会话跟踪技术

一、HTTP 无状态

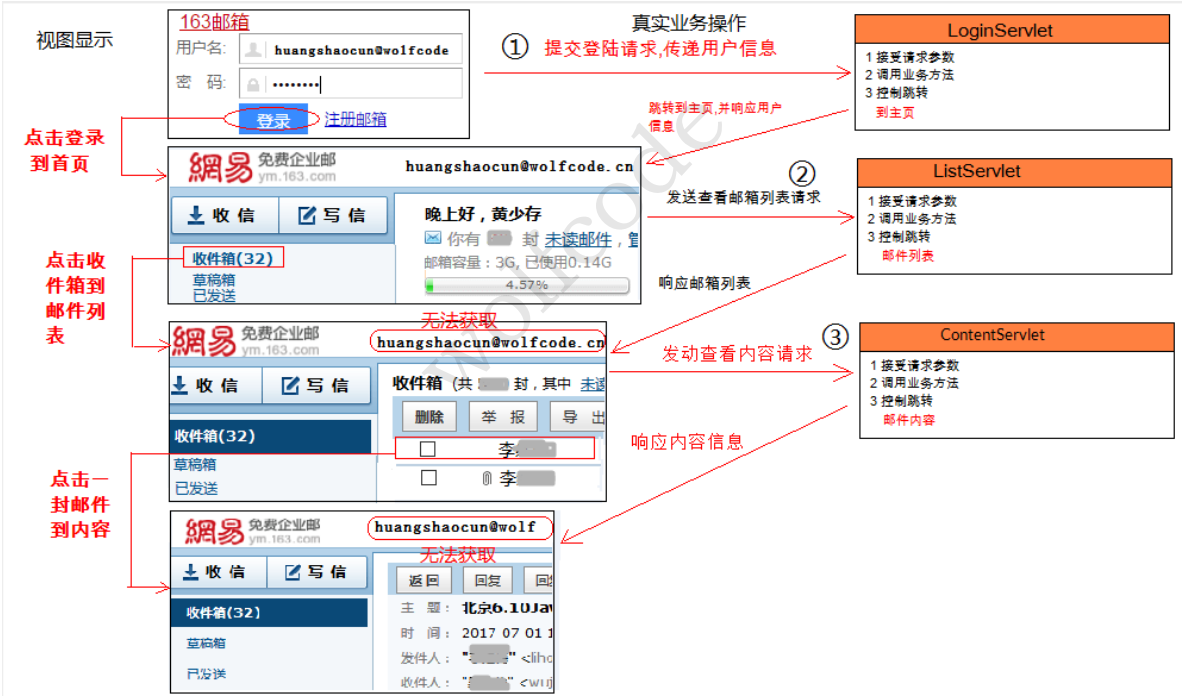
1、问题说明

HTTP 协议是无状态的，也就是没有记忆力，也无法确定发出请求的用户身份。只负责一问一答。

问题：一个会话中的多个请求之间无法共享数据。

如：在操作邮箱的过程中，我们有下面的需求

- 用户访问登录页面，填写账号密码完成登录验证；
- 若验证通过，页面跳转到邮箱首页，显示**当前用户名**；
- 用户点击收件箱，显示**当前用户的邮件列表**，显示登陆的**用户名**；
- 查看其中一封邮件的内容，显示登陆的**用户名**。



②③ 请求的请求对象（req）是各自的，都没有①请求的数据（用户登陆信息），故 ①②③ 请求的 req 都是各自的，无法共享数据。而 **实际开发中，我们需要在多个请求之间来共享数据。**

2、解决方案

借助其他方式或技术来实现会话中多个请求间数据的共享。

2.1、传参方式

直接在路径上带参数（麻烦且数据不安全，不使用）。

2.2、会话跟踪技术（推荐）

- Cookie 客户端会话跟踪技术。
- Session 服务端会话跟踪技术（HttpSession）。

二、邮箱案例实现

1、项目准备

搭建一个新 Web 项目，名为 http-cookie-session，并添加 jstl jar。

2、编写 login.jsp

在 web 目录下新建 login.jsp，在页面点击了登录按钮，发送登录的请求 /http/login，交由 LoginServlet 来处理，文件内容如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
  <head>
    <title>登录</title>
  </head>
  <body>
    <h3>用户登录</h3>
    <form action="/http/login" method="post">
      <p>账号：<input type="text" name="username"></p>
      <p>密码：<input type="text" name="password"></p>
      <input type="submit" value="登录">
    </form>
  </body>
</html>
```

3、编写 LoginServlet.java

处理登录请求，若登录成功，跳转到 index.jsp。

```
package cn.wolfcode.web._01_http;

@WebServlet("/http/login")
public class LoginServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 处理请求编码
        req.setCharacterEncoding("UTF-8");
        // 接受请求参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        // 调用业务方法来处理请求，这里模拟和数据库中账号信息的匹配
        if("admin".equals(username) && "1".equals(password)) {
            // 模拟登录成功，往 request 作用域存入数据
            req.setAttribute("username", username);
            // 控制跳转 index.jsp
            req.getRequestDispatcher("/WEB-INF/views/index.jsp").forward(req,
resp);
            return;
        }
```

```
            System.out.println("登录失败");
        }
    }
```

4、编写 index.jsp

新建 views/index.jsp，显示当前登录的用户名及提供点击查看收件箱，点了查询收件箱，发的请求 /http/list 交由 ListServlet 来处理，文件内容如下：

```
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>主页</title>
</head>
<body>
    欢迎：${username} <br/>
    <a href="/http/list">收件箱</a>
</body>
</html>
```

5、ListServlet.java

查询用户邮件，跳转跳转到邮件列表页面（list.jsp）。

```
package cn.wolfcode.web._01_http;

@WebServlet("/http/list")
public class ListServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 处理请求编码
        req.setCharacterEncoding("UTF-8");
        // 接受请求参数
        String username = req.getParameter("username");
        // 调用业务方法来处理请求，从数据库中查询邮件列表(模拟)
        List<String> list = Arrays.asList("邮件1", "邮件2", "邮件3");
        req.setAttribute("list", list);
        req.setAttribute("username", username);
        req.getRequestDispatcher("/WEB-INF/views/list.jsp").forward(req, resp);
    }
}
```

6、编写 list.jsp

显示当前用户名及其邮件，提供查询邮件内容，点击发送请求 /http/content 交由 ContentServlet 来处理，文件内容如下：

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
<head>
    <title>邮件列表</title>
</head>
<body>
    欢迎：${username} <br/>
    <c:forEach items="${list}" var="email">
        <a href="/http/content">${email}</a> <br/>
    </c:forEach>
</body>
</html>
```

7、编写 ContentServlet.java

查询邮件内容，跳转跳转到邮件内容页（content.jsp）。

```java
package cn.wolfcode.web._01_http;

@WebServlet("/http/content")
public class ContentServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 处理请求编码
        req.setCharacterEncoding("UTF-8");
        // 接受请求参数
        String username = req.getParameter("username");
        req.setAttribute("username", username);
        // 模拟从数据库中查询到邮件内容，放到 request 作用域中
        req.setAttribute("content", "这是邮件内容");
        req.getRequestDispatcher("/WEB-INF/views/content.jsp").forward(req,
resp);
    }
}
```

8、编写 content.jsp

新建 views/content.jsp，显示邮件内容，文件内容如下：

```jsp
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<html>
<head>
    <title>邮件内容</title>
</head>
<body>
    欢迎：${username}<br/>
    ${content}
</body>
</html>
```

9、存在的问题

一次会话中多次请求，不共享数据，导致用户显示不出来。但若通过路径携带的方式解决该问题，会把用户名拼接在路径，给暴露出来，不安全。

三、Cookie

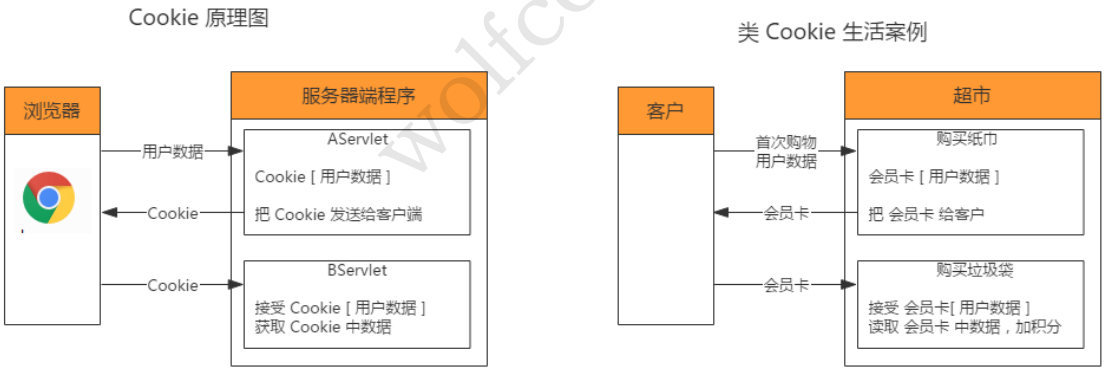
1、Cookie 概述

Cookie 是 **客户端技术**，程序把每个用户的数据以 cookie 的形式写给用户各自的浏览器。当用户使用浏览器再去访问服务器中的 Web 资源时，就会带着各自的数据去。这样，Web 资源处理的就是用户各自的数据了。

思考：

- Cookie 是服务器端创建的还是客户创建的？ 服务器端程序创建
- Cookie 从哪里带到哪里？服务器端 -> 客户端
- Cookie 最终存在什么地方？浏览器
- Cookie 什么时候实现数据共享？再一次访问服务器端资源时
- Cookie 中数据的传递方向？浏览器 -> 服务器（Cookie）-> 浏览器（存入）-> 服务器（获取 Cookie 数据）

2、Cookie 原理图



四、Cookie 基本使用

Cookie 的基本使用无非是：

- 如何在服务器端创建Cookie?
- 如何发送 Cookie 给浏览器?
- 如何获取浏览器带过来的 Cookie?

1、创建 Cookie 对象

创建 Cookie 对象：

```
Cookie cookie = new Cookie (String name, String value)
```

属性	方法	描述
name	getName()	共享数据名称（**唯一**）
value	getValue()	要共享的数据

2、响应 Cookie 给浏览器

使用响应对象中的 addCookie(Cookie 对象) 将数据响应给浏览器。

```
resp.addCookie(cookie);
```

3、服务器端获取 Cookie

浏览器发请求时，自动将 Cookie 发送到服务器，服务器程序直接获取即可。
数据在请求中，使用请求对象中的 getCookies() 方法获取所有的 Cookie 对象。

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

4、代码示例

4.1、准备工作

- 把所有之前所写的 Servlet，拷贝一份放到 cn.wolfcode.web._02_cookie 包下，映射路径中的 http 替换成 cookie。
- 把所有之前缩写的 JSP 中的请求路径中的 http 替换成 cookie。

4.2、修改 LoginServlet.java

创建 Cookie 对象，存入数据响应给浏览器。

```
package cn.wolfcode.web._01_http;

@WebServlet("/cookie/login")
public class LoginServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        req.setCharacterEncoding("UTF-8");
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        if("admin".equals(username) && "1".equals(password)) {
            // 把数据存入 Cookie 中，写给浏览器
            Cookie cookie =  new Cookie("username", username);
            // 把 Cookie 写回浏览器
            resp.addCookie(cookie);
            req.setAttribute("username", username);
            // 模拟登录成功，跳转到主页面
            req.getRequestDispatcher("/WEB-INF/views/index.jsp").forward(req,
resp);
            return;
        }
```

```
            System.out.println("登录失败");
        }
    }
```

4.3、在 Servlet 中获取 Cookie 数据

修改 ListServlet.java，在 service 方法中从 Cookie 中获取登录用户的用户名。

```
package cn.wolfcode.web._02_cookie;

@WebServlet("/cookie/list")
public class ListServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 调用业务方法来处理请求，从数据库中查询邮件列表(模拟)
        List<String> list = Arrays.asList("邮件1", "邮件2", "邮件3");
        req.setAttribute("list", list);
        String username = null;
        // 获取浏览器带过来的 Cookie 数据
        Cookie[] cookies = req.getCookies();
        for(Cookie cookie : cookies) {
            if("username".equals(cookie.getName())) {
                username = cookie.getValue();
                System.out.println(username);
            }
        }
        req.getRequestDispatcher("/WEB-INF/views/list.jsp").forward(req, resp);
    }
}
```

4.4、在 JSP 中获取 Cookie 数据

修改 list.jsp 和 content.jsp，从 Cookie 中获取登录用户的用户名。

4.4.1、脚本方式（不推荐）

```
<%
   String username = null;
   Cookie[] cs = request.getCookies();
   for(Cookie cookie : cs){
       if("username".equals(cookie.getName())){
           username = cookie.getValue();
       }
}
%>
欢迎：<%=username%> <br/>
```

4.4.2、EL 表达式（推荐）

```
欢迎：${cookie.username.value}<br/>
```

五、Cookie 使用细节

1、修改 Cookie 中的数据

1.1、修改 Cookie 数据的方式

- 调用 Cookie 对象的 **setValue** 方法来覆盖原本的数据。
- **重新**创建一个同 name 的 **Cookie 对象**。

以上不管使用哪种方式，修改的都是服务器端内容中的 Cookie 数据，和浏览器中存的 Cookie 没有关系，所以需要重新响应新的 Cookie 到浏览器中进行更新。

注意：需要将修改之后的 Cookie 发送到浏览器中进行更新：

```
resp.addCookie(新的 Cookie 对象);
```

1.2、代码示例

修改 ListServlet，修改 Cookie 中的数据。

```
package cn.wolfcode.web._02_cookie;

@WebServlet("/cookie/list")
public class ListServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 调用业务方法来处理请求，从数据库中查询邮件列表(模拟)
        List<String> list = Arrays.asList("邮件1", "邮件2", "邮件3");
        req.setAttribute("list", list);

        String username = null;
        Cookie[] cookies = req.getCookies();
        for(Cookie cookie : cookies) {
            if("username".equals(cookie.getName())) {
                // 方式一：修改 Cookie 中的数据
                cookie.setValue("wolfcode1");
                // 方式二：创建新的 Cookie 对象，设置名称为 username
                // cookie = new Cookie("username", "wolfcode2");
                // 记得使用 Response 对象响应回浏览器
                resp.addCookie(cookie);
                break;
            }
        }

        req.getRequestDispatcher("/WEB-INF/views/list.jsp").forward(req, resp);
    }
}
```

2、设置 Cookie 存活时间

2.1、需求

要实现以上功能则需要在创建 Cookie 对象之后给其设置存活时间。代码如下：

```
Cookie 对象的 setMaxAge(int expiry);
```

expiry 数值	例子	含义
大于 0	setMaxAge(60)	Cookie 存活的时间，单位为秒，例：60 秒
等于 0	setMaxAge(0)	立即删除当前 Cookie 对象
小于 0	setMaxAge(-1)	会话 Cookie，浏览器关闭则销毁

2.2、Cookie 分类

Cookie 总是保存在客户端中，按在客户端中的存储位置，可分为内存 Cookie 和硬盘 Cookie。

内存 Cookie 由浏览器维护，保存在内存中，浏览器关闭后就消失了，其存在时间是短暂的。硬盘 Cookie 保存在硬盘里，有一个过期时间，除非用户手工清理或到了过期时间，硬盘 Cookie 不会被删除，其存在时间是长期的。所以，按存在时间，可分为非持久 Cookie 和持久 Cookie。

默认情况下 Cookie 属于会话 Cookie，即浏览器关闭则失效，无法使用。

2.3、代码示例

把之前修改的 Cookie 数据的代码注释掉之后，修改 LoginServlet，设置 Cookie 的存活时间。

```
package cn.wolfcode.web._02_cookie;

@WebServlet("/cookie/login")
public class LoginServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 处理请求编码
        req.setCharacterEncoding("UTF-8");
        // 接受请求参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        // 调用业务方法来处理请求，这里模拟和数据库中账号信息的匹配
        if("admin".equals(username) && "1".equals(password)) {
            // 把数据存入 Cookie 中，写给浏览器
            Cookie cookie =  new Cookie("username", username);

            // 设置 Cookie 存活时间，单位秒
            cookie.setMaxAge(60);
```

```
            // 把 Cookie 写回浏览器
            resp.addCookie(cookie);
            req.setAttribute("username", username);
            // 模拟登录成功，跳转到主页面
            req.getRequestDispatcher("/WEB-INF/views/index.jsp").forward(req,
resp);
            return;
        }

        System.out.println("登录失败");
    }
}
```

3、删除 Cookie

3.1、如何删除 Cookie

删除 Cookie 只需要使用设置 Cookie 存活时间的方法即可，当然需要注意的是一般是中途删除 Cookie，不是创建了后马上删除，所以在设置删除之后需要重新发送给浏览器去更新删除。

```
Cookie 对象.setMaxAge(0);
resp.addCookie(Cookie 对象);
```

3.2、代码示例

修改 ListServlet，删除 Cookie 。

```
package cn.wolfcode.web._02_cookie;

@WebServlet("/cookie/list")
public class ListServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 调用业务方法来处理请求，从数据库中查询邮件列表(模拟)
        List<String> list = Arrays.asList("邮件1", "邮件2", "邮件3");
        req.setAttribute("list", list);

        String username = null;
        Cookie[] cookies = req.getCookies();
        for(Cookie cookie : cookies) {
            if("username".equals(cookie.getName())) {
                // 方式一：修改 Cookie 中的数据
                // cookie.setValue("wolfcode1");
                // 方式二：创建新的 Cookie 对象，设置名称为 username
                // cookie = new Cookie("username", "wolfcode2");

                // 删除 Cookie
                cookie.setMaxAge(0);

                // 记得使用 Response 对象响应回浏览器
                resp.addCookie(cookie);
                break;
            }
```

```
        }

        req.getRequestDispatcher("/WEB-INF/views/list.jsp").forward(req, resp);
    }
}
```

4、Cookie 中文问题（了解）

- 在 Tomcat8.5 以下，Cookie 中的 name 属性和 value 都不支持中文，若有中文，抛以下异常：
  HTTP Status 500 - Control character in cookie value or attribute.
- Tomcat8.5 自动处理了 Cookie 中文问题，所以无需手动解决。

Tomcat 8.5 以下解决思路：将中文进行处理，转换成另一种非中文字符串存，获取时再转化回来。转化的 API 如下：

类	方法	作用
URLEncoder	encode("中文", "UTF-8")	把中文字符串转化为非中文
URLDecoder	decode(“非中文”, "UTF-8")	把非中文转化为中文

```
// 往 Cookie 中存中文数据
Cookie cookie = new Cookie("username", URLEncoder.encode("小狼", "UTF-8"));

// 从 Cookie 中获取中文数据
Cookie[] cookies = request.getCookies();
for (Cookie cookie : cookies) {
    if("username".equals(cookie.getName())) {
        System.out.println(URLDecoder.decode(cookie.getValue(), "UTF-8"));
    }
}
```


5、Cookie 的域和路径（了解）

5.1、域与路径的作用

为了让浏览器可以识别 Cookie 发送给对应的服务器，以及识别哪些请求需要携带 Cookie，默认 Cookie 都带了服务器的识别标识以及需要带 Cookie 的资源标识，出于安全考虑，为了保护 Cookie 中的数据。

5.2、域

域用来**识别服务器**，包含 IP:端口或域名:端口，可通过 Cookie 对象的 setDomain 方法设置。默认 Cookie 中的域是创建 Cookie 的服务器的域名。域名分类：

- 一级（主）域名：baidu.com
- 多级域名：news.baidu.com

若想要在相同的主域名下来共享 Cookies 数据，例如，百度和百度地图，百度音乐共用账号，则只需要设置 Cookie 的 domain 即可。若主域不同，是无法共享 Cookie 数据。

```
cookie.setDomain(".baidu.com");
```

5.3、路径

路径用来**识别资源**，可通过 Cookie 对象的 setPath 方法设置。默认为创建 Cookie 的资源的路径。例：/cookie/login 创建 Cookie，则 path 为 /cookie，则访问 /cookie 开头的资源都会携带该 Cookie。

若想要在访问服务器上的任意资源都带上 Cookie，则只需要在创建 Cookie 之后设置下 path 为 / 即可。

```
cookie.setPath("/");
```

5.4、域和路径总结

- 若主域不同，不管 path 如何，都无法带上其他主域的 Cookie。
- 若主域相同，且 path 也为 /，则访问该域下的任意资源都会携带该域的 Cookie。

6、Cookie 问题及应用场景

6.1、存在的问题

- 若是一个用户一台电脑，没有问题，但是若多个人公用一个电脑就存在不安全问题。
- 存中文数据 (Tomcat8.5 之内) 比较麻烦（编码，解码）。
- 一个 Cookie 只能设置一个值，值须字符串类型。
- 一台服务器在一个客户端存储的 Cookie 大小和数量有限：
  - Cookie 大小限制在 4KB 之内；
  - 一台服务器在一个客户端最多保存 20 个 Cookie；
  - 一个浏览器最多可以保存 300 个 Cookie。

6.2、应用场景

- 存用户标识，解决 HTTP 无状态问题。
- 登录时记住用户名。
- 未登录情况实现购物车。

六、Session

1、Session 概述

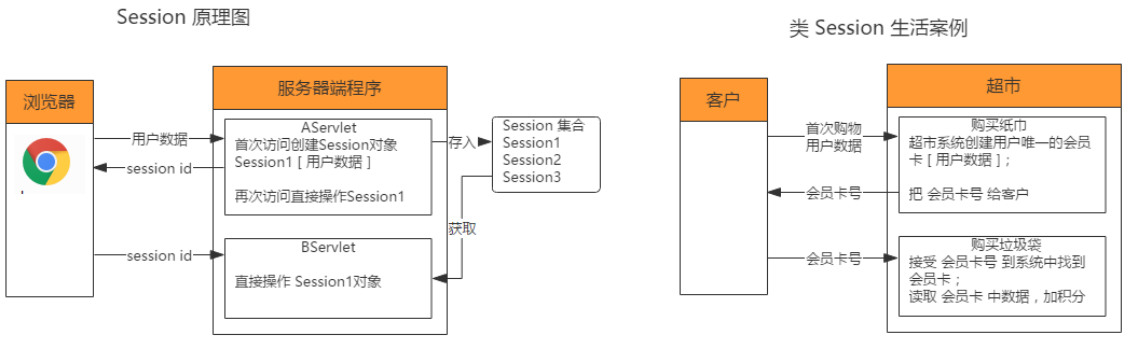
Session 是**服务器端技术**，利用这个技术，服务器在运行时可以为每一个**用户的浏览器**创建一个其**独享的 session 对象**，由于 session 为用户浏览器独享，所以用户在访问服务器的 Web 资源时，可以把各自的数据放在各自的 session 中，当用户再去访问服务器中的其它 Web 资源时，其它 Web 资源再从用户各自的 session 中取出数据为用户服务。

思考：

- Session 谁来创建？服务器
- Session 存在什么地方？服务器
- Session 什么时候实现数据共享？再一次访问服务器端资源时
- Session 中数据的传递反向？浏览器 -> 服务器（Session）-> 数据在服务器
- 服务器如何识别不同用户的 Session?

- 浏览器 -> 服务器（Session）sessionId
- 浏览器 sessionId -> 服务器

2、Session 的原理图



Session 其底层依然需要依赖 Cookie 来传递 Session 的 id 值。存 sessionId 的 Cookie 为会话 Cookie。

所以 Session 在浏览器关闭之后就无法使用了。原因是 sessionId 丢失了，服务器会在 30 分钟内清除未有操作的 Session 对象。

七、Session 的基本使用

Session 的基本使用无非是如何得到 Session 对象以及如何往其中存入数据，实现那共享。

1、Session API

1.1、获取 Session

获取 Session 对象可通过使用 HttpServletRequest 的 API：

方法	作用
getSession(true)	判断是否存在 Session，存在则获取，不存在则创建新 Session 对象返回
getSession(false)	判断是否存在 Session，存在则获取，不存在则返回 null （不符合需求）
getSession()	判断是否存在 Session（浏览器是否带了 sessionId），存在则获取，不存在则创建新 Session 对象返回（推荐）

推荐使用 getSession() 满足需求的同时又比较简洁。

1.2、Session 数据共享

HttpSession API：

方法	作用
setAttribute(String name, Object value)	设置属性名和属性值
getAttribute(String name)	通过属性名去获取属性值
removeAttribute(String name)	从 Session 中移除指定属性名的属性值
invalidate()	移除整个 Session 对象，删除所有的属性和属性值

removeAttribute 和 invalidate 视情况选择，删除部分数据使用 removeAttribute，删除全部数据使用 invalidate。

2、代码示例

2.1、准备工作

- 把所有之前所写的 Servlet，拷贝一份放到 cn.wolfcode.web._03_session 包下，映射路径中的 cookie 替换成 session，且删除所有之前关于 Cookie 操作得代码。
- 把所有之前缩写的 JSP 中的请求路径中的 cookie 替换成 session。

2.2、修改 LoginServlet.java

登录成功之后往 session 中存入数据。

```
package cn.wolfcode.web._03_session;

@WebServlet("/session/login")
public class LoginServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 处理请求编码
        req.setCharacterEncoding("UTF-8");
        // 接受请求参数
        String username = req.getParameter("username");
        String password = req.getParameter("password");
        // 调用业务方法来处理请求，这里模拟和数据库中账号信息的匹配
        if("admin".equals(username) && "1".equals(password)){

            // 往 session 中存入共享数据
            req.getSession().setAttribute("username", "admin");

            // 模拟登录成功，跳转到主页面
            req.getRequestDispatcher("/WEB-INF/views/index.jsp").forward(req,
resp);
            return;
        }

        System.out.println("登录失败");
    }
}
```

2.3、在 Servlet 中获取 Session 数据

修改 ListServlet.java 和 ContentServlet，在 service 方法中从 Session 中获取登录用户的用户名。

```
package cn.wolfcode.web._03_session;

@WebServlet("/session/list")
public class ListServlet extends HttpServlet {
    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 调用业务方法来处理请求，从数据库中查询邮件列表(模拟)
        List<String> list = Arrays.asList("邮件1", "邮件2", "邮件3");
        req.setAttribute("list", list);

        // 从 Session 中获取数据
        System.out.println(req.getSession().getAttribute("username"));

        req.getRequestDispatcher("/WEB-INF/views/list.jsp").forward(req, resp);
    }
}
```

```
package cn.wolfcode.web._03_session;

@WebServlet("/session/content")
public class ContentServlet extends HttpServlet {

    @Override
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 模拟从数据库中查询到邮件内容，放到 request 作用域中
        req.setAttribute("content", "这是邮件内容");

        // 从 Session 中获取数据
        System.out.println(req.getSession().getAttribute("username"));

        req.getRequestDispatcher("/WEB-INF/views/content.jsp").forward(req,
resp);
    }
}
```


2.4、在 JSP 中获取 Session 数据

修改 index.jsp、list.jsp 和 content.jsp，使用 EL 表达式从 Session 中获取登录用户的用户名。

```
欢迎：${username}<br/>
```

2.5、移除 session 中的数据

修改 ListServlet.java，在 service 方法中从 Session 中移除登录用户的用户名。

```
package cn.wolfcode.web._03_session;

@WebServlet("/session/list")
public class ListServlet extends HttpServlet {
    @Override
```

```
    protected void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        // 调用业务方法来处理请求，从数据库中查询邮件列表(模拟)
        List<String> list = Arrays.asList("邮件1", "邮件2", "邮件3");
        req.setAttribute("list", list);

        // 从 Session 中获取数据
        // System.out.println(req.getSession().getAttribute("username"));
        // 从 session 中移除数据
        // req.getSession().removeAttribute("username");
        // 销毁 session
        session.invalidate()

        req.getRequestDispatcher("/WEB-INF/views/list.jsp").forward(req, resp);
    }
}
```

八、Session 的细节使用

1、Session 超时管理

设置会话的有效时间，比如金融项目，一般用户登陆状态只保留 1~3 分钟，不会一直保留。中途离开，回来需要重新登陆。

1.1、全局修改

Tomcat 默认是 30 分钟，实际上只保存 20 来分钟（server.xml）。

```
<session-config>
    <session-timeout>30</session-timeout>
</session-config>
```


1.2、局部修改

在 LoginServlet.java 的 service 方法中使用如下 API 修改 Session 超时时间：

```
// 设置 session 超时时间（单位秒）
session.setMaxInactiveInterval(int interval);
```

2、Session 使用规范

- Session 的属性命名，一般我们命名格式为：**XX_IN_SESSION**，而且这个属性名是**唯一**的。

  ```
  session.setAttribute("USER_IN_SESSION", 值);
  ```

- Session 可以存放多个数据，但是若数据之间是有联系的，比如用户的用户名与密码，我们会封装成一个对象，之后再存入到 Session 中。
- 服务器在做 Session 数据共享时，Session 中的存储的对象类型须实现 java.io.Serializable 接口（了解）。

3、URL 重写（了解）

3.1、存在的问题

Session 是基于 Cookie 的，sessionId 是存在浏览器上，用户可选择不接受 Cookie 或者禁用 Cookie，此时 Session 失效。

3.2、解决方案

只要在请求路径后面拼接 jsessionid，使用；来间隔，
比如：http://localhost/session/list;jsessionid=6225FDD3988C3B11BD8320899E9B1D2F，
就可以解决上述问题。

3.2.1、在 Servlet 中 URL 重写

在请求的 URL 后面拼接 jsessionid（自动检测用户是否开启 Cookie 接收，若开启了，则不拼接 jsessionid）

```
response.encodeURL("/session/list");
```

3.2.2、在 JSP 中 URL 重写

- 脚本重写（了解）

```
<%
    String url =  response.encodeURL("/session/list");
%>
<a href="<%=url%>">收件箱</a>
```

- EL 重写

```
${pageContext.response.encodeURL("/session/list")}
```

注意：一般开发中不去屏蔽 Cookie，一般用户也不会去做。