# 一、

1. 由于HTTP协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是Session.典型的场景比如购物车，当你点击下单按钮时，由于HTTP协议无状态，所以并不知道是哪个用户操作的，所以服务端要为特定的用户创建了特定的Session，用用于标识这个用户，并且跟踪用户，这样才知道购物车里面有几本书。这个Session是保存在服务端的，有一个唯一标识。在服务端保存Session的方法很多，内存、数据库、文件都有。集群的时候也要考虑Session的转移，在大型的网站，一般会有专门的Session服务器集群，用来保存用户会话，这个时候 Session 信息都是放在内存的，使用一些缓存服务比如Memcached之类的来放 Session。  
   2. 思考一下服务端如何识别特定的客户？这个时候Cookie就登场了。每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用 Cookie 来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在 Cookie 里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了。有人问，如果客户端的浏览器禁用了 Cookie 怎么办？一般这种情况下，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户。  
   3. Cookie其实还可以用在一些方便用户的场景下，设想你某次登陆过一个网站，下次登录的时候不想再次输入账号了，怎么办？这个信息可以写到Cookie里面，访问网站的时候，网站页面的脚本可以读取这个信息，就自动帮你把用户名给填了，能够方便一下用户。这也是Cookie名称的由来，给用户的一点甜头。  
   所以，总结一下：  
   Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；  
   Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式。

# 二、

1. **不要混淆 session 和 session 实现。**
2. 本来 session 是一个抽象概念，开发者为了实现中断和继续等操作，将 user agent 和 server 之间一对一的交互，抽象为“会话”，进而衍生出“会话状态”，也就是 session 的概念。
3. 而 cookie 是一个实际存在的东西，http 协议中定义在 header 中的字段。可以认为是 session 的一种后端无状态实现。
4. 而我们今天常说的 “session”，是为了绕开 cookie 的各种限制，通常借助 cookie 本身和后端存储实现的，一种更高级的会话状态实现。
5. 所以 cookie 和 session，你可以认为是同一层次的概念，也可以认为是不同层次的概念。具体到实现，session 因为 session id 的存在，通常要借助 cookie 实现，但这并非必要，只能说是通用性较好的一种实现方案。

# 三、

1，session 在服务器端，cookie 在客户端（浏览器）  
2，session 默认被存在在服务器的一个文件里（不是内存）  
3，session 的运行依赖 session id，而 session id 是存在 cookie 中的，也就是说，如果浏览器禁用了 cookie ，同时 session 也会失效（但是可以通过其它方式实现，比如在 url 中传递 session\_id）  
4，session 可以放在 文件、数据库、或内存中都可以。  
5，用户验证这种场合一般会用 session

因此，维持一个会话的核心就是客户端的唯一标识，即 session id

# 四、

Cookie是保存在浏览器上的一些数据，一般通过HTTP响应头set cookie来设置，当然也可以通过JS脚本来直接设置，Cookie是按照网站来进行组织和保存的，每一个网站都可以在浏览器中保存一些Cookie，保存好了之后，浏览器向这个网站发出的请求都会携带这些Cookie，然后后台就可以分析这些Cookie。

Session这个单词在不同的语境下可以有不同的含义。

它可以理解为一个抽象概念，即会话，会话用于记录一个用户在我们网站上的一些行为、一些状态，可以理解为一个上下文，Context。这些用户状态可以利用Cookie直接保存在前端，也可以保存在后台，然后利用Cookie中的Session ID来标识。

在另外的一些语境下，Session又可以指在后台保存用户状态来实现会话的方式，它把用户状态存储在后台的内存、数据库等介质中，然后我们利用请求的Cookie中保存的Session ID来为这个请求找到它对应的会话。

Session在一些语境下还可能专指后台保存的用户状态。

# 五、

**COOKIE和SESSION有什么区别？**  
**cookie保存在客户端，session保存在服务器端，**  
**cookie**目的可以**跟踪会话，**也可以**保存用户喜好或者保存用户名密码**  
**session用来跟踪会话**  
①当我们登录网站勾选保存用户名和密码的时候，一般保存的都是cookie，将用户名和密码的cookie保存到硬盘中，这样再次登录的时候浏览器直接将cookie发送到服务端验证，直接username和password保存到客户端，当然这样不安全，浏览器也可以加密解密这样做，每个浏览器都可以有自己的加密解密方式，这样方便了用户，再比如用户喜欢的网页背景色，比如QQ空间的背景，这些信息也是可以通过cookie保存到客户端的，这样登录之后直接浏览器直接就可以拿到相应的偏好设置。  
  
②跟踪会话，比如某些网站中网页有不同的访问权限，有只能登录的用户访问的网页或者用户级别不同不能访问的，但是http请求是无状态的，每次访问服务端是不知道是否是登录用户，很自然的想到在http请求报文中加入登录标识就可以了，这个登录标识就可以是cookie，这样的cookie服务端要保存有所有登录用户的cookie，这样请求报文来了之后拿到登录标识cookie，在服务端进行比较久可以了。再比如购物网站，多次点击添加商品到购物车客户端很容易知道哪些物品在购物车中，但是服务端怎么知道每次添加的物品放到哪个登录用户的购物车中呢？也需要请求报文中带着cookie才行（在不登陆的情况下京东也是可以不断添加商品的，推测应该是登录的时候一并创建cookie并且发送物品信息），这些cookie都是为了跟踪会话用的，所以客户端有，服务端也有，并且服务端有全部的会话cookie。  
  
后面衍生出session技术,session技术是要使用到cookie的，之所以出现session技术，主要是为了安全。

http是无状态的协议，客户每次读取web页面时，服务器都打开新的会话，而且服务器也不会自动维护客户的上下文信息，那么要怎么才能实现网上商店中的购物车呢，session就是一种保存上下文信息的机制，它是针对每一个用户的，变量的值保存在服务器端，通过SessionID来区分不同的客户,session是以cookie或URL重写为基础的，默认使用cookie来实现，系统会创造一个名为JSESSIONID的输出cookie，我们叫做session cookie,以区别persistent cookies,也就是我们通常所说的cookie,注意session cookie是存储于浏览器内存中的，并不是写到硬盘上的，这也就是我们刚才看到的JSESSIONID，我们通常情是看不到JSESSIONID的，但是当我们把浏览器的cookie禁止后，web服务器会采用URL重写的方式传递Sessionid，我们就可以在地址栏看到 sessionid=KWJHUG6JJM65HS2K6之类的字符串。

大家请看在HTTP请求报文头的最后一行有cookie，不过是JSessionID的cookie值

Cookie: $Version=1; Skin=new;jsessionid=5F4771183629C9834F8382E23BE13C4C

比如前两个值，应该属于偏好设置之类的。  
服务端是怎么知道客户端的多个请求是隶属于一个Session呢？注意到后台的那个jsessionid=5F4771183629C9834F8382E23BE13C4C木有？原来就是通过HTTP请求报文头的Cookie属性的jsessionid的值关联起来的！（当然也可以通过重写URL的方式将会话ID附带在每个URL的后面哦）。  
明白了原理，我们就可以很容易的分辨出persistent cookies和session cookie的区别了，网上那些关于两者安全性的讨论也就一目了然了，session cookie针对某一次会话而言，会话结束session cookie也就随着消失了，而persistent cookie只是存在于客户端硬盘上的一段文本（通常是加密的），而且可能会遭到cookie欺骗以及针对cookie的跨站脚本攻击，自然不如 session cookie安全了。  
通常session cookie是不能跨窗口使用的，当你新开了一个浏览器窗口进入相同页面时，系统会赋予你一个新的sessionid，这样我们信息共享的目的就达不到了，此时我们可以先把sessionid保存在persistent cookie中，然后在新窗口中读出来，就可以得到上一个窗口SessionID了，这样通过session cookie和persistent cookie的结合我们就实现了跨窗口的session tracking（会话跟踪）。  
在一些web开发的书中，往往只是简单的把Session和cookie作为两种并列的http传送信息的方式，session cookies位于服务器端，persistent cookie位于客户端，可是session又是以cookie为基础的，明白的两者之间的联系和区别，我们就不难选择合适的技术来开发web service了。  
部分参考自：[session与cookie的区别](https://link.zhihu.com/?target=http%3A//www.cnblogs.com/xulb597/archive/2012/07/02/2573252.html)  
==================分割线==============  
举个QQ空间的例子：  
① 当我们登录QQ空间的时候，可以选择保存用户名和密码，这样下次登录的时候浏览器可以自动填充或者自动登陆，此时使用的是cookie技术，将于

http://qzone.qq.com/

域名对应的cookie保存到硬盘中，下次访问的时候浏览器查找保存在硬盘中的与该域名对应的cookie填充。  
②登录之后，我们可能做些操作，比如删除日志，发表说说，这些只有登录用户才能做的事情可以使用cookie也可以使用session进行会话跟踪  
③空间的喜好设置可以保存到硬盘cookie当中。  
-------------------------------------------------------------------------------------------------------------------------  
其实说白了session就是用来保存会话的cookie。  
下面介绍下Java中Servlet的session管理

http://lavasoft.blog.51cto.com/62575/275589 深入理解HTTP Session  
session在web开发中是一个非常重要的概念，这个概念很抽象，很难定义，也是最让人迷惑的一个名词，也是最多被滥用的名字之一，在不同的场合，session一次的含义也很不相同。这里只探讨HTTP Session。  
为了说明问题，这里基于Java Servlet理解Session的概念与原理，这里所说Servlet已经涵盖了JSP技术，因为JSP最终也会被编译为Servlet，两者有着相同的本质。  
在Java中，HTTP的Session对象用javax.servlet.http.HttpSession来表示。  
1、概念：Session代表服务器与浏览器的一次会话过程，这个过程是连续的，也可以时断时续的。在Servlet中，session指的是HttpSession类的对象，这个概念到此结束了，也许会很模糊，但只有看完本文，才能真正有个深刻理解。  
2、Session创建的时间是：  
一个常见的误解是以为session在有客户端访问时就被创建，然而事实是直到某server端程序调用 HttpServletRequest.getSession(true)这样的语句时才被创建，注意如果JSP没有显示的使用 <% @page session="false"%> 关闭session，则JSP文件在编译成Servlet时将会自动加上这样一条语句 HttpSession session = HttpServletRequest.getSession(true);这也是JSP中隐含的 session对象的来历。  
由于session会消耗内存资源，因此，如果不打算使用session，应该在所有的JSP中关闭它。  
引申：  
**1）、访问\*.html的静态资源因为不会被编译为Servlet，也就不涉及session的问题。**  
2）、当JSP页面没有显式禁止session的时候，在打开浏览器第一次请求该jsp的时候，服务器会自动为其创建一个session，并赋予其一个sessionID，发送给客户端的浏览器。**以后客户端接着请求本应用中其他资源的时候，会自动在请求头上添加：**  
**Cookie:JSESSIONID=客户端第一次拿到的session ID**  
这样，**服务器端在接到请求时候，就会收到session ID，**并根据ID在内存中找到之前创建的session对象，提供给请求使用。这也是session使用的基本原理----搞不懂这个，就永远不明白session的原理。  
下面是两次请求同一个jsp，请求头信息：  
  
**通过图可以清晰发现，第二次请求的时候，已经添加session ID的信息。**  
3、Session删除的时间是：  
1）Session超时：超时指的是连续一定时间服务器没有收到该Session所对应客户端的请求，并且这个时间超过了服务器设置的Session超时的最大时间。  
2）程序调用HttpSession.invalidate()  
3）服务器关闭或服务停止  
4、session存放在哪里：服务器端的内存中。不过session可以通过特殊的方式做持久化管理。  
5、session的id是从哪里来的，sessionID是如何使用的：当客户端第一次请求session对象时候，服务器会为客户端创建一个session，并将通过特殊算法算出一个session的ID，用来标识该session对象，当浏览器下次（session继续有效时）请求别的资源的时候，浏览器会偷偷地将sessionID放置到请求头中，服务器接收到请求后就得到该请求的sessionID，服务器找到该id的session返还给请求者（Servlet）使用。一个会话只能有一个session对象，对session来说是只认id不认人。  
6、session会因为浏览器的关闭而删除吗？  
不会，session只会通过上面提到的方式去关闭。  
7、同一客户端机器多次请求同一个资源，session一样吗？  
一般来说，每次请求都会新创建一个session。  
  
其实，这个也不一定的，总结下：对于多标签的浏览器（比如360浏览器）来说，在一个浏览器窗口中，多个标签同时访问一个页面，session是一个。对于多个浏览器窗口之间，同时或者相隔很短时间访问一个页面，session是多个的，和浏览器的进程有关。对于一个同一个浏览器窗口，直接录入url访问同一应用的不同资源，session是一样的。  
8、session是一个容器，可以存放会话过程中的任何对象。  
9、session因为请求（request对象）而产生，同一个会话中多个request共享了一session对象，可以直接从请求中获取到session对象。  
10、其实，session的创建和使用总在服务端，而浏览器从来都没得到过session对象。但浏览器可以请求Servlet（jsp也是Servlet）来获取session的信息。客户端浏览器真正紧紧拿到的是session ID，而这个对于浏览器操作的人来说，是不可见的，并且用户也无需关心自己处于哪个会话过程中。  
---------------------------------------------------------------------------------------------------  
比如下面一段使用session的代码

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

response.setContentType("text/html");

response.setCharacterEncoding("utf-8");

PrintWriter out = response.getWriter();

// 得到用户名和密码，验证

String u = request.getParameter("username");

String p = request.getParameter("password");

UserBeanBO ubb = new UserBeanBO();

if (ubb.checkUser(u, p))

{

// 1.把成功登陆的用户所有信息放入session

UserBean ub = ubb.getUserBean(u);

request.getSession().setAttribute("userInfo", ub);

// 2.把购物车的信息取出

MyCartBO mcb = (MyCartBO)request.getSession().getAttribute("mycart");

ArrayList al = mcb.showMyCart();

// 把al放入request

request.setAttribute("mycartInfo", al);

// 用户合法

request.getRequestDispatcher("success.jsp").forward(request,response);

} else

{

// 用户不合法

request.getRequestDispatcher("error.jsp").forward(request, response);

}

}

HttpSession javax.servlet.http.HttpServletRequest.getSession()

Returns the current session associated with this request, or if the request

does not have a session, creates one.

Returns: the HttpSession associated with this request

See Also:

getSession(boolean)

javax.servlet.http.HttpServletRequest.getSession() 将会返回当前request相关联的HttpSession对象，如果不存在，将会创建一个。  
翻译一下，当一个浏览器请求来到之后，Servlet处理程序（Servlet容器内部实现）将会主动检查请求信息Cookie当中是否有JSESSIONID，若有，找到对应JSESSION的HttpSession对象，如果没有，创建一个，具体的机制在Servlet容器的实现当中。

# 六、

## 【cookie和session】

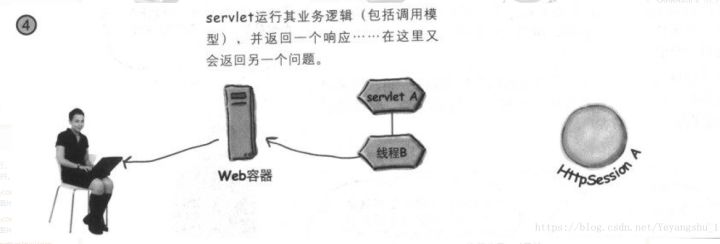
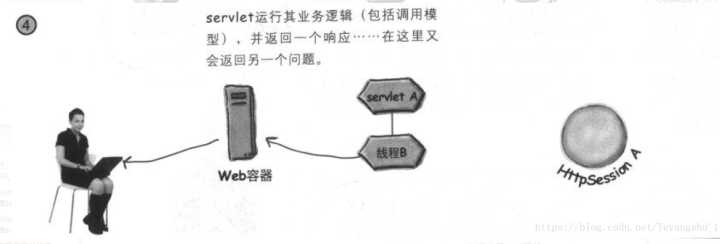
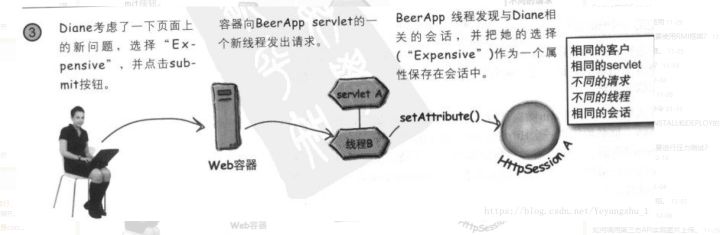
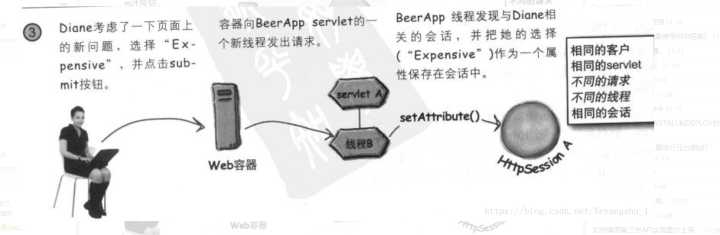
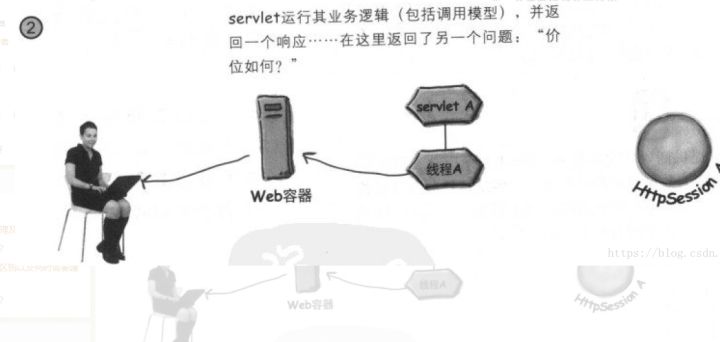
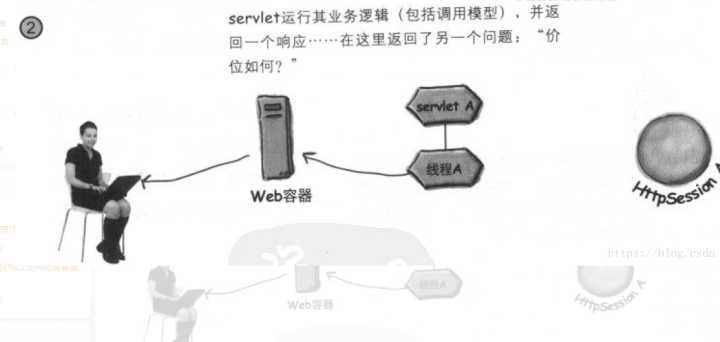
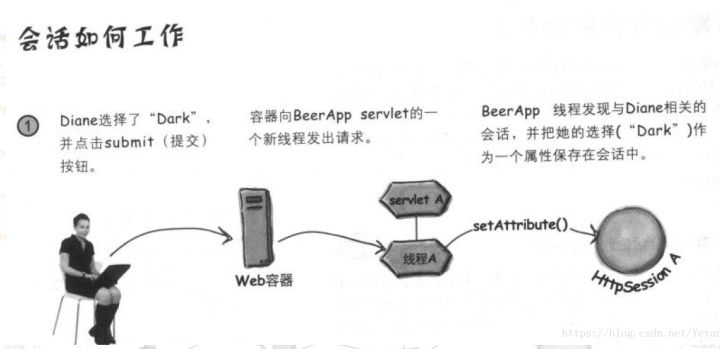
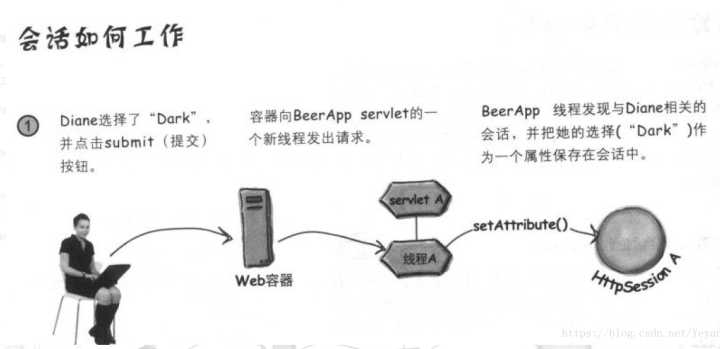
## 1 背景介绍

**什么是会话？**

用户打开一个浏览器, 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

HTTP协议是一种"无状态"协议，客户浏览器与服务器建立连接，发出请求，得到相应，然后关闭连接，这意味着每次客户端检索网页时，客户端打开一个单独的连接到 Web 服务器，服务器会自动不保留之前客户端请求的任何记录。

所以容器不能辨认下一个请求和之前的请求是不是同一个请求，对于容器而言，每个请求都是新的。



使用浏览器与服务器进行会话的过程中,不可避免会产生一些数据, Web服务器没有短期记忆，如何保存这些用户数据？

客户需要一个唯一的会话ID

客户的第一次请求，容器会生成一个唯一的会话ID，并通过响应把它发回客户端，客户在以后的每一个请求中发回这个会话ID。容器看到后，就会找到匹配的会话，并把这个会话与请求关联。

如何交换会话信息？

保存会话数据的两种技术:

cookie和session

## 2 知识剖析

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。

## 3 常见问题

浏览器禁用Cookie怎么办？

## 4 解决方案

URL重写、隐藏表单字段

## 5 编码实战

cookie：

cookie工具类

public class CookieUtil {  
/\*\*  
\* 添加cookie  
\* @param response  
\* @param key cookie主键  
\* @param value cookie值  
\*/  
public static void addCookie(HttpServletResponse response,String key,String value){  
Cookie cookie = new Cookie(key,value);  
//设置路径  
cookie.setPath("/");  
//设置保存时间为1天,单位为s  
cookie.setMaxAge(24\*60\*60);  
//通过response.addCookie将此条cookie添加到客户端  
response.addCookie(cookie);  
}  
/\*\*  
\* 删除cookie  
\* @param response  
\* @param request  
\* @param key  
\*/  
public static void deleteCookie(HttpServletResponse response, HttpServletRequest request,String key) {  
//获取浏览器访问服务器时传递来的cookie数组  
Cookie cookies[] = request.getCookies();  
if (cookies != null) {  
for (int i = 0; i < cookies.length; i++) {  
if (cookies[i].getName().equals(key)) {  
Cookie cookie = new Cookie(key, null);  
//此路径需与之前创建时相同  
cookie.setPath("/");  
//设置为0即为删除  
cookie.setMaxAge(0);  
response.addCookie(cookie);  
}  
}  
}  
}  
/\*\*  
\* 获取指定cookie  
\* @param request  
\* @param key  
\* @return  
\* @throws UnsupportedEncodingException  
\*/  
public static String getCookieValue(HttpServletRequest request,String key) throws UnsupportedEncodingException {  
//通过request.getCookies获取客户端提交的所有cookie  
for (Cookie cookie : request.getCookies()) {  
if (cookie.getName().equals(key)) {  
return URLDecoder.decode(cookie.getValue(), "UTF-8");  
}  
}  
return null;  
}  
}

**controller**

public class CookieController {

@RequestMapping(value = "/cookie", method = [RequestMethod.POST](https://link.zhihu.com/?target=http%3A//requestmethod.post/))

public String cookie(@RequestParam String name, String password, HttpServletResponse response, Model model){

CookieUtil.addCookie(response,"name",name);

CookieUtil.addCookie(response,"password",password);

model.addAttribute("name",name);

model.addAttribute("password",password);

return "cookie";

}

@RequestMapping(value = "getcookie",method = RequestMethod.GET)

public String getcookie(HttpServletRequest request,Model model) throws UnsupportedEncodingException {

String name =CookieUtil.getCookieValue(request,"name");

String password = CookieUtil.getCookieValue(request,"password");

model.addAttribute("name",name);

model.addAttribute("password",password);

return "cookie";

}

@RequestMapping(value = "deletecookie",method = RequestMethod.GET)

public String deletecookie(HttpServletRequest request,HttpServletResponse response,Model model) throws UnsupportedEncodingException {

// CookieUtil.deleteCookie(response,request,"name");

CookieUtil.deleteCookie(response,request,"password");

String name = CookieUtil.getCookieValue(request,"name");

String password = CookieUtil.getCookieValue(request,"password");

model.addAttribute("name",name);

model.addAttribute("password",password);

return "redirect:test.jsp";

}

}

表单jsp

<div align="center">

<body>

<form action="/cookie" method="post">

名称<input type="text" name="name">

<br />

密码<input type="text" name="password" /><br>

<input type="submit" value="提交" />

</form>

<a href="/getcookie">获取cookie</a>

<a href="/deletecookie">删除cookie</a>

</body>

</div>



session：摘自菜鸟教程

public class seesion extends HttpServlet {

private static final long serialVersionUID = 1L;

public void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException

{

// 如果不存在 session 会话，则创建一个 session 对象

HttpSession session = request.getSession(true);

// 获取 session 创建时间

Date createTime = new Date(session.getCreationTime());

// 获取该网页的最后一次访问时间

Date lastAccessTime = new Date(session.getLastAccessedTime());

//设置日期输出的格式

SimpleDateFormat df=new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");

String title = "Servlet Session 实例";

Integer visitCount = new Integer(0);

String visitCountKey = new String("visitCount");

String userIDKey = new String("userID");

String userID = new String("Runoob");

if(session.getAttribute(visitCountKey) == null) {

session.setAttribute(visitCountKey, new Integer(0));

}

// 检查网页上是否有新的访问者

if (session.isNew()){

title = "Servlet Session 实例";

session.setAttribute(userIDKey, userID);

} else {

visitCount = (Integer)session.getAttribute(visitCountKey);

visitCount = visitCount + 1;

userID = (String)session.getAttribute(userIDKey);

}

session.setAttribute(visitCountKey, visitCount);

// 设置响应内容类型

response.setContentType("text/html;charset=UTF-8");

PrintWriter out = response.getWriter();

String docType = "<!DOCTYPE html>\n";

out.println(docType +

"<html>\n" +

"<head><title>" + title + "</title></head>\n" +

"<body bgcolor=\"#f0f0f0\">\n" +

"<h1 align=\"center\">" + title + "</h1>\n" +

"<h2 align=\"center\">Session 信息</h2>\n" +

"<table border=\"1\" align=\"center\">\n" +

"<tr bgcolor=\"#949494\">\n" +

" <th>Session 信息</th><th>值</th></tr>\n" +

"<tr>\n" +

" <td>id</td>\n" +

" <td>" + session.getId() + "</td></tr>\n" +

"<tr>\n" +

" <td>创建时间</td>\n" +

" <td>" + df.format(createTime) +

" </td></tr>\n" +

"<tr>\n" +

" <td>最后访问时间</td>\n" +

" <td>" + df.format(lastAccessTime) +

" </td></tr>\n" +

"<tr>\n" +

" <td>用户 ID</td>\n" +

" <td>" + userID +

" </td></tr>\n" +

"<tr>\n" +

" <td>访问统计：</td>\n" +

" <td>" + visitCount + "</td></tr>\n" +

"</table>\n" +

"</body></html>");

}

}

<servlet>

<!-- 类名 -->

<servlet-name>seesion</servlet-name>

<!-- 所在的包 -->

<servlet-class>com.ppteng.controller.seesion</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>seesion</servlet-name>

<!-- 访问的网址 -->

<url-pattern>/test/session</url-pattern>

</servlet-mapping>1234567891011

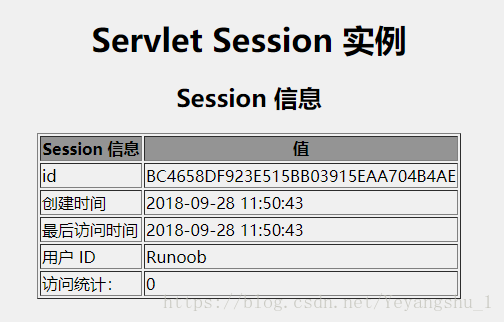
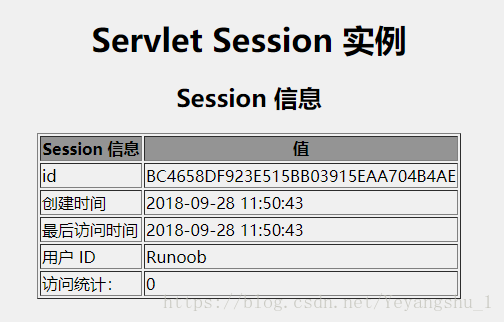
设置会话时间（分钟）：方便进行对比

<session-config>

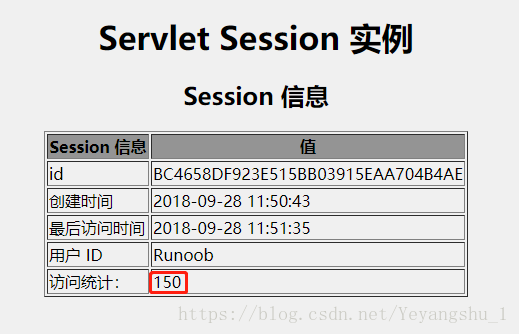
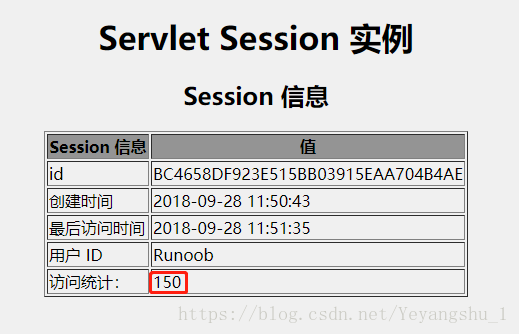
<session-timeout>1</session-timeout>

</session-config>123

运行访问http://localhost:8080/test/session

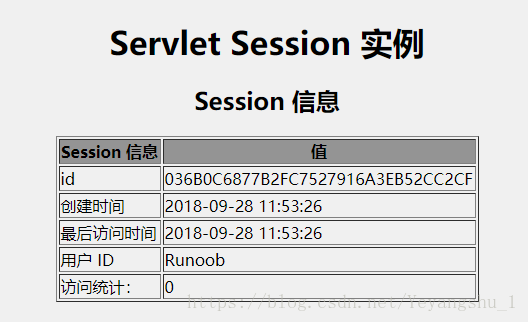
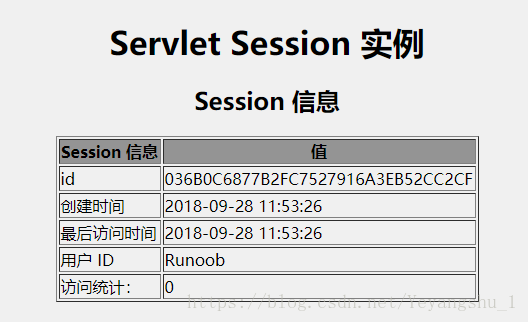


**刷新页面**



两次会话只要不超过1min，就会累加

无操作1min后，session就会过期，刷新页面，session重建



## 6 更多讨论

**Q1：cookie有哪些不足呢？**

A1：cookie是客户端技术，对客户端是可见的，数据安全性较低。

**Q2：cookie和session存取方式不同**

A2：cookie中只能保管ASCII字符串，假如需求存取Unicode字符或者二进制数据，需求先进行编码。Cookie中也不能直接存取Java对象。若要存储略微复杂的信息，运用Cookie是比拟艰难的。

Session中能够存取任何类型的数据，包括而不限于String、Integer、List、Map等。Session中也能够直接保管Java Bean乃至任何Java类，对象等，运用起来十分便当。能够把Session看做是一个Java容器类。

**Q3：网站如何选择？**

A3：Session是保管在服务器端的，每个用户都会产生一个Session。假如并发访问的用户十分多，会产生十分多的Session，耗费大量的内存。发访问量极高的网站，是不太可能运用Session来追踪客户会话的。

Cookie保管在客户端，不占用服务器资源。假如并发阅读的用户十分多，Cookie是很好的选择。