# HTTP协议

1、HTTP协议的主要特点可概括如下：

1. 支持客户/服务器模式
2. 简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快
3. 灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记
4. 无连接：**无连接的含义是限制每次连接只处理一个请求**。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间
5. 无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快

## HTTP协议详解

### HTTP-URL

1、http(超文本传输协议)是一个基于请求与响应模式的、无状态的、应用层的协议，常基于TCP的连接方式，HTTP1.1版本中给出一种持续连接的机制，绝大多数的Web开发，都是构建在HTTP协议之上的Web应用。

2、HTTP URL(URL是一种特殊类型的URI，包含了用于查找某个资源的足够的信息)的格式如下：

http://host[":"port]/[abs\_path]

1. http表示要通过HTTP协议来定位网络资源
2. host表示合法的Internet主机域名或者IP地址
3. port指定一个端口号，为空则使用缺省端口80
4. abs\_path指定请求资源的URI
5. 如果URL中没有给出abs\_path，那么当它作为请求URI时，必须以"/"的形式给出，通常这个工作浏览器自动帮我们完成。

3、eg:

1. 输入：www.guet.edu.cn

浏览器自动转换成：http://www.guet.edu.cn/

1. http:192.168.0.116:8080/index.jsp

### HTTP请求

1、http请求由三部分组成，分别是：请求行、消息报头、请求正文

1. 请求行以一个方法符号开头，以空格分开，后面跟着请求的URI和协议的版本，格式如下：Method Request-URI HTTP-Version CRLF
2. 其中Method表示请求方法
3. Request-URI是一个统一资源标识符
4. HTTP-Version表示请求的HTTP协议版本
5. CRLF表示回车和换行(除了作为结尾的CRLF外，不允许出现单独的CR或LF字符)

2、请求方法(所有方法全为大写)有多种，各个方法的解释如下：

1. GET：请求获取Request-URI所标识的资源
2. POST：在Request-URI所标识的资源后附加新的数据
3. HEAD：请求获取由Request-URI所标识的资源的响应消息报头
4. PUT：请求服务器存储一个资源，并用Request-URI作为其标识
5. DELETE：请求服务器删除Request-URI所标识的资源
6. TRACE：请求服务器回送收到的请求信息，主要用于测试或诊断
7. CONNECT：保留将来使用
8. OPTIONS：请求查询服务器的性能，或者查询与资源相关的选项和需求

3、应用举例：

1. GET方法：在浏览器的地址栏中输入网址的方式访问网页时，浏览器采用GET方法向服务器获取资源

eg:GET /form.html HTTP/1.1 (CRLF)

1. POST方法要求被请求服务器接受附在请求后面的数据，常用于提交表单。

eg：POST /reg.jsp HTTP/ (CRLF)

Accept:image/gif,image/x-xbit,... (CRLF)

...

HOST:www.guet.edu.cn (CRLF)

Content-Length:22 (CRLF)

Connection:Keep-Alive (CRLF)

Cache-Control:no-cache (CRLF)

(CRLF) //该CRLF表示消息报头已经结束，在此之前为消息报头

user=jeffrey&pwd=1234 //此行以下为提交的数据

1. HEAD方法与GET方法几乎是一样的，对于HEAD请求的回应部分来说，它的HTTP头部中包含的信息与通过GET请求所得到的信息是相同的。利用这个方法，不必传输整个资源内容，就可以得到Request-URI所标识的资源的信息。该方法常用于测试超链接的有效性，是否可以访问，以及最近是否更新。

2、请求报头后述

3、请求正文(略)

### HTTP响应

### HTTP报头

### 利用telnet观察http协议通讯过程

## HTTP方法的安全性和幂等性

1、Http协议规定了不同方法的安全特性和幂等特性，作为服务提供者的服务器必需为客户端提供这些特性

* **安全性**：仅指该方法的多次调用不会产生副作用，不涉及传统意义上的"安全"，这里的副作用是指资源状态。即，安全的方法不会修改资源状态，尽管多次调用的返回值可能不一样(被其他非安全方法修改过)
* **幂等性**：是指该方法多次调用返回的效果(形式)一致，客户端可以重复调用并且期望同样的结果。幂等的含义类似于编程语言中的setter方法，一次调用和多次调用产生的效果是一致的，都是对一个变量进行赋值。安全性和幂等性含义有些接近，容易搞混

2、HTTP方法的安全性幂等性见下表

|  |  |  |
| --- | --- | --- |
| **方法名** | **安全性** | **幂等性** |
| GET | 是 | 是 |
| HEAD | 是 | 是 |
| OPTIONS | 是 | 是 |
| DELETE | 否 | 是 |
| PUT | 否 | 是 |
| POST | 否 | 否 |

3、可以认为安全的方法都是只读的方法(GET, HEAD, OPTIONS)，不会改变资源状态，显然，这三个方法也是幂等的

4、DELETE方法的语义表示删除服务器上的一个资源，第一次删除成功后该资源就不存在了，资源状态改变了，所以DELETE方法不具备安全特性。然而HTTP协议规定DELETE方法是幂等的，每次删除该资源都要返回状态码200 OK，服务器端要实现幂等的DELETE方法，必须记录所有已删除资源的元数据(Metadata)，否则，第二次删除后返回的响应码就会类似404 Not Found了

5、PUT和POST方法语义中都有修改资源状态的意思，因此都不是安全的。但是PUT方法是幂等的，POST方法不是幂等的，这么设计的理由是

* HTTP协议规定，POST方法修改资源状态时，URL指示的是该资源的父级资源，待修改资源的ID信息在请求体中携带
* 而PUT方法修改资源状态时，URL直接指示待修改资源。因此，同样是创建资源，重复提交POST请求可能产生两个不同的资源，而重复提交PUT请求只会对其URL中指定的资源起作用，也就是只会创建一个资源

## POST和GET

1、POST和GET本质上没有区别，一个用于传递数据，另一个用于修改数据。GET具有安全性和幂等性，POST不具有安全性也不具有幂等性

### GET和POST与数据如何传递没有关系

1、GET和POST是由HTTP协议定义的。在HTTP协议中，Method和Data(URL，Body，Header)是正交的两个概念，也就是说，使用哪个Method与应用层的数据如何传输是没有相互关系的

2、HTTP没有要求POST方法必须将数据放在BODY中，同样没有要求GET方法必须将数据放在URL中而不能放在BODY中

3、而上述这些(POST方法必须将数据放在BODY中，GET方法必须将数据放在URL中而不能放在BODY中)只HTML标准对HTTP协议的用法的约定

### HTTP协议对GET和POST都没有对长度的限制

1、HTTP协议明确地指出了，HTTP头和Body都没有长度的要求。而对于URL长度上的限制，有两方面的原因造成

1. 浏览器。据说早期的浏览器会对URL长度做限制。据说IE对URL长度会限制在2048个字符内(流传很广，而且无数同事都表示认同)。但我自己试了一下，我构造了90K的URL通过IE9访问live.com，是正常的。网上的东西，哪怕是Wikipedia上的，也不能信
2. 服务器。URL长了，对服务器处理也是一种负担。原本一个会话就没有多少数据，现在如果有人恶意地构造几个几M大小的URL，并不停地访问你的服务器。服务器的最大并发数显然会下降。另一种攻击方式是，把告诉服务器Content-Length是一个很大的数，然后只给服务器发一点儿数据，嘿嘿，服务器你就傻等着去吧。哪怕你有超时设置，这种故意的次次访问超时也能让服务器吃不了兜着走。有鉴于此，多数服务器出于安全啦、稳定啦方面的考虑，会给URL长度加限制。但是这个限制是针对所有HTTP请求的，与GET、POST没有关系

### 99%的人都理解错了HTTP中GET与POST的区别

1、GET和POST是HTTP请求的两种基本方法，要说它们的区别，接触过WEB开发的人都能说出一二，最直观的区别就是GET把参数包含在URL中，POST通过request body传递参数。当你在面试中被问到这个问题，你的内心充满了自信和喜悦

1. GET在浏览器回退时是无害的，而POST会再次提交请求
2. GET产生的URL地址可以被Bookmark，而POST不可以
3. GET请求会被浏览器主动cache，而POST不会，除非手动设置
4. GET请求只能进行url编码，而POST支持多种编码方式
5. GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留
6. GET请求在URL中传送的参数是有长度限制的，而POST么有
7. 对参数的数据类型，GET只接受ASCII字符，而POST没有限制
8. GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息
9. GET参数通过URL传递，POST放在Request body中

2、如果我告诉你GET和POST本质上没有区别你信吗？让我们扒下GET和POST的外衣，坦诚相见吧

* GET和POST是什么?HTTP协议中的两种发送请求的方法
* HTTP是什么？HTTP是基于TCP/IP的关于数据如何在万维网中如何通信的协议
* HTTP的底层是TCP/IP。所以GET和POST的底层也是TCP/IP，也就是说，GET/POST都是TCP链接。GET和POST能做的事情是一样一样的。你要给GET加上request body，给POST带上url参数，技术上是完全行的通的。
* 那么，"标准答案"里的那些区别是怎么回事？
* 在我大万维网世界中，TCP就像汽车，我们用TCP来运输数据，它很可靠，从来不会发生丢件少件的现象
* 但是如果路上跑的全是看起来一模一样的汽车，那这个世界看起来是一团混乱，送急件的汽车可能被前面满载货物的汽车拦堵在路上，整个交通系统一定会瘫痪
* 为了避免这种情况发生，交通规则HTTP诞生了。HTTP给汽车运输设定了好几个服务类别，有GET，POST，PUT，DELETE等等，HTTP规定，当执行GET请求的时候，要给汽车贴上GET的标签(设置method为GET)，而且要求把传送的数据放在车顶上(url中)以方便记录。如果是POST请求，就要在车上贴上POST的标签，并把货物放在车厢里
* 当然，你也可以在GET的时候往车厢内偷偷藏点货物，但是这是很不光彩;也可以在POST的时候在车顶上也放一些数据，让人觉得傻乎乎的。HTTP只是个行为准则，而TCP才是GET和POST怎么实现的基本
* 但是，我们只看到HTTP对GET和POST参数的传送渠道(url还是requrest body)提出了要求。"标准答案"里关于参数大小的限制又是从哪来的呢？
* 在我大万维网世界中，还有另一个重要的角色：运输公司。不同的浏览器(发起http请求)和服务器(接受http请求)就是不同的运输公司。虽然理论上，你可以在车顶上无限的堆货物(url中无限加参数)。但是运输公司可不傻，装货和卸货也是有很大成本的，他们会限制单次运输量来控制风险，数据量太大对浏览器和服务器都是很大负担。业界不成文的规定是，(大多数)浏览器通常都会限制url长度在2K个字节，而(大多数)服务器最多处理64K大小的url。超过的部分，恕不处理。如果你用GET服务，在request body偷偷藏了数据，不同服务器的处理方式也是不同的，有些服务器会帮你卸货，读出数据，有些服务器直接忽略，所以，虽然GET可以带request body，也不能保证一定能被接收到哦。
* 好了，现在你知道，GET和POST本质上就是TCP链接，并无差别。但是由于HTTP的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同
* GET和POST还有一个重大区别，简单的说：GET产生一个TCP数据包，POST产生两个TCP数据包
* 对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200(返回数据);
* 而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok(返回数据)。
* 也就是说，GET只需要汽车跑一趟就把货送到了，而POST得跑两趟，第一趟，先去和服务器打个招呼"嗨，我等下要送一批货来，你们打开门迎接我"，然后再回头把货送过去。
* 因为POST需要两步，时间上消耗的要多一点，看起来GET比POST更有效。因此Yahoo团队有推荐用GET替换POST来优化网站性能。但这是一个坑!跳入需谨慎。为什么
* GET与POST都有自己的语义，不能随便混用。
* 据研究，在网络环境好的情况下，发一次包的时间和发两次包的时间差别基本可以无视。而在网络环境差的情况下，两次包的TCP在验证数据包完整性上，有非常大的优点。
* 并不是所有浏览器都会在POST中发送两次包，Firefox就只发送一次

### 何时用POST何时用GET

1、满足下列任一条件采用POST

1. 请求的结果有持续性的副作用，例如，数据库内添加新的数据行
2. 若使用GET方法，则表单上收集的数据可能让URL过长
3. 要传送的数据不是采用7位的ASCII编码。

2、满足下列任一条件采用GET

1. 请求是为了查找资源，HTML表单数据仅用来帮助搜索
2. 请求结果无持续性的副作用
3. 收集的数据及HTML表单内的输入字段名称的总长不超过1024个字符

## Cookie

1、(Session)跟踪是Web程序中常用的技术，用来跟踪用户的整个会话。常用的会话跟踪技术是Cookie与Session。Cookie通过在客户端记录信息确定用户身份，Session通过在服务器端记录信息确定用户身份

### Cookie机制

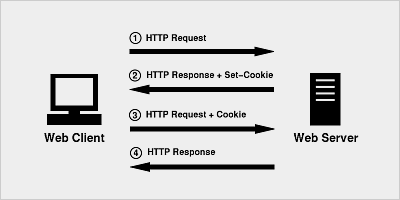
1、Cookie技术是客户端的解决方案，Cookie就是由服务器发给客户端的特殊信息，而这些信息以文本文件的方式存放在客户端，然后客户端每次向服务器发送请求的时候都会带上这些特殊的信息。让我们说得更具体一些：当用户使用浏览器访问一个支持Cookie的网站的时候，用户会提供包括用户名在内的个人信息并且提交至服务器；接着，服务器在向客户端回传相应的超文本的同时也会发回这些个人信息，当然这些信息并不是存放在HTTP响应体(Response Body)中的，而是存放于HTTP响应头(Response Header)；当客户端浏览器接收到来自服务器的响应之后，浏览器会将这些信息存放在一个统一的位置，对于Windows操作系统而言，我们可以从：[系统盘]:\Documents and Settings[用户名]\Cookies目录中找到存储的Cookie；自此，客户端再向服务器发送请求的时候，都会把相应的Cookie再次发回至服务器。而这次，Cookie信息则存放在HTTP请求头(Request Header)了。有了Cookie这样的技术实现，服务器在接收到来自客户端浏览器的请求之后，就能够通过分析存放于请求头的Cookie得到客户端特有的信息，从而动态生成与该客户端相对应的内容。通常，我们可以从很多网站的登录界面中看到"请记住我"这样的选项，如果你勾选了它之后再登录，那么在下一次访问该网站的时候就不需要进行重复而繁琐的登录动作了，而这个功能就是通过Cookie实现的

2、在程序中，会话跟踪是很重要的事情。理论上，一个用户的所有请求操作都应该属于同一个会话，而另一个用户的所有请求操作则应该属于另一个会话，二者不能混淆。例如，用户A在超市购买的任何商品都应该放在A的购物车内，不论是用户A什么时间购买的，这都是属于同一个会话的，不能放入用户B或用户C的购物车内，这不属于同一个会话。而Web应用程序是使用HTTP协议传输数据的。HTTP协议是无状态的协议。一旦数据交换完毕，客户端与服务器端的连接就会关闭，再次交换数据需要建立新的连接。这就意味着服务器无法从连接上跟踪会话。即用户A购买了一件商品放入购物车内，当再次购买商品时服务器已经无法判断该购买行为是属于用户A的会话还是用户B的会话了。要跟踪该会话，必须引入一种机制

3、Cookie就是这样的一种机制。它可以弥补HTTP协议无状态的不足。在Session出现之前，基本上所有的网站都采用Cookie来跟踪会话

4、如果你把Cookies看成为http协议的一个扩展的话，理解起来就容易的多了，其实本质上cookies就是http的一个扩展。有两个http头部是专门负责设置以及发送cookie的，它们分别是Set-Cookie以及Cookie。当服务器返回给客户端一个http响应信息时，其中如果包含Set-Cookie这个头部时，意思就是指示客户端建立一个cookie，并且在后续的http请求中自动发送这个cookie到服务器端，直到这个cookie过期。如果cookie的生存时间是整个会话期间的话，那么浏览器会将cookie保存在内存中，浏览器关闭时就会自动清除这个cookie。另外一种情况就是保存在客户端的硬盘中，浏览器关闭的话，该cookie也不会被清除，下次打开浏览器访问对应网站时，这个cookie就会自动再次发送到服务器端。一个cookie的设置以及发送过程分为以下四步

1. 客户端发送一个http请求到服务器端
2. 服务器端发送一个http响应到客户端，其中包含Set-Cookie头部
3. 客户端发送一个http请求到服务器端，其中包含Cookie头部
4. 服务器端发送一个http响应到客户端



5、在客户端的第二次请求中包含的Cookie头部中，提供给了服务器端可以用来唯一标识客户端身份的信息。这时，服务器端也就可以判断客户端是否启用了cookies。尽管，用户可能在和应用程序交互的过程中突然禁用cookies的使用，但是，这个情况基本是不太可能发生的，所以可以不加以考虑，这在实践中也被证明是对的

6、除了cookies,客户端还可以将发送给服务器的数据包含在请求的url中，这两种传递数据的方式，比起用cookies来传递数据更稳定，因为cookie可能被禁用，但是以GET以及POST方式传递数据时，不存在这种情况

### 什么是Cookie

1、Cookie意为"甜饼"，是由W3C组织提出，最早由Netscape社区发展的一种机制。目前Cookie已经成为标准，所有的主流浏览器如IE、Netscape、Firefox、Opera等都支持Cookie。

2、由于HTTP是一种无状态的协议，服务器单从网络连接上无从知道客户身份。怎么办呢？就给客户端们颁发一个通行证吧，每人一个，无论谁访问都必须携带自己通行证。这样服务器就能从通行证上确认客户身份了。这就是Cookie的工作原理。

3、Cookie实际上是一小段的文本信息。客户端请求服务器，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容

4、查看某个网站颁发的Cookie很简单。在浏览器地址栏输入javascript:alert (document.cookie)就可以了(需要有网才能查看)。JavaScript脚本会弹出一个对话框显示本网站颁发的所有Cookie的内容

5、注意：Cookie功能需要浏览器的支持。如果浏览器不支持Cookie(如大部分手机中的浏览器)或者把Cookie禁用了，Cookie功能就会失效。不同的浏览器采用不同的方式保存Cookie

### 记录用户访问次数

1、Java中把Cookie封装成了javax.servlet.http.Cookie类。每个Cookie都是该Cookie类的对象。服务器通过操作Cookie类对象对客户端Cookie进行操作。通过request.getCookie()获取客户端提交的所有Cookie(以Cookie[]数组形式返回)，通过response.addCookie(Cookiecookie)向客户端设置Cookie

2、Cookie对象使用key-value属性对的形式保存用户状态，一个Cookie对象保存一个属性对，一个request或者response同时使用多个Cookie。因为Cookie类位于包javax.servlet.http.\*下面，所以JSP中不需要import该类

### Cookie的不可跨域名性

1、很多网站都会使用Cookie。例如，Google会向客户端颁发Cookie，Baidu也会向客户端颁发Cookie。那浏览器访问Google会不会也携带上Baidu颁发的Cookie呢？或者Google能不能修改Baidu颁发的Cookie呢？

2、答案是否定的。Cookie具有不可跨域名性。根据Cookie规范，浏览器访问Google只会携带Google的Cookie，而不会携带Baidu的Cookie。Google也只能操作Google的Cookie，而不能操作Baidu的Cookie。

3、Cookie在客户端是由浏览器来管理的。浏览器能够保证Google只会操作Google的Cookie而不会操作Baidu的Cookie，从而保证用户的隐私安全。浏览器判断一个网站是否能操作另一个网站Cookie的依据是域名。Google与Baidu的域名不一样，因此Google不能操作Baidu的Cookie。

4、需要注意的是，虽然网站images.google.com与网站www.google.com同属于Google，但是域名不一样，二者同样不能互相操作彼此的Cookie。

5、注意：用户登录网站www.google.com之后会发现访问images.google.com时登录信息仍然有效，而普通的Cookie是做不到的。这是因为Google做了特殊处理。本章后面也会对Cookie做类似的处理。

### Unicode编码：保存中文

1、中文与英文字符不同，中文属于Unicode字符，在内存中占4个字符，而英文属于ASCII字符，内存中只占2个字节。Cookie中使用Unicode字符时需要对Unicode字符进行编码，否则会乱码

2、提示：Cookie中保存中文只能编码。一般使用UTF-8编码即可。不推荐使用GBK等中文编码，因为浏览器不一定支持，而且JavaScript也不支持GBK编码

### BASE64编码：保存二进制图片

1、Cookie不仅可以使用ASCII字符与Unicode字符，还可以使用二进制数据。例如在Cookie中使用数字证书，提供安全度。使用二进制数据时也需要进行编码。

2、该小节仅用于展示Cookie中可以存储二进制内容，并不实用。由于浏览器每次请求服务器都会携带Cookie，因此Cookie内容不宜过多，否则影响速度。Cookie的内容应该少而精

### 设置Cookie的所有属性

1、String name：

* 该Cookie的名称。Cookie一旦创建，名称便不可更改

2、Object value：

* 该Cookie的值
* 如果值为Unicode字符，需要为字符编码
* 如果值为二进制数据，则需要使用BASE64编码

3、int maxAge：

* 该Cookie失效的时间，单位秒
* 如果为正数，则该Cookie在maxAge秒之后失效
* 如果为负数，该Cookie为临时Cookie，关闭浏览器即失效，浏览器也不会以任何形式保存该Cookie
* 如果为0，表示删除该Cookie
* 默认为–1

4、boolean secure：

* 该Cookie是否仅被使用安全协议传输。安全协议
* 安全协议有HTTPS，SSL等，在网络上传输数据之前先将数据加密
* 默认为false

5、String path：

* 该Cookie的使用路径。如果设置为"/sessionWeb/"，则只有contextPath为"/sessionWeb"的程序可以访问该Cookie
* 如果设置为"/"，则本域名下contextPath都可以访问该Cookie
* 注意最后一个字符必须为"/"

6、String domain：

* 可以访问该Cookie的域名
* 如果设置为".google.com"，则所有以"google.com"结尾的域名都可以访问该Cookie
* 注意第一个字符必须为"."

7、String comment：

* 该Cookie的用处说明
* 浏览器显示Cookie信息的时候显示该说明

8、int version：

* 该Cookie使用的版本号
* 0表示遵循Netscape的Cookie规范
* 1表示遵循W3C的RFC 2109规范

### Cookie的有效期

1、Cookie的maxAge决定着Cookie的有效期，单位为秒(Second)。Cookie中通过getMaxAge()方法与setMaxAge(int maxAge)方法来读写maxAge属性。如果maxAge属性为正数，则表示该Cookie会在maxAge秒之后自动失效。浏览器会将maxAge为正数的Cookie持久化，即写到对应的Cookie文件中。无论客户关闭了浏览器还是电脑，只要还在maxAge秒之前，登录网站时该Cookie仍然有效。下面代码中的Cookie信息将永远有效

Cookie cookie = new Cookie("username","helloweenvsfei"); // 新建Cookie

cookie.setMaxAge(Integer.MAX\_VALUE); // 设置生命周期为MAX\_VALUE

response.addCookie(cookie); // 输出到客户端

2、如果maxAge为负数，则表示该Cookie仅在本浏览器窗口以及本窗口打开的子窗口内有效，关闭窗口后该Cookie即失效。maxAge为负数的Cookie，为临时性Cookie，不会被持久化，不会被写到Cookie文件中。Cookie信息保存在浏览器内存中，因此关闭浏览器该Cookie就消失了。Cookie默认的maxAge值为–1

3、如果maxAge为0，则表示删除该Cookie。Cookie机制没有提供删除Cookie的方法，因此通过设置该Cookie即时失效实现删除Cookie的效果。失效的Cookie会被浏览器从Cookie文件或者内存中删除

Cookie cookie = new Cookie("username","helloweenvsfei"); // 新建Cookie

cookie.setMaxAge(0); // 设置生命周期为0，不能为负数

response.addCookie(cookie); // 必须执行这一句

4、response对象提供的Cookie操作方法只有一个添加操作add(Cookie cookie)。要想修改Cookie只能使用一个同名的Cookie来覆盖原来的Cookie，达到修改的目的。删除时只需要把maxAge修改为0即可。

5、注意：从客户端读取Cookie时，包括maxAge在内的其他属性都是不可读的，也不会被提交。浏览器提交Cookie时只会提交name与value属性。maxAge属性只被浏览器用来判断Cookie是否过期

### Cookie的修改、删除

1、Cookie并不提供修改、删除操作。

* 如果要修改某个Cookie，只需要新建一个同名的Cookie，添加到response中覆盖原来的Cookie
* 如果要删除某个Cookie，只需要新建一个同名的Cookie，并将maxAge设置为0，并添加到response中覆盖原来的Cookie。注意是0而不是负数。负数代表其他的意义

2、注意：修改、删除Cookie时，新建的Cookie除value、maxAge之外的所有属性，例如name、path、domain等，都要与原Cookie完全一样。否则，浏览器将视为两个不同的Cookie不予覆盖，导致修改、删除失败

### Cookie的域名

1、Cookie是不可跨域名的。域名www.google.com颁发的Cookie不会被提交到域名www.baidu.com去。这是由Cookie的隐私安全机制决定的。隐私安全机制能够禁止网站非法获取其他网站的Cookie。

2、正常情况下，同一个一级域名下的两个二级域名如www.helloweenvsfei.com和images.helloweenvsfei.com也不能交互使用Cookie，因为二者的域名并不严格相同。如果想所有helloweenvsfei.com名下的二级域名都可以使用该Cookie，需要设置Cookie的domain参数，例如：

Cookie cookie = new Cookie("time","20080808"); // 新建Cookie

cookie.setDomain(".helloweenvsfei.com"); // 设置域名

cookie.setPath("/"); // 设置路径

cookie.setMaxAge(Integer.MAX\_VALUE); // 设置有效期

response.addCookie(cookie); // 输出到客户端

3、注意：domain参数必须以点"."开始。另外，name相同但domain不同的两个Cookie是两个不同的Cookie。如果想要两个域名完全不同的网站共有Cookie，可以生成两个Cookie，domain属性分别为两个域名，输出到客户端

### Cookie的路径

1、domain属性决定运行访问Cookie的域名，而path属性决定允许访问Cookie的路径(ContextPath)。例如，如果只允许/sessionWeb/下的程序使用Cookie，可以这么写

Cookie cookie = new Cookie("time","20080808"); // 新建Cookie

cookie.setPath("/session/"); // 设置路径

response.addCookie(cookie); // 输出到客户端

2、设置为"/"时允许所有路径使用Cookie。path属性需要使用符号"/"结尾。name相同但domain不同的两个Cookie也是两个不同的Cookie。

3、注意：页面只能获取它属于的Path的Cookie。例如/session/test/a.jsp不能获取到路径为/session/abc/的Cookie。使用时一定要注意

4、domain表示的是cookie所在的域，默认为请求的地址，如网址为www.test.com/test/test.aspx，那么domain默认为www.test.com。而跨域访问，如域A为t1.test.com，域B为t2.test.com，那么在域A生产一个令域A和域B都能访问的cookie就要将该cookie的domain设置为.test.com；如果要在域A生产一个令域A不能访问而域B能访问的cookie就要将该cookie的domain设置为t2.test.com。

5、path表示cookie所在的目录，默认为/，就是根目录。在同一个服务器上有目录如下：/test/，/test/cd/，/test/dd/，现设一个cookie1的path为/test/，cookie2的path为/test/cd/，那么test下的所有页面都可以访问到cookie1，而/test/和/test/dd/的子页面不能访问cookie2。这是因为cookie能让其path路径下的页面访问。

6、浏览器会将domain和path都相同的cookie保存在一个文件里，cookie间用\*隔开

### Cookie的安全属性

1、HTTP协议不仅是无状态的，而且是不安全的。使用HTTP协议的数据不经过任何加密就直接在网络上传播，有被截获的可能。使用HTTP协议传输很机密的内容是一种隐患。如果不希望Cookie在HTTP等非安全协议中传输，可以设置Cookie的secure属性为true。浏览器只会在HTTPS和SSL等安全协议中传输此类Cookie。下面的代码设置secure属性为true

Cookie cookie = new Cookie("time", "20080808"); // 新建Cookie

cookie.setSecure(true); // 设置安全属性

response.addCookie(cookie); // 输出到客户端

2、提示：secure属性并不能对Cookie内容加密，因而不能保证绝对的安全性。如果需要高安全性，需要在程序中对Cookie内容加密、解密，以防泄密

## Session

### 什么是Session

1、对Tomcat而言，Session是一块在服务器开辟的内存空间，其存储结构为ConcurrentHashMap

### Session的目的

1、Http协议是一种无状态协议，即每次服务端接收到客户端的请求时，都是一个全新的请求，服务器并不知道客户端的历史请求记录

2、Session的主要目的就是为了弥补Http的无状态特性。简单的说，就是服务器可以利用session存储客户端在同一个会话期间的一些操作记录

### 实现机制

1、首先看两个问题

1. 服务器如何判断客户端发送过来的请求是属于同一个会话?

* 回答：用Session id区分，Session id相同的即认为是同一个会话，在Tomcat中Session id用JSESSIONID表示

1. 服务器、客户端如何获取Session id？Session id在其之间是如何传输的呢？

* 回答

1. 服务器第一次接收到请求时，开辟了一块Session空间(创建了Session对象)，同时生成一个Session id，并通过响应头的Set-Cookie："JSESSIONID=XXXXXXX"命令，向客户端发送要求设置cookie的响应
2. 客户端收到响应后，在本机客户端设置了一个JSESSIONID=XXXXXXX的cookie信息，该cookie的过期时间为浏览器会话结束
3. 接下来客户端每次向同一个网站发送请求时，请求头都会带上该cookie信息(包含Session id)
4. 然后，服务器通过读取请求头中的Cookie信息，获取名称为JSESSIONID的值，得到此次请求的Session id

* ps：服务器只会在客户端第一次请求响应的时候，在响应头上添加Set-Cookie："JSESSIONID=XXXXXXX"信息，接下来在同一个会话的第二第三次响应头里，是不会添加Set-Cookie："JSESSIONID=XXXXXXX"信息的，而客户端是会在每次请求头的cookie中带上JSESSIONID信息

# Web Server和Web Service

1、概念上根本就是两个东西，像IIS/APACHE/ZEUS这类的软件都叫做Web Server，可以让用户通过用IE访问服务器的IP看到一个页面或者完成从页面上传递来的数据交互

2、WebService 仅仅是一个API，比如你想创建一个Web service它的作用是返回当前的天气情况，那么你可已建立一个ASP页面，它接受邮政编码作为查询字符串，然后返回一个由逗号隔开的字符串，包含了当前的气温和天气，要调用这个ASP页面，客户端需要发送下面的这个HTTP GET请求  
http://www.domain.com/weather.asp?zipcode=20171。返回的数据就应该是这样

21,晴

这个ASP页面就应该可以算作是Web service了，因为它基于HTTP GET请求，暴露出了一个可以通过Web调用的API

2、关于开发

1. 开发Web Server就是重写一个APACHE/IIS
2. 开发Web Service就是写一个类似于我刚才那个例子的程序
3. 有WebServer并且安装了解释语言比如ASP/PHP的就可以支持WebService