**一．Servlet概述**

Servlet在前，JSP在后。JSP前身是Servlet。

1. **Servlet是**在服务器端上运行的程序。一个Servlet就是一个Java类

可以通过“请求-响应”编程模型来访问的这个驻留在服务器内存里的Servlet程序。

Servlet是SUN公司制订的一种用来扩展web服务器功能的组件规范：

用来扩展web服务器功能，让web服务器可以处理动态资源的请求（即可以通过计算生成动态页面）

注：web服务器(比如Apache等)只能处理静态资源的请求(即是需要事先将html文件写好)，

\*组件规范

a.什么是组件？

是符合规范，实现部分功能，需要部署到容器里才能运行的软件模块。

servlet就是一个组件，需要部署到servlet容器里面才能运行。

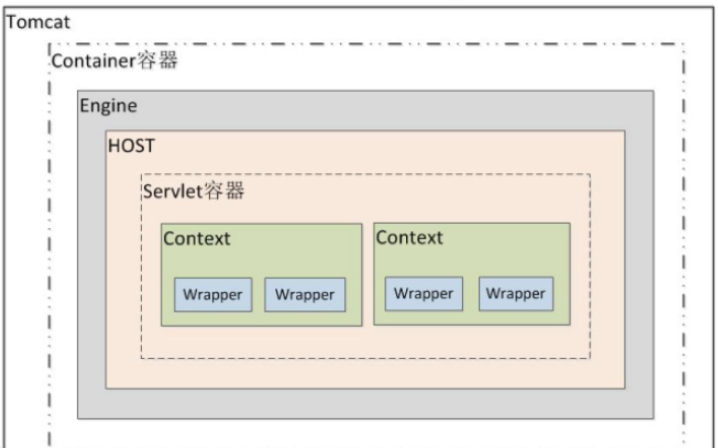
b.什么是容器？

是符合规范，提供组件的运行环境的程序，

比如，Tomcat就是一个提供了Servlet/JSP运行环境的程序。

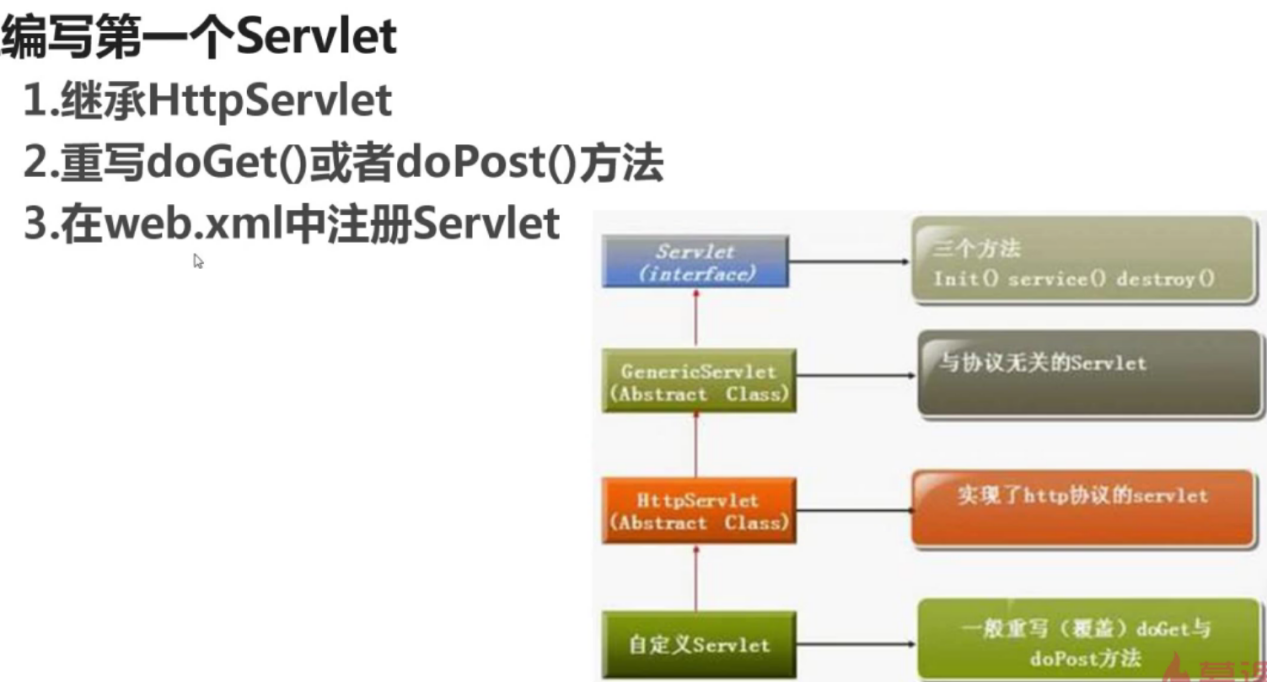
**2.Tomcat容器等级：**

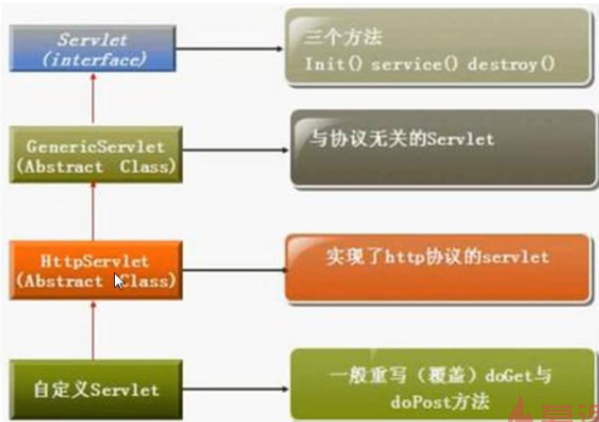
分为四个等级。Servlet的容器管理Context容器，一个Context对应一个Web工程



Engine引擎容器；HOST主机容器；Servlet容器；Context容器；

**3.编写Servlet**





**HttpServlet**类有两个sevrvice( )方法；

自定义Servlet继承了HttpServlet类,一般只需覆盖doPost或者doGet方法,不必覆盖sevrvice( )方法.因为sevrvice( )方法会调用doPost或者doGet方法

在servlet中默认情况下，无论是get还是post 提交过来 都会经过service（）方法来处理，然后转向到doGet 或是doPost方法

如果在自己的servlet类中覆盖了service方法，那么这时service就不是用来转向的，而是用来处理业务的，现在不论你的客户端是用post还是get来请求此servlet

**Servlet**中，service方法是一直存在的，因为最高层的接口Servlet（像HttpServlet等具体的servlet都是直接或者间接实现了这个接口）里面就有这个方法，所以不管是怎样的servlet类，都有service方法，没有service就不能称为一个Servlet了。  
而对于service方法，一般来说这个方法是不需要重写的，因为在HttpServlet中已经有了很好的实现，它会根据请求的方式，调用doGet，doPos以及其他的doXXXt方法，也就是说service是用来转向的，所以我们一般写一个servlet，只需要重写doGet或者doPost就可以了。如果重写了service方法，那么servlet容器就会把请求交给这个方法来处理，倘若你重写的service方法没有调用doXXX，即使你在Servlet中又重写了其他doGet doPost等也是不回被调用的 因为Servlet的service被自动调用（就像init destory一样），所以如果你由于某种需要，需要重写service方法，并且更具不同的method调用doPost doGet等时，就要在末尾加上一句super.service(),这样就可以解决问题了。

通过API可以看到在HttpServlet类中的两个service方法：service(ServletRequest, ServletResponse)方法是重写的其父类GenericServlet类的方法，这个方法是公共的（public），其作用是接受客户端的请求并将其传递给service(HttpServletRequest, HttpServletResponse)方法；service(HttpServletRequest, HttpServletResponse)方法是HttpServlet类定义的方法，是受保护的（protected），主要作用是接受标准的Http请求（HttpServletRequest），并根据请求方式不同分发到不同的doXXX(HttpServletRequest, HttpServletResponse)方法。 这就是HttpServlet有两个service方法的原因了。一般情况下我们的Servlet只需重写受保护的service方法就够了

**Servlet的执行流程：**

**两种访问方式：**

1.http://ip:port/web01【项目应用名】/hello【<url-pattern>】

<http://localhost:8888/WEB01/hello?name=zhangsan&salary=500>

2.http://ip:port/web01【项目应用名】/index.jsp(html)【页面】

step1:浏览器会根据ip,port建立连接

step2:浏览器会将相关的数据(比如请求参数)打包(按照http协议要求，生成一个请求数据包)，发送。

step3:servlet容器解析请求数据包的内容，然后将解析的结果保存到request对象中，【服务器后台自动执行】，同时，还会创建一个response对象。

解析数据包之后跳转至服务器项目下的web.xml开始匹配

<servlet>

<servlet-name>helloServlet</servlet-name> 同名跳转至此处

<servlet-class>web.HelloServlet</servlet-class> 第三步

</servlet>

<servlet-mapping>

<servlet-name>helloServlet</servlet-name> 第二步

<url-pattern>/hello</url-pattern> 第一步

</servlet-mapping>

第三步读取web.HelloServlet【包名，类名】跳转至HelloServlet服务器

step4:servlet容器会创建一个servlet对象，然后调用该对象的service()方法。

step5:servlet容器从response对象中获取处理的结果，然后打包(生成响应数据包)，发送给浏览器。

step6:浏览器解析响应数据包，取出处理的结果，生成相应的页面。

**Servlet的生命周期：**

初始化阶段：

1. 实例化：容器调用servlet的构造方法，创建servlet对象

情况1：容器收到请求之后

情况2：容器事先创建对象

需要配置：<load-on-startup>参数</load-on-startup>

参数值是一个大于等于0的保证书，越小优先级越高(对象越先被创建)

注：容器在默认的情况下，对于某个类型的servlet,只会创建一个实例。

1. 初始化：调用init()方法或带参的（ServletConfig config）方法。

init()在整个生命周期只调用一次

初始化参数配置：在web.xml文件中<Servlet></Servlet>之间添加

step1:配置

<servlet>

<servlet-name>GetInitParameterServlet</servlet-name>

<servlet-class>xxx.xxxxxxxx</servlet-class>

<init-param>

<param-name>username</param-name>

<param-value>admin</param-value>

</init-param>

<init-param>

<param-name>password</param-name>

<param-value>123456</param-value>

</init-param>

</servlet>

step2:通过ServletConfig接口的getInitParameter方法读取

//1.

通过继承自GenericServlet的方法获得ServletConfig对象

ServletConfig config=getServletConfig();

读取初始化参数

String computer=config.getInitParameter("computer");

//2.

在servlet中将web.xml的参数当成属性封装一下

**private** String username**;//用户名**

**private** String password**; //密码**

**public** **void** init() **throws** ServletException {

**this**.setUsername(**this**.getInitParameter("username"));

**this**.setPassword(**this**.getInitParameter("password"));

}

doPost{

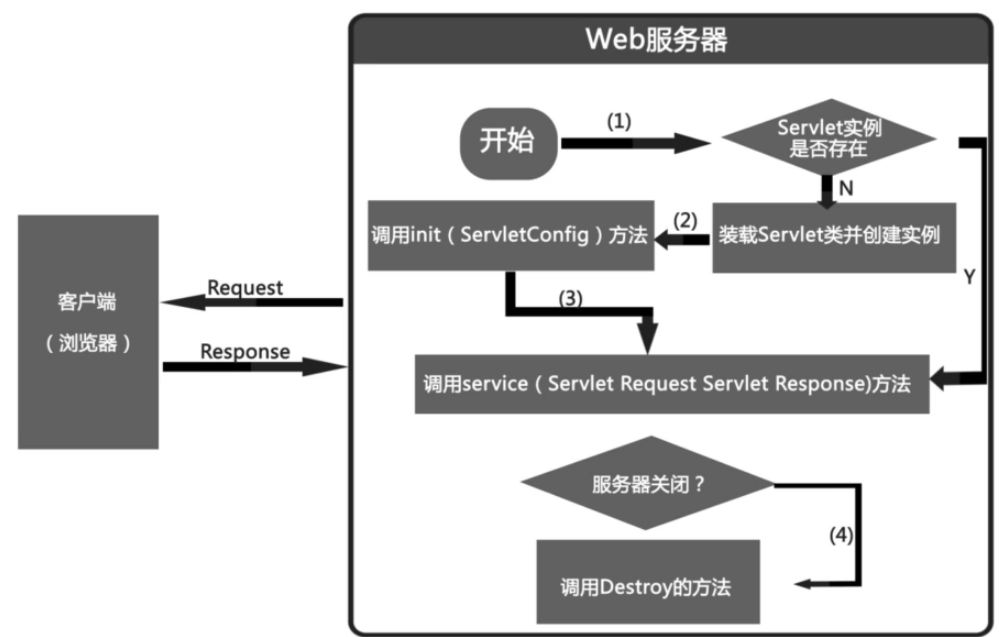
out.println("<h2>"+"用户名："+**this**.getUsername()+"</h2>");

out.println("<h2>"+"密码："+**this**.getPassword()+"</h2>");

}

3.响应客户请求阶段：调用service()方法。由service()方法根据提交方式选择执行doget()或dopost()方法

4.终止阶段：调用destroy()方法



**Tomcat容器部署Servlet的三种情况**

Servlet部署之后长期驻留在内存。

第一种：Tomcat容器启动时自动执行某些Servlet的构造方法创建一个Servlet实例并且调用init()方法进行初始化【Web项目已部署】

只需在web.xml文件中<Servlet></Servlet>之间添加

<load-on-startup>1</load-on-startup> 数字越小优先级越高

第二种：部署后【未添加load-on-startup】

当客户端首次向Servlet发送请求的时候，执行Servlet的构造方法创建一个Servlet实例并且调用init()方法进行初始化

第三种：Servlet类文件或JSP页面被更新修改之后，自动重新部署Servlet,部署位置：

\Workspaces\apache-tomcat-7.0.47\webapps\4.Servlet\WEB-INF\classes\servlet

该路径可查看Tomcat容器部署Servlet的时间，已经部署的前提下才会生成WEB-INF文件夹。

**4.常见的错误及解决办法**

1).404

含义：服务器依据请求地址找不到对应的资源

原因：

a.没有部署项目

b.<servlet-name>上下不一致

c.没有严格按照http://ip:port/项目应用名(appname)/url-pattern来访问。

注意：

404是一个状态码，状态码是一个三位数字，表示服务器处理请求的一种状态。

2).500

含义：系统错误

错误原因：

a.源代码错误(没有按规范来写servlet)比如，没有继承HttpServlet

b.<servlet-class>写错

c.程序不严谨

比如：对请求的参数没有做检查就做转换

3).405

含义：找不到处理方法

错误原因：service方法签名错误

4).删除Workspaces内的JavaWeb项目导致server出错

删除C:\Workspaces\.metadata\.plugins\org.eclipse.core.runtime\.settings\

第一个文件

**5.http(超文本传输协议)（hypertext transfer protocol）**

由w3c制订的一种网络应用层协议，它规定了浏览器和web服务器之间如何通信以及通信所使用的数据格式。

1)**通信方式**

step1:建立连接

step2:浏览器将请求数据打包，发送

step3:web服务器将响应数据打包，发送

step4:web服务器关闭连接

特点：一次请求，一次连接

2)**数据格式**

a.请求数据包

i:请求行： 请求方式 请求资源路径 协议类型和版本

ii:若干消息头：

由w3c定义的一些键值对，

浏览器与web服务器之间可以通过发送这些消息头来传递一些特定的信息。

iii:实体内容：

只有当发送请求为post时，才会有数据(请求参数)

b.响应数据包

i:状态行：协议类型和版本 状态码 状态描述

状态码：状态码是一个三位数字，表示服务器处理请求的一种状态

200：正确

500：系统错误

404：依据请求地址找不到对应的资源

ii:若干消息头

web服务器也可以给浏览器发送一些消息头

iii:实体内容

程序处理的结果

**6.两种请求方式**

1)get方式

i:那些情况下发送get请求：

a.直接输入某个地址

b.点击链接 <a href="url"></a>超链接使用的是get请求

c.表单默认提交方式

ii:get请求有哪些特点

a.会将请求参数添加到请求资源路径后面，

并且只能提交少量的数据(因为请求行最多只能存放大约是2K左右的数据)

b.会将请求参数显示在浏览器地址栏中，不安全

2)post方式

i:哪些情况下发送post请求：

a.设置表单提交方式时 method="post"

ii:post请求的特点

a.会将请求参数添加到实体内容里面，并且可以提交大量数据。

b.不会把请求参数添加到浏览器地址栏中，相对安全。

**7. servlet输出中文**

1)乱码

out.println方法在默认情况下，会使用

"iso-8859-1"来编码。

2)解决方式

response.setContentType("text/html;charset=utf-8");

注：

作用1：out.println方法会使用指定的字符集(比如"utf-8")来编码。

作用2：设置content-type消息头的值，告诉浏览器，服务器返回的数据类型。

**8.如果表单包含有中文参数值，如何处理?**

1)为什么会有乱码?

表单提交时，浏览器会对中文参数值进行编码。

注：

浏览器打开该表单所在的页面时使用的字符集来编码。

服务器端默认情况下，会使用"iso-8859-1"来解码。

2)解决方式

step1:保证浏览器按照指定的字符集来编码。

比如，对于一个静态页面(html),添加

<meta http-equiv="content-type"

content="text/html;charset=utf-8">

step2:服务器端，按照对应的字符集来解码。

request.setCharacterEncoding("utf-8");

注：

a. 该方法必须要添加到所有的

request.getParameter方法的前面。

b. 只针对post请求有效。

c. 如果是get请求，可以采取如下的方式来处理：

name = new String(name.getBytes("iso-8859-1"),"utf-8");

作用：先获得浏览器发送过来的原始的字节数组，然后使用指定的字符集解码。

**9.Servlet与九大内置对象**



JSP的out对象等价于Servlet中用response参数的getWriter方法创建的对象；

严格意义讲是二者是不匹配，后者获得是一个PrintWriter对象，前者属于JSPWriter对象

**10.重定向**

1)什么是重定向？

服务器通知浏览器向一个地址发送请求。

注：

服务器可以发送一个302状态码和一个Location消息头(值是一个地址，一般为重定向地址)

浏览器接收到请求后，会立即向重定向地址发请求。

2)如何重定向？

response.sendRedirect(String url);

3)细节

重定向之前，容器会清空response对象上缓存的所有数据

4)特点：

a.重定向的地址是任意的

b.重定向之后，浏览器地址栏的地址会发生改变

当javabean的作用域范围是scope=“request”时，超链接<a href=url>相当于重定向

**11.转发**

1)什么是转发？

一个web组件(servlet/jsp)将未完成的处理转交给另外一个web组件来继续做。

比如：一个servlet将未完成的任务转交给另一个jsp来展现。

2)如何转发？

step1:先绑定数据到request

request.setAttribute(String name,Object obj);

step2:获得转发器

RequestDispatcher rd=request.getRequestDispatcher(String uri);

step3:开始转发

rd.forward(request,response);

3)细节

转发共用request和response对象

4)转发的特点：

a.转发之后，浏览器的地址栏不变。

b.转发的目的地必须是同一个应用。

**12.转发与重定向的区别**

1)能否共享request?

转发可以，重定向不可以

当容器收到请求时，会立即创建request,response对象，

当响应发送完毕，会立即销毁这两个对象。

2)浏览器的地址栏是否会发生变化？

转发不会，重定向会

3)目的地？

转发的目的地必须是同一个应用，

重定向的目的地没有限制是任意的。

req.getRequestDispatcher("main.jsp").forward(request,response)

req.getRequestDispatcher("main.jsp").include(request,response)

forward方法是把请求的内容转发到另外的一个servlet.而include是把另一个servlet处理过后的内容拿过来

forward与include基本上都是转发到context内部的资源，而redirect可以重定向到外部的资源

调用forward() 的话,有关response对象的一切方法或者属性都会失去作用..只有request能被转向到下一个页面.

调用include()的话,response跟request都能被传递到转向的下一个页面

**13.路径问题**

(1).什么是路径问题？

链接，表单提交，重定向和转发...如何去写相应的路径(地址)

<a href="addEmp.jsp">

<form action="add.do">

response.sendRedirect("list.do");

request.getRequestDispatcher("listEmp.jsp")

(2).相对路径

不以"/"开头的路径。

如果退至到上一级目录以"../"开头

(3).绝对路径

以"/"开头的路径。不以当前文件的位置作为起始，

而是以一个固定的位置作为起始到达目标文件所经过的路径。

(4).如何写绝对路径

链接，表单提交，重定向从应用名开始写

转发从应用名之后开始写。

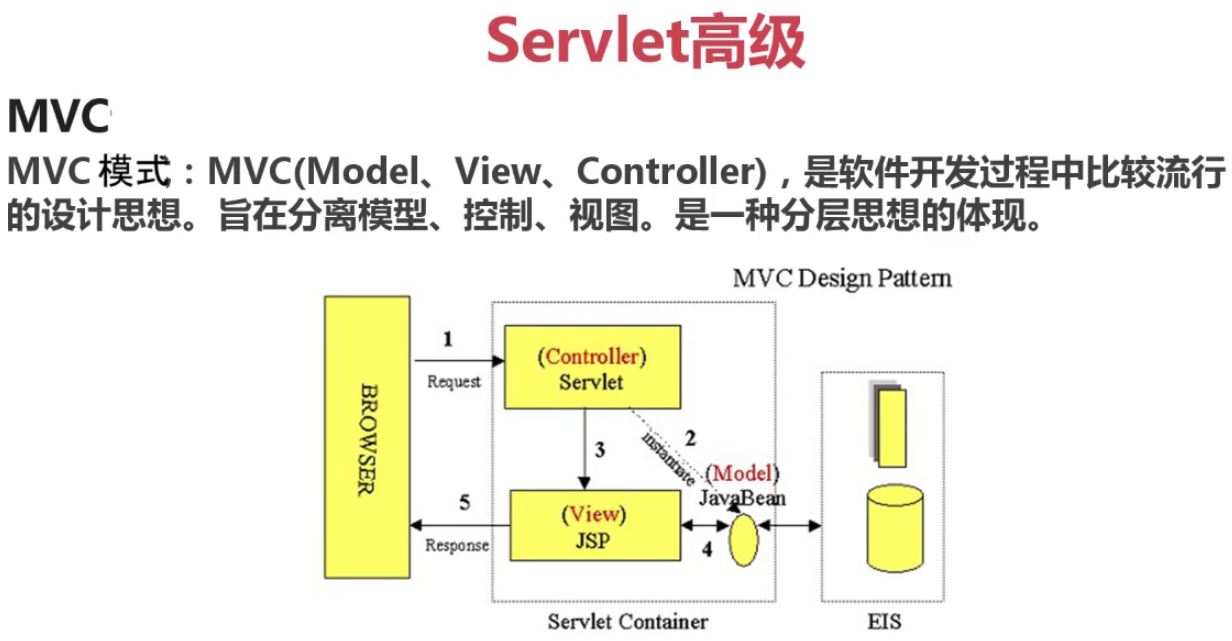
request.getContextPath();获得实际部署项目时的应用名

<a href="<%=request.getContextPath()%>/biz01/b1.jsp">to b1</a>

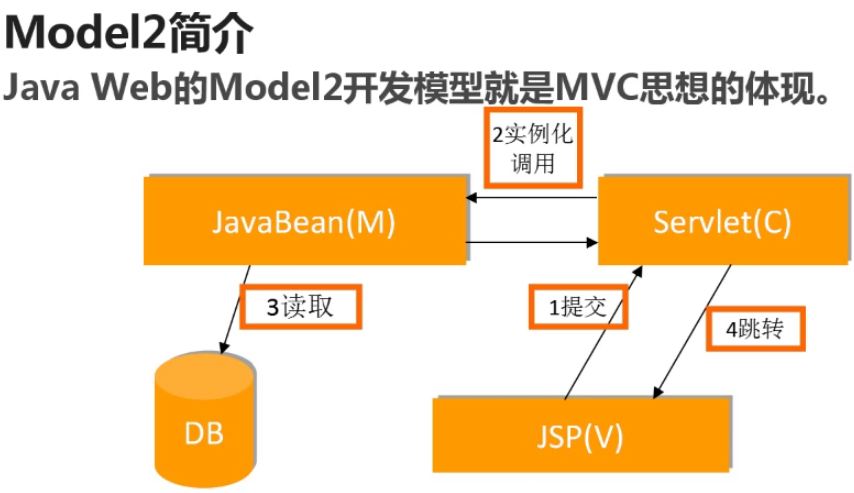
<form action="<%=request.getContextPath()%>/add.do">

response.sendRedirect("<%=request.getContextPath()%>/list.do");

request.getRequestDispatcher("/biz01/b1.jsp")



EIS：企业信息系统



**14.利用转发处理异常**

request.setAttribute("error", "系统繁忙，稍后重试");

request.getRequestDispatcher("error.jsp").forward(request, response);

**15.容器处理异常**

step1：将异常抛给容器,

throw new servletException(e);

step2:配置异常处理页面，

<error-page>

<exception-type>javax.servlet.ServletException</exception-type>

<location>/error2.jsp</location>

</error-page>

step3:写异常页面

error2.jsp

**16.容器如何处理请求资源路径**

比如在浏览器地址栏中输入：

http://ip:port/web05/abc.html

浏览器会将 "/web05/abc.html" 作为请求资源路径发送给容器

step1：依据应用名找到应用所在的文件夹

step2:容器认为访问的是一个servlet,

去web.xml中查找有没有和abc.html匹配的servlet。

精确匹配：

<url-pattern>/abc.html</url-pattern>

通配符匹配:

" \* "匹配0到多个字符

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

后缀匹配：

使用" \*. "开头，后接任意字符(一般写do/form/action)

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

匹配所有以" .do "结尾的请求

step3:如果找不到匹配的servlet,则查找相应的文件。

**17.将多个servlet合并成一个**

step1:后缀匹配

<servlet-mapping>

<servlet-name>ActionServlet</servlet-name>

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

</servlet-mapping>

step2:分析请求资源路径

String uri = request.getRequestURI();

String action=uri.substring(uri.lastIndexOf("/"),uri.lastIndexOf("."));

if("/list".equals(action)){...}