1. 使用Windows系统的主机不区分URL的大小写，而使用Unix或Linux系统的主机则区分大小写。
2. TOMCAT配置：如果没有自动导入tomcat的jar，是新建工程的时候没有在Target runtime中选定默认的运行服务器。
3. TOMCAT可以不配置环境变量。
4. 在tomcat的service.xml中设置：reloadable=“true”来得到每次运行时
5. 解决tomcat启动问题：
   * + 1. SetPropertiesRule警告：解决方法是在Server的配置界面中选中"Publish module contexts to separate XML files"选项。
       2. 如果发生无法在Tomcat中启动一个web项目，可能的情况是工作空间发生问题，创建其他工作空间再次部署
6. 警告问题 ：multiple markers at this line：

Eclipse 默认把这些受访问限制的API设成了ERROR。只要把Windows-Preferences-Java-Complicer- Errors/Warnings里面的Deprecated and restricted API中的Forbidden references(access rules)选为Warning就可以编译通过。

1. 警告问题
2. 一个JSP页面分为HTML页面内容；JSP注释；JSP指令；JSP脚本元素；JSP动作元素
3. HTML语言中的元素都可以被JSP容器所解析。
4. 实际上，JSP页面最终会编译成Servlet类，而容器中只会存在一个Servlet类的实例。在JSP中声明的变量是成员变量，它只在创建Servlet实例时被初始化一次。直到该Servlet实例被销毁。
5. JSP汉字编码问题，charset=UTF-8" pageEncoding="UTF-8"
6. 解决乱码<http://www.blogjava.net/tbwshc/archive/2012/06/18/381005.html>
7. http协议的特点是每次连接只完成一个请求:建立连接、发送请求、提供服务、发送相应、关闭连接。
8. servlet是运行在服务器端的java程序，可以生成动态的web页面。
9. servlet生命周期：

1）加载和实例化；容器通过Java的反射机制来创建Servlet的实例

2）初始化（init()）；容器调用Servlet的init方法来初始化该Servlet对象，对于每个Servlet实例，只会调用一次init方法

3）执行（service()方法、doGet()、doPost()）；当客户端的请求来到后，Servlet容器首先创建ServletRequest和ServletResponse两个对象，然后Servlet容器会自动调用service()方法来响应客户端请求，同时把ServletRequest和ServletResponse两个对象传递给service()方法。通过ServletRequest对象，Servlet实例可以获得客户端的请求信息，处理完成后，则将响应信息放置在ServletResponse对象中

4）清理（destroy()）

1. servlet需要在web.xml中注册(主要是需要url)，或者在@webservlet（”/”）中写明
2. 有两格Servlet类：GenericServlet抽象类：实现service()方法

HttpServlet抽象类：不要重新service方法

实现doPost(HttpServletRequest request, HttpServletResponse response) throw …处理和响应Http Get请求

(是一个JSPform提交信息，然后显示处理结果)

实现doGet(HttpServletRequest request, HttpServletResponse response) throw …处理和响应Http post请求

(是一个JSP，可以直接显示内容)

1. javabean是用java语言描述的软件组件模型。
2. 在JSP中调用JavaBean，JSP提供了3个标准的动作指令：
   1. <jsp:useBean>
   2. <jsp:getProperty>
   3. <jsp:setProperty>
3. 关于Spring的依赖注入（DI）和控制反转（IOC）

依赖注入和控制反转本质是一个概念。在传统的程序设计中，通常由调用者来创建被调用者的实例，而在依赖注入和控制反转的定义中，调用者不负责被调用者的实例创建工作，该工作有Spring框架中的容器来负责，它通过开发者的配置来判断实例的类型，创建后再注入调用者。

1. Spring框架需要自己的jar包。自己制作类库：导入自己的Lib:buildPath>userlibrary
2. 注意，使用Junit时一定要添加common-logging的jar包
3. Spring使用javabean来配置应用程序。Javabean指的是类中有getter和setter方法的java类；

17.1 有property标签的bean其实现类必须包括property中name属性的setter和getter方法。

17.2 一般为属性添加Setter方法

1. 在Spring中，类的实例并不是main方法来创建的，而是通过将xml文件加载后，交付给Spring框架，当执行getBean方法时，就可以得到要创建的实例的引用。因此，调用者并没有创建实例，而是通过Spring框架将创建好的实例注入调用者，这就是依赖注入。而Spring的强大之处在于它可以使用依赖注入将一个Bean注入到另一个Bean中。
2. IoC容器为管理对象之间的依赖关系提供了基础功能。Spring提供了两种容器
   1. BeanFactory：Boolean containsBean(String);

Object getBean(String);

Class getType(String)

* 1. ApplicationContext:在BeanFactory基础上构建的。是BeanFactory的子接口，也称为应用上下文，提供了更多的功能：

AppCon接口的实现类有3个：

ClassPathXmlApplicationContext:从类加载路径下的XML文件中获取上下文定义信息，创建ApplicationContext。

FileSystemXmlApplicationContext:从文件系统中的XML文件中获取上下文定义信息，创建ApplicationContext

XmlWebApplicationContext:从Web系统中的XML文件中获取上下文定义信息，创建ApplicationContext

1. Bean的几个属性：

id: Bean的唯一标识名。它必须是合法的XML ID，在整个XML文档中唯一。

name: 用来为id创建一个或多个别名。它可以是任意的字母符合。多个别名之间用逗号或空格分开。

class: 用来定义类的全限定名（包名＋类名）。只有子类Bean不用定义该属性。

1. 工厂模式：工厂负责制造汽车，调用者需要汽车时，只要找到工厂，向工厂请求一个对象即可。（工厂模式专门负责对大量有公共接口的类进行实例化）
2. 在Spring DI 模式下，所有的对象都存放在容器中，调用者直接拿对象而不用管怎么生成。
3. 依赖注入的三种实现方式：
   1. 设值注入：IoC容器利用setter方法来注入被依赖的实例
   2. 构造注入：IoC容器利用构造方法来注入被依赖的实例
   3. 接口注入
4. 在数据库的操作上新建一个接口。通过接口的实现类来实现对数据库的操作。针对接口编程的好处是接口可以有多个实现类，这些类可以分别操作Oracle、DB2、MySQL  
   等不同的数据库，当数据库发生变化时，只需要调整实现类即可。
5. 在使用ssh开发时，一般将项目分为三层，web层、service层、dao层。开发流程基本上是先定义dao接口，然后实现该接口，再定义同类型的service接口，再实现service接口（使用dao接口注入），接着再从web层去调用service层。
6. Dao完成的是底层的数据操作，service层完成纯粹的业务逻辑，该层的数据操作部分是通过注入dao来实现的。Service层的作用是将从dao层取到的数据做更贴近业务的实现，dao层只实现对数据的增、删、改、查
7. Java反射是java语言特有的，指能够进行自描述和自控制的一类应用。

API: java.lang.reflect包中：Constructor（描述类的构造方法）

Field（描述类的成员变量）；Method（描述类的方法）

1. 反射，首先利用getClass()获得类的实例
2. 利用JDOM解析xml，需要JDOM的jar包。
3. JSP特点
   1. 能够在任何Web或者应用程序服务器上运行
   2. 将逻辑和页面显示分离：web开发人员使用HTML来设计界面，使用JSP标签和脚本来生成页面上的动态内容。在服务器端，JSP引擎负责解释JSP标签和脚本程序生成所请求的内容，将结果以HTML的形式返回到浏览器。
4. JSP基本语法：
   1. JSP注释：<!-- -->//html

<%-- --%>//jsp:页面不会显示注释（也可以使用java注释//）

* 1. JSP指令：
     1. Page指令：<%@ page 属性1 = “属性值1” 属性2 = “属性值2”….%>
     2. Include指令：<%@ include file = “被包含文件的地址”%>
     3. Taglib指令：<%@ taglib url = “tagLibraryURI” prefix = “tagPrefix”%>
  2. JSP脚本元素
     1. JSP声明语句 <%！变量或方法定义 %>
     2. JSP表达式 <%= %>
     3. JSP Scriptlets <% Java代码 %> 输出:out.println();
  3. JSP动作元素 控制JSP行为：如动态插入文件、重用JavaBean、自定义标签等
     1. <jsp:include> page=“relative URL”
     2. <jsp:forward> 页面重定向：转发
     3. <jsp:param> 提供参数<jsp:param name=” ” value= “ ”>
     4. <jsp:plugin>

//和JavaBean链接（可以在JSP中封装Java代码并从内容中分离出逻辑表达）

* + 1. <jsp:useBean> 用来装载一个将在JSP中使用的JavaBean

<jsp:useBean id=”beanInstanceName” class=”classname” scope=”page|request|session|application” />

* + 1. <jsp:setProperty> 设置和修改Bean中的属性值

<jsp:setProperty name=”beanInstanceName” property=”propertyName” value=””/>

* + 1. <jsp:getProperty> 获取Bean属性的值，并将其转换为字符串输出

<jsp:getProperty name=”beanInstanceName” property=”propertyName”/>

1. JSP内置对象：
   1. Request 用于获取客户端信息，实际上，JSP会将客户端的请求信息封装在request对象中。getParameter(String name):获取参数名为name的参数值

setCharacterEncoding(String name)

getParameterNames()

getAttribute(String name)获取指定的属性值

setAttribute(String name, object value)

* 1. Response 包含了从JSP页面返回客户端的所有信息，作用域是他所在页面，经常用于设置Http标题、添加cookie、设置响应内容类型和状态、发送Http重定向和编码URL

sendRedirect(“.jsp”)

* 1. pageContext
  2. session 会话对象，用来记录每个客户端的访问状态

boolean isNew();

getAttribute(String name)获取session范围内name属性的值

setAttribute(String name, Object value)设置

getId()

* 1. application 用于获取和设置Servlet的相关信息，他的生命知道服务器关闭

setAttribute(String name, Object value);

getAttribute(String name)

* 1. out 缓冲输出流

clear()清除缓冲区

getBufferSize()获取缓冲区大小

print(String str)

println(String str)

* 1. config 是ServletConfig类的一个实例，在Servlet初始化的时候，可以通过config向servlet传递信息。所传递的信息可以是属性名和属性值构成的名值对，也可以是通过ServletContext对象传递的服务器相关信息。使用的不多，只有在编写Servlet时需要重载Servlet的init()方法时，才会用到config对象
  2. page
  3. exception 处理页面错误和异常。常用方法是利用page指令的errorPage属性，然后将exception对象传送到page指令中设置的异常处理页面中。

1. JSP对象的声明：通过观察代码后我们发现，原先声明在<%! %>内的变量和方法是一个类内的变量和方法也就是成员变量和成员方法。声明在<% %>内的变量是一个方法的变量也就是局部变量。我们学习JAVA的时候就知道成员变量会被多个对象共享，而局部变量单独对象访问。所以count会自动增加，i不对递增。在这里还有个问题要告诉读者，如果在<% %>内添加一个method()方法那么该程序就会报错，其实原因不难想象，这样声明一个方法等于就是一个类里的方法里又声明了个方法结果肯定是错误的。
2. 页面处理中文请求参数时，可以先将获取的中文参数转换为字节，并保存在字节数组中，

接着将字节数组重新组合成字符串

Byte b[] = str.getBytes(“ISO-8859-1”);

Str = new String(b);

1. 设置javabean属性的4种方式
2. 在ServletResponse接口中提供两个输出流:PrintWriter用于输出字符串数据；ServletOutputStream用于输出字节数据
3. JavaWeb是基于B/S结构的应用，浏览器功能非常单一，只能解析HTML、CSS、JS等，不能解析JavaWeb，如jsp，所以需要把Web应用程序部署到Web应用服务器，由服务器来解析处理
4. 在StrutsDemo工程中，用户单击index.jsp链接，发送http请求，服务器端接受到http请求后，调用web.xml文件中的过滤器的具体方法，通过一系列的内部处理机制，它判断出这个http请求和TestAction类所对应的Action对象相匹配，最后调用TestAction对象中的execute()方法，处理后返回相应的值SUCCESS，然后Struts2通过这个值可查找到success.jsp返回给浏览器。
5. [Servlet3.0规范中的@WebInitParam](mailto:Servlet3.0规范中的@WebInitParam)

之前2.5：

<servlet>

<servlet-name>roseindiaServlet</servlet-name>

<servlet-class>net.roseindia.RoseindiaServlet</servlet-class>

<init-param>

<param-name>myParam</param-name>

<param-value>paramValue</param-value>

</init-param>

</servlet>

现在：

@WebServlet(value="/user",

initParams = {

@WebInitParam(name="rose", value="Hello "),

@WebInitParam(name="india", value=" World!")

})

42 WEB-INF 中的文件是对外不可见的

43 重定向和转发有一个重要的不同：当使用转发时，JSP容器将使用一个内部的方法来调用目标页面，新的页面继续处理同一个请求，而浏览器将不会知道这个过程。 与之相反，重定向方式的含义是第一个页面通知浏览器发送一个新的页面请求。因为，当你使用重定向时，浏览器中所显示的URL会变成新页面的URL, 而当使用转发时，该URL会保持不变。重定向的速度比转发慢，因为浏览器还得发出一个新的请求。同时，由于重定向方式产生了一个新的请求，所以经过一次重定向后，request内的对象将无法使用。

怎么选择是重定向还是转发呢？通常情况下转发更快，而且能保持request内的对象，所以他是第一选择。但是由于在转发之后，浏览器中URL仍然指向开始页面，此时如果重载当前页面，开始页面将会被重新调用。如果你不想看到这样的情况，则选择转发。 转发和重定向的区别 不要仅仅为了把变量传到下一个页面而使用session作用域，那会无故增大变量的作用域，转发也许可以帮助你解决这个问题。重定向：以前的request中存放的变量全部失效搜索，并进入一个新的request作用域。转发：以前的request中存放的变量不会失效，就像把两个页面拼到了一起。

44 在该JSP被翻译成Servlet源代码的过程中,当容器遇到标准动作元素时,就调用与之对应的Servlet的方法来替代它,所有标准动作元素的前面都有一个JSP前缀作为标记,一般形式如下:

<jsp:标签名 ...属性.../>

45 框架 WEBWORK：（WebWork2.1.7+Hibernate4.2.3）

WebWork2建立在Xwork之上，处理HTTP的请求和响应。所有的请求都会被它的前端控制器（ServletDispatcher，最新版本是FilterDispatcher）截获。前端控制器对请求的数据进行包装，初始化上下文数据，根据配置文件查找请求URL对应的Action类，执行Action，将执行结果转发到相应的展现页面。WebWork2支持多视图表示，视图部分可以使用JSP, Velocity, FreeMarker, JasperReports，XML等。

WebWork2使用ServletDispatcher（派遣器）将HTTP请求的变成Action(业务层Action类), session（会话）application（应用程序）范围的映射，request请求参数映射。

Model是所有的实体类；

Action是对某一个实体数据进行维护，最基本至少需要增加、删除、修改、查询单个对象和查询列表等五个Action；

Service提供静态接口即可，主要调用数据访问层完成数据库操作并实现业务逻辑，同时提供Action调用。

提出基于三层框架（WEB层、业务层、数据层）的WEB层需要解决的十个问题。

1. 数据的输入：如何获得基于无状态的HTTP的请求数据？如何将请求的字符数据转换为对应的模型对象？
2. 数据的验证：如何验证数据的合法性并给出明确的错误消息提示？
3. 数据的输出：如何展现复杂的对象结构？如何处理复杂的展现逻辑？
4. 数据的传递和共享：如何在不同的请求或页面之间传递和共享数据？
5. 页面的流程管理：如何管理WEB应用中的页面流程？
6. 模块化管理：如何将复杂的WEB应用以模块化的方式管理？
7. 灵活可扩展的架构：如何支持各种不同的展现层技术？如何与业务层或数据层的各种框架进行整合？
8. 安全和访问控制管理：如何提高基于WEB的安全机制和资源访问控制管理？
9. 代码实现的简洁和高效：如何让开发步骤和代码维护变得简单？如何尽量减少开发的中间环节？如何将公共的功能剥离出来，并可以灵活的组装应用？

10、其他问题：异步调用、国际化支持、文件上传、防止重复提交等等。

WebWork的三个关键部分：

1. Action：一般一个Action代表一次请求或调用。在WebWork中，一般Action类需要实现Action接口，或者直接继承基础类ActionSupport。这时，它要实现默认的execute方法，并返回一个在配置文件中定义的Result（也就是一个自定义的字符串而已）。当然，Action也可以只是一个POJO（普通Java对象），不用继承任何类也不用实现任何接口。Action是一次请求的控制器，同时也充当数据模型的角色，我们强烈建议不要将业务逻辑放在Action中。
2. Results：它是一个结果页面的定义。它用来指示Action执行之后，如何显示执行的结果。Result Type表示如何以及用哪种视图技术展现结果。通过Result Type，WebWork可以方便的支持多种视图技术；而且这些视图技术可以互相切换，Action部分不需做任何改动。
3. Interceptors：WebWork的拦截器，WebWork截获Action请求，在Action执行之前或之后调用拦截器方法。这样，可以用插拔的方式将功能注入到Action中。WebWork框架的很多功能都是以拦截器的形式提供出来。例如：参数组装，验证，国际化，文件上传等等。

例子：

1. 输入页面。Welcome.jsp
2. WebWork的Action类。它实现了Action接口，获得welcome.jsp页面传入的数据即输入的用户名，根据这个用户名生成了一条欢迎信息，并在结果页面中打印出来。
3. 结果页面。welcomeResult.jsp页面，将通过表达式语言，取得Action执行之后暴露出来的数据对象。
4. 定义配置xwork.xml(在系统中，每个模块都有特别多的action，如何处理，（全部配置到xwork.xml?），就在该模块中，如何处理识别)
5. 为什么有的页面需要jsp，有的却是resultjson.jsp

< xwork >

< include file = "webwork-default.xml" />

< package name = "default" extends = "webwork-default" >

< action name = "welcome" class = "com.noname.web.action.WelcomeAction" >

< result name = "success" type = "dispatcher" > /welcomeResult.jsp </ result >

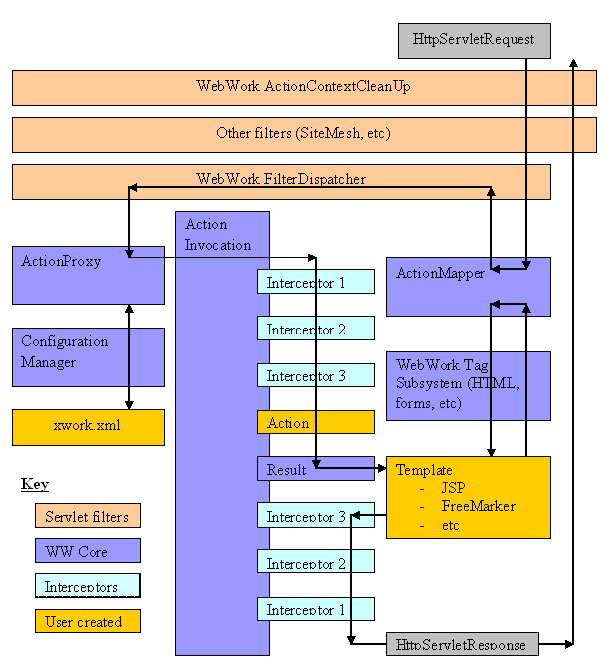
< interceptor-ref name = "params" />

</ action >

</package>

</xwork>

WebWork框架图：



44. textarea不顶个表示，编辑方式有问题：

<textarea>

内容内容

</textarea> 不行，要写成一行

1. Script 脚本可位于 HTML 的 <body> 或 <head> 部分中，或者同时存在于两个部分中。通常的做法是把函数放入 <head> 部分中，或者放在页面底部。这样就可以把它们安置到同一处位置，不会干扰页面的内容。

也可以把脚本保存到外部文件中。外部文件通常包含被多个网页使用的代码。外部 JavaScript 文件的文件扩展名是 .js。<script src="myScript.js"></script>

外部脚本不能包含 <script> 标签，直接写function即可

1. Document.write会覆盖所有的HTML
2. JavaScript 对大小写是敏感的。
3. JavaScript 只有一种数字类型。数字可以带小数点，也可以不带; 极大或极小的数字可以通过科学（指数）计数法来书写：var y=123e5; // 12300000 var z=123e-5; // 0.00123
4. JS的数据类型：数字(无区别)、字符串、boolean、数组(var cars=new Array();)、

对象(var person={firstname:"Bill", lastname:"Gates", id:5566};)、null和undefined

1. ModelDriven：和属性驱动的Action有很大的区别，下面一一列举：

属性驱动和模型驱动，前一个是传递属性，后一个是传递对象

1模型驱动的Action必须实现ModelDriven接口，而且要提供相应的泛型，这里当然就是具体使用的Java Bean了。

2实现ModelDriven的getModel方法，其实就是简单的返回泛型的一个对象。

3在Action提供一个泛型的私有对象，这里就是定义一个User的user对象，并提供相应的getter与setter。

好了，上面的三件事做完之后，Action就会去自动调用User的setter将表单中的name属性的值赋给User中的属性。而Action的后续处理的Jsp页面后者是Servlet就可以使用user对象了。

1. 对于Struts2如果项目没有用到Plaxus、Sitegraph、Spring，需要将struts2-plexus-plugin.2.0.11.jar、struts2- Sitegraph -plugin.2.0.11.jar、struts2- jsf -plugin.2.0.11.jar和struts2- Spring-plugin.2.0.11.jar删掉
2. Struts隐藏了常规的Servlet API如request和response等操作
3. 关于memcached导致无法启动：服务有冲突，如果在同一台服务器上有两个项目同时利用的memcached会导致此问题
4. Struts2的工作流程：

Struts放弃了request、response等servlet API。在调用Action的执行方法(execute等)前，struts2会从request中获取参数，并通过setter方法设置到Action中

访问JSP页面——>提交表单后数据提交给action——>struts2截获所有的请求，包括\*.action的请求——>查找struts2.xml，action所对应的类——>生成该类的一个实例，将提交的数据设置到该实例中——>调用实例中的execute()方法——>跳转到JSP中显示结果，丢弃该实例

1. Struts1中所有的action只有一个实例，该action会被反复使用。而struts2中每个Action会有多个实例，处理一次请求生成一个实例，这些实例彼此独立，处理完即销毁。因此struts2的action是线程安全的，而struts1线程不安全。
2. 程序入口FilterDispatcher
3. Struts2的配置文件：web.xml：配置Struts2的分发器Filter、struts.xml：配置Struts2的Action、struts.properties：配置一些Struts2的一些属性，例如Struts2的后缀、上传文件的大小、上传文件夹等。
4. Action的可执行方法
   1. Dd通过URL执行Action中的非默认方法：Action!method.action Action为struts.xml中配置的Action名字。
   2. 把method配置到Action中（问题：可能同一个Action要被配置多次）

<action name="login" class="com.bit.oa.login.action.LoginAction" method=“login”>

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

<result name="login">/login.jsp</result>

</action>

<action name="logou" class="com.bit.oa.login.action.LoginAction" method=“login”>

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

<result name="logout">/logout.jsp</result>

</action>

* 1. 使用通配符

<action name="log\*" class="com.bit.oa.login.action.LoginAction" method=“{1}”>

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

<result name="{1}">/{1}.jsp</result>

</action>

1. 注意:对于POJO或需要设置值的对象类来说，一定要有set和get方法，如果没有会报错，可能为无result设置等错误
2. Struts2数据类型转换器：JSP提交的数据全部都是String类型的，Struts2也是通过request.getParameter(name)获取到String类型数据，并通过拦截器将各种常用的数据类型，如Date、Time、Timestamp等，然后通过getter、setter方法设置到Action对应的属性上。

有时候struts2默认的转换器并不能满足要求，需要自定义数据转换器，实现ongl.TypeConverter接口。一般的直接继承DefaultTypeConverter类，并实现convertValue方法

1. 问题：

3>关于struts2服务器端数据校验问题validator date不能起作用

4>自定义拦截器有问题，session中无值

1. 属性不能写！
2. Struts 标签遍历：

<struts:iterator id="book" value="bookList">

<tr>

<td><struts:property value="#book.name"/></td>

<td><struts:property value="#book.author"/></td>

<td><struts:property value="#book.publishedDate"/></td>

</tr>

</struts:iterator>

1. doubleSelect 必须在form中**forName doubleselect所对应的表单名，默认为当前表单，若不在表单中，或是没有指定的表单名不存在，则会报错！**
2. Set和get方法不能是静态方法