第3章 Servlet 容器模型

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。

本章首先介绍 ServletContext 接口，通过该对象可以存储应用作用域数据、登录日志以及获得服务器资源等。接下来介绍使用 HttpSession 实现会话管理机制，最后介绍 Cookie 及其应用。

3.1 ServletContext 接口

Servlet 可以使用 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 中可以有两种方法得到 ServletContext 引用。直接调用 getServletContext()，例如：

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

还可以先得到 ServletConfig 引用，再调用它的 getServletContext()，例如：

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

得到 ServletContext 引用后，就可以使用 ServletContext 接口定义的方法，检索 Web 应用程序的初始化参数、检索 Servlet 容器的版本信息、通过属性共享数据以及登录日志等。

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。正像 Servlet 具有初始化参数一样，ServletContext 也有初始化参数。Servlet 上下文初始化参数指定应用程序范围内的信息，如开发人员的联系信息以及数据库连接信息等。

可以使用下面两个方法检索 Servlet 上下文初始化参数。

(1) 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。

(2) public Enumeration getInitParameterNames()：返回一个包含所有初始化参数名的 Enumeration 对象。

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>
```

注意，<context-param>元素是针对整个应用的，所以并不嵌套在某个<servlet>元素中。该元素是<web-app>元素的直接子元素。

在 Servlet 中可以使用下面的代码检索 adminEmail 参数值。

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

注意：Servlet 上下文初始化参数与 Servlet 初始化参数是不同的。Servlet 上下文初始化参数是属于 Web 应用程序的，可以被 Web 应用程序的所有的 Servlet 和 JSP 页面访问。Servlet 初始化参数是属于定义它们的 Servlet 的，不能被 Web 应用程序的其他组件访问。

3.1.3 通过 ServletContext 对象获得资源

本节将讨论 getResource()和 getResourceAsStream()，Servlet 使用这些方法可以访问任何资源而不必关心资源所处的位置。

(1) public URL getResource(String path)：返回由给定路径指定的资源的 URL 对象。这里，路径必须以“/”开头，它相对于该 Web 应用程序的文档根目录。

(2) public InputStream getResourceAsStream(String path)：如果想从资源上获得一个 InputStream 对象，这是一个简捷的方法，它等价于 getResource(path).openStream()。

(3) public String getRealPath(String path)：返回给定的相对路径的绝对路径。

下面程序使用 getResourceAsStream ()获得指定资源的输入流对象，然后从中读取数据，然后写到输出流中。注意，这里资源的路径使用的是相对路径。

程序 3.1 FileDownloadServlet.java

```
package com.demo;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/fileDownload.do")
public class FileDownloadServlet extends HttpServlet{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                     throws ServletException, IOException {
        //设置文件的内容类型
        response.setContentType("image/gif");
        //设置 Content-Disposition 响应头，指定文件名
        response.setHeader("Content-Disposition",
                           "attachment;filename=duke.gif");
        //获得输出流对象
        OutputStream os = response.getOutputStream();
        ServletContext context = getServletContext();
        //返回输入流对象
        InputStream is =
                context.getResourceAsStream("/files/duke.gif");
        byte[] bytearray = new byte[1024];
        int bytesread = 0;
        //从输入流中读取 1KB，然后写到输出流中
        while((bytesread = is.read(bytearray)) != -1 ){
            //将数据发送到客户端
            os.write(bytearray, 0, bytesread);
        }
        os.flush();
        is.close();
    }
}
```

在上述程序中,为 duke.gif 文件指定了一个相对路径,它相对于 Web 应用程序的文档根目录。只要 duke.gif 文件位于 Web 应用程序的 files 目录中,Servlet 就可以找到它。

访问该 Servlet,首先打开如图 3-1 所示“文件下载”对话框,单击“保存”按钮可以将文件下载到本地磁盘。

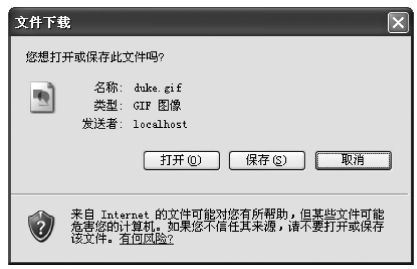

图 3-1 “文件下载”对话框

下面是 getResource()和 getResourceAsStream()的局限:

(1) 不能传递任何活动资源的 URL,例如将 JSP 页面或 Servlet 传递给该方法。

(2) 如果使用不当,该方法可能成为安全漏洞,它可以读取属于 Web 应用程序的所有文件,包括 WEB-INF 目录下的文件。

当然,Servlet 可以通过使用 ServletContext 的 getRealPath(String path)把相对路径转换为绝对路径访问一个资源。

也可以编写一个 HTML 页面实现文件下载,通过超链接访问该 Servlet,代码如下。

```
下载文件<a href = "fileDownload.do">duke.gif</a>
```

单击该超链接,浏览器同样打开“文件下载”对话框。

3.1.4 登录日志

使用 ServletContext 接口的 log()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。

(1) public void log(String msg):参数 msg 为写到日志文件中的消息。默认情况下把日志信息写到<tomcat-install>\logs\localhost.2013-09-10.log 文件中,文件名中的日期为写入日志的日期。

(2) 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 使用 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。

(1) 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。如果不能返回一个 RequestDispatcher 对象,该方法将返回 null。

(2) 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。

ServletContext 和 ServletRequest 的 getRequestDispatcher()的区别是:对 ServletContext 的 getRequestDispatcher()只能传递以“/”开头的路径,而对 ServletRequest 的 getRequestDispatcher(),可以传递一个相对路径。

例如,request.getRequestDispatcher("../html/copyright.html")是合法的,该方法相对于请求的路径计算路径。

3.1.6 使用 ServletContext 对象存储数据

前面讨论了使用请求对象存储数据的方法。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，如下所示。

(1) 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。

(2) 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。

(3) public Enumeration getAttributeNames()：返回绑定到上下文对象上的所有属性名的 Enumeration 对象。

(4) public void removeAttribute(String name)：从上下文对象中删除指定名称的属性。

在 3.2 节要讨论的 HttpSession 接口中也定义了 4 个同样的方法，也就是在 HttpSession 对象中也可以共享对象。

尽管可以使用这些容器的任何一个共享数据，但在这些容器中的数据具有不同的作用域。简单地说，使用 HttpServletRequest 共享的对象仅在请求的生存期中可被访问，使用 HttpSession 共享的对象仅在会话的生存期中可被访问，而使用 ServletContext 共享的对象可在 Web 应用程序的生存期中可被访问。

3.1.7 检索 Servlet 容器的信息

检索容器有关信息的方法如下：getServerInfo()返回 Servlet 所运行的容器的名称和版本。getMajorVersion()和 getMinorVersion()可以返回容器所支持的 Servlet API 的主版本号和次版本号。getServletContextName()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。

3.2.1 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。

1. HTTP 的无状态特性

HTTP 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。

在某些情况下服务器不需要记住客户，HTTP 无状态的特性对这样的应用会工作得很好。例如，一个在线图书馆目录就不需要维护客户的状态。但某些 Web 应用程序中客户与服务器的交互中就需要有状态的。一个典型的例子是购物车应用。一个客户可以多次向购物车中添加商品，也可以清除商品。在处理过程中，服务器应该能够显示购物车中的商品并计算总价格。为了实现这一点，服务器必须跟踪所有的请求并把它们与客户关联。

2. 会话的概念

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 javax.servlet.http.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。容器使用 HttpSession 对象管理会话的过程如图 3-2 所示。

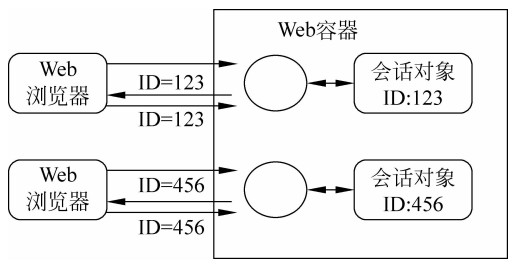

图 3-2 会话管理示意图

(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。此时,该会话处于新建状态,可以使用 HttpSession 接口的 isNew()来确定会话是否属于该状态。

(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) 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
…
```

(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)~(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。对于服务器来说,此时客户的请求被认为是第一次请求(如第 1 步),它不与某个存在的会话对象关联。服务器可以为客户创建一个新的会话对象。

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。如果用户再发送另一个请求,就会创建一个新的会话。显然,有多少个会话,服务器就会创建多少个 HttpSession 对象。换句话说,对每个会话(用户)都有一个对应的 HttpSession 对象。然而,无须担心 HttpSession 对象与客户的关联,容器会为我们做这一点,一旦接收到请求,它会自动返回合适的会话对象。

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。

(1) 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。

(2) 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。

(3) public long getLastAccessedTime()：返回会话最后被访问的时间。

(4) public boolean isNew()：如果会话对象还没有同客户关联，则返回 true。

(5) public ServletContext getServletContext()：返回该会话所属的 ServletContext 对象。

(6) 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。

(7) 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。

(8) public Enumeration getAttributeNames()：返回存储在会话上的所有属性名的一个枚举对象。

(9) public void removeAttribute(String name)：从会话中删除存储的指定名称的属性。

(10) 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。

(11) 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。

(12) 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()，该方法有以下两种格式。

(1) public HttpSession getSession(boolean create)：返回或创建与当前请求关联的会话对象。如果没有与当前请求关联的会话对象，当参数为 true 时创建一个新的会话对象，当参数为 false 时返回 null。

(2) public HttpSession getSession()：该方法与调用 getSession(true)等价。

下面的 Servlet 可显示客户会话的基本信息。程序调用 request.getSession()获取现存的会话，在没有会话的情况下创建新的会话。然后，在会话对象上查找类型为 Integer 的 accessCount 属性。如果找不到这个属性，则使用 1 作为访问计数。然后，对这个值进行递增，并用 setAttribute()与会话关联起来。

程序 3.2 ShowSessionServlet.java

```
package com.demo;
```

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Date;
import javax.servlet.annotation.WebServlet;

@WebServlet("/ShowSessionServlet")
public class ShowSessionServlet extends HttpServlet{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                       throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        //创建或返回用户会话对象
        HttpSession session = request.getSession(true);
        String heading = null;
        //从会话对象中检索 accessCount 属性
        Integer accessCount = (Integer)session.getAttribute("accessCount");
        if(accessCount==null){
             accessCount = new Integer(1);
            heading = "欢迎您,首次登录该页面!";
        }else{
            heading = "欢迎您,再次访问该页面!";
            accessCount = accessCount+1;
        }
        //将 accessCount 作为属性存储到会话对象中
        session.setAttribute("accessCount",accessCount);
        PrintWriter out = response.getWriter();
        out.println("<html><head>");
        out.println("<title>会话跟踪示例</title></head>");
        out.println("<body><center>");
        out.println("<h4>" + heading
             + "<a href='ShowSessionServlet'>再次访问</a>" + "</h4>");
        out.println("<table border='0'>");
        out.println("<tr bgcolor=\"ffad00\"><td>信息</td><td>值</td>\n");
        String state = session.isNew()?"新会话":"旧会话";
        out.println("<tr><td>会话状态: <td>" + state + "\n");
        out.println("<tr><td>会话 ID:<td>" + session.getId() + "\n");
        out.println("<tr><td>创建时间:<td>");
        out.println("" + new Date(session.getCreationTime()) + "\n");
        out.println("<tr><td>最近访问时间:<td>");
        out.println("" + new Date(session.getLastAccessedTime()) + "\n");
        out.println("<tr><td>最大不活动时间:<td>" +
        session.getMaxInactiveInterval() + "\n");
        out.println("<tr><td>Cookie:<td>" + request.getHeader("Cookie") + "\n");
        out.println("<tr><td>已被访问次数:<td>" + accessCount + "\n");
        out.println("</table>");
        out.println("</center></body></html>");
    }
}
```

第一次访问该 Servlet,将显示如图 3-3 所示的页面,此时计数变量 accessCount 值为 1,Cookie 请求头的值为 null。再次访问页面(单击“再次访问”链接或通过刷新页面)显示结果如图 3-4 所示,计数变量 accessCount 值增 1,但会话 ID 的值相同。如果再打开一个浏览器窗口访问该 Servlet,计数变量仍从 1 开始,因为又开始了一个新的会话,服务器将为该会话创建一个新的会话对象并分配一个新的会话 ID。

注意,这里没有写任何代码来标识用户,仅调用了请求对象的 getSession()并假设每次处理同一客户的请求时都返回相同的 HttpSession 对象。如果请求是用户的首次请求,不能使

图 3-3　首次访问结果

图 3-4　再次访问结果

用会话，这时，将为新用户创建一个新的会话。

3.2.5　会话超时与失效

会话对象会占用一定的系统资源，我们不希望会话不必要地长久保留。然而，HTTP 没有提供任何机制让服务器知道客户已经离开，但可以规定当用户在一个指定的期限内处于不活动状态时，就将用户的会话终止，这称为会话超时(session timeout)。

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 30min。如果用户在指定期间内没有执行任何动作，服务器就认为用户处于不活动状态并使会话对象无效。

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()。

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

```
package com.demo;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/GuessNumberServlet")
public class GuessNumberServlet extends HttpServlet{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {
      int magic = (int)(Math.random()*101);
      HttpSession session = request.getSession();
      session.setAttribute("num",new Integer(magic));

      response.setContentType("text/html;charset=utf-8");
      PrintWriter out = response.getWriter();
      out.println("<html><body>");
      out.println("我想出一个 0 到 100 之间的数,请你猜!");
      out.println("<form action='/helloweb/GuessNumberServlet'
                   method='post'>");
      out.println("<input type='text' name='guess' />");
      out.println("<input type='submit' value='确定'/>");
      out.println("</form>");
      out.println("</body></html>");
    }

    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                   throws ServletException, IOException {
      int guess = Integer.parseInt(request.getParameter("guess"));
      HttpSession session = request.getSession();
      int magic = (Integer)session.getAttribute("num");

      response.setContentType("text/html;charset=utf-8");
      PrintWriter out = response.getWriter();
      out.println("<html><body>");
      if(guess==magic){
           session.invalidate(); //销毁会话对象
           out.println("祝贺你,答对了!");
           out.println("<a href = '/helloweb/GuessNumberServlet'>
                        再猜一次.</a>");
      }else if(guess>magic){
           out.println("太大了! 请重猜!");
      }else{
           out.println("太小了! 请重猜!");
      }
      out.println("<form action='/helloweb/GuessNumberServlet'
                   method='post'>");
      out.println("<input type='text' name='guess' />");
      out.println("<input type='submit' value='确定'/>");
      out.println("</form>");
```

```
            out.println("</body></html>");
        }
    }
```

程序中当用户猜对时调用了会话对象的 invalidate()使会话对象失效，再通过链接发送一个 GET 请求允许用户再继续猜。程序运行结果如图 3-5 和图 3-6 所示。

图 3-5　GET 请求显示的页面

图 3-6　猜正确的页面

3.3　Cookie 及其应用

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。

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。

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

(2) public String getValue()：返回 Cookie 的值。

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

(4) public void setMaxAge(int expiry)：设置 Cookie 在浏览器中的最长存活时间，单位为秒。如果参数值为负，表示 Cookie 并不永久存储，如果是 0 表示删除该 Cookie。

(5) public int getMaxAge()：返回 Cookie 在浏览器上的最大存活时间。

(6) public void setDomain(String pattern)：设置该 Cookie 所在的域。域名以点号(.)开头，例如，.foo.com。默认情况下，只有发送 Cookie 的服务器才能得到它。

(7) 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 setXxx()设置各种属性，通过响应对象的 addCookie(cookie)把 Cookie 加入响应头。具体步骤如下。

1. 创建 Cookie 对象

调用 Cookie 类的构造方法可以创建 Cookie 对象。下面的语句创建了一个 Cookie 对象：

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

2. 设置 Cookie 的最大存活时间

在默认情况下,发送到客户端的 Cookie 对象只是一个会话级别的 Cookie,它存储在浏览器的内存中,用户关闭浏览器后 Cookie 对象将被删除。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()设置 Cookie 的最大存活时间。下面的代码将 userCookie 对象的最大存活时间设置为一个星期。

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

3. 向客户发送 Cookie 对象

要将 Cookie 对象发送到客户端,需要调用响应对象的 addCookie()将 Cookie 添加到 Set-Cookie 响应头,如以下代码所示。

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

下面的 Servlet 向客户发送一个 Cookie 对象。

程序 3.4 SendCookieServlet.java

```
package com.demo;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/SendCookie")
public class SendCookieServlet extends HttpServlet{
   public void doGet(HttpServletRequest request,
                     HttpServletResponse response)
                    throws IOException,ServletException{
      Cookie userCookie = new Cookie("username", "hacker");
      userCookie.setMaxAge(60 * 60 * 24 * 7);
      response.addCookie(userCookie);

      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      out.println("<html><title>发送 Cookie</title>");
      out.println("<body><h3>已向浏览器发送一个 Cookie.</h3></body>");
      out.println("</html>");
   }
}
```

访问该 Servlet,服务器将在浏览器上写一个 Cookie 文件,该文件是一个文本文件,一般存放在 C:\Document and Settings*username*\Cookies 文件夹中。

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。

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。例如:

程序 3.5 ReadCookieServlet.java

```
package com.demo;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/ReadCookie")
public class ReadCookieServlet extends HttpServlet{
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws IOException,ServletException{
        String cookieName = "username";
        String cookieValue = null;
        Cookie[] cookies = request.getCookies();
        if (cookies!= null){
          for(int i = 0;i<cookies.length;i++){
            Cookie cookie = cookies[i];
            if(cookie.getName().equals(cookieName))
              cookieValue = cookie.getValue();
          }
        }
      response.setContentType("text/html;charset=utf-8");
      PrintWriter out=response.getWriter();
      out.println("<html><title>读取 Cookie</title>");
      out.println("<body><h3>从浏览器读回一个 Cookie</h3>");
      out.println("Cookie 名:" + cookieName + "<br>");
      out.println("Cookie 值:" + cookieValue + "<br>");
      out.println("</body></html>");
    }
}
```

访问该 Servlet 将从客户端读回此前写到客户端的 Cookie。

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 4KB,因此 Cookie 不会占据硬盘太多空间。

为了保证安全,许多浏览器还提供了设置是否使用 Cookie 的功能。例如,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。

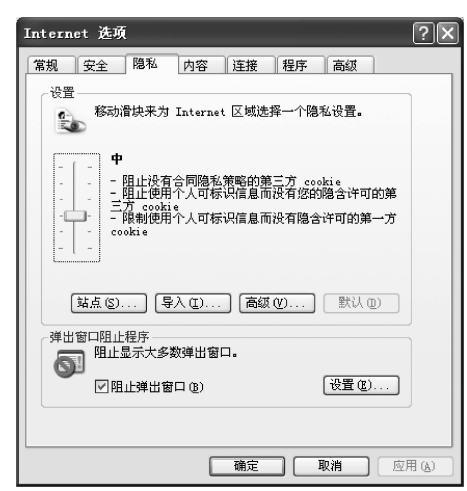

图 3-7 “Internet 选项”对话框的“隐私”选项卡

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,将滑块移到最上方,浏览器将阻止所有的 Cookie,即来自所有网

站的 Cookie 都将被阻止,并且计算机上现有的 Cookie 不能被网站读取。将滑块移到最下方,浏览器将接收所有的 Cookie,即所有 Cookie 都将被保存到计算机上,其他级别处于二者之间。

在“Internet 选项”对话框的“常规”选项卡中,单击“删除 Cookies”按钮可以删除所有 Cookies。单击“设置”按钮,在打开的对话框中单击“查看文件”按钮,就会打开存放 Cookie 文件的临时文件夹,可以有选择地删除 Cookie。

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。下面的 login.jsp 是登录页面。

程序 3.6 login.jsp

```
<%@ page contentType="text/html;charset=UTF-8"
         pageEncoding="UTF-8"%>
<html>
<head><title>登录页面</title></head>
<body>
 ${sessionScope.message}<br>
  <form action="CheckUserServlet" method="post">
       请输入用户名和口令:<br>
       用户名:<input type="text" name="username" /><br>
       口 令:<input type="password" name="password" /><br>
       <input type="checkbox" name="check" value="check" />自动登录<br>
       <input type="submit" value="提交"/>
       <input type="reset" value="重置"/>
    </form>
</body>
</html>
```

该 JSP 页面运行结果如图 3-8 所示。

图 3-8 login.jsp 页面运行结果

下面是 CheckUserServlet 的代码。

程序 3.7 CheckUserServlet.java

```
package com.demo;
import java.io.IOException;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet("/CheckUserServlet")
public class CheckUserServlet extends HttpServlet {
    String message = null;
    protected void doGet(HttpServletRequest request,
                         HttpServletResponse response)
                         throws ServletException, IOException {
        response.setContentType("text/html;charset=utf-8");
        String value1 = "",value2 = "";
        Cookie cookie = null;
        Cookie[] cookies = request.getCookies();
        if (cookies!=null){
           for(int i = 0;i<cookies.length;i++){
             cookie = cookies[i];
```

```
            if(cookie.getName().equals("username"))
              value1 = cookie.getValue();
            if(cookie.getName().equals("password"))
              value2 = cookie.getValue();
          }
           if(value1.equals("admin")&&value2.equals("admin")){
              message = "欢迎您!" + value1 + "再次登录该页面!";
              request.getSession().setAttribute("message", message);
              response.sendRedirect("welcome.jsp");
           }else{
              response.sendRedirect("login.jsp");
           }
        }else{
          response.sendRedirect("login.jsp");
        }
    }
    protected void doPost(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
      response.setContentType("text/html;charset=utf-8");
      String username = request.getParameter("username").trim();
      String password = request.getParameter("password").trim();
      if(!username.equals("admin") || !password.equals("admin")){
          message = "用户名或口令不正确,请重试!";
          request.getSession().setAttribute("message",message);
          response.sendRedirect("login.jsp");
      }else{
          //如果用户选中了"自动登录"复选框,向浏览器发送两个 Cookie
          if((request.getParameter("check")!= null) &&
                (request.getParameter("check").equals("check"))){
            Cookie nameCookie = new Cookie("username", username);
            Cookie pswdCookie = new Cookie("password", password);
            nameCookie.setMaxAge(60*60);
            pswdCookie.setMaxAge(60*60);
            response.addCookie(nameCookie);
            response.addCookie(pswdCookie);
          }
        message = "你已成功登录!";
        request.getSession().setAttribute("message",message);
        response.sendRedirect("welcome.jsp");
      }
    }
}
```

以 GET 方法访问 CheckUserServlet(在浏览器地址栏中输入该 Servlet 的 URL 地址),由于是首次访问,请求中并不包含 Cookie,该 Servlet 将响应重定向到 login.jsp 页面。在该页面中如果用户输入了正确的用户名和口令,且选中“自动登录”复选框,单击“提交”按钮,将发送 POST 请求由 CheckUserServlet 的 doPost()处理。在该方法中使用用户名和口令创建两个 Cookie 对象并发送到客户端。

之后再发送 GET 请求,Servlet 将从 Cookie 中检索出用户名和口令,并对其验证。验证通过后将响应重定向到 welcome.jsp 页面。

小　　结

容器在启动时会加载每个 Web 应用程序,并为每个 Web 应用创建唯一的 ServletContext 对象。使用该对象可以获得 Web 应用程序的初始化参数或 Servlet 容器的版本等信息,也可

以用来共享数据、获得资源输入流、实现请求转发与登录日志等。

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 上实现状态。一个会话是一个完整的客户与服务器的交互序列,在一个会话期间,服务器会记住客户并将所有来自该客户的请求与客户的唯一会话对象关联。

服务器通过为其指定一个唯一的标识符实现一个会话。使用 Cookie 会话 ID 都会在响应中发送给客户,并在每个后续的请求中返回给服务器。当超时或被设置无效,会话都会结束。会话超时期限可以通过 DD 文件配置,这将影响所有会话的超时期限。要改变指定会话的超时时间,可以使用会话对象的 setMaxInactiveInterval()。

习　　题

1. 下面哪个方法用于从 ServletContext 中检索属性?(　　)

A. String getAttribute(int index)　　B. String getObject(int index)

C. Object getAttribute(int index)　　D. Object getObject(int index)

E. Object getAttribute(String name)　　F. String getAttribute(String name)

2. 下面哪个方法用来检索 ServletContext 初始化参数?(　　)

A. Object getInitParameter(int index)

B. Object getParameter(int index)

C. Object getInitParameter(String name)

D. String getInitParameter(String name)

E. String getParameter(String name)

3. 为 Servlet 上下文指定初始化参数,下面的 web.xml 片段哪个是正确的?(　　)

A.
```
<context-param>
  <name>country</name>
  <value>China</value>
</context-param>
```

B.
```
<context-param>
  <param name="country" value="China" />
</context-param>
```

C.
```
<context>
  <param name="country" value="China" />
</context>
```

D.
```
<context-param>
  <param-name>country</param-name>
  <param-value>China</param-value>
</context-param>
```

4. 下面哪个接口或类检索与用户相关的会话对象?(　　)

A. HttpServletResponse　　B. ServletConfig

C. ServletContext　　D. HttpServletRequest

5. 给定 request 是一个 HttpServletRequest 对象,下面哪两行代码会在不存在会话的情况下创建一个会话?(　　)

A. request.getSession()　　B. request. getSession(true)

C. request.getSession(false)　　D. request.createSession()

6. 关于会话属性，下面哪两个说法是正确的？（　　）

A. HttpSession 的 getAttribute(String name)返回类型为 Object

B. HttpSession 的 getAttribute(String name)返回类型为 String

C. 在一个 HttpSession 上调用 setAttribute("keyA","valueB")时，如果这个会话中对应键 keyA 已经有一个值，就会导致抛出一个异常

D. 在一个 HttpSession 上调用 setAttribute("keyA","valueB")时，如果这个会话中对应键 keyA 已经有一个值，则这个属性的原先值会被 valueB 替换

7. 调用下面哪个方法将使会话失效？（　　）

A. session.invalidate();　　B. session.close();

C. session.destroy();　　D. session.end();

8. 是否能够通过客户机的 IP 地址实现会话跟踪？

9. 如何理解会话失效与超时？如何通过程序设置最大失效时间？如何通过 Web 应用程序部署描述文件设置最大超时时间？二者有什么区别？

10. 关于 HttpSession 对象，下面哪两个说法是正确的？（　　）

A. 会话的超时时间设置为－1，则会话永远不会到期

B. 一旦用户关闭所有浏览器窗口，会话就会立即失效

C. 在部署描述文件中定义的超时时间之后，会话会失效

D. 可以调用 HttpSession 的 invalidateSession()使会话失效

11. 给定一个会话对象 s，有两个属性，属性名分别为 myAttr1 和 myAttr2，下面哪行(段)代码会把这两个属性从会话中删除？（　　）

A. s.removeAllValues();

B. s.removeAllAttributes();

C. s.removeAttribute("myAttr1");
s. removeAttribute("myAttr2");

D. s.getAttribute("myAttr1",UNBIND);
s. getAttribute("myAttr2",UNBIND);

12. 将下面哪个代码片段插入到 doGet()中可以正确记录用户的 GET 请求的数量？（　　）

A.
```
HttpSession session = request.getSession();
int count = session.getAttribute("count");
session.setAttribute("count", count++);
```

B.
```
HttpSession session = request.getSession();
int count = (int) session.getAttribute("count");
session.setAttribute("count", count++);
```

C.
```
HttpSession session = request.getSession();
int count = ((Integer) session.getAttribute("count")).intValue();
session.setAttribute("count", count++);
```

D.
```
HttpSession session = request.getSession();
int count = ((Integer) session.getAttribute("count")).intValue();
session.setAttribute("count", new Integer(++count));
```

13. 以下哪段代码能从请求对象中获取名为 ORA-UID 的 Cookie 的值？（　　）

A.
```
String value = request.getCookie("ORA-UID");
```

B.
```
String value = request.getHeader("ORA-UID");
```

C.
```
Cookie[] cookies = request.getCookies();
String cName = null;
String value = null;
if(cookies != null){
   for(int i = 0 ;i < cookies.length; i++){
    cName = cookies[i].getName();
    if(cName!= null && cName.equalsIgnoreCase("ORA_UID")){
     value = cookies[i].getValue();
    }
   }
}
```

D.
```
Cookie[] cookies = request.getCookies();
if(cookies.length > 0){
  String value = cookies[0].getValue();
}
```