**会话管理**

# 授课内容

授课科目：JavaEE

授课内容：会话管理

授课类型：讲授+实践

授课时间：3+1学时

主讲教师：解卫静

# 教学目标要求

## 能力目标：

1. 培养学生掌握会话管理的基本技术实现；
2. 培养学生掌握Session技术实现，并能够使用Session解决实际问题。

## 知识目标：

1. 培养学生理解隐藏域、Cookie、URL重写三种基本会话管理技术的实现过程和基本原理；
2. 培养学生理解Session会话管理的原理和相关接口。

# 课件分析

概 述：首先介绍Web应用程序的基础协议——HTTP协议，由于HTTP协议基于请求/响应模式、无连接、无状态的特点，当一个用户需要在多次请求/响应之间建立关联时，产生会话管理的概念。然后介绍隐藏域、Cookie和URL重写会话管理技术的原理和实现方式。重点介绍Session的原理、实现和应用案例。

教学重点：Cookie、Session的编程实现。

教学难点：Cookie与Session的原理和区别。

# 教学方法

讲授法、案例法、实践教学法、类比法、讨论法等。

# 教学过程

## 课前补充

（0分钟）

## 上节回顾

（2分钟）

上节课学习了Servlet的生命周期，分为四个阶段：

1. 加载和实例化：由Servlet容器完成；
2. 初始化：实例化Servlet后会调用init()方法；
3. 处理客户请求：调用service()方法允许Servlet响应请求(根据请求方式调用doGet()或doPost())；

4.销毁：调用destroy()方法。

## 作业点评

（0分钟）

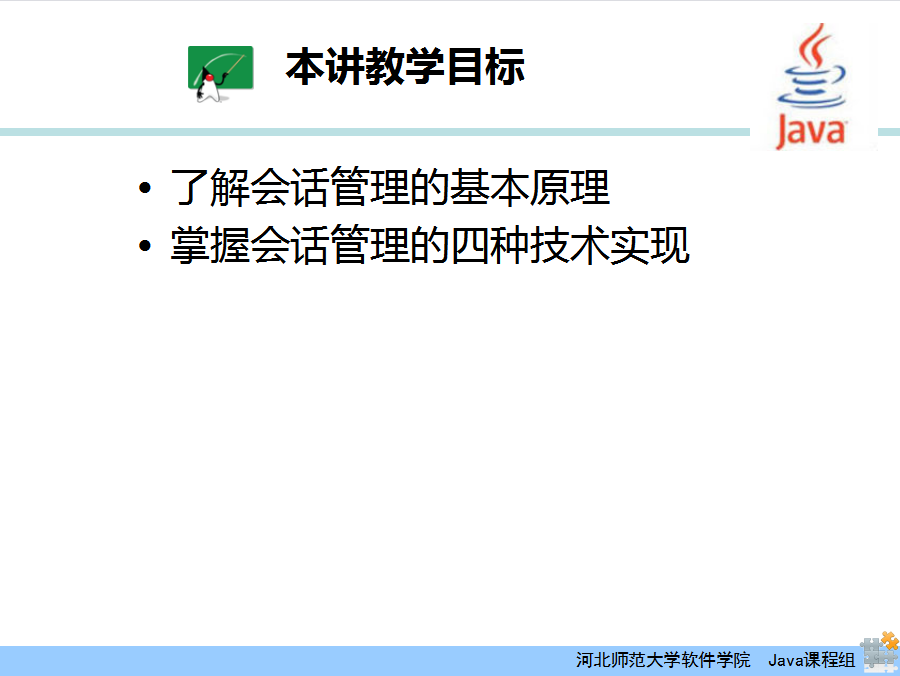
## 导入新课

（2分钟）

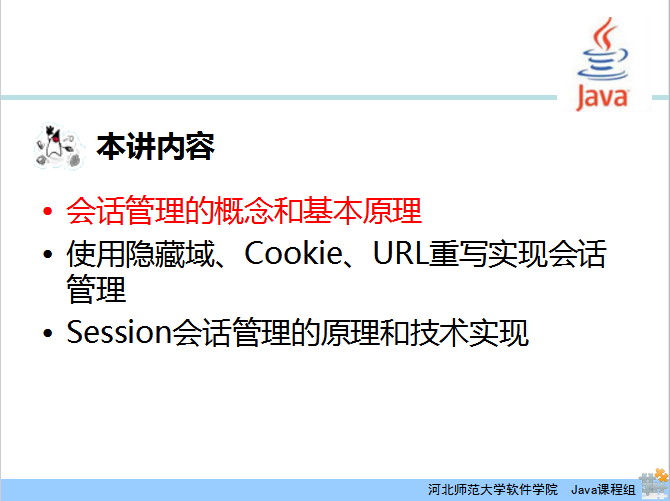
学完前面两章Servlet模型，可以通过向服务器发送账号、密码进行登录，但是登录以后想要进行其他操作，比如修改个人信息，服务器如何知道这些操纵来自同一个用户呢？通过本章会话管理的学习，这个问题会得以解答。本章将介绍会话管理的原理和技术实现，包括隐藏域、Cookie、URL重写和Session，将重点介绍Servlet容器中会话管理技术Session。

## 讲授新课

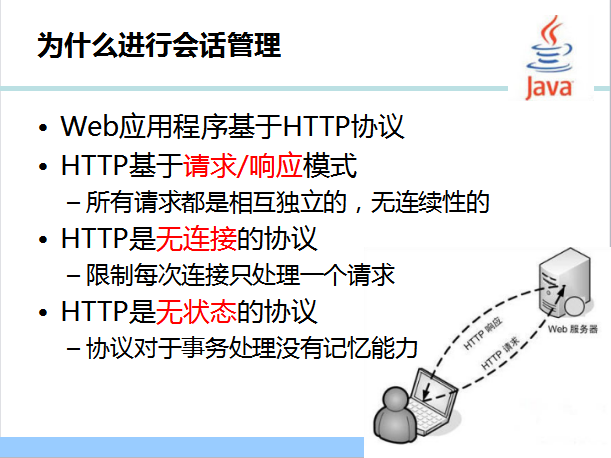
（120分钟）



本章的教学目标是了解会话管理的基本原理和相关技术实现，包括三种基本技术，隐藏域、Cookie、URL重写，以及Servlet容器中的会话管理机制Session。



学习会话管理技术之前，先来了解会话管理的概念和基本原理。

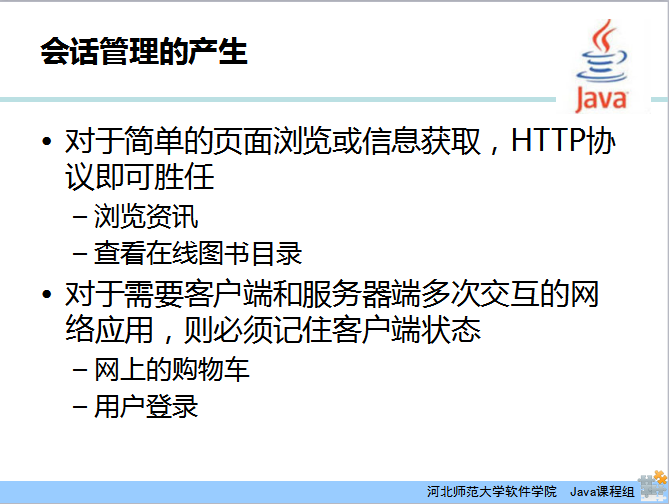


在Web应用程序中进行会话管理的一个重要因素是Web应用程序是基于HTTP协议的，即Web应用程序是使用HTTP协议传输数据的。在此简单回顾HTTP协议的一些特点。

HTTP协议基于请求/响应模式，也就是客户端浏览器向服务器进行一次请求，服务器就会返回给客户端浏览器一个响应，请求一次响应一次，一个请求对应一个响应。而所有请求都是相互独立的，无连续性的，也就是根据HTTP协议，所有请求之间是没有任何关联的。

HTTP的一个特点是无连接。无连接就是请求时建连接、请求完释放连接，即客户端浏览器发出一个请求并受到服务器的应答后，客户端浏览器和服务器就会断开连接。而且限制每次连接只处理一个请求。

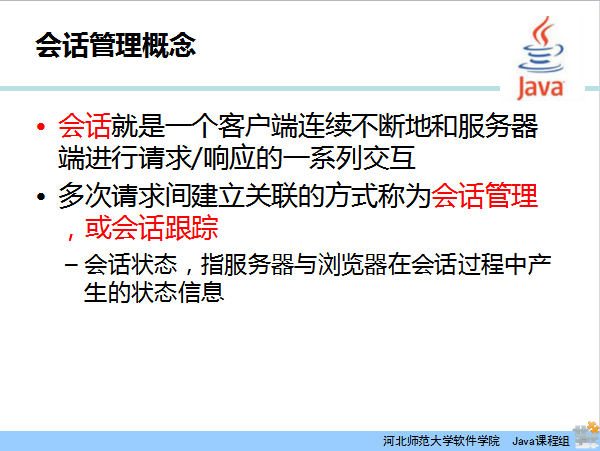
HTTP是无状态的协议。无状态是指协议对于事务处理没有记忆能力。也就是说在客户端浏览器与服务器之间的请求和响应结束后，在服务器上并不保存任何客户端的信息。这就意味着如果后续处理需要前面的信息，则必须重传。



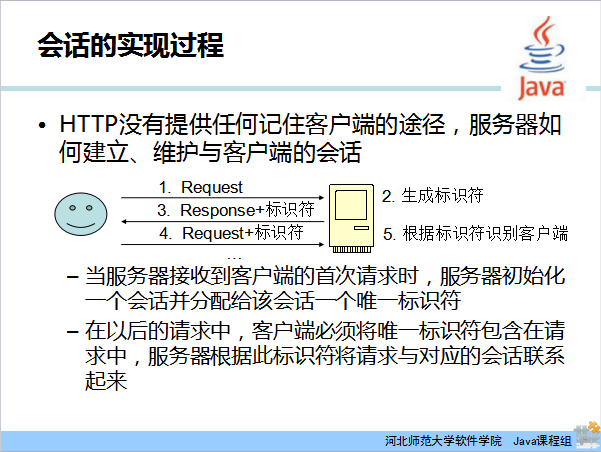
在实际的应用场景中，比如需要浏览资讯，随便在网易、搜狐、新浪等网站看看新闻，或者查看在线图书目录，这些简单的页面浏览或信息获取，一次请求/响应就可以完成，所以对于这些功能，HTTP协议可以完全胜任。

但是对于需要客户端浏览器和服务器端多次交互的网络应用，也就是用户需要进行多次请求服务器才能完成的功能，HTTP协议还能胜任吗？比如网上的购物车功能，加第二件或第三、四件商品进购物车时，用户能看到之前加进来的商品，这时候仅依靠无状态的HTTP协议是不行的，因为这时需要服务器知道这一组请求是同一个用户发来的。再比如此章开头提到的用户登录，服务器需要知道是哪个用户登录的，修改个人信息时才不会修改了别人的，这就需要服务器记录客户端的信息。

但是根据HTTP协议，服务器端不会记录客户端的信息，那这时候就需要程序员手动在同一个用户的请求之间建立联系，也就是进行会话管理，达到服务器记录客户端信息的效果。



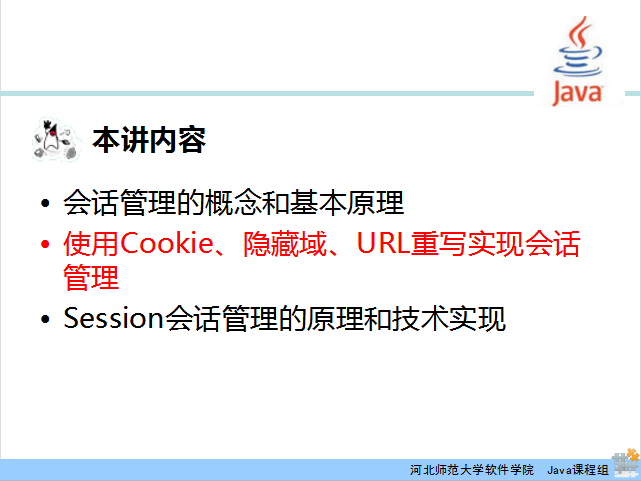
会话就是一个客户端浏览器连续不断地和服务器端进行请求/响应的一系列交互。这里交互就是请求/响应的过程，客户端浏览器连续不断地与服务器进行交互，那一个小时交互一次算连续不断呢?还是一分钟交互一次算连续不断？还是一秒钟？下面来解释。HTTP协议是无状态的，所以服务器不知道哪些请求是同一个用户的，现在我们给每次请求/响应建立一些关联，只要这些关联没有失效，那么这时进行的请求/响应就是连续不断的，它们属于同一个会话。比如，从登录进入系统到注销退出系统，这整个过程称之为一个会话。而多次请求间建立关联的方式称为会话管理，又叫会话跟踪。使多次请求间产生关联的客户端信息，也就是服务器与客户端浏览器在会话过程中，每次请求所产生的状态信息，为会话状态。比如在购物车中每一次添加的商品信息，即为回话状态。借助会话状态，服务器能够把属于同一会话中的一系列的请求和响应过程关联起来。



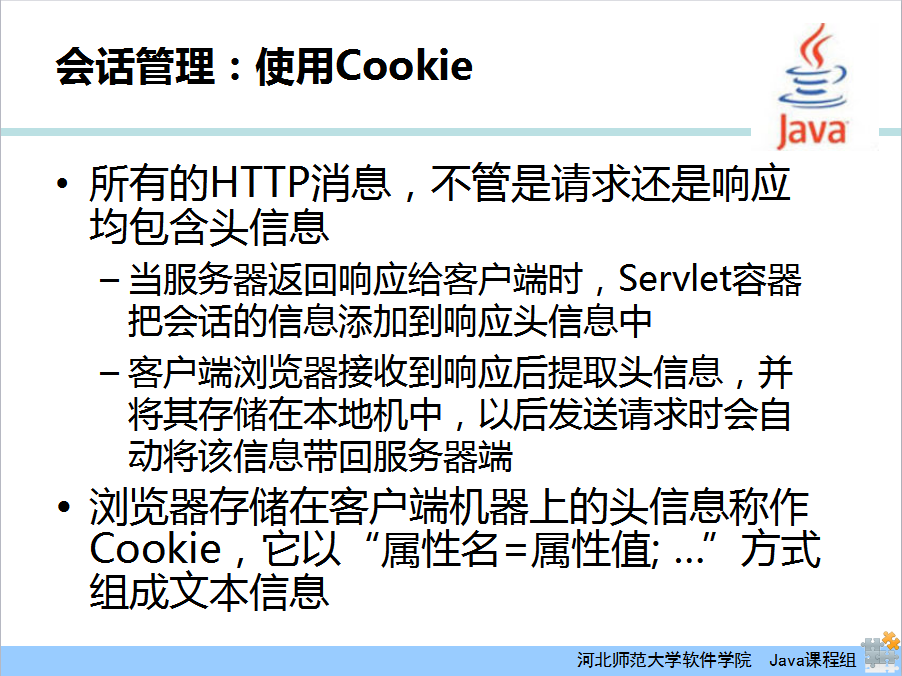
HTTP没有提供任何记住客户端的途径，服务器如何建立、维护与客户端浏览器的会话？

当服务器接收到客户端浏览器的首次请求时，服务器会初始化一个会话并分配给该会话一个唯一标识符，服务器将这个标识符响应给客户端浏览器，在以后的请求中，客户端浏览器必须将唯一标识符包含在请求中，服务器根据此标识符将请求与对应的会话联系起来。

比如用户登录，客户端浏览器第一此访问服务器时，服务器为此用户生成一个标识符，这个标识符我们可以理解为一个填有信息的单子，服务器响应客户端浏览器时，把这个单子一并递过去，客户端浏览器下次请求时，附带着这个单子，以便服务器辨别这是之前登录的用户。



所有会话管理技术实现会话管理的原理是一样的，但有不同的实现方式，基本方式有三种——Cookie、隐藏域和URL重写。



三种基本方式中使用最广泛的是Cookie。由于所有的HTTP消息，不管是请求还是响应都包含头信息，所以当服务器返回响应给客户端浏览器时，Servlet容器把会话的信息添加到响应头信息中；客户端浏览器接收到响应后提取头信息，并将其存储在本地机中；客户端浏览器以后发送请求时，会将该信息放在请求头信息中回传给服务器端，从而服务器就可以识别这是同一个用户的请求。

浏览器存储在客户端机器上的头信息称作Cookie,它以“属性名=属性值; …”这种键值对方式组成文本信息。这里注意:

1.一个Web站点可以给一个客户端浏览器发送多个Cookie，一个客户端浏览器也可以存储多个Web站点提供的Cookie。服务器传送给各个客户端浏览器的数据是可以各不相同的。

2.一旦客户端浏览器保存了某个Cookie，那么它在以后每次访问该服务器时，都应在HTTP请求头中将这个Cookie回传给服务器。但是客户端浏览器也不是发送它所接收到的所有Cookie，它会检查当前请求的域名和路径，只要这两项分别与Cookie中的域名和路径匹配，才会发送，只要有一项不匹配，它就不会发送。所以客户端浏览器不会把网站一的Cookie发送给网站二，也不会把网站二的Cookie发送给网站一。



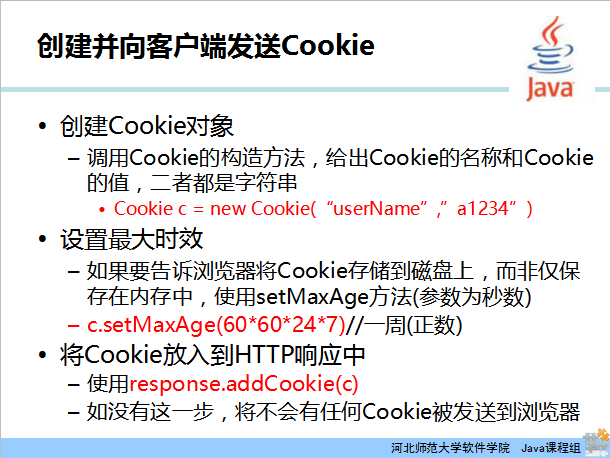
比如用Cookie实现用户自动登录功能，其编程思路如下。

1.用户填写账号、密码，点击登录按钮向服务器发出HTTP请求；

2.服务器收到请求后，创建Cookie并放入HTTP响应中；

3.客户端浏览器收到服务器的响应后，保存头信息中的Cookie数据；

4.当客户端浏览器再次访问服务器进行其他操作时，会在请求头信息中将Cookie回传给服务器，服务器通过读取Cookie识别这个用户。



为了封装Cookie信息，Servlet API中提供了一个javax.servlet.http.Cookie类，该类包含了生成Cookie信息和提取Cookie信息各个属性的方法。服务器端创建并向客户端浏览器发送Cookie，的过程如下。

1.首先调用Cookie的构造方法创建一个Cookie对象。

|  |
| --- |
| Cookie c = new Cookie(”userID”,”a1234”); |

这个方法需要设置两个参数，第一个参数是Cookie的名称，第二个参数是Cookie的值，两个参数都以字符串形式的给出。这里自动登录账户的Cookie名称设置为“userId”，Cookie值为“a1234”

2.通过setMaxAge()方法设置Cookie的最大时效。

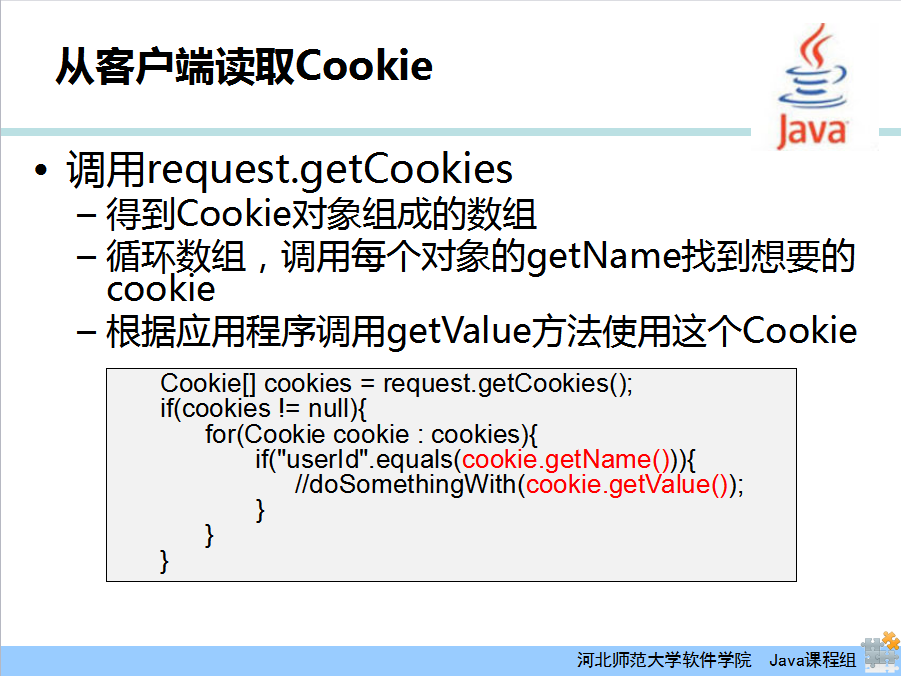
|  |
| --- |
| c.setMaxAge(60\*60\*24\*7); |

这个方法需要设置一个int型参数，这个参数代表Cookie的最长有效时间，单位为妙。当这个数为正数时，客户端浏览器会将Cookie存储到磁盘上，而非仅保存在内存中。这里60\*60\*24\*7代表一周。

3.将Cookie放入到HTTP响应中。

|  |
| --- |
| response.addCookie(c); |

Cookie是放在响应头信息中传递的。如果没有这一步，将不会有任何Cookie被发送到客户端浏览器。



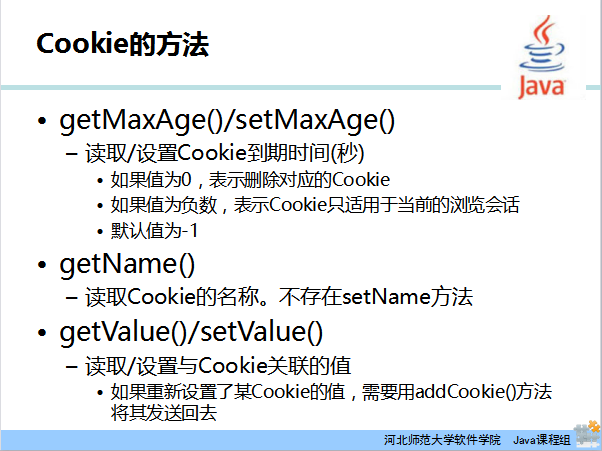
客户端浏览器保存了这个Cookie以后，再次访问该服务器时，会在HTTP请求头中将这个Cookie回传给服务器，让服务器读取，读取过程如下。

|  |
| --- |
| Cookie[] cookies = request.getCookies();  if(cookies != null){  for(Cookie cookie : cookies){  if("userId".equals(cookie.getName())){  //doSomethingWith(cookie.getValue());  }  }  } |

1.得到Cookie对象组成的数组;

2.循环数组，调用每个对象的getName()找到想要的Cookie;

3.根据应用程序调用getValue()方法使用这个Cookie的值。



Cookie常用的方法如下。

getMaxAge()/setMaxAge()，即读取/设置Cookie的到期时间。这里注意：

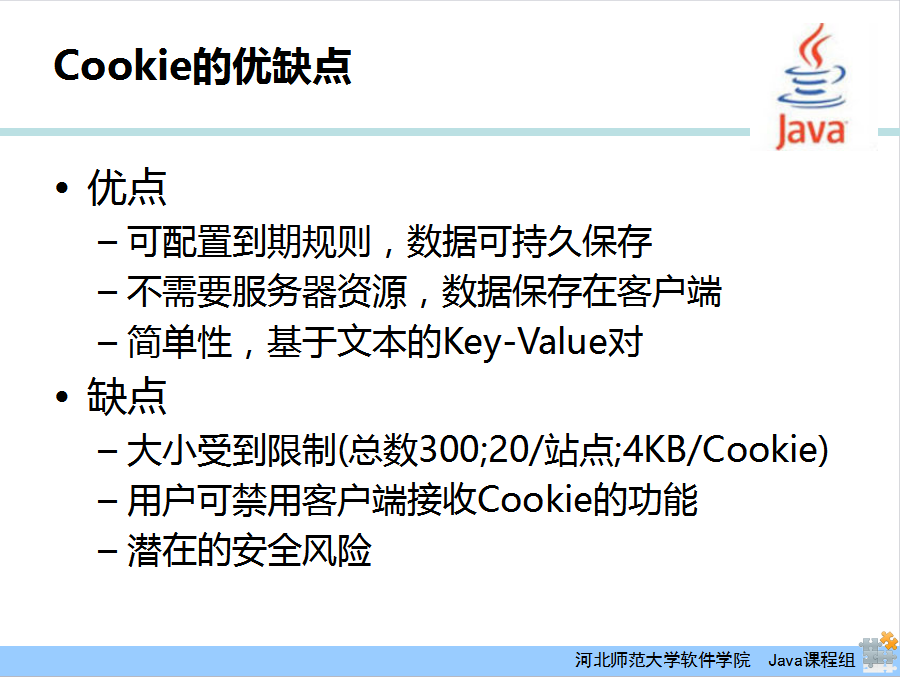
1.setMaxAge()的参数为正数时，客户端浏览器会将Cookie存储到磁盘上，而非仅保存在内存中。

2.参数为0，表示删除对应的Cookie。不存在专门删除Cookie的方法。

3.参数为负数，表示Cookie只适用于当前的浏览会话。默认值是-1，表示关闭客户端浏览器，Cookie消失。

getName()方法为读取Cookie的名称。不存在setName()方法。

getValue()/setValue()，读取/设置与Cookie关联的值。如果重新设置了某Cookie的值，需要用response.addCookie()将其发送回去。



Cookie的优点：

1.可配置到期规则，数据可持久保存。可以设置一个比较大的值比如5年5\*365\*24\*60\*60；

2.数据保存在客户端，所以不需要服务器资源；

3.简单性，基于文本的键值对，Cookie名称对应Cookie内容。

Cookie的缺点：

1.大小受限制。客户端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。

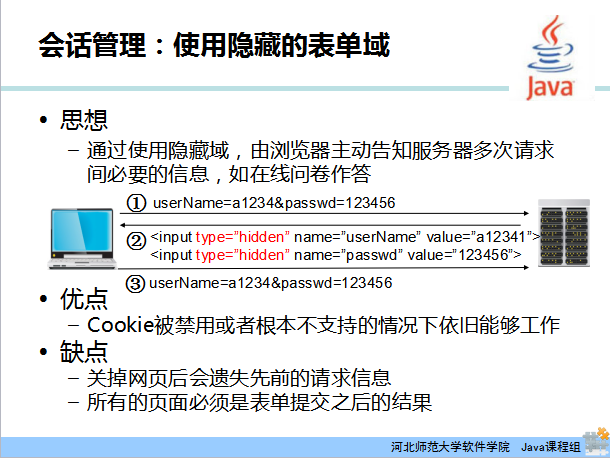
2.用户可禁用客户端浏览器接收Cookie的功能

潜在的安全风险。Cookie作为用户身份的替代，如果被截获并提交给服务器，用户的信息就可能会被盗用。

另外，在谷歌浏览器中查看Cookie(如下图所示)的方法:

设置->高级->内容设置->Cookie；或直接访问chrome://settings/content/cookies。





Cookie技术是将信息保存在客户端浏览器，以头信息的方式与服务器进行传递。使用隐藏域是由客户端浏览器主动告知服务器多次请求间必要的信息，将请求间必要的信息保存在客户端的表单里，以表单的方式向服务器传递信息。隐藏域，即隐藏的表单域，当表单元素input的type属性值为hidden时，这个输入字段在页面中是隐藏的，对于用户是不可见的，这就是一个隐藏域。

比如使用隐藏域实现用户登录功能，过程如下。

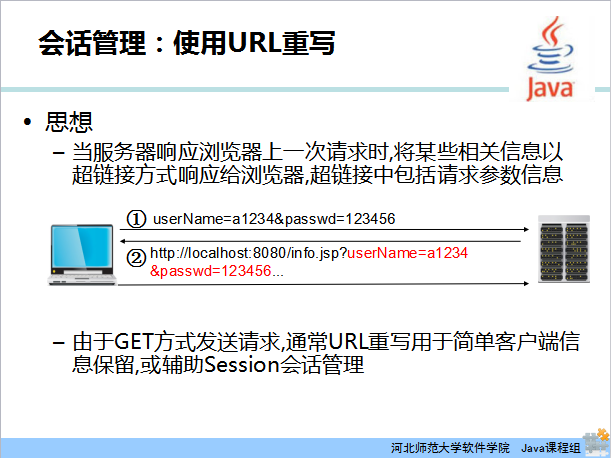
1.用户填写账号、密码，点击登录按钮向服务器发出HTTP请求；

2.服务器收到请求并进行处理后，将账号、密码响应给客户端浏览器；

3.客户端浏览器收到服务器的响应后，将账号、密码保存在隐藏域中；

4.当客户端浏览器再次访问服务器进行其他操作时，会将隐藏域中的账号、密码再次传给服务器，服务器通过验证账号和密码来确定用户。

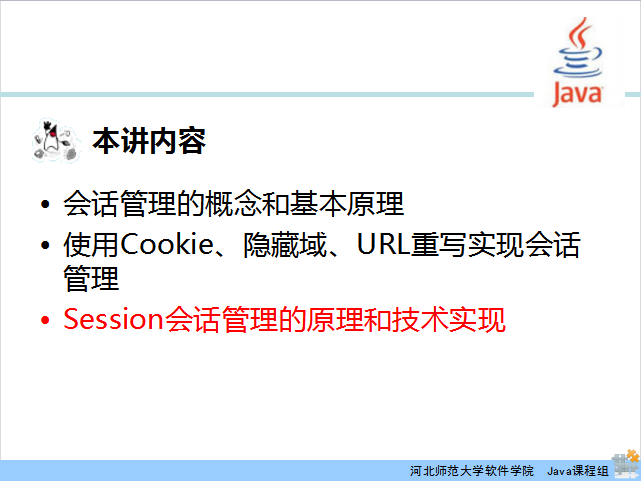
使用隐藏的表单域缺点是关掉网页后会遗失先前的请求信息，而且所有的页面必须是表单提交之后的结果。优点是Cookie被禁用或者根本不支持Cookie的情况下依旧能够工作。



所谓URL重写，其实就是GET请求参数的应用，当服务器响应客户端浏览器上一次请求时,将某些相关信息以超链接方式响应给客户端浏览器,超链接中包括请求参数信息。

比如用URL重写的方式实现用户登录功能。服务器在响应结果中加入超链接，并把账号、密码以超链接的请求参数形式发送给服务器。这样服务器就可以进行验证了。

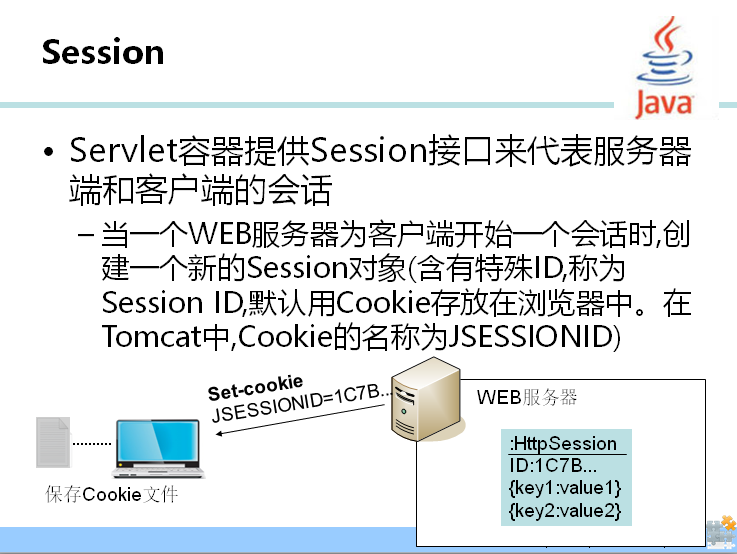
显然，因为URL重写是在超链接之后附加信息，所以必须以GET方式发送请求，再加上GET本身可以携带的请求参数长度有限，因此大量的客户端信息保留并不适合使用URL重写。通常URL重写是用在一些简单的客户端信息保留，或者辅助Session会话管理，这一点将在下一节讲到。



上一节介绍三个会话管理的基本方式。无论哪个方式，都必须在服务器自行处理对客户端浏览器的响应，将需要传递的所有具体的关联信息传送给客户端浏览器，并在之后的请求中将这些信息发送给服务器，服务器辨识请求间的关联。这其中传递的信息量都比较大，而在Servlet/JSP中有一种更好的会话管理的机制Session。

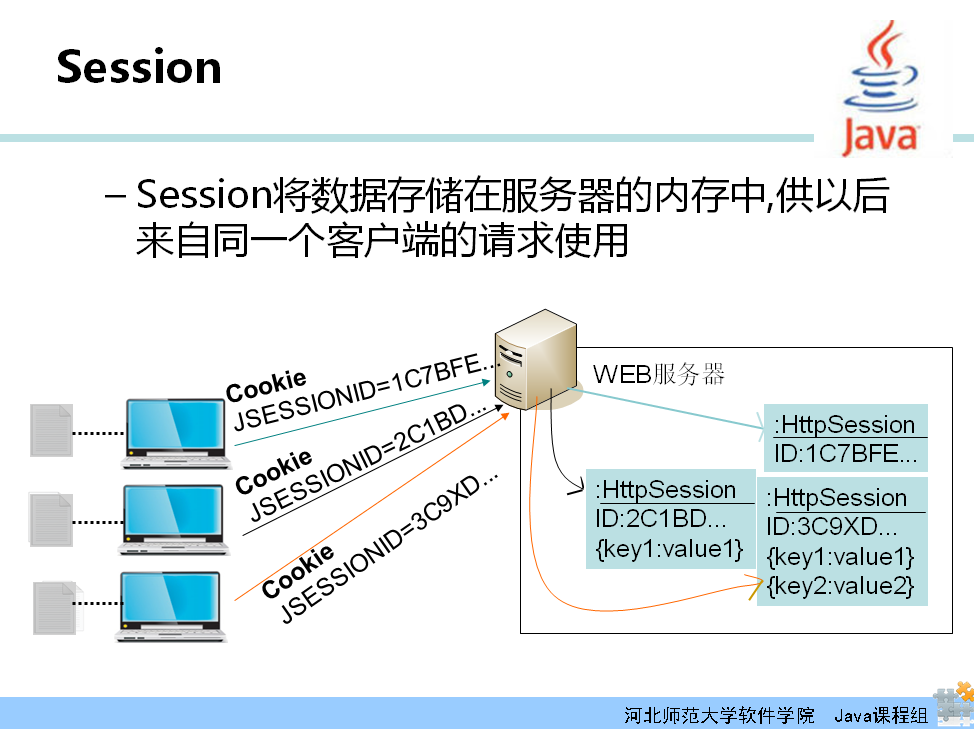


Servlet进行会话管理时，可以在服务器中创建专门用来保存请求之间信息的对象Session。Session是Java里HttpSession类的实例的简称。可以使用HttpServletRequest的getSession()方法取得Session对象。使用Session的setAttribute()方法可以将信息设置为对象的属性。在会话期间就可以当作服务器“记得”客户端的信息，如果想取出这些信息，使用Session的getAttribute()方法。

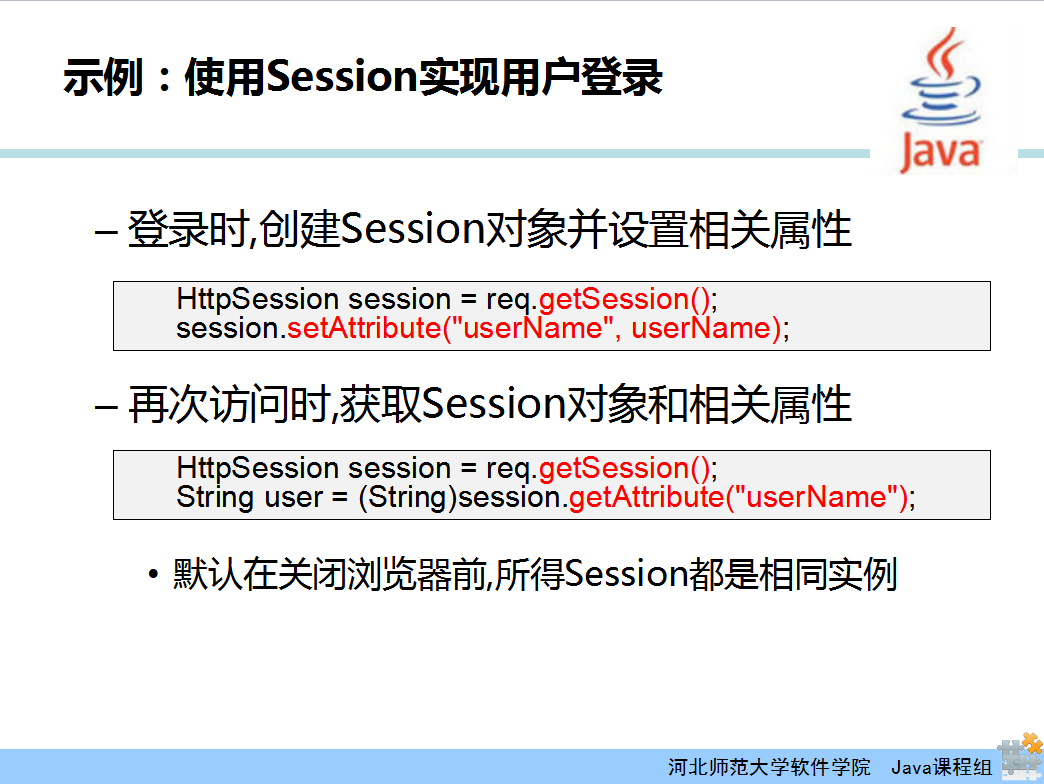


Servlet容器提供Session接口来代表服务器端和客户端浏览器的会话。当服务器接到客户端浏览器发来的请求时，Servlet容器会为客户端浏览器开始一个会话，也就是使用getSession()方法为这个客户端浏览器创建一个新的Session对象，每个对象都含有一个特殊的ID,称为Session ID。在服务器响应客户端浏览器时，默认使用Cookie将Session ID存放在客户端浏览器中。在Tomcat中,Cookie的名称为JSESSIONID。

所以，使用Session来进行会话管理时，设定为属性的Session对象存储在服务器端，Cookie保存在客户端浏览器，Cookie中存放的是只服务器传过来的Session对象的ID。



所以真正保存请求间信息的是Session对象。Session将数据信息存储在服务器端的内存中,供以后来自同一个客户端浏览器的请求使用。所以客户端浏览器下次请求服务器时，自动将存储Session ID的Cookie发送给服务器，服务器通过这个ID号来查找对应的Session对象为用户服务。



使用Session实现自动登录的过程如下：

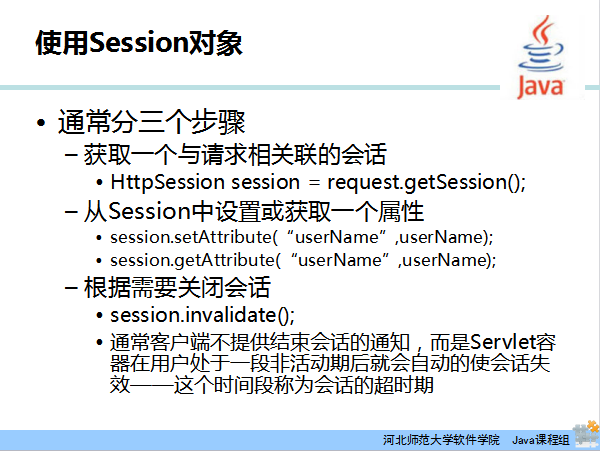
1.客户端浏览器第一次发出登录请求后，服务器创建Session对象并设置属性，这样就将账号保存在了Session属性中；

2.服务器作出响应时将Session ID传给客户端浏览器，客户端浏览器将Session ID以Cookie的形式保存在客户端浏览器；

3.用户登录以后进行其他操作时，客户端浏览器将Cookie自动发送给服务器，服务器用getSession()方法通过客户端浏览器传过来的Session ID获取之前的Session对象，并获取之前保存在这个对象属性中的账号信息。

默认在关闭客户端浏览器前，取得的Session都是相同的实例。因为我们并没有设置Cookie的最大时效，所以Cookie的最大时效是它的默认值-1，也就是在关闭客户端浏览器时Cookie就会消失。在Cookie消失以后，那就谁也找不到那个Session对象了。

如果关闭客户端浏览器并重新打开，并向服务器发出请求，这时存Session ID的Cookie已经消失了，所以没有Session ID发给服务器，服务器找不到原来的Session对象，就会重新创建一个Session对象。



使用Session对象通常分三个步骤。

1.通过request.getSession()方法获取一个与请求相关联的Session。如果客户端浏览器中有这个站点的Cookie，并且通过Cookie中的Session ID，服务器找到了对应的Session对象，那获取的就是这个对象；如果客户端浏览器没有这个站点的Cookie或者通过Session ID没有找到对应的Session对象，那么通过这个方法获得的就是一个新创建的Session对象；

2.通过setAttribute()方法设置Session的属性或者通过getAttribute()方法获取Session的属性；

3.根据需要使用invalidate()方法关闭会话。因为Session保存在服务器，会占用服务器的内存空间,所以不需要使用Session时，可以执行invalidate()使其失效，也就是将这个Session删除。

通常客户端浏览器不提供结束会话的通知，而是Servlet容器在用户处于一段非活动期后就会自动的使会话失效—这个时间段称为会话的超时期。所以如果会话超过一定时间还不进行请求的话，就可以使用invalidate()方法将为这个用户服务的Session对象销毁。



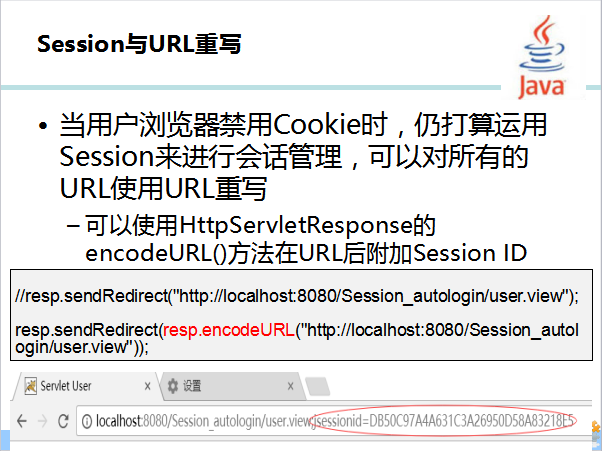
Session对象失效:当用户超出指定会话期时间处于非活动状态时,会话自动结束。比如说会话超时期为一小时，那么如果用户在停止操作超过一小时的情况下，再次访问的时候，Session失效。典型的应用案例就是用户登录之后走开了一小时后回来的时候，需要重新登录。

可以使用setMaxInactiveInterval()方法设置会话的超时期，或者通过web.xml的<session-timeout>标签设置，也可以通过程序设定条件，使用方法invalidate()使Session失效。

这里注意，除非程序通知服务器删除一个Session，否则服务器会一直保留，程序一般都是在用户做注销的时候发个指令去删除Session。然而客户端浏览器从来不会主动在关闭之前通知服务器它将要关闭，因此服务器根本不会有机会知道客户端浏览器已经关闭，之所以会有这种错觉，是大部分Session机制都使用会话Cookie来保存Session ID，而关闭客户端浏览器后这个Session ID就消失了，再次连接服务器时也就无法找到原来的Session。如果服务器设置的Cookie被保存到硬盘上，或者使用某种手段改写客户端浏览器发出的HTTP请求头，把原来的Session ID发送给服务器，则再次打开客户端浏览器仍然能够找到原来的Session。

恰恰是由于关闭客户端浏览器不会导致Session被删除，迫使服务器为Session设置了一个失效时间，当距离客户端浏览器上一次使用Session的时间超过这个失效时间时，服务器就可以认为客户端浏览器已经停止了活动，才会把Session删除以节省存储空间。

使用Session进行会话管理时，默认客户端浏览器关闭时Cookie失效，如果需要设置Cookie的最大时效，可以通过SessionCookieConfig接口，但是要注意设定SessionCookieConfig必须在ServletContext初始化前，所以必须在Web.xml中设定存储Session ID的Cookie存活期限等信息。



当用户客户端浏览器禁用Cookie时，Session没起作用，是因为客户端浏览器没有得到Session ID，所有就没有Session ID发送给服务器供其查找对应的Session对象。如果用户客户端浏览器禁用Cookie时，仍打算运用Session来进行会话管理，可以使用Session搭配URL重写来实现。URL重写在这里的作用就是传递Session ID。使用HttpServletResponse的encodeURL()可以向客户端浏览器响应的URL后会附加Session ID，这样当用户再次请求服务器时，会将Session ID发送给服务器。

例如URL重写辅助Session实现用户登录。当服务器响应给客户端浏览器一个URL时，使用encodeURL()处理这个URL。如将代码一修改为代码二。

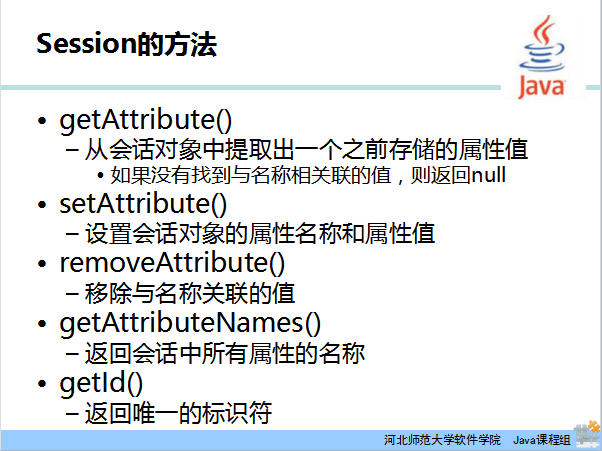
代码一：

|  |
| --- |
| resp.sendRedirect("http://localhost:8080/Session\_autologin/user.view"); |

代码二：

|  |
| --- |
| resp.sendRedirect(resp.encodeURL("http://localhost:8080/Session\_autologin/user.view")); |

当客户端浏览器以这个URL访问服务器时，会发现URL后附带了参数Session ID。



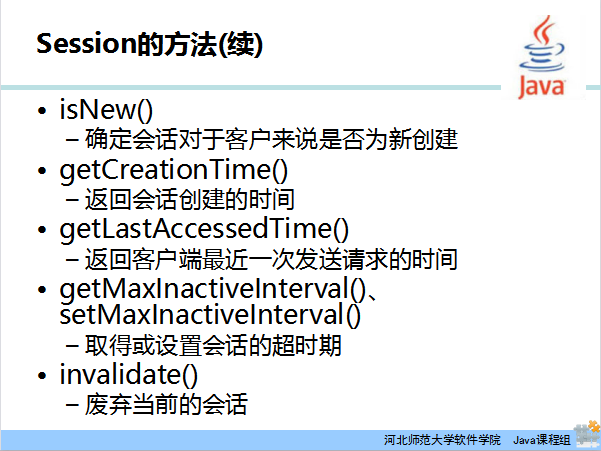
Session的getAttribute()方法，是从从会话对象中提取出一个之前存储的值，这里注意如果没有找到与名称相关联的值，则返回null。

Session的setAttribute()方法,设置会话对象的属性名称和属性值。

removeAttribute()方法，是移除与属性名称关联的值。

getAttributeNames()是获取当前会话中所有属性的名称。

getId()方法是返回当前会话的唯一标识符Session ID。



isNew()方法的返回值为boolean型，确定会话对于客户来说是否为新创建；

getCreationTime()，返回会话创建的时间；

getLastAccessedTime()，返回客户端浏览器最近一次发送请求的时间；

getMaxInactiveInterval()、setMaxInactiveInterval()，取得或设置会话的超时期，即当前Session对象多长时间不被访问将无效；

Invalidate()方法，废弃、销毁当前的会话。

## 课时小结

（10分钟）

1. HTTP协议的特点。
2. 会话和会话管理的概念。
3. 使用隐藏域实现会话管理的原理和技术实现。
4. 使用Cookie实现会话管理的原理和技术实现。
5. 使用URL重写实现会话管理的原理和技术实现。
6. 使用Session实现会话管理的原理和技术实现

# 布置作业

（5分钟）

1. 完成会话管理的实验手册。
2. 分别使用Cookie和Session技术实现用户登录功能。
3. 使用Session实现购物车功能。