第3章

Servlet容器模型

本章内容

- 3.1 ServletContext接口
- 3.2 会话管理
- 3.3 Cookie及其应用

3.1 ServletContext接口

- 3.1.1 得到ServletContext引用
- 3.1.2 获取应用程序的初始化参数
- 3.1.3 通过ServletContext对象获得资源
- 3.1.4 登录日志
- 3.1.5 用RequestDispatcher实现请求转发
- 3.1.6 使用ServletContext对象存储数据
- 3.1.7 检索Servlet容器的信息

3.1.1 得到ServletContext引用

- Web容器在启动时会加载每个Web应用程序，并为每个Web应用程序创建一个唯一的ServletContext实例对象，该对象一般称为Servlet上下文对象。
- Servlet可以用javax.servlet.ServletContext对象来获得Web应用程序的初始化参数或Servlet容器的版本等信息，它也可以被Servlet用来与其他的Servlet共享数据。

3.1.1 得到ServletContext引用

- 在Servlet中有两种方法得到ServletContext引用。

1．直接调用getServletContext()方法，

```
ServletContext context = getServletContext();
```

2．先得到ServletConfig引用，再调用它的getServletContext()方法，

```
ServletContext context =
  getServletConfig().getServletContext();
```

3.1.2　获取应用程序的初始化参数

- ServletContext对象是在Web应用程序装载时初始化的。可以使用下面两个方法检索Servlet上下文初始化参数：
- public String getInitParameter(String name)：返回指定参数名的字符串参数值，如果参数不存在则返回null。
- public Enumeration getInitParameterNames()：返回一个包含所有初始化参数名的Enumeration对象。

3.1.2 获取应用程序的初始化参数

- 应用程序初始化参数应该在web.xml文件中使用<context-param>元素定义，而不能通过注解定义。下面是一个例子：

```
<context-param>
    <param-name>adminEmail</param-name>
    <param-value>webmaster@163.com</param-value>
</context-param>
```

3.1.2　获取应用程序的初始化参数

- 在Servlet中可以使用下面代码检索adminEmail参数值。

```
ServletContext context = getServletContext();
String email =
   context.getInitParameter("adminEmail");
```

3.1.3通过ServletContext对象获得资源

- public URL getResource(String path)：返回由给定路径指定的资源的URL对象。
- public InputStream getResourceAsStream(String path)：如果想从资源上获得一个InputStream对象，这是一个简洁的方法，它等价于getResource(path).openStream()。
- public String getRealPath(String path)：返回给定的相对路径的绝对路径。
- 程序3.1 FileDownloadServlet.java

3.1.4 登录日志

- 使用ServletContext接口的log()方法可以将指定的消息写到服务器的日志文件中，该方法有下面两种格式。
- public void log(String msg)：参数msg为写到日志文件中的消息。
- public void log(String msg, Throwable throwable)：将msg指定的消息和异常的栈跟踪信息写入日志文件。

3.1.5 用RequestDispatcher实现请求转发

- 使用ServletContext接口的下列两个方法也可以获得RequestDispatcher对象，实现请求转发。
- RequestDispatcher getRequestDispatcher(String path)：参数path表示资源路径，它必须以“/”开头，表示相对于Web应用的文档根目录。
- RequestDispatcher getNamedDispatcher(String name)：参数name为一个命名的Servlet对象。Servlet和JSP页面都可以通过Web应用程序的DD文件指定名称。

3.1.6 使用ServletContext对象存储数据

- 使用ServletContext对象也可以存储数据，该对象也是一个作用域对象，它的作用域是整个应用程序。在ServletContext接口中也定义了4个处理属性的方法.
- public void setAttribute(String name, Object object)：将给定名称的属性值对象绑定到上下文对象上。

3.1.6 使用ServletContext对象存储数据

- public Object getAttribute(String name)：返回绑定到上下文对象上的给定名称的属性值，如果没有该属性，则返回null。
- public Enumeration getAttributeNames()：返回绑定到上下文对象上的所有属性名的Enumeration对象。
- public void removeAttribute(String name)：从上下文对象中删除指定名称的属性。

3.1.7 检索Servlet容器的信息

- getServerInfo()方法返回Servlet所运行的容器的名称和版本。
- getMajorVersion()和getMinorVersion()方法可以返回容器所支持的Servlet API的主版本号和次版本号。
- getServletContextName()方法返回与该ServletContext对应的Web应用程序名称，它是在web.xml中使用<display-name>元素定义的名称。

3.2 会话管理

- 在很多情况下，Web服务器必须能够跟踪客户的状态。跟踪客户状态可以使用数据库实现，但在Servlet容器中通常使用会话机制维护客户状态。

3.2 会话管理

- 3.2.1 理解状态与会话
- 3.2.2 会话管理机制
- 3.2.3 HttpSession API
- 3.2.4 使用HttpSession对象
- 3.2.5 会话超时与失效

3.2.1　理解状态与会话

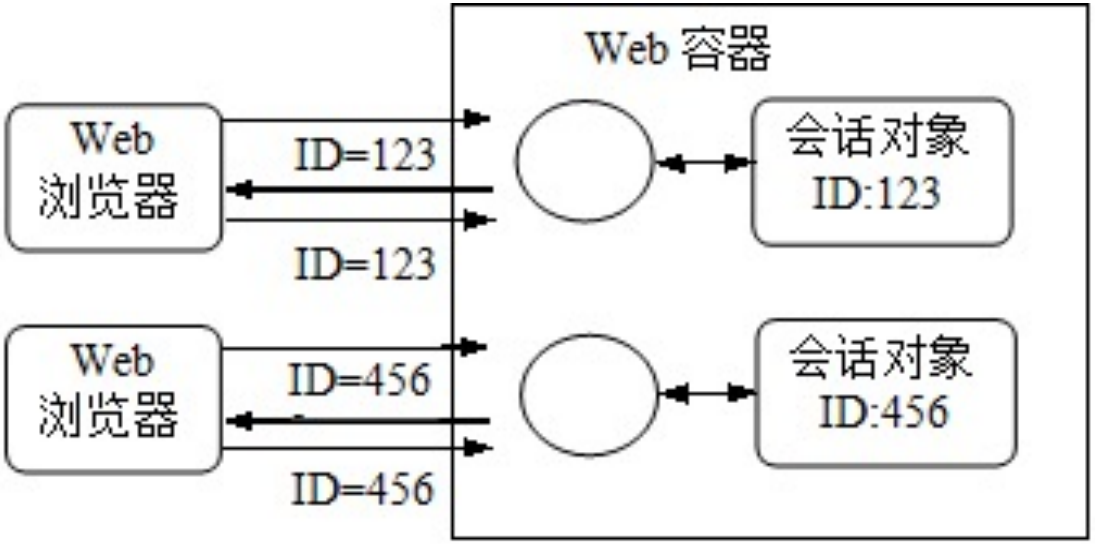
- 协议记住用户及其请求的能力称为状态（state）。按这个观点，协议分成两种类型：有状态的和无状态的。
- HTTP协议是一种无状态的协议，HTTP服务器对客户的每个请求和响应都是作为一个分离的事务处理。服务器无法确定多个请求是来自相同的客户还是不同的客户。这意味着服务器不能在多个请求中维护客户的状态。

3.2.1　理解状态与会话

- 会话（session）是一个客户与服务器之间的不间断的请求响应序列。
- 当一个客户向服务器发送第一个请求时就开始了一个会话。对该客户之后的每个请求，服务器能够识别出请求来自于同一个客户。当客户明确结束会话或服务器在一个预定义的时限内没从客户接收任何请求时，会话就结束了。当会话结束后，服务器就忘记了客户以及客户的请求。

3.2.2 会话管理机制

- 容器通过HttpSession接口抽象会话的概念。该接口由容器实现并提供了一个简单的管理用户会话的方法。



3.2.2 会话管理机制

（1）当客户向服务器发送第一个请求时，服务器就可以为该客户创建一个HttpSession会话对象，并将请求对象与该会话对象关联。服务器在创建会话对象时为其指定一个唯一标识符，称为会话ID，它可作为该客户的唯一标识。

3.2.2 会话管理机制

（2）当服务器向客户发送响应时，服务器将该会话ID与响应数据一起发送给客户，这是通过Set-Cookie响应头实现的，响应消息可能为：

HTTP/1.1 200 OK

Set-Cookie:JSESSIONID=61C4F23524521390E70993E5120263C6

Content-Type:text/html

…

- 这里，JSESSIONID的值即为会话ID，它是32位的十六进制数。

3.2.2 会话管理机制

（3）客户在接收到响应后将会话ID存储在浏览器的内存中。当客户再次向服务器发送一个请求时，它将通过Cookie请求头把会话ID与请求一起发送给服务器。这时请求消息可能为：

POST /helloweb/selectProduct.do HTTP/1.1

Host:www.mydomain.com

Cookie:
JSESSIONID=61C4F23524521390E70993E5120263C6

…

3.2.2 会话管理机制

（4）服务器接收到请求后，从请求对象中取出会话ID，在服务器中查找之前创建的会话对象，找到后将该请求与之前创建的ID值相同的会话对象关联起来。

- 上述过程的第（2）到第（4）步一直保持重复。
- 如果客户在指定时间没有发送任何请求，服务器将使会话对象失效。一旦会话对象失效，即使客户再发送同一个会话ID，会话对象也不能恢复。

3.2.2 会话管理机制

- 通过会话机制可以实现购物车应用。当用户登录购物网站时，Web容器就为客户创建一个HttpSession对象。
- 实现购物车的Servlet使用该会话对象存储用户的购物车对象，购物车中存储着用户购买的商品列表。当客户向购物车中添加商品或删除商品时，Servlet就更新该列表。当客户要结账时，Servlet就从会话中检索购物车对象，从购物车中检索商品列表并计算总价格。一旦客户结算完成，容器就会关闭会话。

3.2.2 会话管理机制

- 注意，不能使用客户的IP地址唯一标识客户。因为，客户可能是通过局域网访问Internet。尽管在局域网中每个客户有一个IP地址，但对于服务器来说，客户的实际IP地址是路由器的IP地址，所以该局域网的所有客户的IP地址都相同！因此也就无法唯一标识客户。

3.2.3 HttpSession API

- 下面是HttpSession接口中定义的常用方法。
- public String getId()：返回为该会话指定的唯一标识符，它是一个32位的十六进制数。
- public long getCreationTime()：返回会话创建的时间。时间为从1970年1月1日午夜到现在的毫秒数。
- public long getLastAccessedTime()：返回会话最后被访问的时间。
- public boolean isNew()：如果会话对象还没有同客户关联，则返回true。

3.2.3 HttpSession API

- public ServletContext getServletContext()：返回该会话所属的ServletContext对象。
- public void setAttribute (String name, Object value)：将一个指定名称和值的属性存储到会话对象上。
- public Object getAttribute(String name)：返回存储到会话上的指定名称的属性值，如果没有指定名称的属性，则返回null。
- public Enumeration getAttributeNames()：返回存储在会话上的所有属性名的一个枚举对象。

3.2.3 HttpSession API

- public void setMaxInactiveInterval(int interval)：设置在容器使该会话失效前客户的两个请求之间最大间隔的时间，单位为秒。参数为负值表示会话永不失效。
- public int getMaxInactiveInterval()：返回以秒为单位的最大间隔时间，在这段时间内，容器将在客户请求之间保持该会话打开状态。
- public void invalidate()：使会话对象失效并删除存储在其上的任何对象。

3.2.4 使用HttpSession对象

- 使用HttpSession对象通常需要三步：

（1）创建或返回与客户请求关联的会话对象。

（2）在会话对象中添加或删除“名/值”对属性。

（3）如果需要可使会话失效。

- 创建或返回HttpSession对象需要使用HttpServletRequest接口提供的getSession()方法，该方法有两种格式：

3.2.4 使用HttpSession对象

- public HttpSession getSession(boolean create)：返回或创建与当前请求关联的会话对象。
- public HttpSession getSession()：该方法与调用getSession(true)等价。

3.2.5 会话超时与失效

- 可以在DD文件中设置会话超时时间。

```
<session-config>
    <session-timeout>10</session-timeout>
</session-config>
```

- <session-timeout>元素中指定的以分钟为单位的超时期限。0或小于0的值表示会话永不过期。如果没有通过上述方法设置会话的超时期限，默认情况下是30分钟。

3.2.5 会话超时与失效

- 在DD文件中设置的会话超时时间针对Web应用程序中的所有会话对象，但有时可能需要对特定的会话对象指定超时时间，可使用会话对象的setMaxInactiveInterval()。要注意，该方法仅对调用它的会话有影响，其他会话的超时期限仍然是DD文件中设置的值。
- 在某些情况下，可能希望通过编程的方式结束会话。例如，在购物车的应用中，我们希望在用户付款处理完成后结束会话。这样，当客户再次发送请求时，就会创建一个购物车中不包含商品的新的会话。可使用HttpSession接口的invalidate()。

3.2.5 会话超时与失效

- 下面是一个猜数游戏的Servlet。当使用GET请求访问它时，生成一个在0~100之间的随机整数，将其作为一个属性存储到用户的会话对象中，同时提供一个表单供用户输入猜测的数。
- 如果该Servlet接收到一个POST请求，它将比较用户猜的数和随机生成的数是否相等，若相等在响应页面中给出信息，否则，应该告诉用户猜的数是大还是小，并允许用户重新猜。
- 程序3.3 GuessNumberServlet.java

3.3 Cookie及其应用

- 3.3.1 Cookie API
- 3.3.2 向客户端发送Cookie
- 3.3.3 从客户端读取Cookie
- 3.3.4 Cookie的安全问题
- 3.3.5 实例：用Cookie实现自动登录

3.3.1 Cookie API

- Cookie是客户访问Web服务器时，服务器在客户硬盘上存放的信息，好像是服务器送给客户的“点心”。
- Cookie实际上是一小段文本信息，客户以后访问同一个Web服务器时浏览器会把它们原样发送给服务器。
- 通过让服务器读取它原先保存到客户端的信息，网站能够为浏览者提供一系列的方便，例如，在线交易过程中标识用户身份、安全要求不高的场合避免客户登录时重复输入用户名和密码等等。

3.3.1 Cookie API

- 对Cookie的管理需要使用javax.servlet.http.Cookie类，构造方法如下：

  public Cookie(String name, String value)

- 参数name为Cookie名，value为Cookie的值，它们都是字符串。
- Cookie类的常用方法如下：

  public String getName()：返回Cookie名称，名称一旦创建不能改变。

  public String getValue()：返回Cookie的值。

  public void setValue(String newValue)：在Cookie创建后为它指定一个新值。

3.3.1 Cookie API

- public void setMaxAge(int expiry)：设置Cookie在浏览器中的最长存活时间，单位为秒。
- public int getMaxAge()：返回Cookie在浏览器上的最大存活时间。
- public void setDomain(String pattern)：设置该Cookie所在的域。
- public String getDomain()：返回为该Cookie设置的域名。

Cookie的管理包括两个方面：将Cookie对象发送到客户端和从客户端读取Cookie。

3.3.2 向客户端发送Cookie

- 要把Cookie发送到客户端，Servlet先要使用Cookie类的构造方法创建一个Cookie对象，通过set*Xxx*()方法设置各种属性，通过响应对象的addCookie(cookie)方法把Cookie加入响应头。具体步骤如下：

1）创建Cookie对象

```
Cookie userCookie = new Cookie("username",
   "hacker");
```

3.3.2　向客户端发送Cookie

2）如果希望浏览器将Cookie对象存储到磁盘上，需要使用Cookie类的setMaxAge()方法设置Cookie的最大存活时间。

```
userCookie.setMaxAge(60*60*24*7);
```

3）向客户发送Cookie对象

```
response.addCookie(userCookie);
```

- 程序3.4　SendCookieServlet.java

3.3.3 从客户端读取Cookie

- 要从客户端读入Cookie，Servlet应该调用请求对象的getCookies()，该方法返回一个Cookie对象的数组。大多数情况下，只需要用循环访问该数组的各个元素寻找指定名字的Cookie，然后对该Cookie调用getValue()取得与指定名字关联的值。
- 具体步骤如下：

3.3.3 从客户端读取Cookie

1）调用请求对象的getCookies方法

- 该方法返回一个Cookie对象的数组。如果请求中不含Cookie，返回null值。

```
Cookie[] cookies=request.getCookies();
```

2）对Cookie数组循环

- 有了Cookie对象数组后，就可以通过循环访问它的每个元素，然后调用每个Cookie的getName()，直到找到一个与希望的名称相同的对象为止。找到所需要的Cookie对象后，一般要调用它的getValue()，并根据得到的值做进一步处理。

3.3.3 从客户端读取Cookie

程序3.5 ReadCookieServlet.java

3.3.4 Cookie的安全问题

- Cookie是服务器向客户机上写的数据，因此有些用户认为Cookie会带来安全问题，认为Cookie会带来病毒。事实上，Cookie并不会造成安全威胁，Cookie永远不会以任何方式执行。
- 另外，由于浏览器一般只允许存放300个Cookie，每个站点的Cookie最多存放20个，每个Cookie的大小限制为4 KB，因此Cookie不会占据硬盘多大空间。
- 为了保证安全，许多浏览器还是提供了设置是否使用Cookie的功能。

3.3.4 Cookie的安全问题

- 在IE浏览器中打开“工具”菜单中的“Internet选项”对话框，在“隐私”选项卡中可以设置浏览器是否接受Cookie，如图3-7所示。
- 在该对话框中可以通过一个滑块设置浏览器接收Cookie的级别。其中有6个级别
- 注意，即使客户将Cookie设置为“阻止所有Cookie”，浏览器仍然自动支持会话级的Cookie。

3.3.5　实例：用Cookie实现自动登录

- 许多网站都提供用户自动登录功能，即用户第一次登录网站，服务器将用户名和密码以Cookie的形式发送到客户端。
- 当客户之后再次访问该网站时，浏览器自动将Cookie文件中的用户名和密码随请求一起发送到服务器，服务器从Cookie中取出用户名和密码并且通过验证，这样客户不必再次输入用户名和密码登录网站，这称为自动登录。
- 程序3.6　login.jsp
- 程序3.7　CheckUserServlet.java

3.4 小 结

- 容器在启动时会加载每个Web应用程序，并为每个创建唯一的ServletContext对象。
- 通过使用HttpSession对象可以跟踪客户与服务器的交互，Web应用程序需要在本来无状态的HTTP协议上实现状态。
- 服务器通过为其指定一个唯一的标识符实现一个会话。