return null;
    }
}
```

3.3.2 注册类型转换器

在 Web 应用中,注册类型转换器有两种常用方式:

- 注册局部类型转换器:仅仅对某个 Action 的属性起作用。
- 注册全局类型转换器:所有 Action 的特定属性都会生效。

1. 注册局部类型转换器

在 Action 所在的包中建立 properties 文件,文件命名格式为 ActionName-conversion.properties。ActionName 是需要转换器生效的 Action 的类名,后面的-conversion.properties 是固定部分。

对于 3.3 节中的 GoodsConvertAction 类,应该提供的类型转换器注册文件的文件名为 GoodsConvertAction-conversion.properties,该文件是一个典型的 properties 文件,文件由 key-value 对组成。文件内容为:

```
propertyName=类型转换器类
```

ActionName-conversion.properties 文件由多个“propertyName=类型转换器类”项组成,其中 propertyName 是 Action 中需要类型转换器转换的属性名,类型转换器类是开发者实现的类型转换器的全限定类名(需要加包名)。

下面是 GoodsConvertAction-conversion.properties 文件的内容(使用的自定义类型转换器类是 3.3.1 节中的 GoodsConverter1.java):

```
goods=converter.GoodsConverter1
```

至此，局部类型转换器注册成功。

2. 注册全局类型转换器

假设应用中有多个 Action 都包含了 Goods 类型的属性，如果多次重复注册局部类型转换器，则将是很繁琐的事情。幸运的是，Struts 2 提供了全局类型转换器，它对指定类型的全部属性有效。

注册全局类型转换器应该在 classpath(src)下提供一个 xwork-conversion.properties 文件，其内容由多个“目标类型＝对应类型转换器”项组成，其中“目标类型”指定需要完成类型转换的类(全限定名)，“对应类型转换器”指定类型转换的转换器类(全限定名)。例如，指定 model.Goods 类的类型转换器为 3.3.1 节中的 converter. GoodsConverter2，则注册全局类型转换器的注册文件代码如下：

```
model.Goods=converter.GoodsConverter2
```

一旦注册了上面的全局类型转换器，该全局类型转换器就会对所有 Goods 类型属性起作用。

注意：如果 Action 的某个属性类型既被注册了局部类型转换器，又被注册了全局类型转换器，则执行局部类型转换器。

3.3.3 实践环节

创建一个 Web 应用 project333pratice，该应用具体实施步骤如下：

(1) 编写一个 JSP 页面 input.jsp，该页面运行效果如图 3.4 所示。

(2) 编写 POJO 类 User。

(3) 编写 Action 类，在 Action 类中，类型转换器会自动将请求过来的值转换成 User 类型。

(4) 通过继承 StrutsTypeConverter 类的方式，编写自定义类型转换器类 UserConverter。

(5) 注册全局类型转换器。

(6) 配置 Action。

(7) 编写用户信息输出页面 showUser.jsp，页面效果如图 3.5 所示。

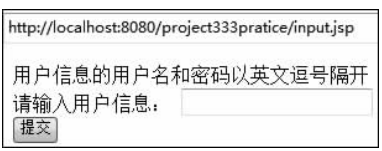

图 3.4 实践环节的首页面

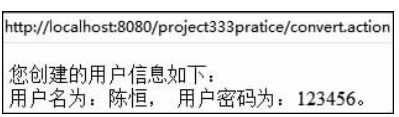

图 3.5 实践环节的结果页面

3.4 数组属性的类型转换器

在 3.3 节中，一直只处理字符串数组的第一个数组元素——请求参数是单个值，而不是一个字符串数组。实际上，请求参数经常是字符串数组，如图 3.6 所示。

图 3.6　数组类型转换器的首页面

上述页面 inputs.jsp 的代码如下：

```
<body>
    <h3>数组类型转换器</h3>
    <form action="converts.action" method="post">
    请输入商品信息(格式为：苹果,10.58,200)：<br>
    商品 1：<input type="text" name="goods"/><br>
    商品 2：<input type="text" name="goods"/><br>
    商品 3：<input type="text" name="goods"/><br>
    <input type="submit" value="提交"/>
    </form>
</body>
```

在此页面中包含三个商品信息的请求参数，名称都是 goods。此时 goods 请求参数必须是数组类型或 List 类型。

下面是处理该请求的 Action 类的代码：

```
package action;
import model.Goods;
import com.opensymphony.xwork2.ActionSupport;
public class GoodsConvertActions extends ActionSupport{
    private Goods[] goods;
    public Goods[] getGoods() {
        return goods;
    }
    public void setGoods(Goods[] goods) {
        this.goods=goods;
    }
    public String execute(){
        return SUCCESS;
    }
}
```

上面的 Action 使用了 Goods[]数组类型属性来封装 goods 请求参数。下面是本应用的类型转换器代码：

```
package converter;
import java.util.Map;
```

```
import model.Goods;
import org.apache.struts2.util.StrutsTypeConverter;
public class GoodsConverter3 extends StrutsTypeConverter{
    //将字符串值转换为指定的类型
    @Override
    public Object convertFromString(Map context, String[] values, Class toClass) {
        if(values.length >1){
            //创建一个数组 Goods 实例
            Goods goods[]=new Goods[values.length];
            //遍历数组
            for(int i=0; i <values.length; i++){
                //将每个数组元素转换成一个 Goods 实例
                Goods agoods=new Goods();
                String stringValues[]=values[i].split(",");
                //为 agoods 实例赋值
                agoods.setGoodsname(stringValues[0]);
                agoods.setGoodsprice(Double.parseDouble(stringValues[1]));
                agoods.setGoodsnumber(Integer.parseInt(stringValues[2]));
                goods[i]=agoods;
            }
            return goods;
        }else{
            return null;
        }
    }
    //将指定的对象转换为字符串
    @Override
    public String convertToString(Map context, Object obj) {
        if(obj instanceof Goods[]){
            Goods goods[]=(Goods[])obj;
            String result="[";
            //遍历数组
            for(Goods agoods : goods){
                result=result +"<" +
                        agoods.getGoodsname() +","
                        +agoods.getGoodsprice() +","
                        +agoods.getGoodsnumber() +">";
            }
            return result +"]";
        }
        return null;
    }
}
```

注册上述类型转换器和配置 Action 的代码略。

当在图 3.6 中的三个输入框内分别输入：“苹果 1,10.58,100”、“苹果 2,11.58,200”、

“苹果 3,12.58,300”时,然后提交请求,将看到如图 3.7 所示的页面。

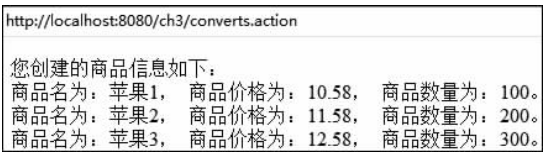

图 3.7 数组类型转换器的结果页面

上述结果页面 showGoodses.jsp 的代码如下:

```
<%@page language="java" import="java.util.*" pageEncoding="utf-8"%>
<%
String path=request.getContextPath();
String basePath=request.getScheme()+"://"+request.getServerName()+":"+
request.getServerPort()+path+"/";
%>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <base href="<%=basePath%>">
        <title>My JSP 'showGoods.jsp' starting page</title>
    </head>
    <body>
        您创建的商品信息如下:<br>
        <c:forEach var="mf" items="${goods}">
        商品名为:${mf.goodsname },
        商品价格为:${mf.goodsprice },
        商品数量为:${mf.goodsnumber }。<br>
        </c:forEach>
    </body>
</html>
```

3.5 集合属性的类型转换器

如果 3.4 节应用中 goods 属性不是使用字符串数组封装,而是使用集合属性来处理,则修改后的 Action 代码如下:

```
package action;
import java.util.List;
import model.Goods;
import com.opensymphony.xwork2.ActionSupport;
public class GoodsConvertActionsList extends ActionSupport{
    private List<Goods>goods;
    public List<Goods>getGoods() {
```

```
        return goods;
    }
    public void setGoods(List<Goods>goods) {
        this.goods=goods;
    }
    public String execute(){
        return SUCCESS;
    }
}
```

因为 Action 里的 goods 属性是一个 List 类型，应该在类型转换器中也提供对应的转换，因此提供如下的类型转换器：

```
package converter;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import model.Goods;
import org.apache.struts2.util.StrutsTypeConverter;
public class GoodsConverter4 extends StrutsTypeConverter{
    //将字符串值转换为指定的类型
    @Override
    public Object convertFromString(Map context, String[] values, Class toClass){
        if(values.length >1){
            //创建一个 List 对象
            List<Goods>result=new ArrayList<Goods>();
            //遍历请求参数数组
            for(int i=0; i <values.length; i++){
                //将每个数组元素转换成一个 Goods 实例
                Goods agoods=new Goods();
                String stringValues[]=values[i].split(",");
                //为 agoods 实例赋值
                agoods.setGoodsname(stringValues[0]);
                agoods.setGoodsprice(Double.parseDouble(stringValues[1]));
                agoods.setGoodsnumber(Integer.parseInt(stringValues[2]));
                result.add(agoods);
            }
            return result;
        }else{
            return null;
        }
    }
    //将指定的对象转换为字符串
    @Override
    public String convertToString(Map context, Object obj) {
        if(obj instanceof Goods[]){
```

```
            List<Goods>goods=(List<Goods>)obj;
            String result="[";
            //遍历数组
            for(Goods agoods : goods){
                result=result +"<" +
                        agoods.getGoodsname() +","
                        +agoods.getGoodsprice() +","
                        +agoods.getGoodsnumber() +">";
            }
            return result +"]";
        }
        return null;
    }
}
```

本节修改后的代码只需要将 3.4 节中对应的配置文件做简单修改即可,JSP 页面不需要做任何修改。实际上,List 对象和数组几乎可以互换使用。即使 Action 类里使用 List 对象来封装请求参数的多个值,类型转换器也可以将字符串数组转换成 Goods 数组(不需要修改类型转换器类),因为 Struts 2 默认支持数组和 List 之间的转换。

3.6 本章小结

本章重点讲解自定义类型转换器的实现和注册。但在实际应用中,开发者很少自定义类型转换器,一般都是使用 Struts 2 内置的转换器。因为,Struts 2 提供了常用类型的转换功能。

习 题 3

(1) 在 MVC 框架中,为什么要进行类型转换?

(2) Struts 2 提供了哪些内置的类型转换器?

(3) 在 Struts 2 框架中,如何自定义类型转换器类?又如何注册类型转换器?

第 4 章　Struts 2 的拦截器

学习的目的

拦截器是 Struts 2 框架的一个重要组成部分，通过本章的学习，要理解拦截器的基本原理，理解拦截器与过滤器的区别，掌握拦截器的配置，并且能够自定义拦截器。

本章主要内容

- 拦截器概述；
- 配置拦截器；
- 自定义拦截器；
- 内置拦截器。

拦截器(Interceptor)是 Struts 2 框架的核心组成部分。在 Struts 2 文档中对拦截器的解释为，拦截器是动态拦截 Action 调用的对象。它提供了一种机制，使开发者可以定义一个特定的功能模块，这个模块可以在 Action 执行之前或之后运行，也可以在一个 Action 执行之前阻止 Action 执行。

在 Struts 2 框架中，当需要使用某个拦截器时，只需要在配置文件中进行相关的配置即可；如果不需要使用某个拦截器，只需要在配置文件中取消该拦截器的配置即可。在实际应用中，不管是否应用某个拦截器，对于整个 Struts 2 框架不会产生任何影响。这是一种可插拔式的设计，具有非常好的可扩展性。

4.1　拦截器概述

4.1.1　拦截器的原理

Struts 2 的拦截器实现相对简单。当请求到达 Struts 2 的 StrutsPrepareAndExecuteFilter 时，Struts 2 会查找配置文件，并根据其配置实例化对应的拦截器对象，然后组成一个拦截器栈(又称拦截器链)，最后一个一个地调用栈中的拦截器。

拦截器围绕着 Action 和 Result 的执行而执行，其工作方式如图 4.1 所示。

从图 4.1 可以看到，在 Action 和 Result 执行之前，为 Action 配置的拦截器将首先被执行，在 Action 和 Result 执行之后，拦截器将重新获得控制权，然后按照与先前调用相反的顺序依次执行。在整个执行过程中，任何一个拦截器都可以选择直接返回，从而终止余下的拦截器、Action 和 Result 的执行。例如，当一个未授权的用户访问受保护的资源时，执行身份验证的拦截器可以直接返回。

4.1.2　拦截器与过滤器的区别

拦截器与过滤器的区别具体如下：

- 拦截器不依赖于 Servlet 容器，而过滤器依赖于 Servlet 容器；

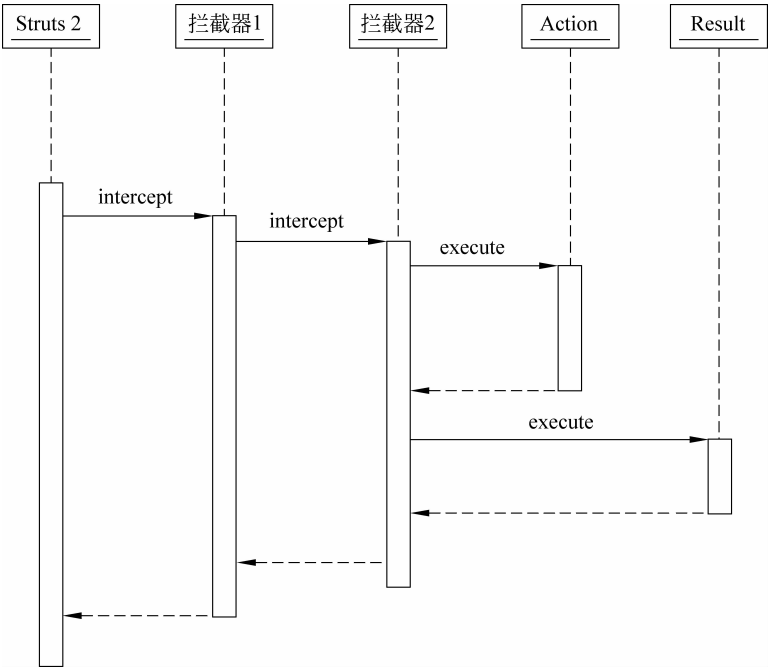

图 4.1　拦截器的工作方式

- 拦截器只能对 Action 请求起作用，而过滤器则可以对几乎所有的请求起作用；
- 拦截器可以访问 Action 上下文、值栈里的对象，而过滤器不能；
- 在 Action 的生命周期中，拦截器可以多次被调用，而过滤器只能在容器初始化时（如 Tomcat 服务器启动）被调用一次；
- 过滤器配置在 web.xml 文件中，而拦截器配置在 struts.xml 文件中。

4.1.3　Struts 2 内置的拦截器

Struts 2 框架利用大量内置的拦截器，完成了框架内的大部分操作，如请求参数解析、文件的上传和下载、国际化、类型转换器和数据校验等。这些内置拦截器以 name-class 对的形式配置在 struts-default.xml 文件中，其中 name 是拦截器的名字，就是以后使用该拦截器的唯一标识；class 则指定了该拦截器的实现类。在配置文件中，如果定义的 package 继承了 Struts 2 的 struts-default 包，则可以自由使用 Struts 2 内置的拦截器，否则必须自己定义这些拦截器。

下面是 Struts 2 内置拦截器的简单介绍。

- alias：实现在不同请求中相似参数别名的转换。
- autowiring：这是个自动装配的拦截器，主要用于当 Struts 2 和 Spring 整合时，Struts 2 可以使用自动装配的方式来访问 Spring 容器中的 Bean。
- chain：构建一个 Action 链，使当前 Action 可以访问前一个 Action 的属性，一般和 <result type="chain"…/>一起使用。
- conversionError：这是一个负责处理类型转换错误的拦截器，负责将类型转换错误从 ActionContext 中取出，并转换成 Action 的 FieldError 错误。

- createSession：该拦截器负责创建一个 HttpSession 对象，主要用于那些需要有 HttpSession 对象才能正常工作的拦截器中。
- debugging：当使用 Struts 2 的开发模式时，这个拦截器会提供更多的调试信息。
- execAndWait：后台执行 Action，负责将等待画面发送给用户。
- exception：该拦截器负责处理异常，它将异常映射为结果。
- fileUpload：这个拦截器主要用于文件上传，负责解析表单中文件域的内容。
- i18n：这是支持国际化的拦截器，负责把所选的语言、区域放入用户 Session 中。
- logger：这是一个负责日志记录的拦截器，主要是输出 Action 的名字。
- model-driven：这是一个用于模型驱动的拦截器，当某个 Action 类实现了 ModelDriven 接口时，负责把 getModel()方法的结果堆入 ValueStack 中。
- scoped-model-driven：如果一个 Action 实现了一个 ScopedModelDriven 接口，该拦截器负责从指定生存范围中找出指定的 Model，并将通过 setModel 方法将该 Model 传给 Action 实例。
- params：这是最基本的一个拦截器，负责解析 HTTP 请求中的参数，并将参数值设置成 Action 对应的属性值。
- prepare：如果 Action 实现了 Preparable 接口，将调用该拦截器的 prepare()方法。
- static-params：该拦截器负责将配置文件中＜action＞标签下＜param＞标签中的参数传入 Action。
- scope：这是范围转换拦截器，可以将 Action 状态信息保存到 HttpSession 范围，或保存到 ServletContext 范围内。
- servlet-config：如果某个 Action 需要直接访问 Servlet API，就是通过该拦截器实现的。
- timer：该拦截器负责输出 Action 的执行时间，在分析该 Action 的性能瓶颈时有用。
- token：该拦截器主要用于阻止重复提交，检查传到 Action 中的 token，从而防止多次提交。
- token-session：该拦截器的作用与前一个基本类似，只是它把 token 保存在 HttpSession 中。
- validation：通过执行在 xxxAction-validation.xml 中定义的校验器，从而完成数据校验。
- workflow：该拦截器负责调用 Action 类中的 validate 方法，如果校验失败，则返回 input 的逻辑视图。

一般情况下，开发者无须手动控制这些内置的拦截器，因为 struts-default.xml 文件中已经配置了这些拦截器，只需要定义的包继承 struts-default 包，就可以直接使用这些拦截器。

4.2 拦截器的配置

Struts 2 拦截器由 struts-default.xml、struts.xml 等配置文件进行管理，在用户开发自己的拦截器时，需要在 struts.xml 文件中进行配置，然后才能使用自己的拦截器。

4.2.1 配置拦截器

拦截器的配置是在 struts.xml 中完成的，定义一个拦截器使用＜interceptor…/＞元素，其格式如下：

```
<interceptor name="拦截器名" class="拦截器实现类"></interceptor>
```

一般情况下，上述这种格式就可完成拦截器的配置。有时，如果需要在配置拦截器时为其传入拦截器参数，只要在＜interceptor..＞与＜/interceptor＞之间配置＜param…/＞子标签即可传入相应的参数。其格式如下：

```
<interceptor name="拦截器名" class="拦截器实现类">
    <param name="参数名">参数值</param>
    ...<!--如果需要传入多个参数,可以一并设置-->
</interceptor>
```

在 struts.xml 中可以配置多个拦截器，它们被包在＜interceptors＞＜/interceptors＞之间。例如，下面的配置：

```
<?xml version="1.0" encoding="UTF-8" ?>
    <!DOCTYPE struts PUBLIC "-//Apache Software Foundation//DTD Struts
    Configuration 2.0//EN" "http://struts.apache.org/dtds/struts-2.1.dtd">
    <struts>
    <package name="default" extends="struts-default">
        <interceptors>
            <interceptor name="拦截器名 1" class="拦截器类 1"></interceptor>
            <interceptor name="拦截器名 2" class="拦截器类 2"></interceptor>
            …
            <interceptor name="拦截器名 n" class="拦截器类 n"></interceptor>
        </interceptors>
        ...<!--action 配置-->
    </package>
    </struts>
```

拦截器是配置在包下的。在包下配置了一系列的拦截器，但仅仅是配置在该包下，并没有得到应用。如果要应用这些拦截器，就需要在＜action＞配置中引用这些拦截器，格式如下：

```
<interceptor-ref name="拦截器名" ></interceptor-ref>
```

在＜action＞配置中引用拦截器示例如下：

```
<action name="Action 名" class="Action 类">
    <interceptor-ref name="拦截器 1"></interceptor-ref>
    <interceptor-ref name="拦截器 2"></interceptor-ref>
    <interceptor-ref name="defaultStack"></interceptor-ref>
</action>
```