Java 应用与开发
HTTP 会话跟踪技术

王晓东

wangxiaodong@ouc.edu.cn

中国海洋大学

November 25, 2018

学习目标

1. 掌握会话的基本概念，理解会话不是仅仅使用 HTTP 协议就能够保证的，而是客户端浏览器和服务器端在 HTTP 协议之上采用额外的技术协同的结果。
2. 掌握常用的会话跟踪技术，了解采用 URL 重写维持会话跟踪的方法；理解 Cookie 和 Session 的协同机制，掌握使用 Cookie 和 Session 实现会话跟踪的技术。
3. 能够使用 Cookie 和 Session 编写会话跟踪代码。

大纲

接下来…

会话基本概念

会话跟踪技术
- URL 重写
- Cookie
- Java EE 会话对象

本节习题

什么是会话

- 在 Web 应用中把客户端浏览器开始请求 Web 服务器，访问不同 Web 文档进行请求/响应，到结束访问的一系列过程称为会话，即一次会话（Session）。
- 当用户访问第一个 Java Web 组件时，Java EE Web 服务器自动为用户创建一个会话对象。

例如，当当网进行图书浏览、购买、完成结算的全过程可能是一次会话；登录 126 邮箱，完成浏览收件箱、编写邮件、发送邮件、登出邮箱可以是一次会话。

会话跟踪

❖ HTTP 的固有设计“缺陷”

由于 Web 应用采用 HTTP 协议，而 HTTP 协议是无状态、不持续的协议，所以需要独立于 HTTP 协议的会话跟踪技术，用于记录会话的状态信息。

❖ 什么是会话跟踪

- 在一个会话内，当用户在次访问时，服务器需要能够定位是先前访问的同一个用户。
- Web 应用需要在用户访问的一个会话内，让 Web 服务器保存客户的信息（如客户的账号或客户的购物车），称为**会话跟踪**，即 Web 服务器必须使用某种技术保存客户的信息。

Java EE Web 会话跟踪方法

1. **重写 URL** 将客户端的信息附加在请求 URL 地址的参数中，Web 服务器取得参数信息，完成客户端信息的保存。
2. **隐藏表单字段** 将要保存的客户信息，如用户登录账号使用隐藏表单字段发送到服务器端，完成 Web 服务器保持客户状态信息。
3. **Cookie** 使用 Java EE API 提供的 Cookie 对象，可以将客户信息保存在 Cookie 中，完成会话跟踪功能。
4. **HttpSession 对象** Java EE API 专门提供了 HttpSession 会话对象保存客户的信息来实现会话跟踪。

一般 3 和 4 组合使用。

接下来…

会话基本概念

会话跟踪技术
- URL 重写
- Cookie
- Java EE 会话对象

本节习题

URL 重写

接下来…

URL 重写的实现

❶ 浏览器端构造 URL 请求

- 在进行 HTTP 请求时，可以在 URL 地址后直接附加请求参数，把客户端的数据传输到 Web 服务器端。
- Web 服务器通过 HttpServletRequest 请求对象取得这些 URL 地址后面附加的请求参数。
- 这种 URL 地址后附加参数的方式称为 URL **重写**。

URL 重写示例

```
1    <a href="../product/main.do?userid=9001&category=11">产品管理</a>
```

此例中，将客户 ID 附加在地址栏上，以?name=value 形式附加在 URL 后，多个参数使用 & 符号进行间隔。

URL 重写的实现

❷ 服务器端解析 URL 获取用户会话标识

Web 服务器端使用请求对象取得 URL 后附加的客户端参数数据。

```
1    String userid = request.getParameter("userid"); // 取得用户 ID 参数数据
```

URL 重写的实现

❸ 浏览器和服务器两端持续带会话标识通信

为保证 Web 应用在能在以后持续的请求/响应中实现会话跟踪，必须保证每次请求都要在 URI 地址中加入 userid=9001 参数，进而实现会话跟踪。
如下为 Servlet 重定向请求的附加参数：

```
1    response.sendRedirect("../product/view.do?productid=1201&userid=" + userid);
```

URL 重写的缺点

- URL 传递参数的限制
- 安全性缺陷
- 编程繁杂

接下来…

什么是 Cookie

- Cookie 在 Java EE 之前就已经存在，是由 Netscape 浏览器引入的，用于在客户端保存服务器端数据，实现一种简单有效的客户/服务器的信息交换模式。
- Cookie 是 Web 服务器保存在客户端的小的文本文件，存储许多 name/value 对，可以保存如登录帐号、用户喜好等会话数据。
- Cookie 由 Web 服务器创建，由 Web 服务器在进行 HTTP 响应时，将 Cookie 保存在 HTTP 响应头中并发送给浏览器，浏览器收到 HTTP 响应头，解析出 Cookie，将它保存在客户的本地隐藏文件中。
- 客户浏览器每次向 Web 服务器发出 HTTP 请求时，自动将 Cookie 保存在请求头中，随请求体一起发送到服务器，这个过程不需要人工参与。
- Web 服务器可以从请求对象中取出 Cookie，进而得到 Cookie 中保存的名称/值对，从而实现会话跟踪。

什么是 Cookie

- Cookie 在 Java EE 之前就已经存在，是由 Netscape 浏览器引入的，用于在客户端保存服务器端数据，实现一种简单有效的客户/服务器的信息交换模式。
- Cookie 是 Web 服务器保存在客户端的小的文本文件，存储许多 name/value 对，可以保存如登录帐号、用户喜好等会话数据。
- Cookie 由 Web 服务器创建，由 Web 服务器在进行 HTTP 响应时，将 Cookie 保存在 HTTP 响应头中并发送给浏览器，浏览器收到 HTTP 响应头，解析出 Cookie，将它保存在客户的本地隐藏文件中。
- 客户浏览器每次向 Web 服务器发出 HTTP 请求时，自动将 Cookie 保存在请求头中，随请求体一起发送到服务器，这个过程不需要人工参与。
- Web 服务器可以从请求对象中取出 Cookie，进而得到 Cookie 中保存的名称/值对，从而实现会话跟踪。

什么是 Cookie

- Cookie 在 Java EE 之前就已经存在，是由 Netscape 浏览器引入的，用于在客户端保存服务器端数据，实现一种简单有效的客户/服务器的信息交换模式。
- Cookie 是 Web 服务器保存在客户端的小的文本文件，存储许多 name/value 对，可以保存如登录帐号、用户喜好等会话数据。
- Cookie 由 Web 服务器创建，由 Web 服务器在进行 HTTP 响应时，将 Cookie 保存在 HTTP 响应头中并发送给浏览器，浏览器收到 HTTP 响应头，解析出 Cookie，将它保存在客户的本地隐藏文件中。
- 客户浏览器每次向 Web 服务器发出 HTTP 请求时，自动将 Cookie 保存在请求头中，随请求体一起发送到服务器，这个过程不需要人工参与。
- Web 服务器可以从请求对象中取出 Cookie，进而得到 Cookie 中保存的名称/值对，从而实现会话跟踪。

什么是 Cookie

- Cookie 在 Java EE 之前就已经存在，是由 Netscape 浏览器引入的，用于在客户端保存服务器端数据，实现一种简单有效的客户/服务器的信息交换模式。
- Cookie 是 Web 服务器保存在客户端的小的文本文件，存储许多 name/value 对，可以保存如登录帐号、用户喜好等会话数据。
- Cookie 由 Web 服务器创建，由 Web 服务器在进行 HTTP 响应时，将 Cookie 保存在 HTTP 响应头中并发送给浏览器，浏览器收到 HTTP 响应头，解析出 Cookie，将它保存在客户的本地隐藏文件中。
- 客户浏览器每次向 Web 服务器发出 HTTP 请求时，自动将 Cookie 保存在请求头中，随请求体一起发送到服务器，这个过程不需要人工参与。
- Web 服务器可以从请求对象中取出 Cookie，进而得到 Cookie 中保存的名称/值对，从而实现会话跟踪。

什么是 Cookie

- Cookie 在 Java EE 之前就已经存在，是由 Netscape 浏览器引入的，用于在客户端保存服务器端数据，实现一种简单有效的客户/服务器的信息交换模式。
- Cookie 是 Web 服务器保存在客户端的小的文本文件，存储许多 name/value 对，可以保存如登录帐号、用户喜好等会话数据。
- Cookie 由 Web 服务器创建，由 Web 服务器在进行 HTTP 响应时，将 Cookie 保存在 HTTP 响应头中并发送给浏览器，浏览器收到 HTTP 响应头，解析出 Cookie，将它保存在客户的本地隐藏文件中。
- 客户浏览器每次向 Web 服务器发出 HTTP 请求时，自动将 Cookie 保存在请求头中，随请求体一起发送到服务器，这个过程不需要人工参与。
- Web 服务器可以从请求对象中取出 Cookie，进而得到 Cookie 中保存的名称/值对，从而实现会话跟踪。

Java EE 规范 Cookie API

Java EE API 提供**javax.servlet.http.Cookie**类来表达一个 Cookie 对象。

- HttpServletResponse 接口中定义了保存 Cookie 到浏览器的方法
- HttpServletRequest 接口中定义了取得客户端保存的 Cookie 对象的方法

❖ Cookie 对象的创建

使用 Cookie 类的构造方法 public Cookie(String name, String value)

```
1    Cookie cookie01 = new Cookie("userid", "9001");
```

Cookie 的主要方法

❖ get 方法

1. public String getName()
2. public String getValue()
3. public int getMaxAge()
4. public String getPath()
5. public int getVersion()
6. public String getDomain()

Cookie 的主要方法

❖ set 方法

1. public void setValue(String newValue)
2. **public void setMaxAge(int expiry)**
3. public void setDomain(String pattern)
4. public void setPath(String uri)

将 Cookie 保存到客户端

1. 创建 Cookie 对象

```
1       String userid = request.getParameter("userid"); // 取得登录 ID
2       Cookie cookie01 = new Cookie("userid", useid); // 保存到Cookie中
```

2. 设置 Cookie 属性

```
1       cookie01.setMaxAge(7 * 24 * 60 * 60); // 设置cookie的有效期
```

3. 发送 Cookie 到客户端

```
1       response.addCookie(cookie01);
```

Web 服务器读取客户端保存的 Cookie

❖ public Cookie[] getCookies()

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

3     for (int i = 0; i < cookies.length; i++) {
4       if (cookies[i].getName().equals("userid")) {
5         String name = cookies[i].getValue();
6       }
7     }
```

Cookie 的缺点

- 存储方式单一
- 存储位置限制
- 大小受浏览器限制
- 可用性限制
- 安全性限制（可以采用手动 Cookie 加解密）

接下来…

什么是会话对象

- Java EE 规范提出了一种服务器实现**会话跟踪**的机制，即 HttpSession 接口，实现该接口的对象称为 Session 对象。
- Session 对象保存在 Web 服务器上，每次会话过程创建一个，为用户保存各自的会话信息提供全面支持。
- 注意不要将过多的数据存放在会话对象内，如只在一个请求期间内需要传递的数据，就不要存储在会话对象中，而应该保存在**请求对象**中。

什么是会话对象

- Java EE 规范提出了一种服务器实现**会话跟踪**的机制，即 HttpSession 接口，实现该接口的对象称为 Session 对象。
- Session 对象保存在 Web 服务器上，每次会话过程创建一个，为用户保存各自的会话信息提供全面支持。
- 注意不要将过多的数据存放在会话对象内，如只在一个请求期间内需要传递的数据，就不要存储在会话对象中，而应该保存在**请求对象**中。

什么是会话对象

- Java EE 规范提出了一种服务器实现**会话跟踪**的机制，即 HttpSession 接口，实现该接口的对象称为 Session 对象。
- Session 对象保存在 Web 服务器上，每次会话过程创建一个，为用户保存各自的会话信息提供全面支持。
- 注意不要将过多的数据存放在会话对象内，如只在一个请求期间内需要传递的数据，就不要存储在会话对象中，而应该保存在**请求对象**中。

会话对象的类型和取得

会话对象的类型接口为**javax.servlet.http.HttpSession**，在请求对象类型 HttpServletRequest 中定义了取得会话对象的方法。

1. **public HttpSession getSession()**
   - 如果 Web 服务器内没有此客户的会话对象，则 Web 容器创建新的回话对象并返回；
   - 如果已经存在会话对象，则直接返回此对象的引用。
2. **public HttpSession getSession(boolean create)**
   - boolean 参数为 true 时，同无参数的 getSession 方法；
   - boolean 参数为 false 时，如存在会话对象则返回对象引用，如无会话对象则返回 null，Web 容器不会创建会话对象。

会话对象的类型和取得

会话对象的类型接口为**javax.servlet.http.HttpSession**，在请求对象类型 HttpServletRequest 中定义了取得会话对象的方法。

1. **public HttpSession getSession()**
   - 如果 Web 服务器内没有此客户的会话对象，则 Web 容器创建新的回话对象并返回；
   - 如果已经存在会话对象，则直接返回此对象的引用。
2. **public HttpSession getSession(boolean create)**
   - boolean 参数为 true 时，同无参数的 getSession 方法；
   - boolean 参数为 false 时，如存在会话对象则返回对象引用，如无会话对象则返回 null，Web 容器不会创建会话对象。

会话对象的类型和取得

会话对象的类型接口为**javax.servlet.http.HttpSession**，在请求对象类型 HttpServletRequest 中定义了取得会话对象的方法。

1. **public HttpSession getSession()**
   - 如果 Web 服务器内没有此客户的会话对象，则 Web 容器创建新的回话对象并返回；
   - 如果已经存在会话对象，则直接返回此对象的引用。
2. **public HttpSession getSession(boolean create)**
   - boolean 参数为 true 时，同无参数的 getSession 方法；
   - boolean 参数为 false 时，如存在会话对象则返回对象引用，如无会话对象则返回 null，Web 容器不会创建会话对象。

会话对象的类型和取得

会话对象的类型接口为**javax.servlet.http.HttpSession**，在请求对象类型 HttpServletRequest 中定义了取得会话对象的方法。

1. **public HttpSession getSession()**
   - 如果 Web 服务器内没有此客户的会话对象，则 Web 容器创建新的回话对象并返回；
   - 如果已经存在会话对象，则直接返回此对象的引用。
2. **public HttpSession getSession(boolean create)**
   - boolean 参数为 true 时，同无参数的 getSession 方法；
   - boolean 参数为 false 时，如存在会话对象则返回对象引用，如无会话对象则返回 null，Web 容器不会创建会话对象。

会话对象的类型和取得

会话对象的类型接口为**javax.servlet.http.HttpSession**，在请求对象类型 HttpServletRequest 中定义了取得会话对象的方法。

1. **public HttpSession getSession()**
   - 如果 Web 服务器内没有此客户的会话对象，则 Web 容器创建新的回话对象并返回；
   - 如果已经存在会话对象，则直接返回此对象的引用。
2. **public HttpSession getSession(boolean create)**
   - boolean 参数为 true 时，同无参数的 getSession 方法；
   - boolean 参数为 false 时，如存在会话对象则返回对象引用，如无会话对象则返回 null，Web 容器不会创建会话对象。

会话对象的类型和取得

会话对象的类型接口为**javax.servlet.http.HttpSession**，在请求对象类型 HttpServletRequest 中定义了取得会话对象的方法。

1. **public HttpSession getSession()**
   - 如果 Web 服务器内没有此客户的会话对象，则 Web 容器创建新的回话对象并返回；
   - 如果已经存在会话对象，则直接返回此对象的引用。
2. **public HttpSession getSession(boolean create)**
   - boolean 参数为 true 时，同无参数的 getSession 方法；
   - boolean 参数为 false 时，如存在会话对象则返回对象引用，如无会话对象则返回 null，Web 容器不会创建会话对象。

会话对象的功能和方法

❖ public void setAttribute(String name, Object value)

将数据存入会话对象，以 name/value 模式进存储，value 的类型为通用的 **Object** 类型，可以将任何 Java 对象保存到会话对象中。

```
1    HttpSession session = request.getSession();
2    Collection shopcart = new ArrayList();
3    Session.setAttribute("shopcart", shopcart);
```

例如，使用容器类型 Collection、List、Set 或 Map 可以非常容易的实现电子商务网站购物车的存取功能。

会话对象的功能和方法

❖ public Object getAttribute(String name)

取出保存在会话中指定名称属性的值对象，需要根据保存时使用的类型进行**造型/强制类型转换**。

```
1    Collection shopcart = (Collection) session.getAttribute("shortcart");
```

❖ public void removeValue(String name)

清除会话对象中某个属性对象。

```
1    session.removeAttribute("shopcart");
```

会话对象的功能和方法

❖ public Enumeration getAttributeNames()

取得会话对象中保存的所有属性名称列表，返回一个枚举器类型对象。

```
1    Enumeration enume = session.getAttributeNames();
2    while (enume.hasMoreElements()) {
3      String name = (String) enume.nextElment();
4      out.println(name);
5    }
```

会话对象的功能和方法

❖ public void setMaxInactiveInterval(int interval)

设置会话对象的失效期限，即 2 次请求直接的时间间隔，以秒为单位。

```
1    session.setMaxInactiveInterval(15 * 60);
```

❖ public int getMaxInactiveInterval()

取得会话的有效间隔时间，返回整数，表示间隔的秒数。

```
1    int maxtimes = session.getMaxInactiveInterval();
```

会话对象的功能和方法

❖ public void invalidate()

立即迫使会话对象失效，将当前的会话对象销毁，同时清除会话对象内的所有属性，该方法一般用在注销处理的 Servlet 中。

```
1    session.invalidate();
```

❖ public boolean isNew()

测试取得的会话对象是否是刚刚创建的，true 表示刚刚创建，false 表示会话对象已经存在。

❖ public long getCreateTime()

取得会话对象的创建时间，返回 long 类型数据，表示从 1970.1.1.0 时开始到创建时间所间隔的毫秒数。

会话对象的功能和方法

❖ public String getId()

取得会话对象的 ID，是唯一性的字符串代码。

由于 HTTP 协议的无状态性，为了使用 Web 容器能识别不同客户而确定各自的会话对象，Web 容器需要将会话 ID 保存到客户端。当客户进行 HTTP 请求时，需要发送此 ID 给服务器，Web 服务器根据此 ID 定位服务器内部的会话对象，实现指定客户的会话跟踪。

会话对象的生命周期

会话对象的生命周期比请求对象和响应对象的生命周期要长久，可以跨越多次不同的 Web 组件 JSP 和 Servlet 的请求和响应，因此会话对象可以作为不同 JSP 和 Servlet 之间的数据共享区，保存不同页面需要访问的数据。

创建状态 新的用户请求到来。

活动状态 一个会话有效期内的所有请求，将共享一个会话对象。

销毁状态
- 客户端浏览器所保存的 SessionID 被销毁时；
- 服务器端执行会话对象的 invalidate() 方法时；
- 客户端请求间隔时间超时。

会话对象的生命周期

会话对象的生命周期比请求对象和响应对象的生命周期要长久，可以跨越多次不同的 Web 组件 JSP 和 Servlet 的请求和响应，因此会话对象可以作为不同 JSP 和 Servlet 之间的数据共享区，保存不同页面需要访问的数据。

创建状态 新的用户请求到来。

活动状态 一个会话有效期内的所有请求，将共享一个会话对象。

销毁状态

- 客户端浏览器所保存的 SessionID 被销毁时；
- 服务器端执行会话对象的 invalidate() 方法时；
- 客户端请求间隔时间超时。

会话对象的生命周期

会话对象的生命周期比请求对象和响应对象的生命周期要长久，可以跨越多次不同的 Web 组件 JSP 和 Servlet 的请求和响应，因此会话对象可以作为不同 JSP 和 Servlet 之间的数据共享区，保存不同页面需要访问的数据。

创建状态 新的用户请求到来。

活动状态 一个会话有效期内的所有请求，将共享一个会话对象。

销毁状态

- 客户端浏览器所保存的 SessionID 被销毁时；
- 服务器端执行会话对象的 invalidate() 方法时；
- 客户端请求间隔时间超时。

会话对象的生命周期

会话对象的生命周期比请求对象和响应对象的生命周期要长久，可以跨越多次不同的 Web 组件 JSP 和 Servlet 的请求和响应，因此会话对象可以作为不同 JSP 和 Servlet 之间的数据共享区，保存不同页面需要访问的数据。

创建状态 新的用户请求到来。

活动状态 一个会话有效期内的所有请求，将共享一个会话对象。

销毁状态

- 客户端浏览器所保存的 SessionID 被销毁时；
- 服务器端执行会话对象的 invalidate() 方法时；
- 客户端请求间隔时间超时。

会话对象的生命周期

会话对象的生命周期比请求对象和响应对象的生命周期要长久，可以跨越多次不同的 Web 组件 JSP 和 Servlet 的请求和响应，因此会话对象可以作为不同 JSP 和 Servlet 之间的数据共享区，保存不同页面需要访问的数据。

创建状态 新的用户请求到来。

活动状态 一个会话有效期内的所有请求，将共享一个会话对象。

销毁状态

- 客户端浏览器所保存的 SessionID 被销毁时；
- 服务器端执行会话对象的 invalidate() 方法时；
- 客户端请求间隔时间超时。

会话对象的生命周期

会话对象的生命周期比请求对象和响应对象的生命周期要长久，可以跨越多次不同的 Web 组件 JSP 和 Servlet 的请求和响应，因此会话对象可以作为不同 JSP 和 Servlet 之间的数据共享区，保存不同页面需要访问的数据。

创建状态 新的用户请求到来。

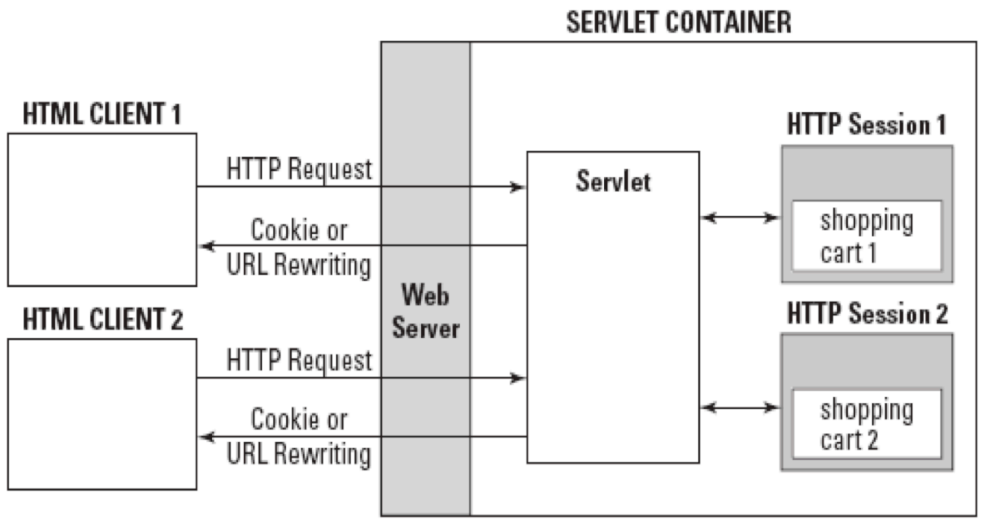
活动状态 一个会话有效期内的所有请求，将共享一个会话对象。

销毁状态

- 客户端浏览器所保存的 SessionID 被销毁时；
- 服务器端执行会话对象的 invalidate() 方法时；
- 客户端请求间隔时间超时。

会话 ID 的保存方式

默认情况下，会话的 ID 值会**自动发送到客户端的 Cookie 中进行保存**。



每次 HTTP 请求时，所有 Cookie 会随求一起发送到 Web 服务器，保存在请求对象中。Web 服务器从 Cookie 中得到 SessionID，进而定位服务器内的 HttpSession 会话对象，实现 Web 的会话跟踪功能。

会话对象的应用示例

❖ 保存用户的登录 ID

```
1   void doPost(HttpServletRequest request, HttpServletResponse response) {
2     String userId = request.getParameter("userId");
3     String password = request.getParameter("password");
4
5     User user = BeanFactory.get("USER"); // 从Bean工厂获取一个User的对象实例
6
7     if (user.validate(userId, password) { // 验证用户的有效性
8       HttpSession session = request.getSession(); // 取得会话对象
9       session.setAttribute("userId", userId); // 将用户Id保存到会话对象
10    } else { // 用户无效
11      response.sendRedirect("../login.jsp"); // 重定向到登录页面
12    }
13  }
```

会话对象的应用示例

❖ 会话对象销毁（编程一般在注销功能中）

```
1  public class LogoutAction extends HttpServlet {
2    public void doGet(HttpServletRequest request, HttpServletResponse response) {
3      HttpSession session = request.getSession();
4      session.invalidate(); // 清除此会话对象
5      response.sendRedirect("login.jsp"); // 跳转到登录页面
6    }
7  }
```

接下来…

会话基本概念

会话跟踪技术
- URL 重写
- Cookie
- Java EE 会话对象

本节习题

本节习题

❖ 简答题

1. 什么是会话跟踪？
2. Java EE 实现 Web 会话跟踪的方法有哪些？
3. URL 和 Cookie 实现会话跟踪的缺点分别有哪些？
4. 简述使用 Cookie 和 HTTP Session 实现会话跟踪的机制。

本节习题

❖ 小编程

实践课堂会话对象的应用实例，编写完整的用户登录、登录后才能浏览的主页，并在主页中包含登出按钮。

1. 登录页面为 login.html，登录处理 Servlet 为 LoginServlet.java，使用表单完成用户登录数据的提交并通过 LoginServlet 查询判断用户名和密码是否合法（可以硬编码，无需查询数据库），如用户名和密码合法则**重定向**到 index.html。

```
1       if ("admin".equals(username) && "admin".equals(password)) {
2         response.sendRedirect("index.html");
3       } else {
4         response.sendRedirect("login.html");
5       }
```

2. index.html 中需要包含 Logout 按钮，用户点击 Logout 按钮则销毁当前会话并**重定向**到 login.html。

THE END

wangxiaodong@ouc.edu.cn