# 会话

## 什么是会话

会话可简单理解为：用户开一个浏览器，点击多个超链接，访问服务器多个web资源，然后关闭浏览器，整个过程称之为一个会话。

## 会话过程中要解决的一些问题？

每个用户在使用浏览器与服务器进行会话的过程中，不可避免各自会产生一些数据，程序要想办法为每个用户保存这些数据。

例如：用户点击超链接通过一个servlet购买了一个商品，程序应该想办法保存用户购买的商品，以便于用户点结帐servlet时，结帐servlet可以得到用户购买的商品为用户结帐。

思考：用户购买的商品保存在request或servletContext中行不行？

## 保存会话数据的两种技术

### Cookie

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。

### Session

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**HttpSession**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

# Cookie API

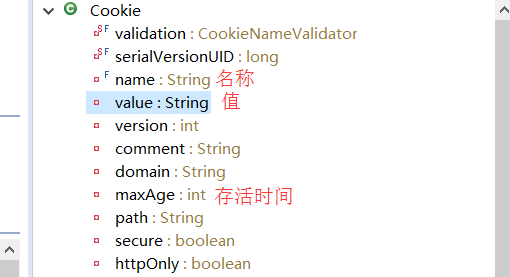
## javax.servlet.http.Cookie

* javax.servlet.http.Cookie类用于创建一个Cookie
* response接口中定义了一个addCookie方法，它用于在其响应头中增加一个相应的Set-Cookie头字段。
* request接口中也定义了一个getCookies方法，它用于获取客户端提交的Cookie。

## Cookie类的方法：

* public Cookie(String name,String value)
* setValue与getValue方法
* setMaxAge与getMaxAge方法 (秒)
* setPath与getPath方法
* setDomain与getDomain方法
* getName方法

## Cookie类的一些属性



## Cookie应用场景

### 记录上次访问时间

#### 代码实现

|  |
| --- |
|  |
|  |
|  |

#### Cookie的maxAge方法

**设置cookies的存活时间**

|  |
| --- |
|  |
|  |

**maxAge**:cookie的缓存时间。默认是-1，默认存在浏览器的缓存中。单位是秒

* 负数：表示cookie的数据存在浏览器缓存中
* 0：表示删除cookie
* 正数：缓存在持久化磁盘上的时间

#### Cookie细节

* 一个Cookie只能标识一种信息，它至少含有一个标识该信息的名称（NAME）和设置值（VALUE）。
* 一个WEB站点可以给一个WEB浏览器发送多个Cookie，一个WEB浏览器也可以存储多个WEB站点提供的Cookie。
* 浏览器一般只允许存放**300个Cookie**，每个站点最多存放20个Cookie，**每个Cookie的大小限制为4KB。**
* 如果创建了一个cookie，并将他发送到浏览器，默认情况下它是一个会话级别的cookie（即存储在浏览器的内存中），用户退出浏览器之后即被删除。若希望浏览器将该cookie存储在磁盘上，则需要使用maxAge，并给出一个以秒为单位的时间。将最大时效设为0则是命令浏览器删除该cookie。

### 记住用户名

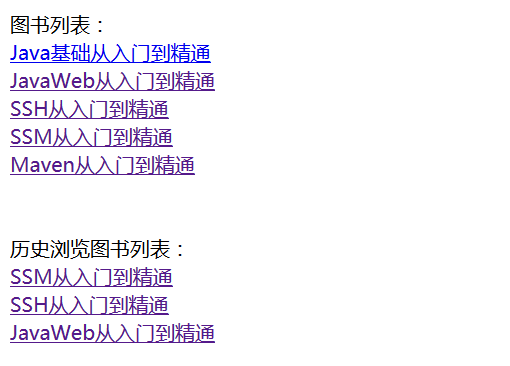
|  |
| --- |
|  |
|  |

#### 步骤：

|  |
| --- |
| **Step 01：创建一个login.jsp页面** |
|  |
| **Step 02: 创建一个Servlet,然后设置Cookies** |
|  |
| Step 03: JSP页面通过写JS，显示登录名称 |
| <script type=*"text/javascript"*>  **var** cookies = document.cookie.split(';');  //通过key获取cookie的值  **function** getCK(mkey) {    **for**(**var** i = 0;i<cookies.length;i++){  **var** kv = cookies[i].split('=');  //key有可能空字符串  **if**( kv[0].trim() == mkey){  **return** kv[1].trim();  }  }    **return** '';  }    window.onload = **function**(){  **var** remember = getCK('remember');  **var** username = getCK('username');    console.log(remember + "---");  console.log(username);  **if**(remember == 'true'){//记住密码  //console.log('记住密码');  **var** userNameInput = document.getElementById('username');  userNameInput.value = username;    **var** rememberInput = document.getElementById('remember');  rememberInput. checked="checked";  }  }  </script> |

### 历史记录

#### 案例：查看书的浏览记录



#### 原理画图讲解

#### 实现步骤

|  |
| --- |
| Step1:写个书的模型 |
|  |
| Step2:写个DBUtil 模仿数据库数据 |
|  |
| Step3:写个showAllBookServelt |
|  |
| Step4:写个ViewBookServlet |
|  |

# Session API

## session概述

* 在WEB开发中，服务器可以为每个用户浏览器创建一个会话对象（session对象），注意：一个浏览器独占一个session对象(默认情况下)。因此，在需要保存用户数据时，服务器程序可以把用户数据写到用户浏览器独占的session中，当用户使用浏览器访问其它程序时，其它程序可以从用户的session中取出该用户的数据，为用户服务。
* Session和Cookie的主要区别在于：
* Cookie是把用户的数据写给用户的浏览器。
* Session技术把用户的数据写到用户独占的session中。
* Session对象由服务器创建，开发人员可以调用request对象的getSession方法得到session对象。

## Session原理图

|  |
| --- |
|  |

* 每个浏览器存储自己的数据到Session中
* Session的数据是不能被其它浏览器共享的
* **Session一般可用于判断用户是否登录**

## Session小实验

Session小实验：使用IE访问某一个servlet，其它IE可以取到这个servlet存的数据吗？

|  |
| --- |
|  |
|  |

## Session的疑问

**疑问：服务器是如何实现一个session为一个用户浏览器服务的？**

* 服务器会为每个浏览器分配一个session ID,然后把Session ID通过Cookie的形式存储在客户端

|  |
| --- |
|  |

## Session的应用场景

### 购物车实现

|  |
| --- |
| Step1购物列表ShowAllBookServlet |
|  |
| Step2购买Servlet |
|  |
| Step3显示购物车Servlet |
|  |

### 验证码登录

|  |
| --- |
| Step1:写一个验证码的Servlet |
|  |
| Step2:写一个登录页面 |
|  |
| Step3:写一个登录处理的Servlet |
|  |

**注意：平时开发，表单使用POST请求，应该在POST请求中处理**

## HttpSession常用方法

* 把数据保存在HttpSession对象中，该对象也是一个域对象。
* void setAttribute(String name,Object value);
* Object getAttribute(String name);
* void removeAttribute(String name);
* HttpSession.getId()
* setMaxInactiveInterval(int interval) 设置session的存活时间
* invalidate() 使此会话无效

## getSession():内部执行原理

**HttpSession request.getSession():内部执行原理**

1、获取名称为JSESSIONID的cookie的值。

2、没有这样的cookie，创建一个新的HttpSession对象，分配一个唯一的SessionID，并且向客户端写了一个名字为JSESSIONID=sessionID的cookie

3、有这样的Cookie，获取cookie的值（即HttpSession对象的值），从服务器的内存中根据ID找那个HttpSession对象：

找到了：取出继续为你服务。

找不到：从2开始。

**HttpSession request.getSession(boolean create)**

参数：

true：和getSession()功能一样。

false：根据客户端JSESSIONID的cookie的值，找对应的HttpSession对象，找不到返回null（不会创建新的，只是查询）。

## Session的状态

### Session的状态三种：

创建：当浏览器第一次访问服务器动态资源就创建

活着：服务器应用运行时

死亡：

Session.invalidate();强制销毁

超时：默认30分钟

setMaxInactiveInterval(int ）单位秒

### 在Web.xml中配置Session的有效时间



### Session的持久化

#### 为什么要持久化Session

#### 持久化的优点：

节约内存空间；

确保在服务器重启或单个Web应用重启后，能回复重启前的会话；

#### 持久化状态转化

Session在其生命周期中，可能会在运行时状态和持久化状态之间转换。

**搁置**

会话从运行时状态变为持久化状态的过程称为 —— 搁置；

在以下情况下，Session会被搁置：

当服务器重启或单个Web应用终止时，Web应用中的Session会被搁置；

会话处于不活动状态的时间太长，达到了特定的限定值；

Web应用中处于运行状态的会话数目太多，达到了特定的限制值，部分Session被搁置

**激活**

会话从持久化状态变为运行时状态的过程称为激活；

在以下情况下，Session会被激活：

当服务器重启或单个Web应用重启时，Web应用中的Session会被激活

处于Session中的客户端想Web应用发出HTTP请求，相应的Session会被激活

#### 持久化对象系列化接口

当存在Session中的对象，重启tomcat时，如果有session对象，会出现下面的问题

表示对象不能被序列化，需要让Book实现序列化接口Serialize

|  |
| --- |
|  |
|  |

**会话持久化的路径是在\work\Catalina\localhost\test**

