

第2章 Servlet技术模型

2.1 Servlet API

2.2 Servlet生命周期

2.3 分析请求

2.4 发送响应

2.5 Web应用程序及结构

2.6 部署描述文件

2.7 @WebServlet和@WebInitParam注解

2.8 ServletConfig接口



2.1 SERVLET API

Sun的Servlet规范提供了一个标准的、平台独立的框架用来实现在**Servlet**和**Servlet**容器之间的通信。该框架是由一组**Java**接口和类组成的，它们统称为**Servlet API**。

<http://tomcat.apache.org/tomcat-7.0-doc/servletapi/>



Servlet3.0 API的组成:

(1) javax.servlet包

开发**独立于协议**的服务器小程序的接口和类。

(2) javax.servlet.http包

定义了开发**采用HTTP协议**通信的服务器小程序的接口和类。

(3) javax.servlet.annotation包

定义**9个注解**类型和**2个枚举**类型。

(4) javax.servlet.descriptor包

定义了访问Web应用程序**配置信息**的类型。



2.1.1 JAVAX.SERVLET包

➤Servlet接口

Servlet API中的核心接口，每个Servlet必须直接或间接**实现**该接口。该接口定义了5个方法。

- `public void init(ServletConfig config)`
- `public void service(ServletRequest request,
ServletResponse response)
throws ServletException, IOException`
- `public void destroy()`
- `public ServletConfig getServletConfig()`
- `public String getServletInfo()`

➤ServletConfig接口

ServletConfig接口为用户提供了有关Servlet的**配置信息**。Servlet配置包括Servlet名称、Servlet环境（context）对象、Servlet初始化参数名和参数等。

➤GenericServlet类

GenericServlet是一个**抽象类**，它**实现**了Servlet接口和ServletConfig接口，它又是**HttpServlet类的超类**。

➤ ServletRequest接口

ServletRequest接口是独立于任何协议的请求对象。它定义获取客户请求信息的方法。

➤ ServletResponse接口

ServletResponse接口是独立于任何协议的响应对象。它定义了向客户发送适当响应的方法。

2.1.2 JAVAX.SERVLET.HTTP包

➤HttpServlet 抽象类

实现针对**HTTP**协议的Servlet, 它**扩展了**
GenericServlet 类

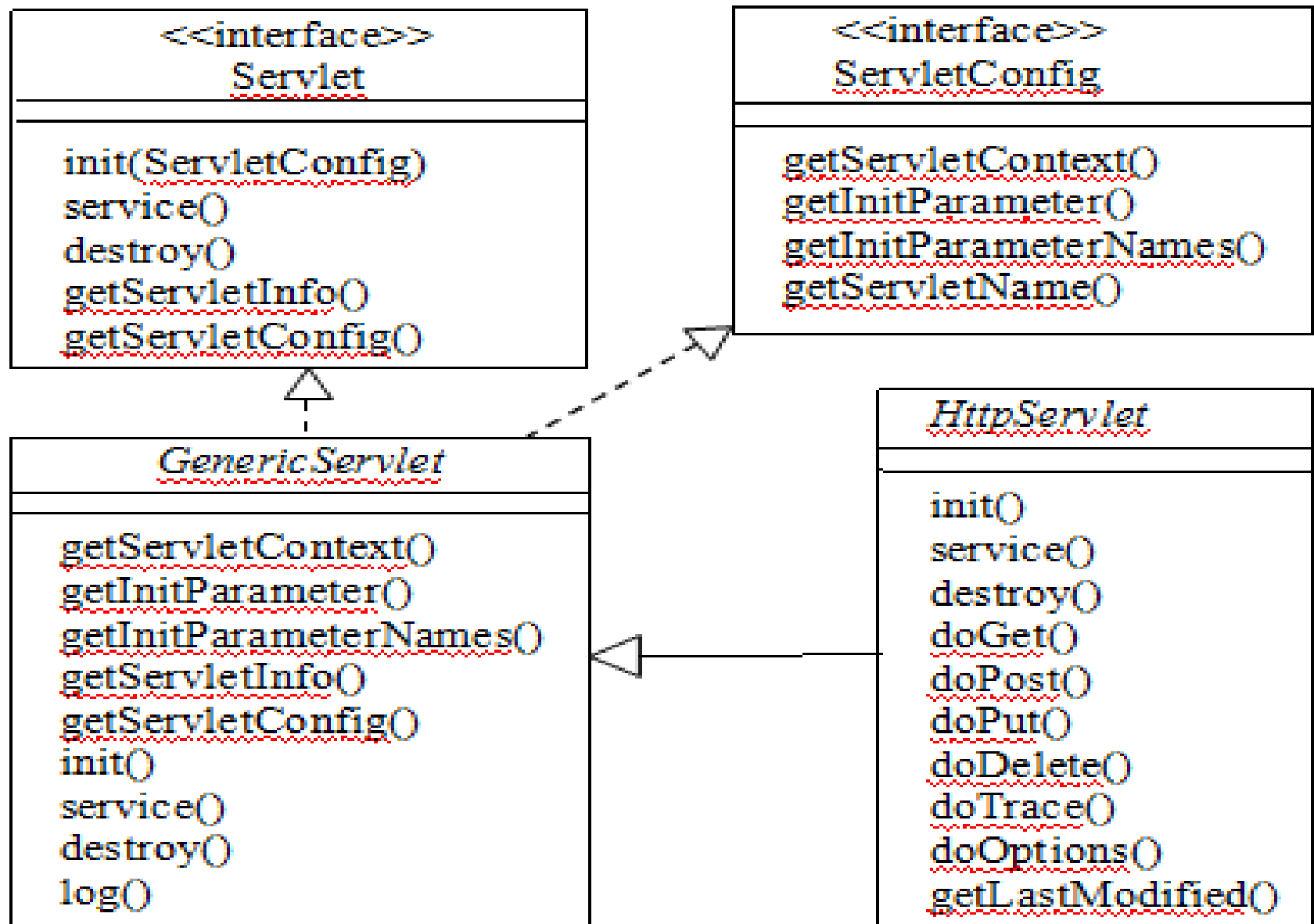
HttpServlet类增加了一个新的**service()**方法:

```
protected void service (HttpServletRequest,  
                        HttpServletResponse)  
throws ServletException, IOException;
```

该方法是Servlet向客户请求提供服务的一个方法。编写Servlet可以覆盖该方法。



Servlet接口和类的层次关系如图2.1所示。



➤ **HttpServletRequest**接口

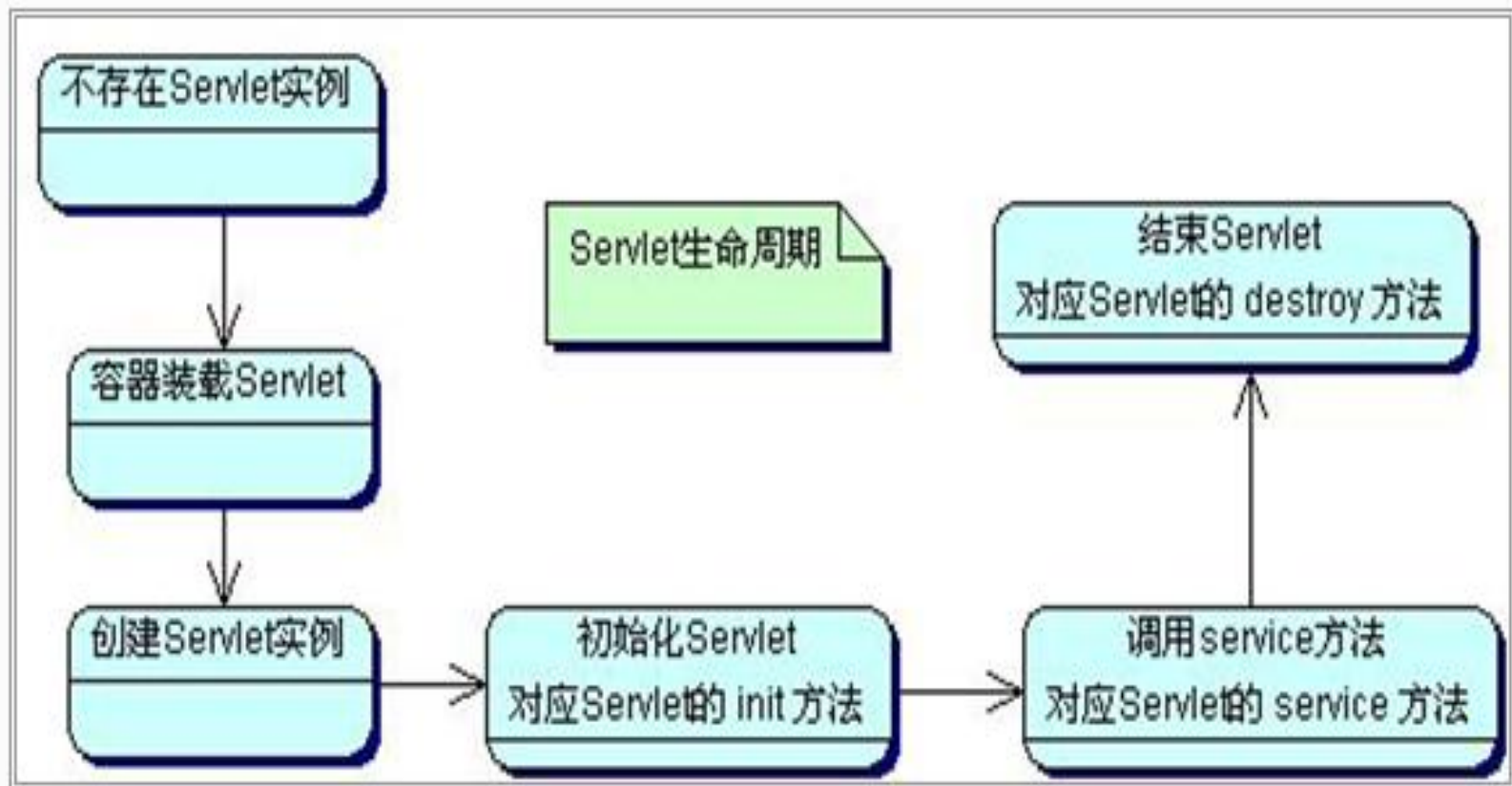
该接口**扩展了ServletRequest**接口；
提供了针对**HTTP**请求操作方法，如定义了从请求对象中获取如**HTTP**请求头、**cookies**信息的方法。

➤ **HttpServletResponse**接口

该接口**扩展了ServletResponse**接口；
提供了针对**HTTP**的发送响应的方法；
定义了为响应设置如**HTTP**响应头、设置 **cookies**信息的方法；

2.2 SERVLET生命周期

❖ **Servlet的生命周期**定义了一个Servlet如何被加载、初始化，以及它怎样接收请求、响应请求、提供服务,最后被销毁的过程。



a)加载和实例化Servlet

对于一个Servlet，可能在Web容器启动时或第一次被访问时加载到容器中。对每个Servlet，容器使用**Class.forName()**方法对其加载并实例化。

b)初始化

一旦容器创建了servlet实例，它将在新创建的实例上调用**init(ServletConfig)**方法初始化这个对象。

调用**init(ServletConfig)**方法后，容器将调用无参数的**init()**方法，在init()方法返回后，Servlet就被初始化了。在Servlet生命周期中init()方法仅被调用一次。

c)为客户请求服务

- 当容器接收到请求时，都会创建一个新的线程，然后调用 **service(ServletRequest,ServletResponse)** 方法。
- **service()** 方法将检查HTTP请求的类型（GET、POST等）并调用相应的 **doGet()**、**doPost()** 等方法。Servlet使用 **响应对象（response）** 获得输出流对象，调用有关方法将响应发送给客户浏览器。
- 之后，线程将被**销毁**或者返回到容器管理的线程池。请求和响应对象已经离开其作用域，也将被销毁。最后客户得到响应。

d) 销毁和卸载Servlet

- 当容器决定不再需要Servlet实例时，它将在Servlet实例上调用**destroy()**方法，Servlet在该方法中释放资源，如它在init()方法中获得的数据库连接。一旦该方法被调用，Servlet实例不能再提供服务。
- 一旦Servlet实例被销毁，它将作为垃圾被回收。如果Web容器关闭，Servlet也将被销毁和卸载。

2.3 分析请求

2.3.1 HTTP请求结构

2.3.2 发送HTTP请求

2.3.3 处理HTTP请求

2.3.4 分析请求

2.2.5 请求转发

2.3.6 使用请求对象存储数据

2.3.7 实例：一个简单的考试系统

2.3.8 文件上传



一个HTTP消息是客户向服务器的**请求**或服务器向客户的**响应**。

HTTP消息组成：

消息部分	说明
请求行或状态行	指定请求或响应消息的目的
请求头或响应头	指定元信息，如关于消息内容的大小、类型、编码方式
空行	
可选的消息体	请求或响应消息的主要内容

2.3.1 HTTP请求结构

客户向服务器发出的 HTTP 消息叫做**HTTP请求** (HTTP Request)。

请求行 → `POST /bookstore/selectBook HTTP/1.1`

请求头 → `Accept = */*`
`Accept-Language = zh-cn`
`Accept-Encoding = gzip, deflate`
`User-Agent = Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1;`
`Host = localhost:8080`
`Connection = Keep-Alive`

空行

数据 → `name=Java+EE`

1) 请求行

- **HTTP 请求行由三部分组成，且由空格分隔：**
 - a) 方法名**
 - b) 请求的资源的局部路径 (URI)**
 - c) 使用的 HTTP 的版本**
- **示例：POST/bookstore/selectBook HTTP 1.1**
 - **POST 是方法名，**
 - **/bookstore/selectBook 是资源的 URI，**
 - **HTTP/1.1 是请求的 HTTP 的版本。**

2) 请求头

- 请求行之后的头行称为请求头,它可以指定:

请求使用的浏览器信息

字符编码信息

客户能处理的页面类型等。

- 示例:

User-Agent: = Mozilla/4.0 (compatible;
MSIE 6.0; Windows NT 5.1;

Content-Type: application/msword

Content-Length:11

3)常用的请求方法

- 请求行中的方法名指定了**客户请求服务器完成的动作**。
- HTTP 1.1版本共定义了8个方法
 - GET**：请求读取一个Web页面
 - POST**：请求向服务器发送数据
 - PUT**：请求存储一个Web页面
 - HEAD**：请求读取一个Web页面的头部
 - DELETE**：移除Web页面
 - TRACE**：返回收到的请求
 - OPTIONS**：查询特定选项
 - CONNECT**：保留作将来使用

4) GET方法和POST方法

- ◆ GET方法用来检索资源。它的含义是“**获得(get)由该URI标识的资源**”。
- ◆ 说明：资源通常是**被动资源**，但是 GET也可以用来请求**主动资源**。
- ◆ 例如：传递一个名为userid参数的请求行。

HTTP方法	URL串	查询串	HTTP版本
GET	/servlet/helloServlet?	userid=john	HTTP/1.1

- ◆ 查询串（query string），是由**&**符号分开的**参数名和参数值对**组成的，如下所示：

name1=value1&name2=value2&...&nameM=valueM

➤ **POST请求用来向服务器发送需要处理数据。**

POST含义是“将数据发送到由该URI标识的主动资源”，数据块在消息体中发送。

➤ **GET方法与POST方法比较**

特征	GET方法	POST方法
资源类型	主动的或被动的	主动的
数据类型	文本	文本或二进制数据
数据量	一般不超过255个字符	没有限制
可见性	数据是URL的一部分，在浏览器的URL地址栏中用户可见	数据不是URL的一部分而是作为请求的消息体发送，因此在浏览器的URL地址栏中对用户不可见
数据缓存	数据可以在浏览器的URL历史中缓存	数据不能在浏览器的URL历史中缓存

两种方法的使用场合：

➤ 使用GET方法

检索HTML文件或图像文件，因为只需要发送文件名。

➤ 使用POST方法

发送大量数据。例如，POST方法非常适合于在线调查，因为查询字符串可能超过255个字符。

上传文件。因为文件的大小可能超过255个字符，而且，文件也可能是二进制的文件。

捕获用户名和口令。因为我们需要防止用户从URL中看到口令。

2.3.2 发送请求

发送请求的事件:

- 用户在浏览器的地址栏中输入URL并按回车键;
- 用户在HTML页面中点击了超链接;
- 用户在HTML页面中添写一个表单并提交;
- 其他 : 用JavaScript函数在当前文档上调用 **reload()** 方法。

在上面的所有事件中，缺省情况下浏览器使用的是**HTTP GET**方法。



指定POST方法:

- 下面的HTML表单通过method的属性值使用HTTP的POST方法:

```
<form name='loginForm' method='POST'  
        action='/loginServlet'>  
  <input type='text' name='userid'>  
  <input type='password' name='passwd'>  
  <input type='submit' name='loginButton'  
    value='Login'>  
</form>
```

注意: 在<form>标签中如果没有指定method属性, 浏览器缺省使用GET方法

2.3.3 处理HTTP请求

在HttpServlet类中,对每个HTTP方法,都有一个相应的**doXXX()**方法,它具有下面一般格式:

```
protected void  
doXXX(HttpServletRequest,  
        HttpServletResponse)  
throws ServletException, IOException;
```

所有的doXXX()方法都有两个参数:

HttpServletRequest对象

HttpServletResponse对象。



□ HTTP方法和doXxx()方法

HTTP方法

doXxx()方法

GET 方法

doGet()方法

POST方法

doPost()方法

HEAD方法

doHead()方法

PUT方法

doPut()方法

DELETE方法

doDelete()方法

OPTIONS方法

doOptions()方法

TRACE方法

doTrace()方法

□ 在HttpServlet类中为每个doXxx()方法提供的是空实现。为实现业务逻辑，应该覆盖这些doXxx()方法。

2.3.4 分析请求

1. 检索请求参数

请求参数是随请求一起发送到服务器的数据，它是以 **名/值对** 的形式发送的。可以使用 `ServletRequest` 接口中定义的方法检索由客户发送的参数

```
public String getParameter(String name)
```

返回由 `name` 指定的请求参数值，如果指定的参数不存在，则返回 `null` 值。使用该方法必须确信指定的参数只有一个值。

public String[] getParameterValues(String name)

返回指定的参数name所包含的所有值，返回一个String数组对象,如果指定的参数不存在，则返回null值。该方法适用于参数有多个值的情况。如果参数只有一个值，则返回的数组的长度为1。

public Enumeration getParameterNames()

返回一个Enumeration对象,它包含请求中所有的请求参数名，元素是String类型的。

public Map getParameterMap ()

返回一个包含所有请求参数的Map对象，该对象以参数名作为键,以参数值作为值。

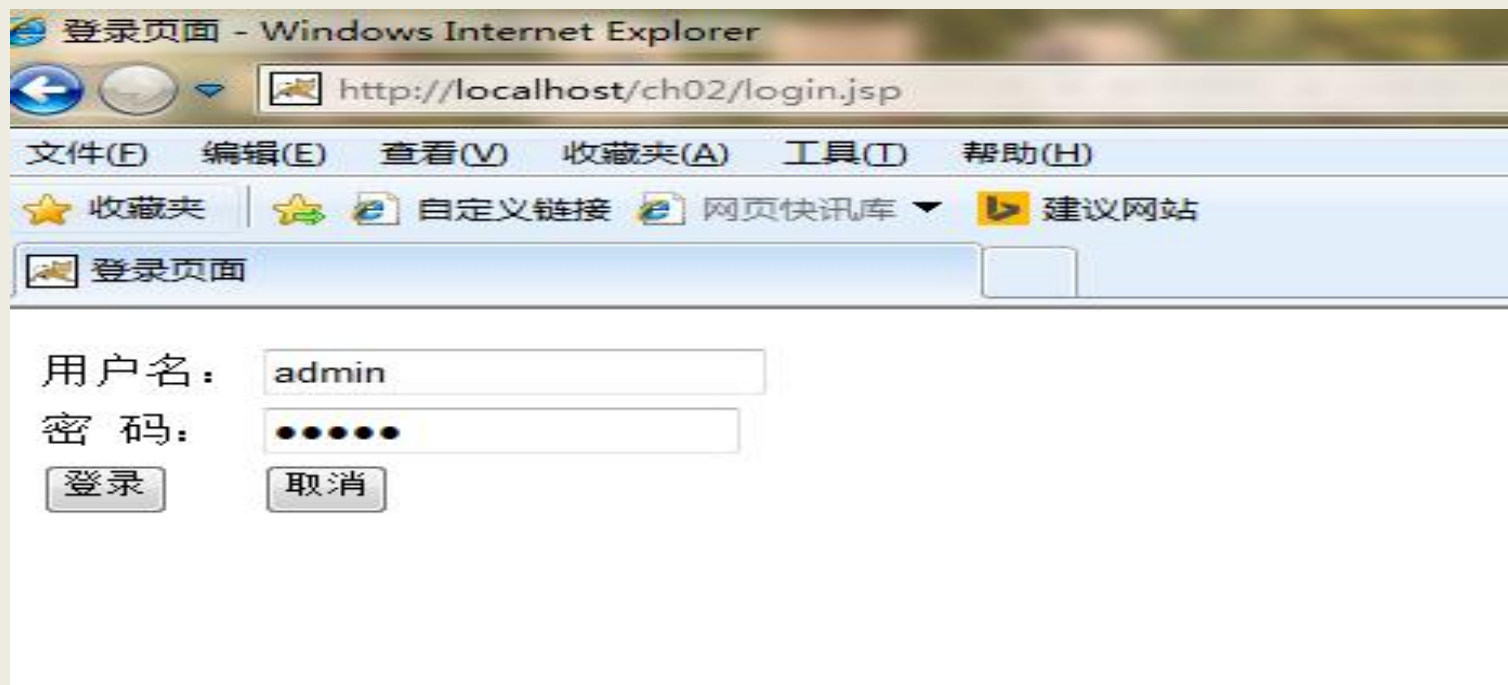
□ 请求参数传递的方法

(1) 通过表单提供请求参数。

源程序代码：[程序2.1 login.jsp](#)

[程序2.2 LoginServlet.java](#)

执行结果<http://localhost/ch02/login.jsp>



(2) 将参数名和值**附加**在请求的URL后面，这种方法只适用于GET请求。

如：访问LoginServlet使用的URL：

`http://localhost/ch02/login.do?username=admin&password=admin`

- 问号后面内容为请求参数名和请求参数值对，称为查询串（query string）
- 若有多个参数，中间用**&**符号分隔，参数名和参数值之间用等号（**=**）分隔。

(3) 在超链接中也可以传递请求参数
例如:

```
<a href="/ch02/login.do?username=admin  
&password=admin">用户登录</a>
```

□ **注意:** 以jsp为前缀的请求参数名都是保留的,下面的用法会产生意想不到的结果, 因此不推荐使用。

```
http://localhost:80/ch02/login?jspTest=myTest
```

2.检索客户端有关信息

String getRemoteHost() 返回客户端的主机名。

String getRemoteAddr() 返回客户端的IP地址。

int getRemotePort() 返回客户端的IP地址端口号。

String getProtocol() 返回客户使用的请求协议和版本。

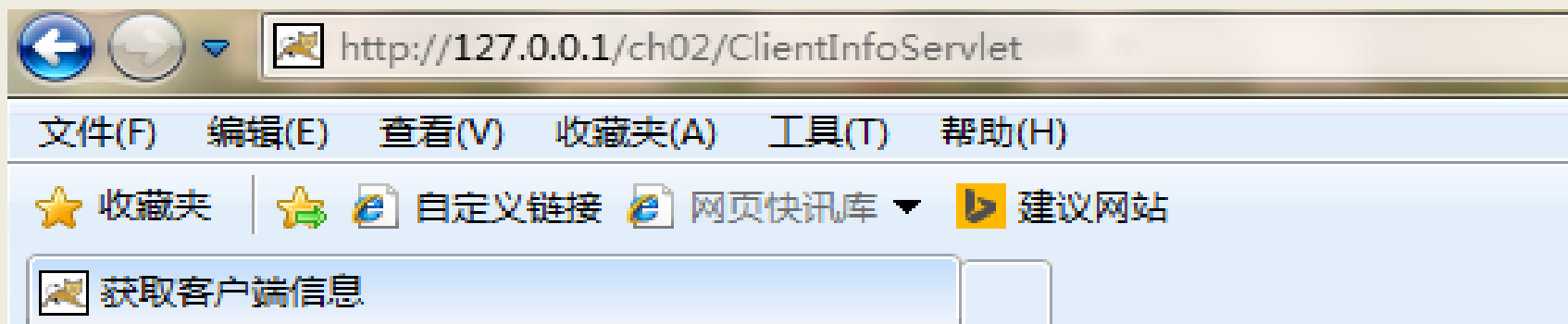
String getContentType() 返回请求体的MIME类型。

String getCharacterEncoding() 返回客户请求的编码方式。

public Cookie[] getCookies() 返回客户与该请求一起发送的Cookie对象数组。

例如：程序2.3 [ClientInfoServlet.java](#)

运行： <http://localhost/ch02/ClientInfoServlet>



客户端信息:

客户主机名	127.0.0.1
客户IP地址	127.0.0.1
端口	49846
请求方法	GET
请求协议	HTTP/1.1
请求URI	/ch02/ClientInfoServlet

3.检索HTTP请求头

HTTP请求头是针对HTTP协议的，因此处理头的方法属于**HttpServletRequest**接口。常用的请求头有：

请求头名	内 容
user-agent	关于浏览器和它的平台的信息
accept	客户能处理的页面的类型
accept-charset	客户可以接受的字符集
accept-encoding	客户能处理的页面编码的方法
accept-language	客户能处理的语言
host	服务器的DNS名字
authorization	客户的信任凭据的列表
cookie	将一个以前设置的cookie送回服务器
date	消息被发送的日期和时间

HttpServletRequest接口中用于检索头信息的方法有：

public String getHeader(String name) 返回指定名称的请求头的**值**。

public Enumeration getHeaders(String name) 返回指定名称的请求头的**Enumeration对象**。

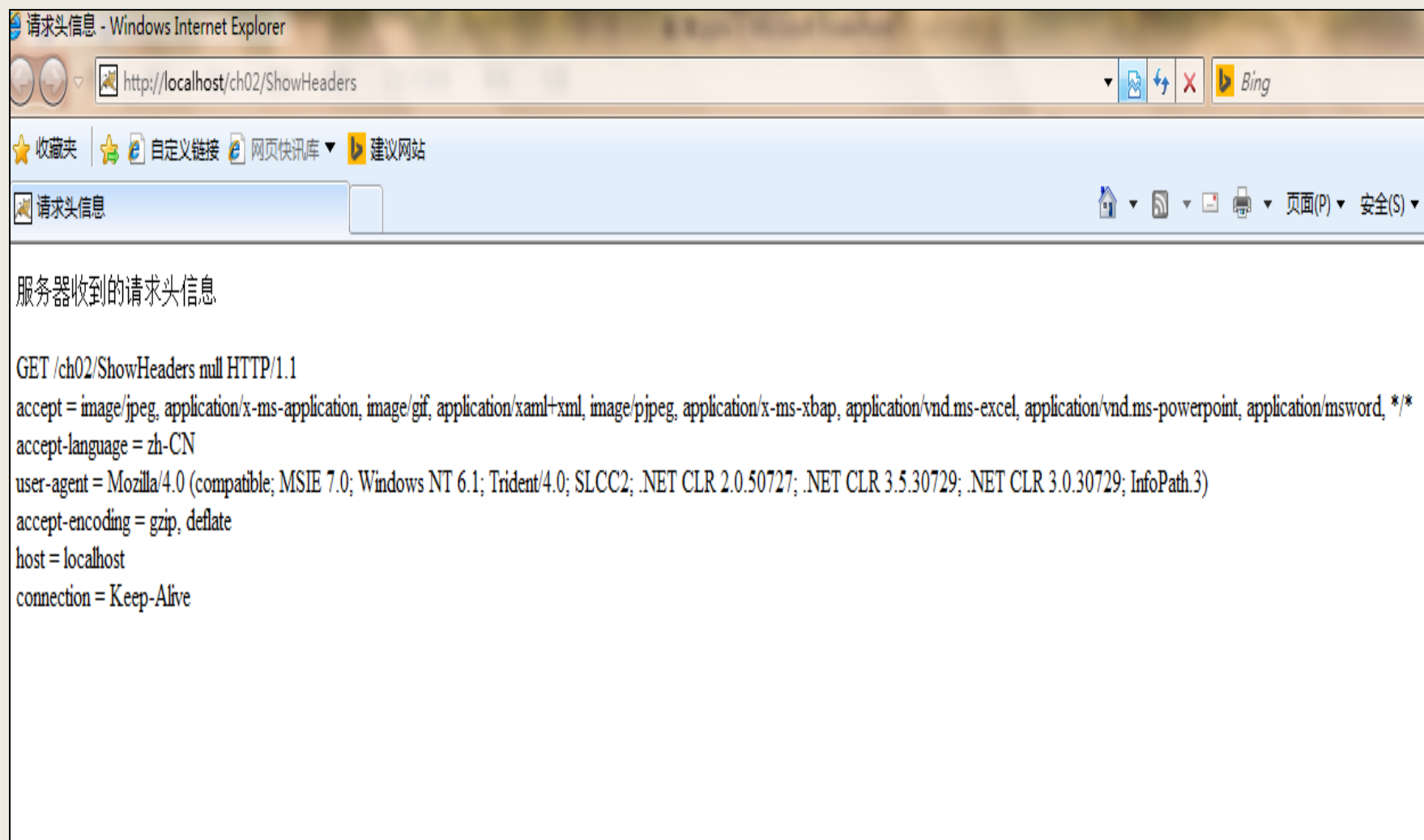
public Enumeration getHeaderNames() 返回一个**Enumeration对象**，它包含所有**请求头名**。

public int getIntHeader(String name) 返回指定名称的请求头的**整数值**。

public long getDateHeader(String name) 返回指定名称的请求头的**日期值**。

举例:程序2.3 ShowHeaderServlet.java

运行: <http://localhost/ch02/ShowHeaders>



2.3.5 请求转发

为实现请求转发,需要使用**RequestDispatcher**对象,通过使用**ServletRequest**接口中定义的方法得到该对象, 格式如下:

```
RequestDispatcher getRequestDispatcher(String path)
```

RequestDispatcher接口定义了两个方法

```
public void forward(ServletRequest request,  
                    ServletResponse response)
```

```
public void include(ServletRequest request,  
                    ServletResponse response)
```

2.3.6 使用请求对象存储数据

ServletRequest接口的方法:

public void setAttribute(String name, Object obj)

public Object getAttribute(String name)

public Enumeration getAttributeNames()

public void removeAttribute(String name)

示例程序:

程序2.1 login.jsp

程序2.5 LoginServlet.java

程序2.6 welcome.jsp

运行:<http://localhost/chap02/login.jsp>

2.3.7 实例：一个简单的考试系统

开发一个简单的考试系统，在JSP页面中**建立**一个表单，通过**POST**方法传递参数。

示例程序：

[程序2.7 questions.jsp](#)

[程序2.8 SimpleTestServlet.java](#)

运行：

<http://localhost/ch02/questions.jsp>



2.3.8 文件上传

开发一个简单的**文件上传系统**，在客户端页面中通过一个表单打开一个文件，然后提交给服务器，并在服务器上保存。

示例程序：

程序2.9 fileUpload.jsp

程序2.10 FileUploadServlet.java

运行：

<http://localhost/ch02/fileUpload.jsp>



@MultipartConfig注解

- 使用 **@MultipartConfig** 该注解告诉容器该 Servlet 能够处理 **multipart/form-data** 的请求。
- 使用该注解，**HttpServletRequest** 对象可以得到表单数据的各部分。

元素名	类 型	说 明
location	String	指定容器临时存储文件的目录位置
maxFileSize	long	指定允许上传文件的最大字节数
maxRequest Size	long	指定允许整个请求的 multipart/form-data 数据的最大字节数
fileSizeThre shold	int	指定文件写到磁盘后阈值的大小

- 除了注解中指定文件的限制外，还可以在web.xml文件中使用<servlet>的子元素<multipart-config>指定这些限制，该元素包括4个子元素，分别为：

<location>、<max-file-size>、<max-request-size>和<file-size-threshold>。

文件输入流

在服务器端，可以使用请求对象的 `getInputStream()` 返回 `ServletInputStream` 输入流对象，**文件内容** 就包含在该对象中，另外其中还包含 **表单域的名称和值**、**上传的文件名**、**内容类型** 等信息。例如，假设上传一个 **Java** 源文件，返回的输入流的内容可能如下。



-----7d81a5209008a

Content-Disposition: form-data; name="mnumber"
223344

-----7d81a5209008a

Content-Disposition: form-data; name="fileName";
filename="C:\study\HelloWorld.java" filename="HelloWorld.java"

Content-Type: application/octet-stream

```
public class HelloWorld {  
    public static void main(String ars[]){  
        System.out.println("Hello,World!");  
    }  
}
```

-----7d81a5209008a

Content-Disposition: form-data; name="submit"
提交

-----7d81a5209008a--

返回Part对象

Part是Servlet 3.0 API新增的一个接口，定义在 `javax.servlet.http`包中，它表示多部分表单数据的一个部分。通过请求对象的下面两个方法能够返回Part对象。

- **public Part `getPart(String name)`:** 返回用 `name`指定名称的Part对象。
- **public Collection<Part> `getParts()`:** 返回所有Part对象的一个集合。

Part接口方法

- `public InputStream getInputStream() throws IOException`: 返回Part对象的输入流对象。
- `public String getContentType()`: 返回Part对象的内容类型。
- `public String getName()`: 返回Part对象的名称。
- `public long getSize()`: 返回Part对象的大小。
- `public String getHeader(String name)`: 返回Part对象指定的MIME头的值。



- **public Collection<String> `getHeaders(String name)`**: 返回name指定的头值的集合。
- **public Collection<String> `getHeaderNames()`**: 返回Part对象头名称的集合。
- **public void `delete()` throws IOException**: 删除临时文件。
- **public void `write(String fileName)` throws IOException**: 将Part对象写到指定的文件中。

2.4 发送响应

2.4.1 HTTP响应结构

2.4.2 理解ServletResponse

2.4.3 理解HttpServletResponse

2.4.4 发送状态码和错误消息



2.4.1 HTTP响应结构

由服务器向客户发送的HTTP消息称为HTTP响应（HTTP response）。

一个典型的HTTP响应消息 为：

状态行 → **HTTP/1.1 200 OK**

响应头 { **Date: Tue, 01 Sep 2004 23:59:59 GMT**
Content-Type: text/html
Content-Length: 52

空行

响应数据 → **<html> <body>**
<h1>Hello, John!</h1>
</body></html>

1. 状态行与状态码

状态行的组成有：

- **HTTP版本**
- **请求结果的响应状态码**
- **描述状态码的短语**

例如：

HTTP/1.1 200 OK

HTTP/1.1 404 Not Found

HTTP/1.1 500 Internal Error

2. 响应头

响应头是服务器向客户端发送的响应消息。

例如：

Date: Tue, 01 Sep 2004 23:59:59 GMT

Content-Type: text/html

Content-Length: 52

3. 响应数据

空行的后面是响应的数据。

例如：

<html> <body>

<h1>Hello, John!</h1>

</body></html>

2.4.2 理解ServletResponse

1. 使用PrintWrite,向客户发送文本数据

```
PrintWriter out = response.getWriter();
```

2. 使用ServletOutputStream向客户发送二进制数据。

```
ServletOutputStream sos = response.  
getOutputStream()
```



3. 设置内容类型

- 在向客户端发送数据之前，一般应该设置发送数据的 **MIME**（**Multipurpose Internet Mail Extensions**）内容类型。

```
response.setContentType("application/pdf");  
response.setContentType("text/html;charset=  
UTF-8");
```

- **MIME**是描述消息内容类型的因特网标准。
- **MIME**消息能包含文本、图像、音频、视频以及其他应用程序专用的数据。

例如:使用制表符分隔数据生成Excel电子表格

程序2.11 : [ExcelServlet.java](#)

运行结果: <http://localhost/ch02/excel.do>

本例中内容类型设置为application/vnd.ms-excel
输出以Excel电子表格的形式发送给客户浏览器。

问题： 如果将响应的内容类型设置为
“application/msword;charset=gb2312”，
在客户端将打开？ 显示输出内容。

常见的MIME内容类型

类型名

含义

application/msword

Microsoft Word文档

application/pdf

Acrobat 的pdf文件

application/vnd.ms-excel

Excel 电子表格

application/vnd.ms-powerpoint

PowerPoint演示文稿

application/jar

JAR文件

application/zip

ZIP压缩文件

audio/midi

MIDI音频文件

image/gif

GIF图像

image/jpeg

JPEG图像

text/html

HTML文档

text/plain


纯文本

video/mpeg

MPEG视频片段

文件(F) 编辑(E) 视图(V) 插入(I) 格式(O) 工具(T) 数据(D) »

地址(D)  http://localhost:8080/chap04/excel

 转到

A1

 学号

	A	B	C	D	E	F
1	学号	姓名	性别	年龄	所在系	
2	95001	李勇	男	20	信息	
3	95002	刘晨	女	19	数学	
4						
5						
6						

⏮ ⏪ ⏩ ⏭

excel/

⏮

⏭

⏩ ⏭

补充：实现文件下载

□ 通过将响应内容类型设置为application/jar实现向客户发送一个JAR文件，对客户来说是从服务器下载文件。

□ 程序为： [FileDownloadServlet.java](#)

□ 运行结果：

<http://localhost/ch02/filedownload.do>



文件下载



您想打开或保存此文件吗？



名称: servlet-api.jar

类型: Executable Jar File

发送者: localhost

打开 (O)

保存 (S)

取消



来自 Internet 的文件可能对您有所帮助，但某些文件可能危害您的计算机。如果您不信任其来源，请不要打开或保存该文件。[有何风险？](#)

2.4.3 理解HttpServletResponse

1. 设置响应头

`public void setHeader(String name, String value)`

`public void setIntHeader(String name, int value)`

`public void setDateHeader(String name, long date)`

`public void addHeader(String name, String value)`

`public void addIntHeader(String name, int value)`

`public void addDateHeader(String name, long date)`

`public boolean containsHeader(String name)`



典型的响应头名

响应头名称	说明
Date	指定服务器的当前时间
Expires	指定内容被认为过时的时间
Last-Modified	指定文档被最后修改的时间
Refresh	告诉浏览器重新装载页面
Content-Type	指定响应的内容类型
Content-Length	指定响应的内容长度
Content-Disposition	为客户指定将响应的文件保存到磁盘上的名称
Content-Encoding	指定页面在传输过程中使用的编码方式

程序2.12: ShowTimeServlet.java

运行结果:

<http://localhost/ch02/ShowTimeServlet>

- 要告诉浏览器在5秒钟后跳转到<http://host/path>页面, 可以使用下面语句:

```
response.setHeader("Refresh","5;URL=http://host/path/");
```

- 在HTML页面中通过在<head>标签内添加下面代码也可以实现这个功能:

```
<meta http-equiv="Refresh" content="5;  
URL= http://host/path/">
```

2. 响应重定向

- Servlet可能决定不直接向浏览器发送响应，而是将响应重定向到其他资源。

`public void sendRedirect(String location)`

- `location`为指定的新的资源的URL。该URL可以是**绝对URL**，也可以是**相对URL**。

程序2.13: [RedirectServlet.java](#)

运行结果: <http://localhost/ch02/redirect.do>

❑ 关于**sendRedirect()**方法，记住下面两点。

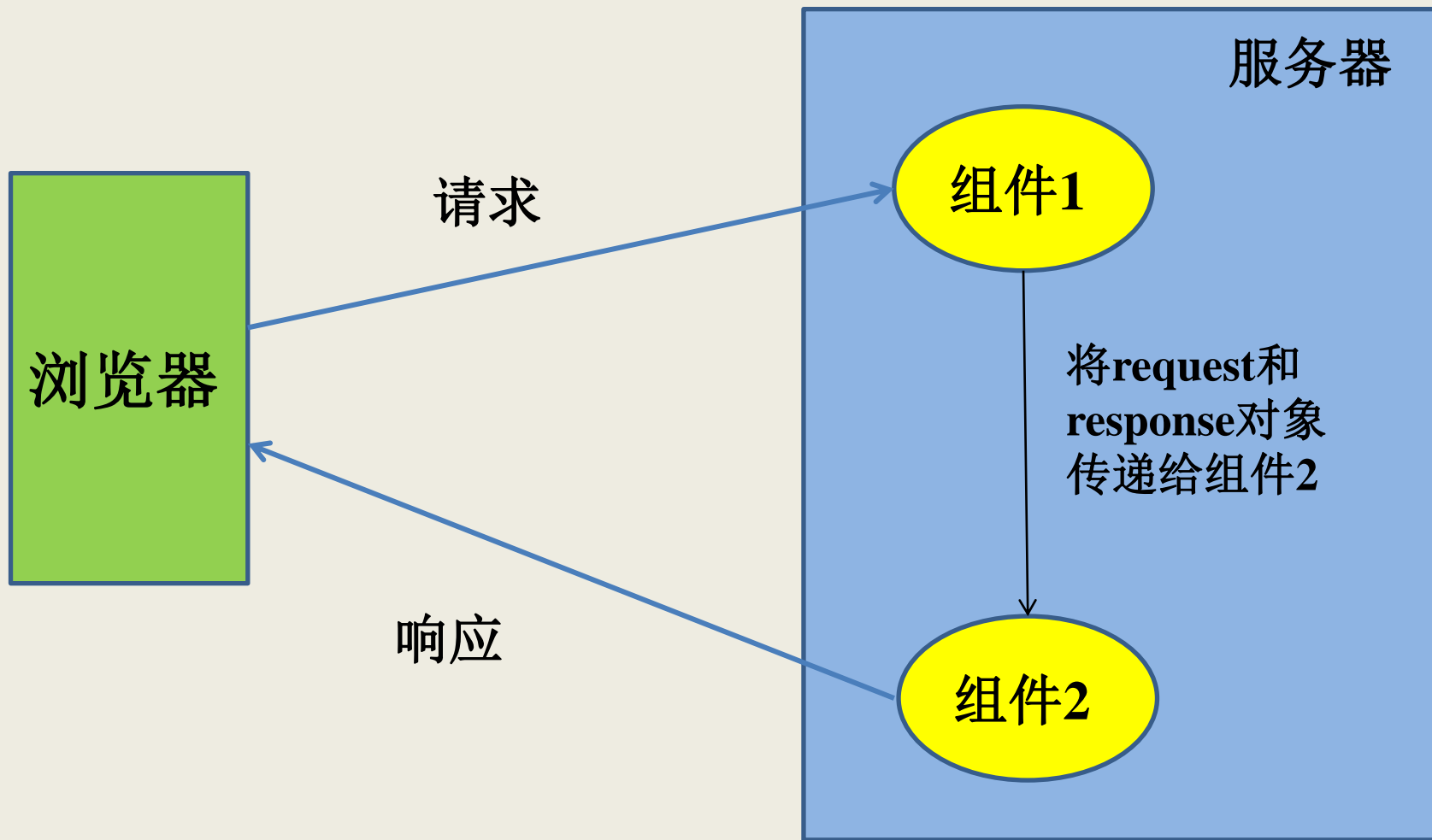
(1) 如果响应被提交，即响应头已经发送到浏览器，就不能调用该方法。此时如果调用该方法，方法将抛出异常。例如：

```
public void doGet(HttpServletRequest req,  
                  HttpServletResponse res){  
    PrintWriter pw = res.getWriter();  
    pw.println("<html><body>Hello  
    orld!</body></html>");  
    pw.flush();  
    res.sendRedirect("http://www.cnn.com"); }
```

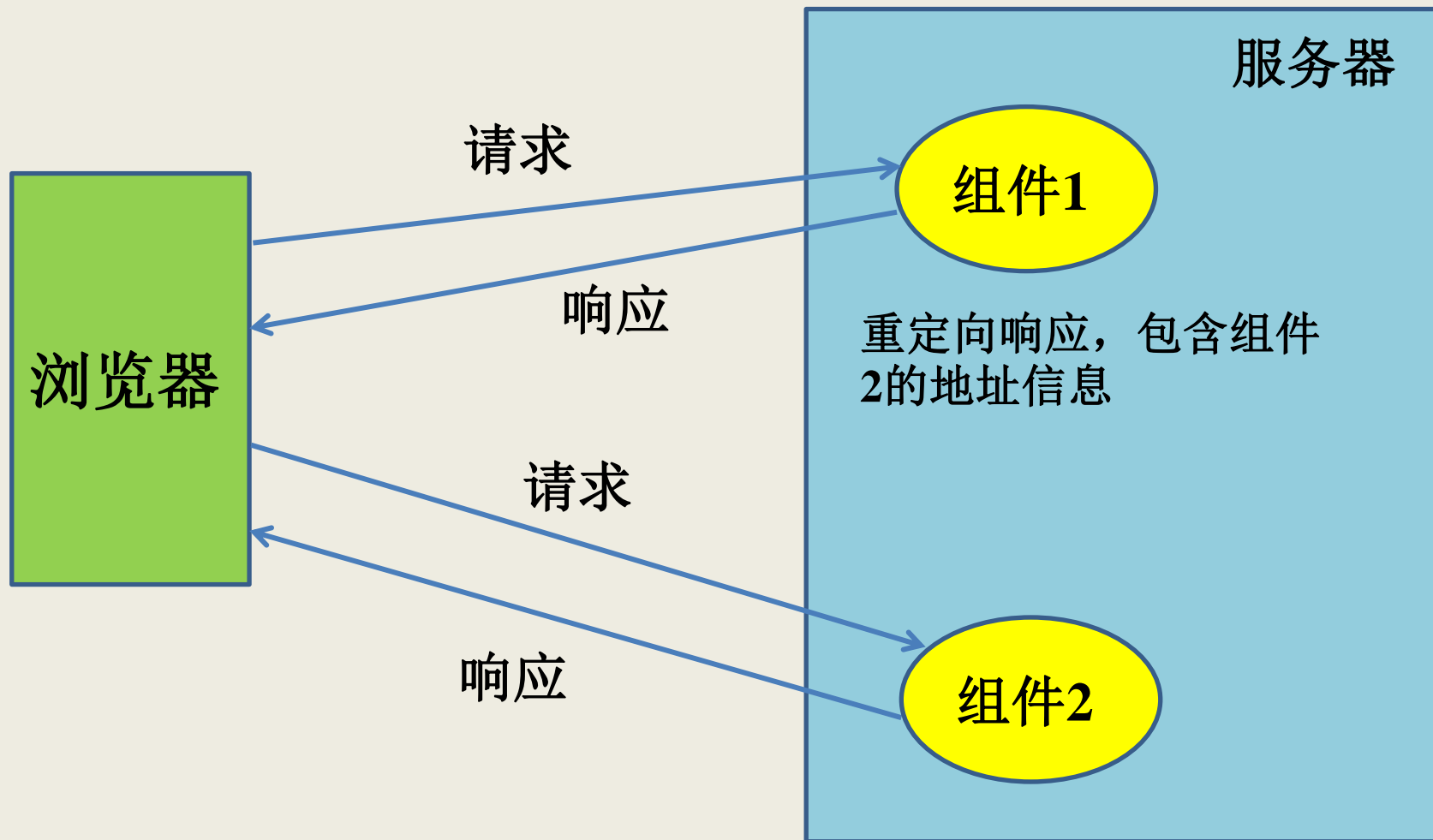
(2) 只有当浏览器从第一个资源收到重定向消息后，它才请求第二个资源。

3. 响应重定向与请求转发的比较

□ 请求转发



□ 响应重定向



- **RequestDispatcher**对象是一个Web资源的包装器，可以用来把当前请求转发到该资源。
- 这种转发是在**服务器端控制权**的转向，客户端发来的请求将交由新的页面进行处理。
- 使用请求转发，在客户的**浏览器地址栏中不会显示转发后的资源地址**。
- 使用请求转发可以共享**请求作用域**中的数据。

- **sendRedirect()**方法实际是服务器向浏览器发送一个特殊的**响应头（Location，状态码302）**，它命令浏览器连接到新的位置。
- 因此，使用这种方法在浏览器的**地址栏中可以看到地址的变化**。
- 重定向是作为不同请求来看待的，因此，所有请求作用域的参数在重定向到下一个页面时都会失效。
- 使用**sendRedirect()**方法重定向时，资源不能位于**WEB-INF**目录中。
- 使用响应重定向可以共享**会话作用域**中的数据。

2.4.4 发送状态码和错误消息

- 如下是Web服务器发送相应的一个典型的状态行：

HTTP/1.1 200 OK

HTTP的版本 状态码 消息

- 状态码可以使用**httpServletResponse**接口中定义的**setStatus()**方法设置，格式为：

public void setStatus (int sc)



在HTTP协议1.1版中状态码的定义：

状态码范围	含 义	示 例
100~199	表示信息	100表示服务器同意处理客户的请求。
200~299	表示请求成功	200表示请求成功， 204表示内容不存在。
300~399	表示重定向	301表示页面移走了， 304表示缓存的页面仍然有效。
400~499	表示客户的错误	403表示禁止的页面， 404表示页面没有找到。
500~599	表示服务器的错误	500表示服务器内部错误， 503表示以后再试。

发送错误消息

- **HttpServletResponse**提供了**sendError()**方法用来向客户发送状态码和错误消息。

public void sendError (int sc)

public void sendError (int sc, String msg)

- 如果Servlet发现客户不应访问其结果，它将调用：

sendError(HttpServletResponse.SC_UNAUTHORIZED)

- **程序2.14:** [StatusServlet.java](#)

- **运行结果:** <http://localhost/ch02/StatusServlet>

2.5 Web应用程序及结构

- 2.5.1 Web应用程序
- 2.5.2 应用服务器
- 2.5.3 Web应用程序的结构



2.5.1 Web应用程序

- 所谓 **Web应用程序** 是一种可以通过Web访问的应用程序。
- 一个Web应用程序是由完成特定任务的各种 **Web组件** 构成的并通过Web将服务展示给外界。

- 多个Servlet
- 多个JSP页面
- HTML文件
- 图像文件等



2.5.2 应用服务器

- **Web** 应用程序驻留在应用服务器上。
- 应用服务器为 **Web** 应用程序提供一种简单的和可管理的对系统资源的访问机制。
 - 它也提供低级的服务，
 - 如 **HTTP** 协议的实现
 - 数据库连接管理。
- **Servlet** 容器仅仅是应用服务器的一部分。



• 市场上可以得到多种Servlet容器，其中包括：

- Apache 的Tomcat 、
- Caucho Technology 的Resin、
- Macromedia 的JRun、
- JBoss
- BEA 的WebLogic、
- IBM 的WebSphere等。

2.5.3 Web应用程序的结构

└ webapps

└ bookstore

└ html (包含所有的HTML文件)

└ images (包含所有的GIF、JPEG和BMP图象文件)

└ javascripts (包含所有的*.js文件)

└ jsp (包含所有的JSP文件)

└ index.html (默认的欢迎文件)

└ WEB-INF

└ classes (应用程序的类文件)

└ com

└ mycompany

└ MyClass.class

└ lib (驱动程序包和标签库包文件)

└ *.jar(jdbcdriver.jar,mytaglib.jar)

└ web.xml (部署描述文件)



1. 理解文档根目录

—假设服务器主机名为**www.myserver.com**，如果要访问**bookstore** Web应用程序根目录下的**index.html** 文件，应该使用下面的URL：

<http://www.myserver.com/bookstore/index.html>

—要访问html目录中的**/hello.html** 文件，使用下面的URL：

<http://www.myserver.com/bookstore/html/hello.html>

2. 理解WEB-INF目录

(1) classes目录

(2) lib目录

(3) web.xml文件

在创建Web应用程序时，可能需要阻止用户访问一些特定资源而允许Servlet容器访问它们。为了**保护**这些资源，应将它们存储在**WEB-INF**目录中。

3. Web归档文件

- 扩展名为`.war`的Web归档文件（web archive），一般称为**WAR**文件。
- 创建一个**WAR**文件:使用Java的**jar**工具将**ch01**目录打包:

```
jar cvf ch01.war *
```

4. 默认的Web应用程序

- Tomcat服务器维护一个默认的Web应用程序。
- CATALINA_HOME>\webapps\ROOT目录被设置为默认的Web应用程序的文档根目录。
- 访问它的资源不需要指定应用程序的名称。
例如，访问默认Web应用程序的URL为：
<http://localhost:8080/>

2.6 部署描述文件

部署描述文件（**Deployment Descriptor**，简称**DD**）可用来初始化Web应用程序的组件。

2.6.1 一个简单的DD

2.6.2 DD文件的定义

2.6.3 使用<servlet>元素

2.6.4 使用<servlet-mapping>元素

2.6.5 使用<welcome-file-list>元素

2.6.1 一个简单的DD

□部署描述文件 程序2.15 [web.xml](#)

- DD文件是一个XML文件。
- 第一行指定了XML的版本及所使用的字符集。
- 下面所有的内容都包含在<web-app>和</web-app>元素中，它是DD文件的根元素



2.6.2 DD文件的定义

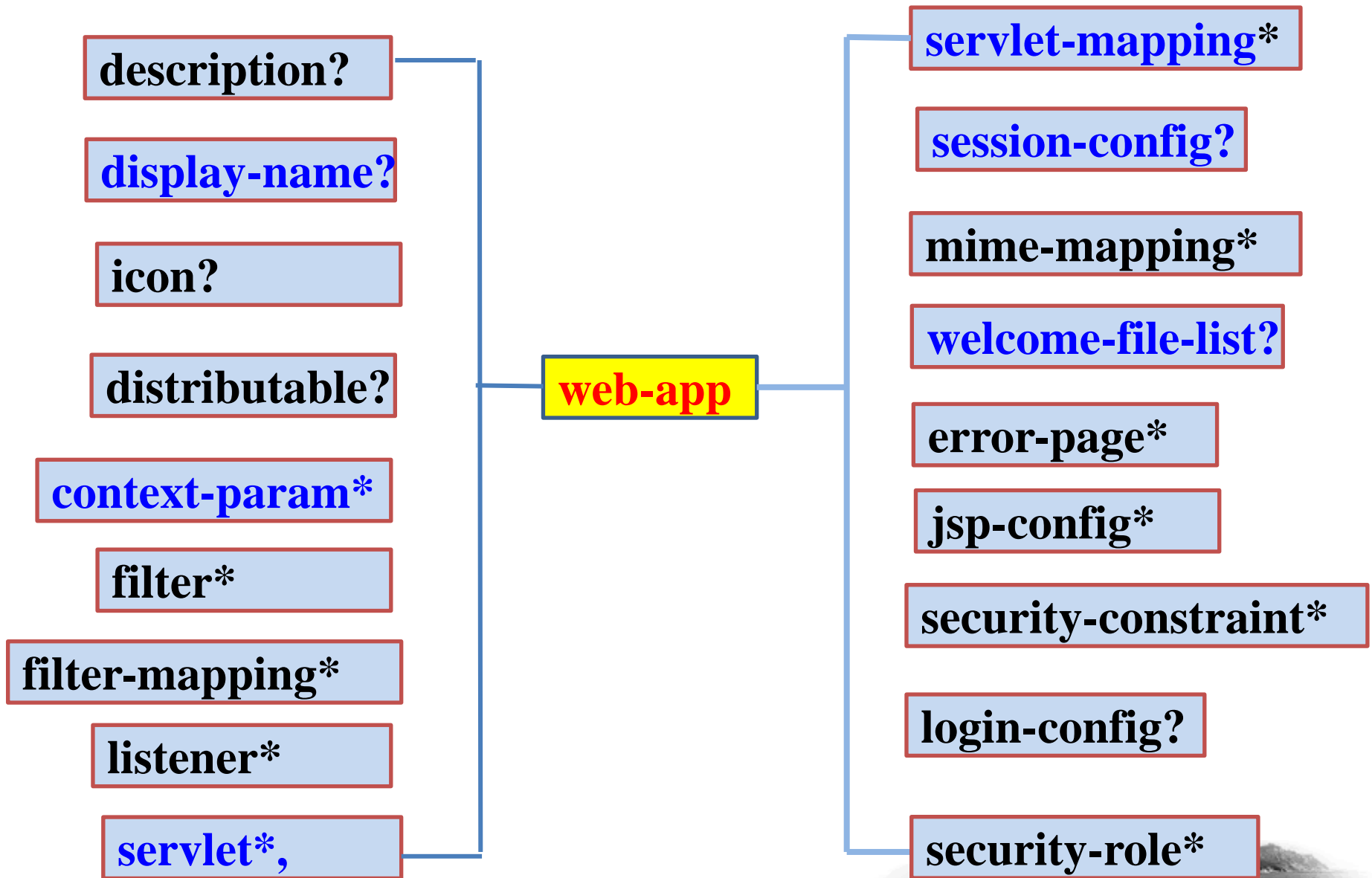
- 为了保证跨Web容器的可移植性，部署描述文件的**文档类型定义**（Document Type Definition, **DTD**）的标准由Sun公司制定。
- DTD规定了XML文档的**语法和标签的规则**，这些规则包括一系列的元素和实体的声明。

http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd

[web-app_2_5.xsd](#)

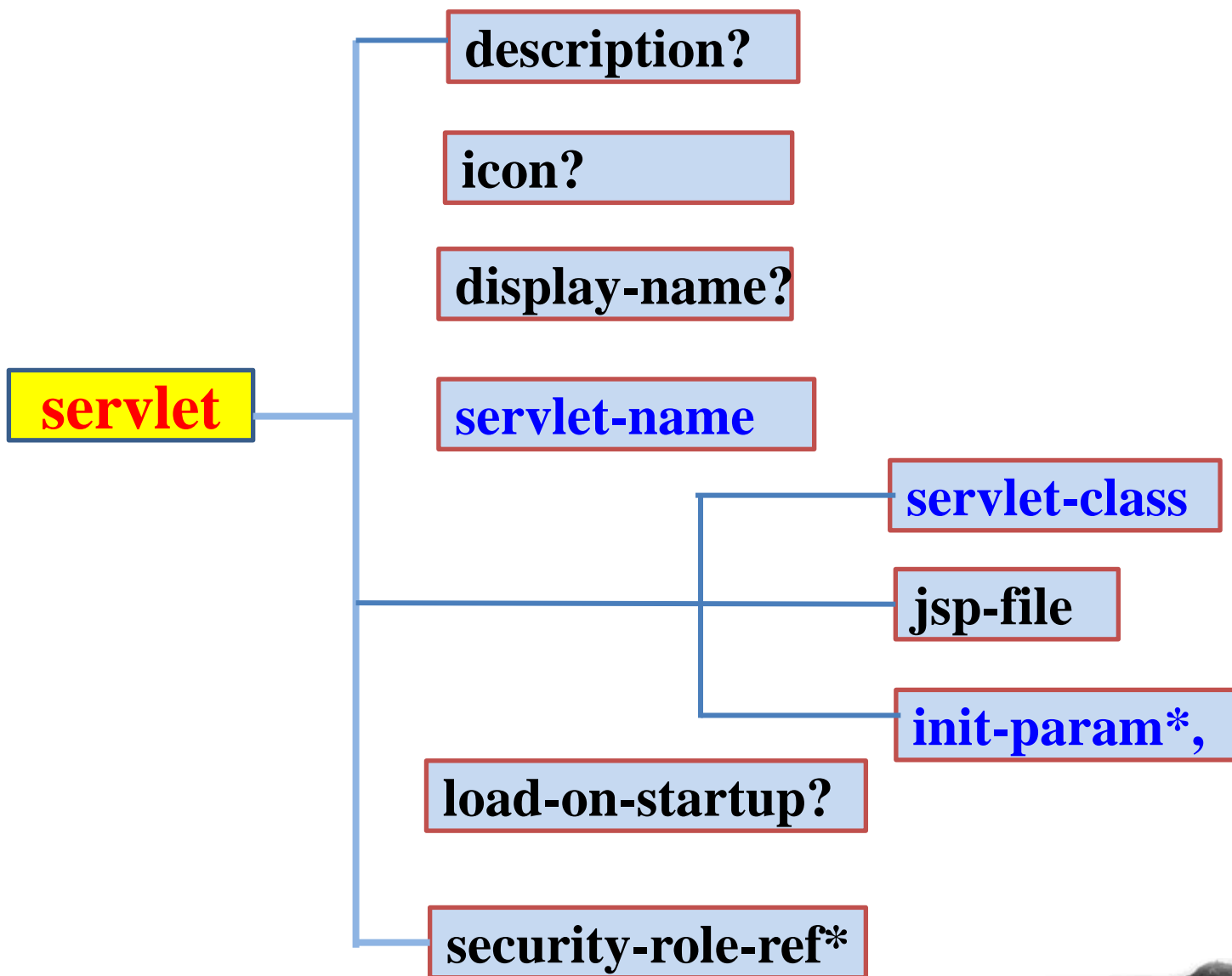


<web-app>元素的DTD定义



说明	元素名
description	对应用程序的简短描述
display-name	定义应用程序的显示名称
context-param	定义应用程序的初始化参数
servlet	定义Servlet
servlet-mapping	定义Servlet映射
welcome-file-list	定义应用程序的欢迎文件
session-config	定义会话时间
listener	定义监听器类
filter	定义过滤器
filter-mapping	定义过滤器映射
error-page	定义错误处理页面
security-constraint	定义Web应用程序的安全约束
mime-mapping	定义常用文件扩展名的MIME类型

2.6.3 使用<servlet>元素



- 下面的代码展示了在部署描述文件中<servlet>元素的一个典型的使用：

<servlet>

<servlet-name>helloServlet</servlet-name>

<servlet-class>

com.myserver.HelloServlet

</servlet-class>

<init-param>

<param-name>email</param-name>

<param-value>hacker@163.com</param-value>

</init-param>

<init-param>

<param-name>limit</param-name>

<param-value>200</param-value>

</init-param>

<load-on-startup>2</load-on-startup>

</servlet>

1. <servlet-name>元素

- 该元素用来定义**Servlet名称**，该元素是必选项。定义的名称在DD文件中应该唯一。
- 可以通过**ServletConfig**的**getServletName()**方法检索Servlet名。



2. <servlet-class>元素

- 该元素指定**Servlet**类的完整名称，即需要带包的名称，例如**com.demo.HelloServlet**。
- 容器将使用该类创建Servlet实例。
- Servlet类以及它所依赖的所有类都应该在Web应用程序的类路径中。**WEB-INF**目录中的**classes**目录和**lib**目录中的JAR文件被自动添加到容器的类路径中，因此如果把类放到这两个地方就不需要设置类路径。




3. <init-param>元素

- 该元素定义向Servlet传递的初始化参数。
- 在一个<servlet>元素中可以定义任意多个<init-param>元素。每个<init-param>元素必须有且仅有一组<param-name>和<param-value>子元素。
- Servlet可以通过ServletConfig接口的getInitParameter()方法检索初始化参数。

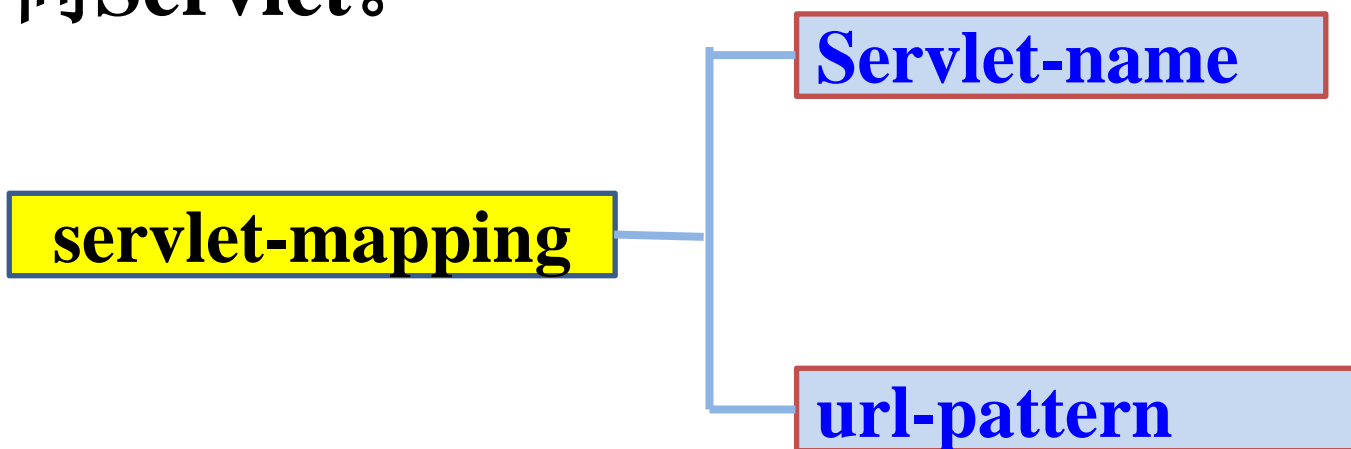


4. <load-on-startup>元素

- <load-on-startup>元素指定**是否在Web应用程序启动时载入该Servlet**。
 - 该元素的值是一个**整数**。如果没有指定该元素或其内容为一个负数，容器将根据需要决定何时装入Servlet。如果其内容为一个正数，则在Web应用程序启动时载入该Servlet。
 - 对不同的Servlet，可以指定**不同**的值，这可以控制容器装入这些Servlet的顺序，值小的先装入。
- 

2.6.4使用<servlet-mapping>元素

- <servlet-mapping>元素用来定义一个**映射**，它为Servlet分配一个**URL**，客户使用该URL访问Servlet。



<servlet-name>元素应该是使用<servlet>元素定义的Servlet名，而**<url-pattern>**可以包含要与该Servlet关联的模式字符串。

- 下面是在DD文件中使用<servlet-mapping>元素的例子：

<servlet-mapping>

<servlet-name>helloServlet**</servlet-name>**

<url-pattern>/helloServlet/hello/***</url-pattern>**

</servlet-mapping>

- 下面的URL就与上面的URL模式匹配：
 - http://www.myserver.com/mywebapp/helloServlet/hello/abc.jsp

1. URL的组成

http://www.myserver.com/mywebapp/helloServlet/hello/abc.jsp?userName=Hacker

协议与主机名

上下文路径

Servlet路径

路径信息

查询串



请求URI

getContextPath() 获得上下文路径

getServletPath() 获得Servlet路径

getPathInfo() 获得路径信息

2. <url-pattern>的三种形式

- **目录匹配**: 以斜杠 “/” 开头, 以 “/*” 结尾的形式。

`/helloServlet/hello/*`

- **扩展名匹配**: 以星号 “*.” 开始, 后接一个扩展名

`<servlet-mapping>`

`<servlet-name>pdfGeneratorServlet</servlet-name>`

`<url-pattern>*.pdf</url-pattern>`

`</servlet-mapping>`

- **精确匹配**: 所有其他字符串都作为精确匹配。

`<servlet-mapping>`

`<servlet-name>reportServlet</servlet-name>`

`<url-pattern>/report</url-pattern>`

`</servlet-mapping>`



3. 容器如何匹配url

例如: **http://www.myserver.com/mywebapp**
/helloServlet/hello/abc.jsp?userName=Hacker

```
<url-pattern>
```

```
    /helloServlet/hello/*
```

```
</url-pattern>
```

➤ 当容器接收到请求URL后，它首先解析出URI，然后从中取出第一部分作为上下文路径。这里是/mywebapp,接下来在Tomcat中查找是否有名称为 mywebapp的Web应用程序。

3. 容器如何匹配url

例如: **http://www.myserver.com/mywebapp**
/helloServlet/hello/abc.jsp?userName=Hacker

```
<url-pattern>  
    /helloServlet/hello/*  
</url-pattern>
```

➤ 如果没有名为mywebapp的Web应用程序,则上下文路径为空, 请求将发送到默认的Web应用程序。



3. 容器如何匹配url


例如： `http://www.myserver.com/mywebapp/helloServlet/hello/abc.jsp?userName=Hacker`

```
<url-pattern>
```

```
    /helloServlet/hello/*
```

```
</url-pattern>
```

➤ 如果有名为mywebapp的Web应用程序，则继续解析下一个部分，容器尝试将Servlet路径与Servlet映射匹配，如果找到一个匹配，则完整的URI请求（上下文路径部分除外）就是Servlet路径，在这种情况下，路径信息为null。



3. 容器如何匹配url

例如: `http://www.myserver.com/mywebapp/helloServlet/hello/abc.jsp?userName=Hacker`

```
<url-pattern>  
    /helloServlet/hello/*  
</url-pattern>
```

➤ 容器沿着URI路径树向下，每次一层目录，使用/作为路径分隔符，反复尝试最长的路径，看是否与一个Servlet映射匹配，如果找到一个匹配，则请求URI的匹配部分就是Servlet路径，剩下的部分是路径信息。

3. 容器如何匹配url

例如: `http://www.myserver.com/mywebapp/helloServlet/hello/abc.jsp?userName=Hacker`

```
<url-pattern>
```

```
    /helloServlet/hello/*
```

```
</url-pattern>
```

➤ 如果不能找到匹配的资源，容器向客户发送一个404错误。



2.6.5 使用<welcome-file-list>元素

❖ 欢迎页面，文件名通常为**index.html**

<http://www.baidu.com/>

□ 在Tomcat中，如果访问的URL是目录，并且没有特定的Servlet与这个URL模式匹配，那么它将在该目录中首先查找**index.html**文件，如果找不到将查找**index.jsp**文件

<welcome-file-list>

<welcome-file>[index.html](#)**</welcome-file>**

<welcome-file>[index.jsp](#)**</welcome-file>**

<welcome-file>[home.html](#)**</welcome-file>**

<welcome-file>[default.jsp](#)**</welcome-file>**

</welcome-file-list>



2.7 @WebServlet和@ WebInitParam注解

在Servlet 3.0中可以使用 **@WebServlet** 注解而不需要在web.xml文件中定义Servlet。该注解属于 **javax.servlet.annotation** 包，因此在定义Servlet时应使用下列语句导入。

```
import javax.servlet.annotation.WebServlet;
```

@WebServlet注解的常用元素



元素名	类 型	说明
Name	String	指定Servlet名称，等价web.xml中的<servlet- name>元素。如果没有显式指定， 则使Servlet的完全限定名作为名称。
urlPatterns	String[]	指定一组Servlet的URL映射模式，该元素等价于web.xml文件中的<url-pattern>元素
Value	String[]	该元素等价于< urlPatterns >元素。两个元素不能同时使用
loadOnStartup	int	指定该Servlet的加载顺序，等价于web.xml文件中的<load-on-startup>元素

元素名	类 型	说明
initParams	WebInitParam[]	指定Servlet的一组初始化参数，等价于<init-param>元素
asyncSupported	boolean	声明Servlet是否支持异步操作模式，等价于web.xml文件中的<async-supported>元素
Description	String	指定该Servlet的描述信息，等价于<description>元素
display-name	String	指定该Servlet的显示名称，等价于<display-name>元素

@WebInitParam注解的常用元素

元素名	类型	说明
Name	String	指定初始化参数名，等价于<param-name>元素
value	String	指定初始化参数值，等价于<param-value>元素
Description	String	关于初始化参数的描述，等价于<description>元素

```
@WebServlet(name = "UserServlet", //servlet名称
urlPatterns = { "/user" }, //url
loadOnStartup = 1, //启动项
initParams = { @WebInitParam(name =
"username", value = "张三") }) //初始化参数
```


2.8 ServletConfig接口

- 在 Servlet 初始化时，容器将调用 **init(ServletConfig)** 方法，并为其传递一个 ServletConfig 对象，该对象称为 **Servlet 配置对象**。
- 使用该对象可以获得 Servlet 初始化参数、Servlet 名称、ServletContext 对象等。

```
public String getInitParameter(String name)
public Enumeration getInitParameterNames()
public String getServletName()
public ServletContext getServletContext()
```

如何得到ServletConfig接口对象

□ 在Servlet中要得到ServletConfig接口对象有两种方法：

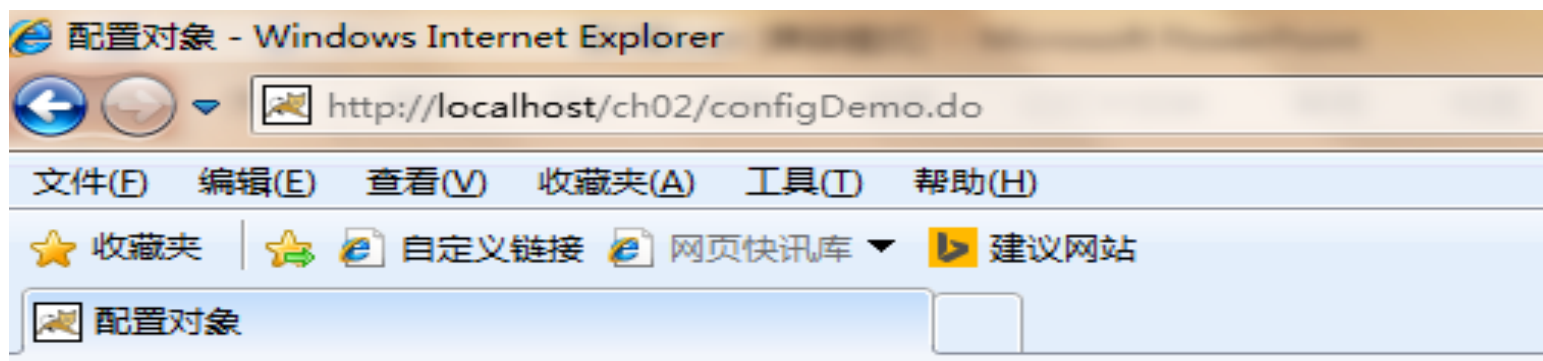
(1) 使用Servlet 的 **getServletConfig()**方法

ServletConfig config = getServletConfig();

(2) 覆盖Servlet 的 **init ServletConfig config)**方法。格式如下：

```
ServletConfig=null;  
public void init(ServletConfig config){  
    super.init(config); //必须调用超类的init()  
    方法。  
    this.config = config;}
```

- 获取Servlet的初始化参数:
- 程序2.16 [ConfigDemoServlet.java](#)
- 运行程序:
<http://localhost/ch02/configDemo.do>



Servlet名称: ConfigDemoServlet
Email地址: hacker@163.com
电话: 8899123

web.xml中如何声明Servlet初始化参数

<servlet>

<servlet-name>configDemoServlet</servlet-name>

<servlet-class> com.demo.ConfigDemoServlet

</servlet-class>

<init-param>

<param-name>email</param-name>

<param-value>hacker@163.com</param-value>

</init-param>

<init-param>

<param-name>telephone</param-name>

<param-value>8899123</param-value>

</init-param>

</servlet>



2.9 小 结

- Servlet API由4个包组成，每个包中都定义了若干接口和类，它们是开发Servlet所需的全部内容。
- 本章介绍了Servlet的执行过程和生命周期，重点介绍了请求和响应模型，其中包括如何获取请求参数、如何检索请求头以及如何发送响应。

