Exploring Struts 2

Day 2 : Saturday
Created Date: 18/08/2012
Created By : Alok Kumar Vishwakarma (Java Trainee)
Mentor : Satheesh Balasubramanian

- Struts Configuration File (struts.xml)
- Web Application Deployment Descriptor file (web.xml)
- context.xml
- MANIFEST.MF

Struts Configuration File (struts.xml)

The core configuration of the struts 2 framework is the default struts.xml file and will be present in the classpath (WEB-INF/classes) or in Source Packages / <default-package> struts.xml initializes its own resources like

- **Action classes -** That can call business logic and data access code.
- **Interceptors-** That can preprocess and postprocess a request.
- **Results-** That can prepare views using JSP, Velocity or any other templates.

At runtime, there is a single configuration for an application. Prior to runtime, the configuration is defined through one or more XML documents, including the default *struts.xml* document. There are several elements that can be configured, including packages, namespaces, includes, actions, results, interceptors, and exceptions.

Structure of the struts.xml file

```
<?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-2.0.dtd">
<struts>
<action name="StrutsApp" class="com.sysbiz.HelloWorld">
<result>/pages/HelloWorld.jsp</result>
</action>
<!-- Add actions here -->
</package>
<!-- Add packages here -->
</struts>
```

Features of struts 2 configuration file:

- The struts.xml file allows to break big struts.xml file into small files and configuration files to be included as needed.
- **Example:**

<struts>

<include file= “file1.xml” />
<include file= “file2.xml” />

</struts>

- can even place struts-plugin.xml file in the JAR, and it will be automatically plugged into the application. This helps the programmers to develop self-configured components.
- If you want to use the frameworks such as Freemaker and Velocity modules, then the templates can also be loaded from classpath. This enables the developer to package entire module just in single JAR file.

Exploring struts.xml

Few important rules to keep in mind before configuring the struts.xml file.

- The XML declaration is case sensitive: it may not begin with “<?XML” or any other variant;
- If the XML declaration appears at all, it must be the very first thing in the XML document: not even whitespace or comments may appear before it; and
- it is legal for a transfer protocol like HTTP to override the encoding value that you put in the XML declaration, so you cannot guarantee that the document will actually use the encoding provided in the XML declaration.

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

This is the first line of code that struts.xml file contains. it indicates the minimum version of xml in use i.e. “1.0” and encoding type i.e. “UTF-8” which is supported by the XML Parsers.

UTF-8:

- UCS Transformation format -8 bit is a variable width encoding that can represent every character in the Unicode character set.
- UTF-8 encodes each of the **1,112,064 code points** in the Unicode character set

using one to four 8-bit bytes (termed "octets" in the Unicode Standard).

- It is the dominant encoding for the World Wide Web.

2. <!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">

The above DTD is the second line of the struts.xml file which is included while configuring the struts.xml. The struts.xml file must confirm to the Struts 2 Document Type Definition (DTD).The DTD provides information about the **structure and the elements** that the struts.xml file should have.

Struts 2 DTD :

```
<!--
  Struts configuration DTD.
  Use the following DOCTYPE

  <!DOCTYPE struts PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 2.0//EN"
      "http://struts.apache.org/dtds/struts-2.0.dtd">
-->

<!ELEMENT struts (package|include|bean|constant)*>

<!ELEMENT package (result-types?, interceptors?, default-interceptor-ref?,
default-action-ref?, default-class-ref?, global-results?, global-exception-mappings?,
action*)>
<!ATTLIST package
   name CDATA #REQUIRED
   extends CDATA #IMPLIED
   namespace CDATA #IMPLIED
   abstract CDATA #IMPLIED
   externalReferenceResolver NMTOKEN #IMPLIED
>

<!ELEMENT result-types (result-type+)>
```

```
<!ELEMENT result-type (param*)>
<!ATTLIST result-type
   name CDATA #REQUIRED
   class CDATA #REQUIRED
   default (true|false) "false"
>

<!ELEMENT interceptors (interceptor|interceptor-stack)+>

<!ELEMENT interceptor (param*)>
<!ATTLIST interceptor
   name CDATA #REQUIRED
   class CDATA #REQUIRED
>

<!ELEMENT interceptor-stack (interceptor-ref*)>
<!ATTLIST interceptor-stack
   name CDATA #REQUIRED
>

<!ELEMENT interceptor-ref (param*)>
<!ATTLIST interceptor-ref
   name CDATA #REQUIRED
>

<!ELEMENT default-interceptor-ref (param*)>
<!ATTLIST default-interceptor-ref
   name CDATA #REQUIRED
>

<!ELEMENT default-action-ref (param*)>
<!ATTLIST default-action-ref
   name CDATA #REQUIRED
>

<!ELEMENT default-class-ref (param*)>
<!ATTLIST default-class-ref
```

```
  class CDATA #REQUIRED
>

<!ELEMENT global-results (result+)>

<!ELEMENT global-exception-mappings (exception-mapping+)>

<!ELEMENT action (param|result|interceptor-ref|exception-mapping)*>
<!ATTLIST action
  name CDATA #REQUIRED
  class CDATA #IMPLIED
  method CDATA #IMPLIED
  converter CDATA #IMPLIED
>

<!ELEMENT param (#PCDATA)>
<!ATTLIST param
  name CDATA #REQUIRED
>

<!ELEMENT result (#PCDATA|param)*>
<!ATTLIST result
  name CDATA #IMPLIED
  type CDATA #IMPLIED
>

<!ELEMENT exception-mapping (#PCDATA|param)*>
<!ATTLIST exception-mapping
  name CDATA #IMPLIED
  exception CDATA #REQUIRED
  result CDATA #REQUIRED
>

<!ELEMENT include (#PCDATA)>
<!ATTLIST include
  file CDATA #REQUIRED
>
```

```
<!ELEMENT bean (#PCDATA)>
<!ATTLIST bean
   type CDATA #IMPLIED
   name CDATA #IMPLIED
   class CDATA #REQUIRED
   scope CDATA #IMPLIED
   static CDATA #IMPLIED
   optional CDATA #IMPLIED
>

<!ELEMENT constant (#PCDATA)>
<!ATTLIST constant
   name CDATA #REQUIRED
   value CDATA #REQUIRED
>
```

3. The **<struts> tag** is the root tag for the **struts.xml.** It may contain the following tags : package, include, bean and constant.

4. The Package Tag :

Packages are a way to group actions, results, result types, interceptors, and interceptor-stacks into a logical configuration unit. Conceptually, packages are similar to objects in that they can be extended and have individual parts that can be overridden by "sub" packages.

The **<package />** tag is used to group together configurations that share common attributes such as interceptor stacks or URL namespaces. It may also be useful to organizationally separate functions, which may be further separated into different configuration files.

The package element has one required attribute, **name**, which acts as the key for later reference to the package. The **extends** attribute is optional and allows one package to inherit the configuration of one or more previous packages - including all interceptor, interceptor-stack, and action configurations.

The optional **abstract** attribute creates a base package that can omit the action configuration.

Attribute	Required	Description
name	**yes**	key for other packages to reference
extends	no	inherits package behavior of the package it extends
namespace	no	provides a mapping from the URL to the package.
abstract	no	declares package to be abstract (no action configurations required in package)

a)**. name** ? unique name is given for a package.

b). **extends** ? the name of a package that this package will extend; all configuration information (including action configurations) from the extended package will be available in the new package, under the new namespace.

c). **namespace** ? the namespace provides a mapping from the URL to the package. i.e. for two different packages, with namespace attributes defined as ?pack1? and ?pack2?, the URLs would be something like ?/webApp/pack1/my.action? and ?/webApp/pack2/my.action?

d). **abstract** ? if this attribute value is ?true? the package is truly a configuration grouping and actions configured will not be accessible via the package name. It is important to make sure you are extending the correct parent package so that the necessary pre-configured features will be available to you.

5. The Include Tag:

The <include /> tag is used to modularize a Struts2 application that needs to include other configuration files. It contains only one attribute ?file? that provides the name of the xml file to be included. This file has exactly the same structure as the ?struts.xml? configuration file. For example, to break a configuration file of a finance application, you might choose to group together the invoices, admin, report configurations etc into separate files:

```
<struts>
<include file="invoices-config.xml" />
<include file="admin-config.xml" />
<include file="reports-config.xml" />
</struts>
```

While including files, order is important. The information from the included file will be available from the point that the include tag is placed in the file.

There are some files that are included implicitly. These are the ?**strutsdefault.xml**? and the ?**struts-plugin.xml**? files. Both contains default configurations for result types, interceptors, interceptor stacks, packages as well as configuration information for the web application execution environment (which can also configured in the ?struts.properties? file). The difference is that ?struts-default.xml? provides the core configuration for Struts2, where ?struts-plugin.xml? provides configurations for a particular plug-in. Each plug-in JAR file should contain a ?struts-plugin.xml? file, all of which are loaded during startup.

6. The Bean Tag:

Most applications won't need to extend the Bean Configuration. The bean element requires the classattribute which specifies the Java class to be created or manipulated. A bean can either

1. be created by the framework's container and injected into internal framework objects, or
2. have values injected to its static methods

The first use, object injection, is generally accompanied by the type attribute, which tells the container that which interface this object implements.

The second use, value injection, is good for allowing objects not created by the container to receive framework constants. Objects using value inject must define the the static attribute.

7. The Constant Tag:

There are two key roles for constants.

1. They are used to override settings like the maximum file upload size or whether the Struts framework should be in devMode(= development mode) or not.

2. They specify which Bean should be chosen, among multiple implementations of a given type.

Constants can be declared in multiple files. By default, constants are searched for in the following order, allowing for subsequent files to override by the previous ones:

- struts-default.xml
- struts-plugin.xml
- struts.xml
- struts.properties
- web.xml

The struts.properties file is provided for backward-compatiblity with WebWork. In the struts.properties file, each entry is treated as a constant. In the web.xml file, any FilterDispatcher initialization parameters are loaded as constants.

Web Application Deployment Descriptor File (web.xml)

The web.xml configuration file is a **J2EE configuration file** that determines how elements of the HTTP request are processed by the servlet container. It is not strictly a Struts 2 configuration file, but it is a file that needs to be configured for Struts2 to work.
The **web.xml** web application descriptor file represents the core of the Java web application, so it is appropriate that it is also part of the core of the Struts framework. In the web.xml file, Struts defines its **FilterDispatcher,** the Servlet Filter class that initializes the Struts framework and handles all requests. This filter can contain initialization parameters that affect what, if any, additional configuration files are loaded and how the framework should behave.
In addition to the FilterDispatcher, Struts also provides an **ActionContextCleanUp** class that handles special cleanup tasks when other filters, such as those used by Sitemesh, need access to an initialized Struts framework.

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <filter>
    <filter-name>struts2</filter-name>
    <filter-class>org.apache.struts2.dispatcher.FilterDispatcher</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>struts2</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
  <session-config>
    <session-timeout>
      30
    </session-timeout>
  </session-config>
</web-app>
```

this file provides an entry point for any web application. The entry point of Struts2 application will be a filter defined in deployment descriptor (web.xml). Hence we will define an entry of *FilterDispatcher* class in web.xml. The web.xml file needs to be created under the folder **WebContent/WEB-INF**.

Note that we map the Struts 2 filter to ***/****, and not to ***/*.action** which means that all urls will be parsed by the struts filter.

context.xml

The context.xml file contains the default Context element used for all web applications in the system. The supported attributes include:

- **cookies**: This is a flag indicating if sessions will be tracked using cookies. The default is true.
- **crossContext**: This is a flag indicating whether the ServletContext.getContext(String path) method should return contexts for other web applications deployed in the calling web application's virtual host.

Its present in the location **/META-INF** in the all struts 2 web applications.

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

First line displays the XML version and encoding mechanism. It follows all all similar rules as struts.xml.

2. <Context antiJARLocking="true" path="/sample"/>

If true, the Tomcat classloader will take extra measures to avoid JAR file locking when resources are accessed inside JARs through URLs. This will impact startup time of applications, but could prove to be useful on platforms or configurations where file locking can occur.

If not specified, the **default value is false**. antiJARLocking is a subset of antiResourceLocking and therefore, to prevent duplicate work and possible issues, only one of these attributes should be set to true at any one time.

MANIFEST.MF

When you create a JAR file, it automatically receives a default manifest file. There can be only one manifest file in an archive, and it always has the pathname
META-INF/MANIFEST.MF

When a JAR file is created with version 1.2 of the Java Development Kit, the default manifest file is very simple. Here are its full contents:
Manifest-Version: 1.0

This line shows that a manifest's entries take the form of "header: value" pairs. The name of a header is separated from its value by a colon. The default manifest shows that it conforms to version 1.0 of the manifest specification.
The manifest can also contain information about the other files that are packaged in the archive. Exactly what file information is recorded in the manifest will depend on what use you intend for the JAR file. The default manifest file makes no assumptions about what information it should record about other files, so its single line contains data only about itself.