Java Web开发技术

本课程简介

- 面向学员
  - 熟悉Java编程语言
  - 熟悉HTML
  - 不熟悉JavaWeb开发技术
- 授课时间
  - 1-2天（不包括学员上机时间）
- 课程内容
  - JavaWeb应用简介
  - Servlet技术
  - JSP技术
  - 使用HTTP会话（Session）
  - 使用JavaBean
  - 使用Servlet过滤器
  - 自定义客户化标签
  - 开发Java Mail Web

JavaWeb应用简介

- 教学目标
  - 理解JavaWeb应用的概念
  - 理解Servlet容器的概念
  - 掌握安装和启动Tomcat服务器的过程
  - 创建第一个JavaWeb应用，了解JavaWeb应用的目录结构，在Tomcat服务器上发布并运行JavaWeb应用的过程。
  - 对JSP和Servlet有初步的了解
  - 对web.xml文件有初步的了解

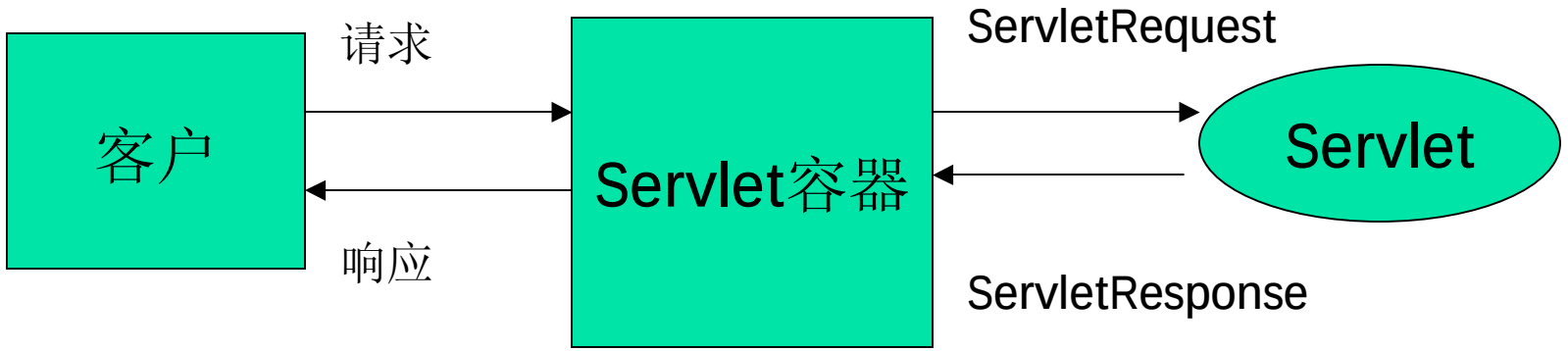
JavaWeb应用的概念

- 在Sun的Java Servlet规范中，对Java Web应用作了这样定义：“Java Web应用由一组Servlet、HTML页、类、以及其它可以被绑定的资源构成。它可以在各种供应商提供的实现Servlet规范的Servlet容器中运行。”
- Java Web应用中可以包含如下内容:
  - Servlet
  - JSP
  - 实用类
  - 静态文档如HTML、图片等
  - 客户端类（如Applet）
  - 描述Web应用的信息（web.xml）

Servlet容器的概念

- Servlet容器为JavaWeb应用提供运行时环境，它负责管理Servlet和JSP的生命周期，以及管理它们的共享数据。
- Servlet容器也称为JavaWeb应用容器，或者Servlet/JSP容器。后文均采用Servlet容器的提法。
- 目前最流行的Servlet容器软件包括:
  - Tomcat
  - Resin
  - J2EE服务器（如Weblogic）中也提供了内置的Servlet容器

Servlet容器响应客户请求的过程



演示运行HelloServlet例子，来说明Servlet容器响应客户请求的过程

http://localhost:8080/helloapp/hello?clientName=小王

Servlet容器响应客户请求的过程

- 当客户请求访问某个Servlet时，Servlet容器将创建一个ServletRequest对象和ServletResponse对象。
- 在ServletRequest对象中封装了客户请求信息，然后Servlet容器把ServletRequest对象和ServletResponse对象传给客户所请求的Servlet。
- Servlet把响应结果写到ServletResponse中，然后由Servlet容器把响应结果传给客户。

Tomcat服务器的结构

- Tomcat服务器是由一系列可配置的组件构成，其中核心组件是Catalina Servlet容器，它是所有其它Tomcat组件的顶层容器。Tomcat的组件可以在<CATALINA_HOME>/conf/server.xml文件中进行配置：

```
<Server>
    <Service>
        <Connector />
        <Engine>
            <Host>
                <Context>
                </Context>
            </Host>
        </Engine>
    </Service>
</Server>
```

Tomcat服务器的结构

1. 顶层类元素

顶层类元素包括<Server>元素和<Service>元素，它们位于整个配置文件的顶层。

2. 连接器类元素

连接器类元素代表了介于客户与服务之间的通信接口，负责将客户的请求发送给服务器，并将服务器的响应结果传递给客户。

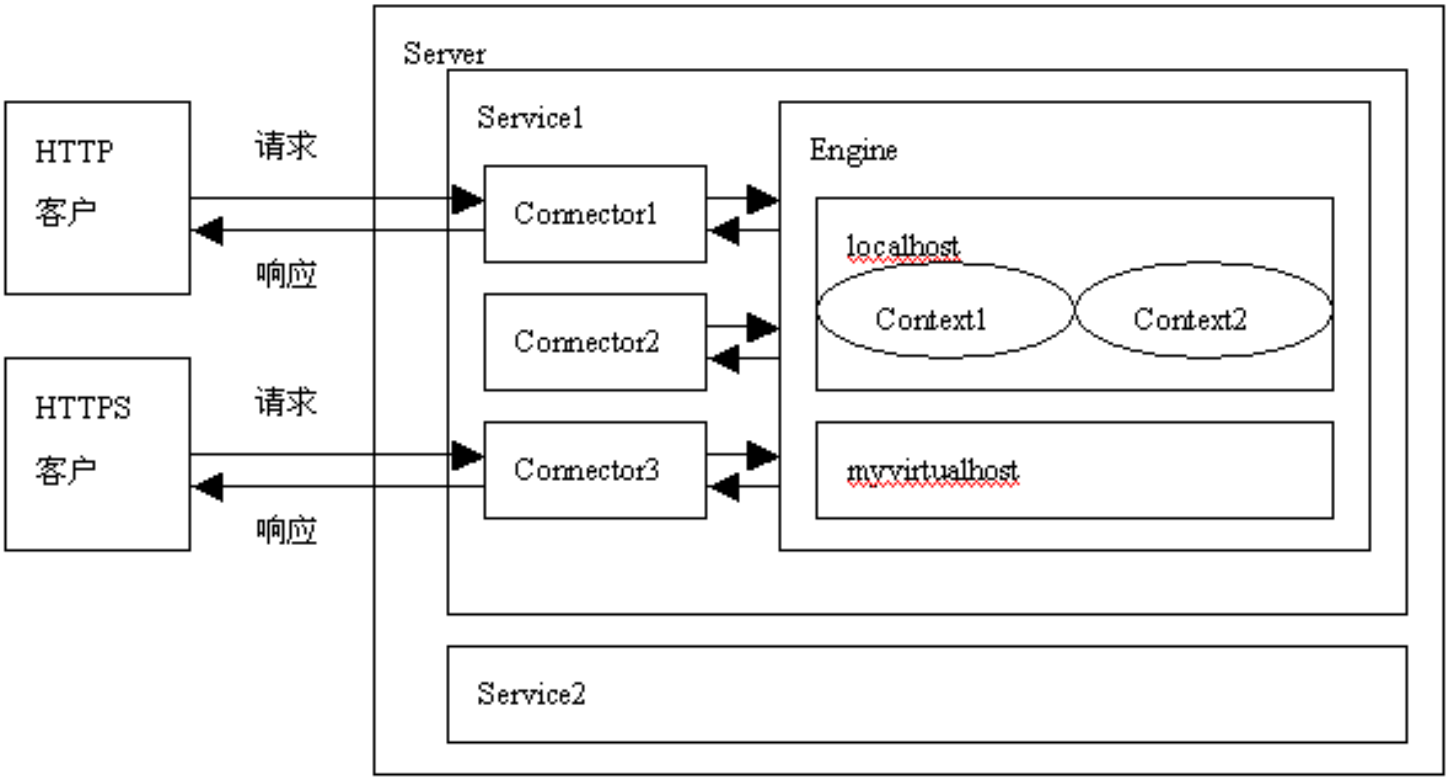
3. 容器类元素

容器类元素代表处理客户请求并生成响应结果的组件，有三种容器类元素，它们是Engine、Host和Context。 Engine组件为特定的 Service组件处理所有客户请求， Host 组件为特定的虚拟主机处理所有客户请求，Context组件为特定的Web应用处理所有客户请求。

4.嵌套类元素

嵌套类元素代表了可以加入到容器中的组件，如<Logger>元素、<Valve>元素和<Realm>元素。

Tomcat各个组件之间的嵌套关系



Connector负责接收客户的请求并向客户返回响应结果，在同一个Service中，多个Connector共享同一个Engine。同一个Engine中可以有多个Host，同一个Host中包含多个Context

安装并运行Tomcat服务器

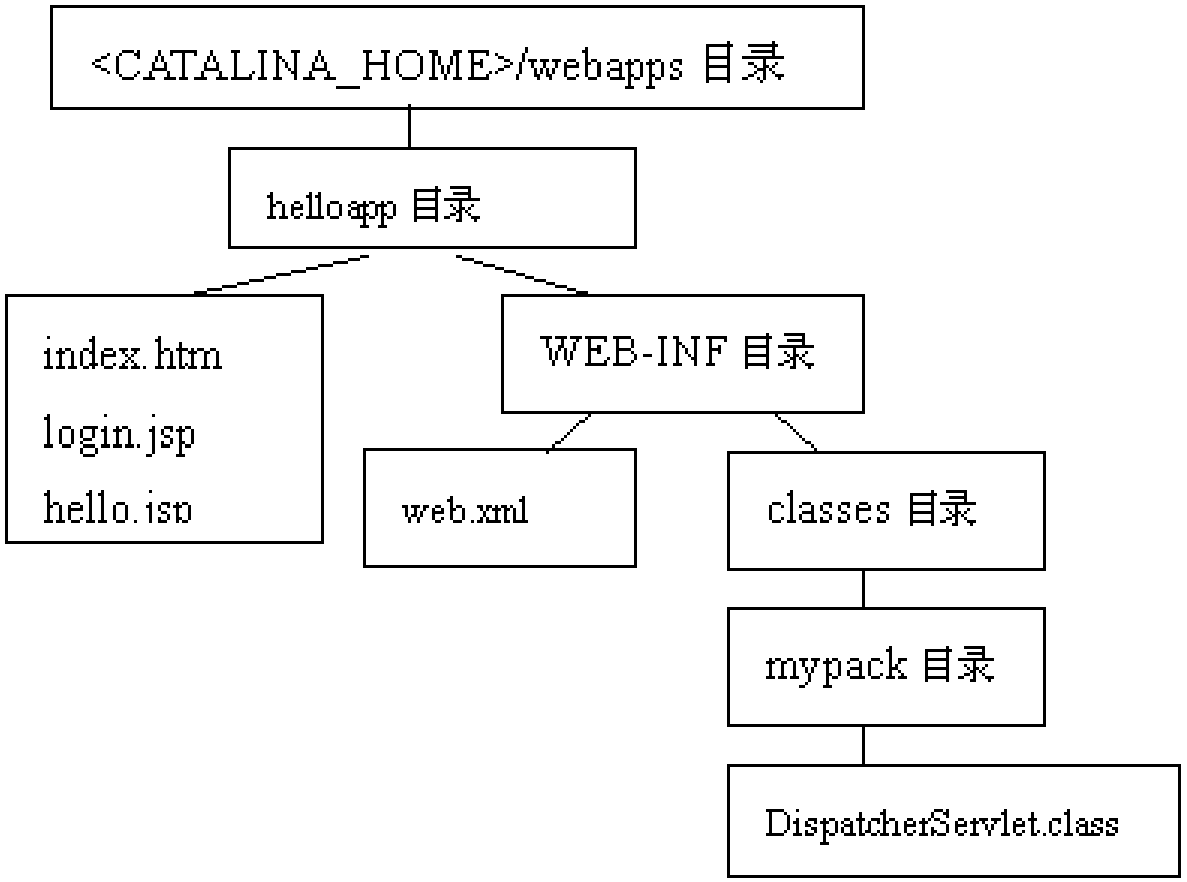
- Tomcat的下载地址: http://jakarta.apache.org
- 安装步骤:
  - 首先安装JDK。
  - 接下来，解压Tomcat压缩文件jakarta-tomcat-5.x.zip。解压Tomcat的压缩文件的过程就相当于安装的过程。假定解压至C:\jakarta-tomcat目录。
  - 然后设定两个环境变量：JAVA_HOME，它是JDK的安装目录；CATALINA_HOME，它是Tomcat的安装目录。

启动和关闭Tomcat服务器

- 启动命令:
  <CATALINA_HOME>\bin\startup.bat
- 关闭命令:
  <CATALINA_HOME>\bin\shutdown.bat
- 访问主页:
  http://localhost:8080/

创建第一个JavaWeb应用

- **helloapp应用的目录结构图**



helloapp应用的目录结构

- /helloapp:Web应用的根目录，所有的JSP和HTML文件存放于此目录
- /helloapp/WEB-INF:存放Web应用的发布描述文件web.xml
- /helloapp/WEB-INF/classes:存放各种class文件，servlet类文件也放于此目录
- /helloapp/WEB-INF/lib:存放Web应用所需的各种JAR文件。例如，在这个目录下，你可以存放JDBC驱动程序的JAR文件
- Web应用发布到Tomcat中的目录为:

  <CATALINA_HOME>\webapps

创建web.xml文件

- 创建一个默认的web.xml文件，并把这个文件放到WEB-INF目录中。

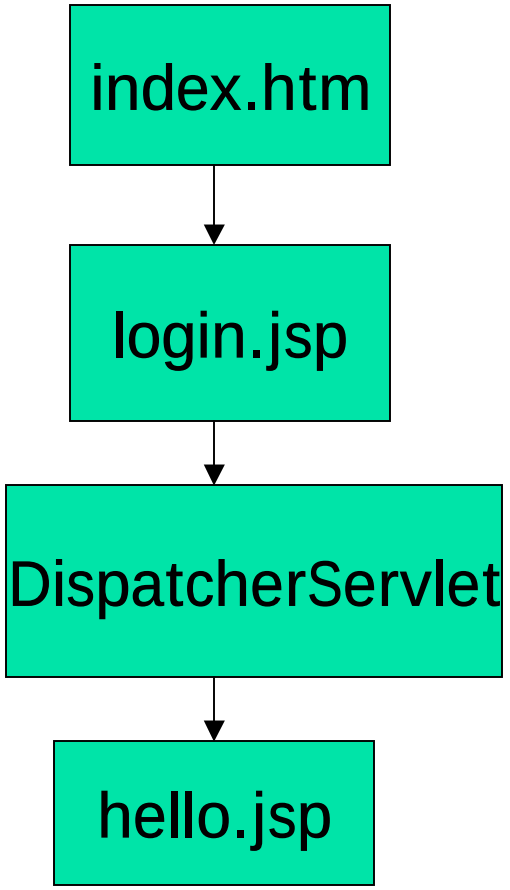
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC
  '-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN'
  'http://java.sun.com/j2ee/dtds/web-app_2_3.dtd'>

<web-app>
</web-app>
```

创建helloapp应用的web组件

- 在helloapp应用中创建如下组件：
  - HTML组件 - index.htm
  - JSP组件 - login.jsp和hello.jsp
  - Servlet组件 - DispatcherServlet
- 组件之间的链接或转发关系



创建index.htm文件

- 这个文件仅仅用来显示一串带链接的字符“Welcome to HelloApp”。以下是index.htm文件的代码:

```
<html>
<head>
<title>helloapp</title>
</head>
<body >
<p><font size="7">Welcome to HelloApp</font></p>
<p><a href="login.jsp">login </a>
</body>
</html>
```

创建login.jsp

- 它显示登录页面，要求输入用户名和口令，这个页面链接到一个名为**DispatcherServlet**的**Servlet**。

```
<form name="loginForm" method="post" action="dispatcher">
<table>
<tr>
<td><div align="right">User Name:</div></td>
<td> <input type="text" name="username"></td></tr>
<tr>
<td><div align="right">Password:</div></td>
<td><input type="password" name="password"></td></tr>
<tr>
<td></td>
<td><input type="Submit" name="Submit" value="Submit"></td>
</tr>
</table></form>
```

创建hello.jsp

- **hello.jsp被DispatcherServlet调用，显示Hello页面。**

```
<html>
<head>
  <title>helloapp</title>
</head>
<body>
  <h1>
    Welcome: < % = request.getAttribute("USER") % >
  </h1>
</body>
</html>
```

创建DispatcherServlet.java

- 它调用HttpServletRequest对象的getParameter方法读取客户提交的loginForm表单数据，获取用户名和口令，然后将用户名和口令保存在HttpServletRequest对象的属性中，再把请求转发给hello.jsp。

编译并发布DispatcherServlet

- 编译DispatcherServlet.java。编译时，需要将Java Servlet API的JAR文件（servlet-api.jar）设置为classpath，servlet-api.jar文件位于<CATALINA_HOME>/common/lib目录下。

- 把编译出来的class文件拷贝到/helloapp/WEB_INF/classes目录下。DispatcherServlet.class的存放位置为/helloapp/WEB_INF/classes/mypack/DispatcherServlet

- 参考编译批处理文件complie.bat

编译并发布DispatcherServlet

- 在web.xml中为DispatcherServlet类加上<servlet>和<servlet-mapping>元素：

```
<web-app>

<servlet>
  <servlet-name>dispatcher</servlet-name>
  <servlet-class>mypack.DispatcherServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>dispatcher</servlet-name>
  <url-pattern>/dispatcher</url-pattern>
</servlet-mapping>

</web-app>
```

在server.xml中加入<Context>元素

- **<Context>**元素是**<CATLINA_HOME>/conf/server.xml**中使用最频繁的元素，它代表了运行在**<Host>**上的单个**Web**应用。一个**<Host>**中可以有多个**<Context>**元素。每个**Web**应用必须有惟一的**URL**路径，这个**URL**路径在**<Context>**元素的**path**属性中设定。

- 在名为“**localhost**”的**<Host>**元素中加入如下**<Context>**元素：

```
<!-- Define the default virtual host -->
<Host name="localhost" debug="0" appBase="webapps"
unpackWARs="true" autoDeploy="true">
……
……
<Context path="/helloapp" docBase="helloapp" debug="0"
reloadable="true"/>

</Host>
```

Context元素的属性

Context 元素的属性

属性	描述
path	指定访问该 Web 应用的 URL 入口
docBase	指定 Web 应用的文件路径。你可以给定绝对路径，也可以给定相对于 Host 的 appBase 属性的相对路径（关于 Host 的 appBase 属性参见 2.3 节）。如果你的 Web 应用采用开放目录结构，那就指定 Web 应用的根目录；如果你的 Web 应用是个 WAR 文件，那就指定 WAR 文件的路径。
reloadable	如果这个属性设为 true，Tomcat 服务器在运行状态下会监视在 WEB-INF/classes 和 WEB-INF/lib 目录下 class 文件的改动。如果监测到有 class 文件被更新，服务器会自动重新加载 Web 应用。

在开发阶段，将 reloadable 属性设为 true，有助于调试 Servlet 和其它的 class 文件。但是由于这一功能会加重服务器的运行负荷，因此建议在 Web 应用的产品发布阶段，将这个属性设为 false。

创建并发布WAR文件

- （1）进入helloapp应用的根目录<CATALINA_HOME>/webapps/helloapp
- （2）把整个Web应用打包为helloapp.war文件，命令如下:

  jar cvf helloapp.war *.*
- （3）把helloapp.war文件拷贝到<CATALINA_HOME>/webapps目录下。
- （4）删除原先的helloapp目录。
- （5）启动Tomcat服务器。

练习题1

- 问题：假设在**helloapp**应用中有一个**hello.jsp**，它的文件路径如下：

  <CATALINA_HOME>/webapps/helloapp/hello/hello.jsp

  那么在浏览器端访问**hello.jsp**的**URL**是什么？

- 选项：

  (A) http://localhost:8080/hello.jsp

  (B) http://localhost:8080/helloapp/hello.jsp

  (C) http://localhost:8080/helloapp/hello/hello.jsp

- 答案：**C**

练习题2

- 问题：假设在**helloapp**应用中有一个**HelloServlet**类，它位于**org.javathinker**包下，那么这个类的**class**文件应该放在什么目录下?
- 选项:

  (A) helloapp/HelloServlet.class

  (B) helloapp/WEB-INF/HelloServlet.class

  (C) helloapp/WEB-INF/classes/HelloServlet.class

  (D)helloapp/WEB-INF/classes/org/javathinker/HelloServlet.class

- 答案：**D**

练习题3

- 问题：假设在**helloapp**应用中有一个**HelloServlet**类，它在**web.xml**文件中的配置如下：

```
<servlet>
  <servlet-name> HelloServlet </servlet-name>
  <servlet-class>org.javathinker.HelloServlet</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name> HelloServlet </servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

那么在浏览器端访问**HelloServlet**的**URL**是什么？

练习题3（续）

- 选项：
  - (A) http://localhost:8080/HelloServlet
  - (B) http://localhost:8080/helloapp/HelloServlet
  - (C) http://localhost:8080/helloapp/org/javathinker/hello
  - (D) http://localhost:8080/helloapp/hello

- 答案：D

练习题4

- 问题：客户请求访问HTML页面与访问Servlet有什么异同？
- 选项：

  (A)相同：都使用HTTP协议

  (B)区别：前者Web服务器直接返回HTML页面，后者Web服务器调用Servlet的方法，由Servlet动态生成HTML页面

  (C)相同：前者Web服务器直接返回HTML页面,后者Web服务器直接返回Servlet的源代码。

  (D)区别：后者需要在web.xml中配置URL路径。

  (E)区别：前者使用HTTP协议，后者使用RMI协议。

- 答案：A，B，D