Servlet只能对应用程序的动态部分进行控制，比如Struts中的Action。静态资源（图片、html文件、js、css等）交由Web容器处理。Filter则处理所有请求。当然，一般静态资源是不需要程序处理的。

可以利用web.xml描述符文件对JSP文件进行保护，使之不能被直接访问。（WebLogic不支持放在WEB-INF下的保护方式）。如下：

<security-constraint>

<web-resource-collection>

<web-resource-name>JSPs</web-resource-name>

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

</web-resource-collection>

<auth-constraint/>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method>

</login-config>

请查阅资料：web.xml配置详解。

Struts2使用FilterDispatcher（org.apache.struts2.dispatcherFilter.Dispatcher）作为调度控制器（中央分发器）。具体的动作交给拦截器（interceptor）执行。Struts2默认的拦截器非常多。

**配置：**

Struts2的配置文件：（xml为主，properties为辅）

系统级：struts.xml，struts.properties

插件：struts-plugin.xml，每个插件自带

用户自定义：

拦截器配置相关：

interceptors包含一个interceptor集合，package内所有action都可引用。

action使用interceptor-ref引用某个interceptor或者interceptor-stack。

interceptor-stack包含一个常用interceptor的集合。供action使用。

default-interceptor-ref，action默认引用，可覆盖。

constant元素：

覆盖struts.properties中的配置，可另写一个properties文件，也可在struts.xml中定义constant元素。如<constant name=”struts.devMode” value=”true” />

action元素：

如果没有为action配置class，则会定位到ActionSupport类。这种配置仅仅是为了方便跳转到某个JSP。struts2使用setter方法为action的属性设值，OGNL使用getter方法取属性值。因此，action的属性一定要有相应的getter/setter方法。

注：配置文件都应该放在类文件路径下，web应用程序一般是WEB-INF/classes目录。

**Action**：

默认情况下，struts2不创建HttpSession。不过JSP仍会创建，要想创建一个无Session的JSP，可以在其头部添加<%page session=”false” %>。

想要在Action中访问Servlet相关对象，可以实现相对应的Aware接口，比如ServletContextAware。SessionAware有点不一样，它返回的不是原生的Session对象，而是一个Map。当然，为了能够自动填充这些对象，还需要Servlet Config拦截器，这个拦截器默认已配置。

除了上面提到的set方法为Action的属性设值外，还可以在action的配置中使用param元素为属性设值。

虽然Action接口足够简单，但为了方便使用，一般都继承ActionSupport类。该类提供了很多实用方法和默认实现。

ResultType（只摘录几个比较重要且应该区分的）：

chain，构建一条动作链；后面的action包含前面action的状态

redirect，重定向到另一个url；状态不能共享，客户端重新发起一个请求

redirect-action，重定向到另一个action；

redirect和redirect-action的区别：前者可以重定向到一个资源，后者只能是action。前者要传递参数，只能在url里设置，后者有专门的配置元素来设置（避免了转码）。

异常处理：

exception-mapping和global-exception-mapping。前者包含在action内部，后者全局使用。每当捕获到一个异常，都会在ValueStack中添加两个对象：exception和exceptionStack，前者是Exception对象，后者是异常堆栈。

通配符：

“\*”不匹配“/”，如果想将“/”包含在内，请使用“\*\*”，转义字符请使用“\”。“{1}、{2}”分别匹配第一个、第二个匹配子串，“{0}”包含整个URI。

谨慎使用通配符！

动态方法调用：

actionName!method.action的方式。都说该方式可能引起安全问题，值得讨论。

**OGNL**：

ValueStack：包含Object Stack和Context Map两部分。Object Stack包含Action和相关对象，Context Map包含各种key/value对象。

Context Map：

parameters，当前请求的请求参数（地址栏）；

request，session，application（ServletContext），

attr，按request---session---application顺序来检索属性。

访问Context Map中的对象时，以#开头；不以#开头，访问Object Stack。

访问方法：

静态字段、静态方法：@类名@静态字段名/方法名(args)

实例方法：object.fieldName/methodName(args)

数组：返回逗号分隔列表，如想访问数组中某一元素，arr[下标]

List：同数组，不过带有方括号[ ]，创建List使用{元素1，元素2…}

Map：{key:value}，创建Map：#{key:value}

OGNL和EL配合使用。

**标签**：

引入标签 <%@ taglib prefix=”xx” uri=”path” %>

标签属性引用OGNL表达式：如 label=”%{labelName}”

特别需要注意checkbox的使用。

list、listKey、listValue：用于radio、combobox、select、checklist、doubleselect等。这三个属性中，list必不可少，另两个可选。可以将String、数组、Collection、Map、Enumeration、Iterator赋给list。

<s:select list=”#{‘1’:’China’,’2’:’Japan’}” /> 生成：

<select>

<option value=”1”>China</option>

<option value=”2”>Japan</option>

</select>

listKey、listValue一般用于Map。

#、$和%：

#：一般用于三个地方：访问Context Map；投影；构造Map

%：使用OGNL计算字符串表达式的值

$：用于配置文件，JSP中不用。可用于EL和JSTL。

主题（theme）：生成的html代码的排版风格

simple：最简单方式，xhtml：默认主题，添加table之类的；css\_xhtml，和xhtml类似，但是使用css来排版；ajax：以xhtml为基础，增加一些ajax功能。

iterator标签：重点status属性。另需注意，对于循环变量，每次都push和pop到ValueStack中，从而在循环过程中不用显式写明循环变量。

**类型转换：**

Http中没有类型，只有String；Java中有各种对象。需要一种工具去转化。在struts2中，这项工作由Parameters拦截器负责（该拦截器在defaultStack中，不需额外配置），该拦截器通过调用各种不同的类型转换器来达成这个目标，而这些转换器都使用了OgnlRuntime。

当类型转换失败时，如果Action实现了ValidationAware接口且Results中包含input结果，则会跳转到该结果，否则抛出异常；如果没有实现该接口，则忽略，继续执行。

如果抛出异常，则转换的异常信息会由Conversion Error拦截器处理。

自定义类型转换器：

实现ognl.TypeConverter接口，不过一般扩展DefaultTypeConverter或者StrutsTypeConverter。

配置：基于字段，需要在Action类所在目录下建立Action-Conversion.properties文件，内容 field1 = customerConverter1；基于类，在类路径下建立conversion.properties文件，类容 全限定类名 = customerConverter1全限定类名。

**输入验证：**

struts2的输入验证做得不好，其实基于字段的类型转换做得也不好（配置散落）。最好能参考Hibernate Validation（基于JSR303）。

**消息处理和国际化：**

内建支持，ActionSupport中的getText()方法，该方法由TextProvider接口提供。TextProvider处理Message和ResourceBundle。

text标签和i18n标签。

**Model Driven 和 Preparable拦截器：**