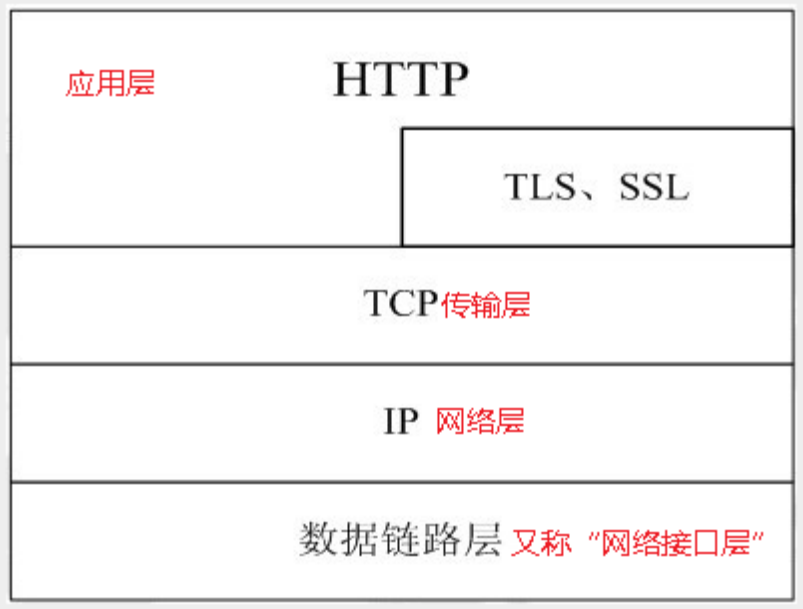
目录

一、概念

协议是指计算机通信网络中两台计算机之间进行通信所必须共同遵守的规定或规则，超文本传输协议(HTTP)是一种通信协议，它允许将超文本标记语言(HTML)文档从Web服务器传送到客户端的浏览器。
HTTP协议，即超文本传输协议(Hypertext transfer protocol)。是一种详细规定了浏览器和万维网(WWW = World Wide Web)服务器之间互相通信的规则，通过因特网传送万维网文档的数据传送协议。
HTTP协议是用于从WWW服务器传输超文本到本地浏览器的传送协议。它可以使浏览器更加高效，使网络传输减少。它不仅保证计算机正确快速地传输超文本文档，还确定传输文档中的哪一部分，以及哪部分内容首先显示(如文本先于图形)等。
HTTP是一个应用层协议，由请求和响应构成，是一个标准的客户端服务器模型。HTTP是一个无状态的协议。
在Internet中所有的传输都是通过TCP/IP进行的。HTTP协议作为TCP/IP模型中应用层的协议也不例外。HTTP协议通常承载于TCP协议之上，有时也承载于TLS或SSL协议层之上，这个时候，就成了我们常说的HTTPS。如下图所示：



HTTP默认的端口号为80，HTTPS的端口号为443。
浏览网页是HTTP的主要应用，但是这并不代表HTTP就只能应用于网页的浏览。HTTP是一种协议，只要通信的双方都遵守这个协议，HTTP就能有用武之地。比如咱们常用的QQ，迅雷这些软件，都会使用HTTP协议(还包括其他的协议)。

二、简史

它的发展是万维网协会（World Wide Web Consortium）和Internet工作小组IETF（Internet Engineering Task Force）合作的结果，（他们）最终发布了一系列的RFC，RFC 1945定义了HTTP/1.0版本。其中最著名的就是RFC 2616。RFC 2616定义了今天普遍使用的一个版本——HTTP 1.1。

三、特点

HTTP协议永远都是客户端发起请求，服务器回送响应。这样就限制了使用HTTP协议，无法实现在客户端没有发起请求的时候，服务器将消息推送给客户端。

HTTP协议的主要**特点**可概括如下：

1、支持客户/服务器模式。支持基本认证和安全认证。

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、HTTP 0.9和1.0使用非持续连接：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。HTTP 1.1使用持续连接：不必为每个web对象创建一个新的连接，一个连接可以传送多个对象，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。

无状态协议： 协议的状态是指下一次传输可以“记住”这次传输信息的能力。 http是不会为了下一次连接而维护这次连接所传输的信息,为了保证服务器内存。 比如客户获得一张网页之后关闭浏览器，然后再一次启动浏览器，再登陆该网站，但是服务器并不知道客户关闭了一次浏览器。 由于Web服务器要面对很多浏览器的并发访问，为了提高Web服务器对并发访问的处理能力，在设计HTTP协议时规定Web服务器发送HTTP应答报文和文档时，不保存发出请求的Web浏览器进程的任何状态信息。这有可能出现一个浏览器在短短几秒之内两次访问同一对象时，服务器进程不会因为已经给它发过应答报文而不接受第二期服务请求。由于Web服务器不保存发送请求的Web浏览器进程的任何信息，因此HTTP协议属于无状态协议（Stateless Protocol）。

HTTP协议是无状态的和Connection: keep-alive的区别： 无状态是指协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。从另一方面讲，打开一个服务器上的网页和你之前打开这个服务器上的网页之间没有任何联系。 HTTP是一个无状态的面向连接的协议，无状态不代表HTTP不能保持TCP连接，更不能代表HTTP使用的是UDP协议（无连接）。 从HTTP/1.1起，默认都开启了Keep-Alive，保持连接特性，简单地说，当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。 Keep-Alive不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如Apache）中设定这个时间。

四、工作流程

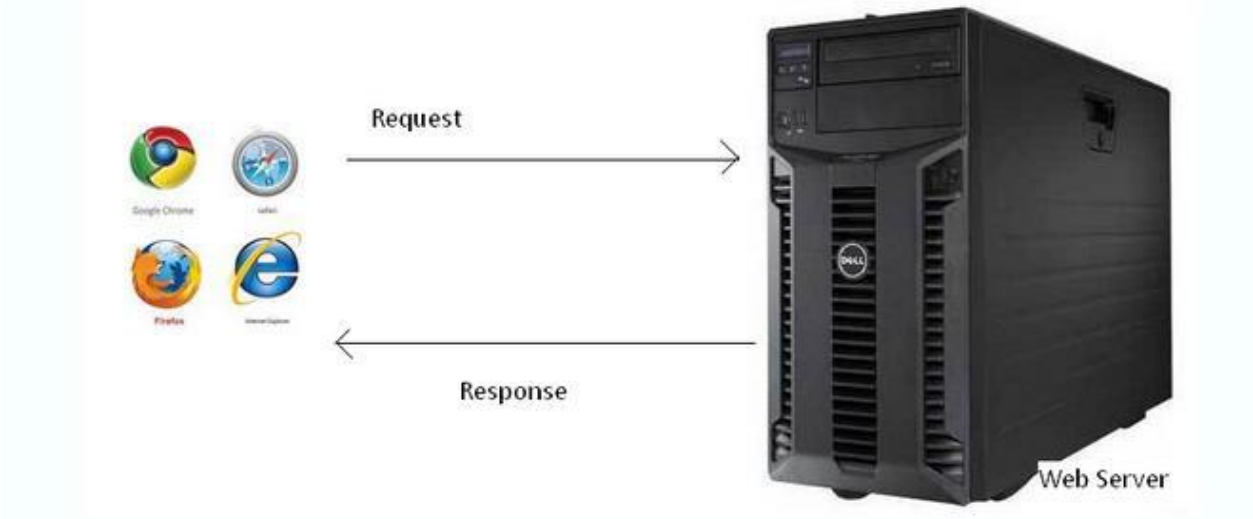
一次HTTP操作称为一个事务，其工作过程可分为四步：

1）首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP的工作开始。

2）建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是MIME信息包括请求修饰符、客户机信息和可能的内容。

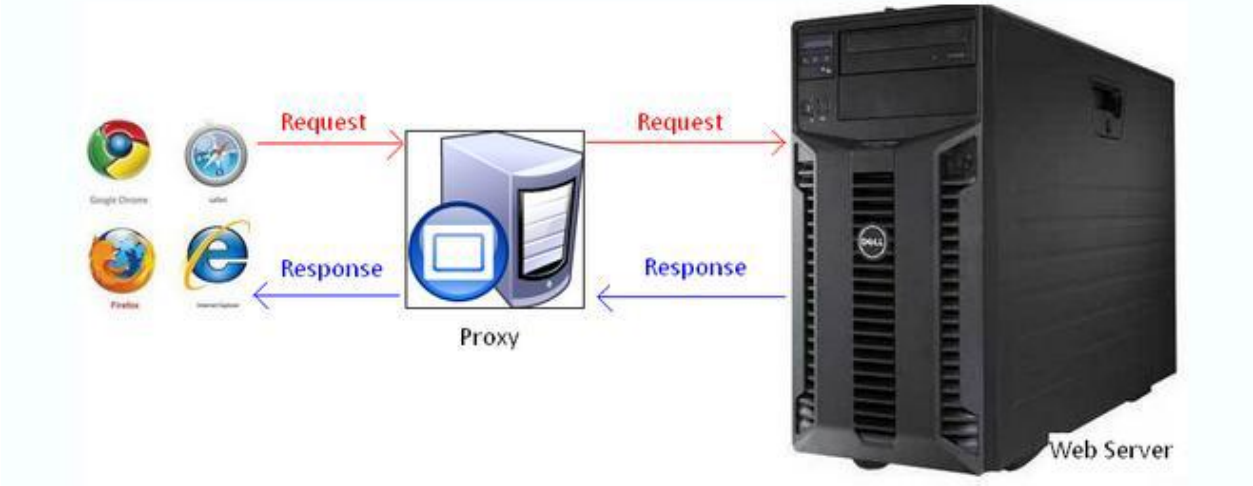
3）服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是MIME信息包括服务器信息、实体信息和可能的内容。

4）客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。 如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，有显示屏输出。对于用户来说，这些过程是由HTTP自己完成的，用户只要用鼠标点击，等待信息显示就可以了。

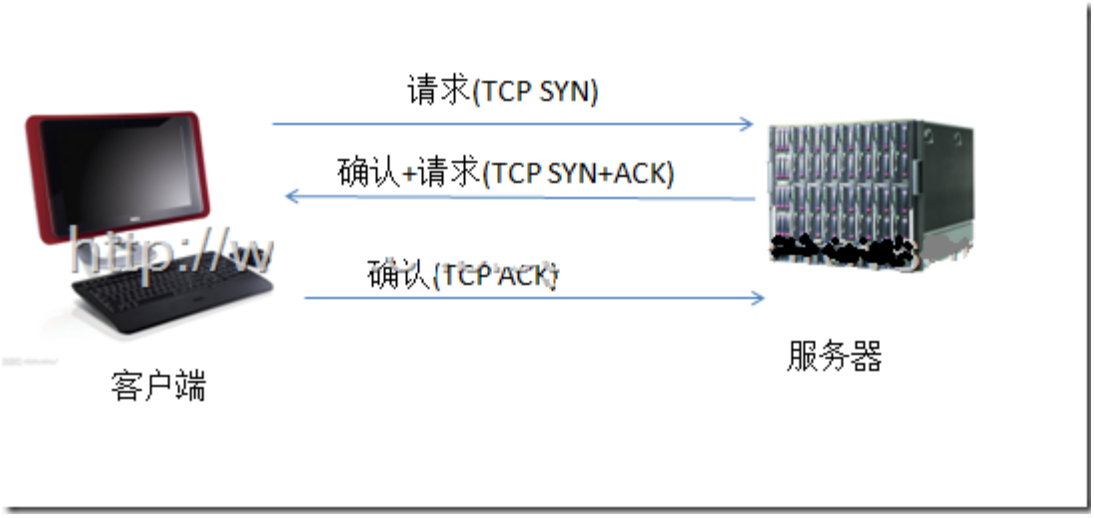


我们的Request 有可能是经过了代理服务器，最后才到达Web服务器的。

过程如下图所示

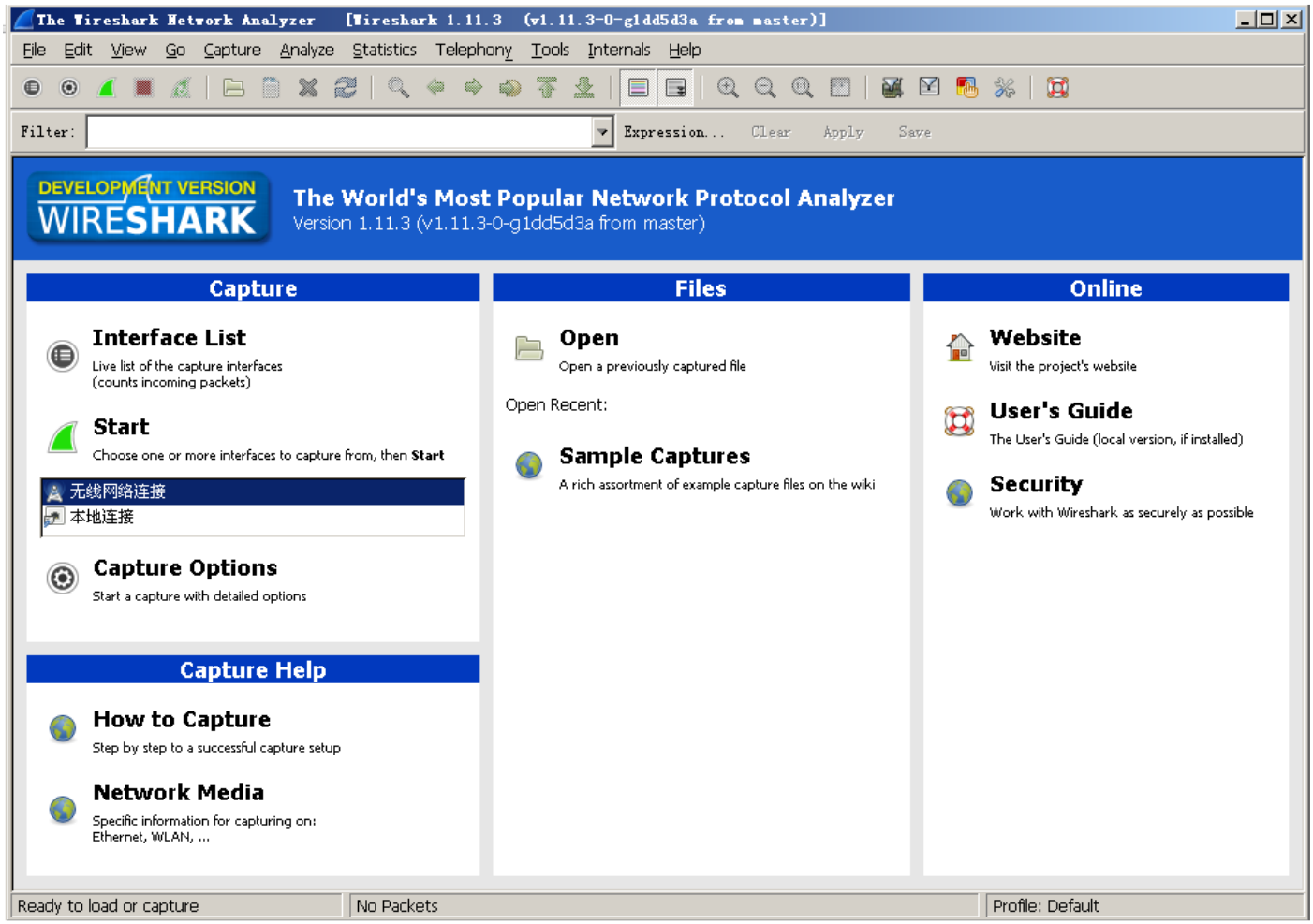


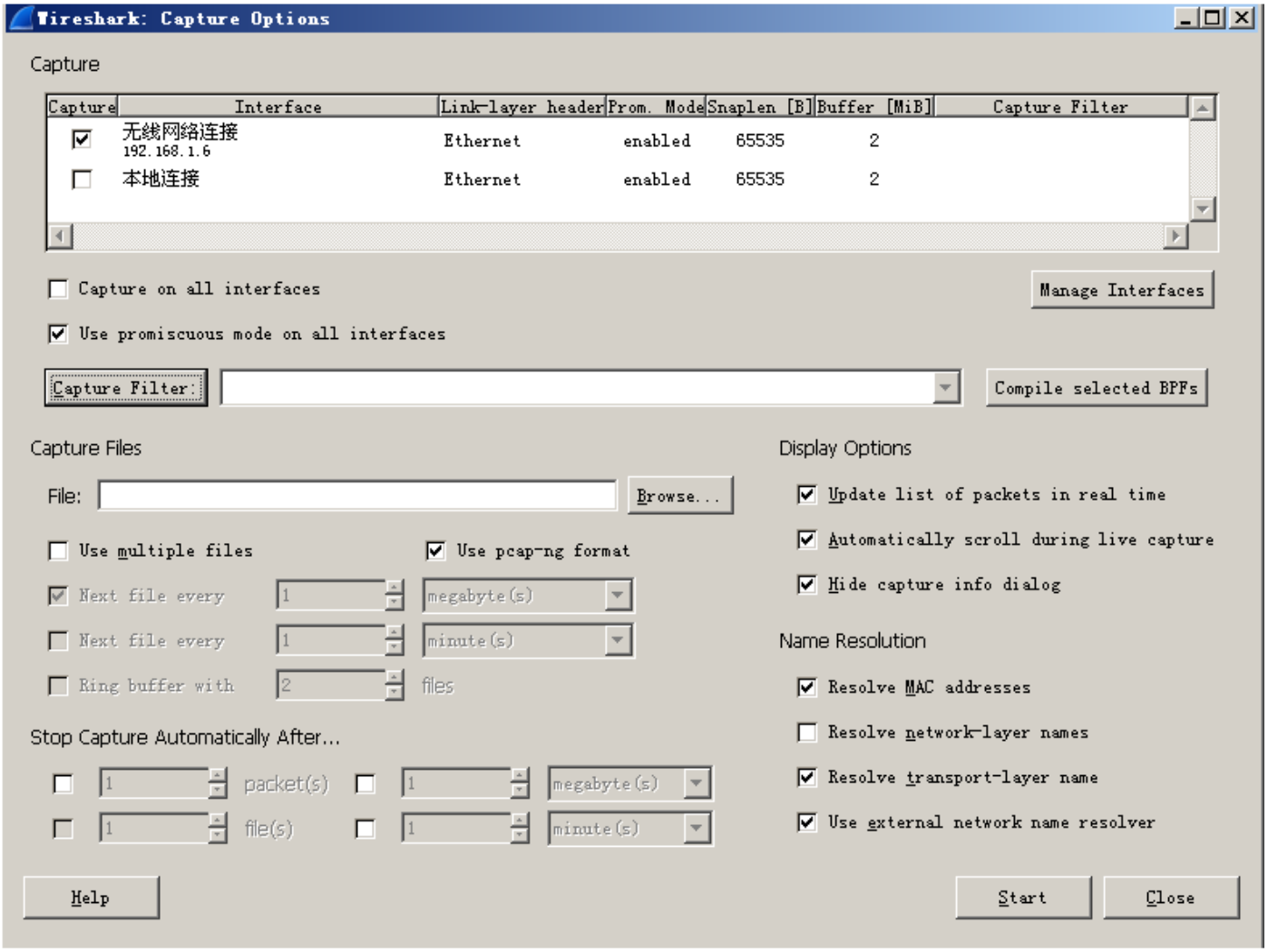
HTTP是基于传输层的TCP协议，而TCP是一个端到端的面向连接的协议。所谓的端到端可以理解为进程到进程之间的通信。所以HTTP在开始传输之前，首先需要建立TCP连接，而TCP连接的过程需要所谓的“三次握手”。下图所示TCP连接的三次握手。 在TCP三次握手之后，建立了TCP连接，此时HTTP就可以进行传输了。一个重要的概念是面向连接，既HTTP在传输完成之间并不断开TCP连接。在HTTP1.1中(通过Connection头设置)这是默认行为。



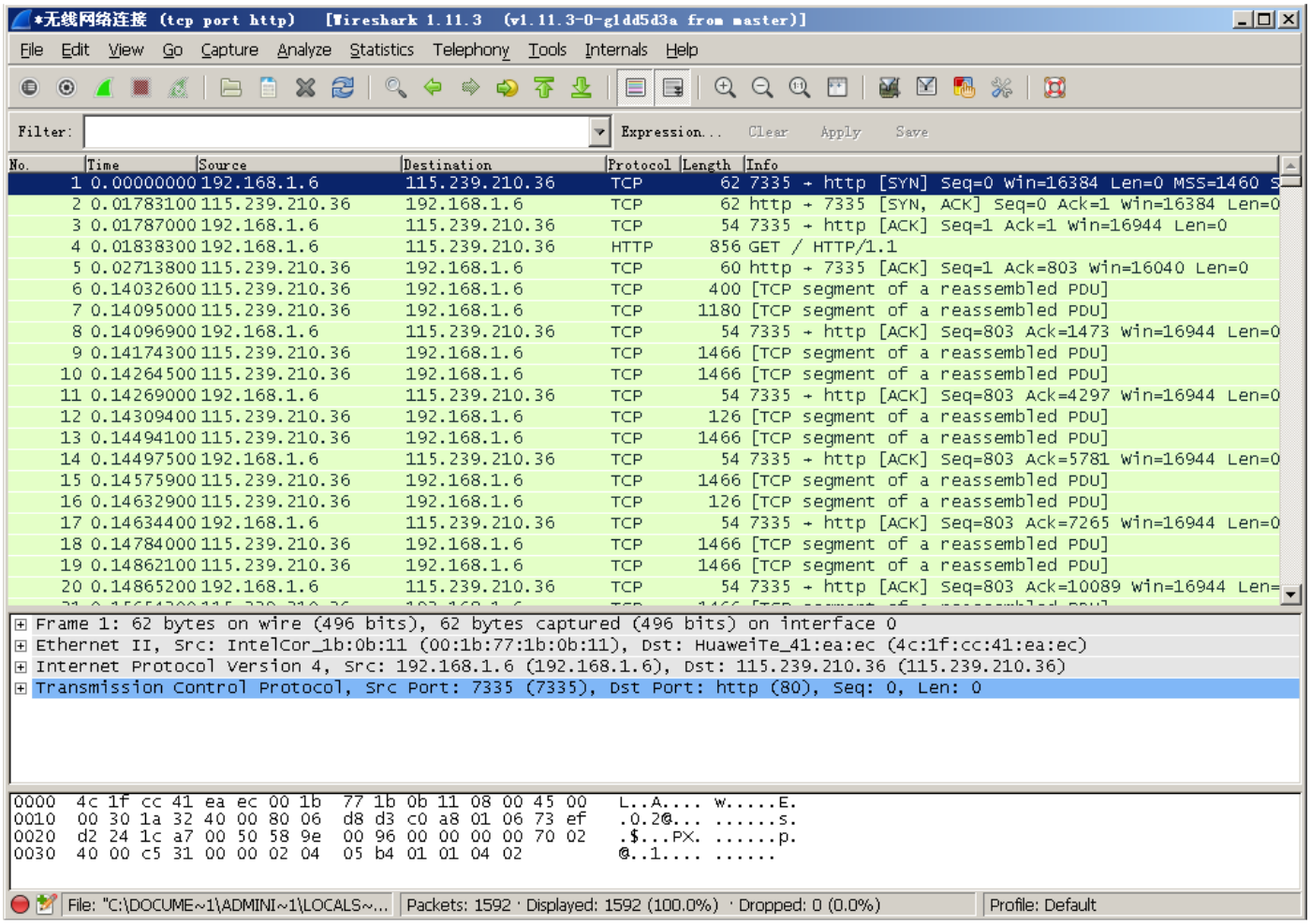
五、使用 Wireshark 抓 TCP、http 包

打开Wireshark，选择工具栏上的"Capture"->"Options"





点击"Capture Filter"，此处选择的是"HTTP TCP port（80）"，选择后点击上图的"Start"开始抓包。 然后在浏览器中打开http://image.baidu.com/，抓包结果如下图所示：



在上图中，可清晰的看到客户端浏览器（ip为192.168.1.6）与服务器（115.239.210.36）的交互过程：

1）No1：浏览器（192.168.1.6）向服务器（115.239.210.36）发出连接请求。此为TCP三次握手第一步，此时从图中可以看出，为SYN，seq:X （x=0）；

2）No2：服务器（115.239.210.36）回应了浏览器（192.168.1.6）的请求，并要求确认，此时为：SYN，ACK，此时seq：y（y为0），ACK：x+1（为1）。此为三次握手的第二步；

3）No3：浏览器（192.168.1.6）回应了服务器（115.239.210.36）的确认，连接成功。为：ACK，此时seq：x+1（为1），ACK：y+1（为1）。此为三次握手的第三步；

4）No4：浏览器（192.168.1.6）发出一个页面HTTP请求；

5）No5：服务器（115.239.210.36）确认；

6）No6：服务器（115.239.210.36）发送数据；

7）No8：客户端浏览器（192.168.1.6）确认；

8）No81：客户端（192.168.1.6）发出一个图片HTTP请求；

9）No202：服务器（115.239.210.36）发送状态响应码200 OK。

六、头域

每个头域由一个域名，冒号（:）和域值三部分组成。域名是大小写无关的，域值前可以添加任何数量的空格符，头域可以被扩展为多行，在每行开始处，使用至少一个空格或制表符。

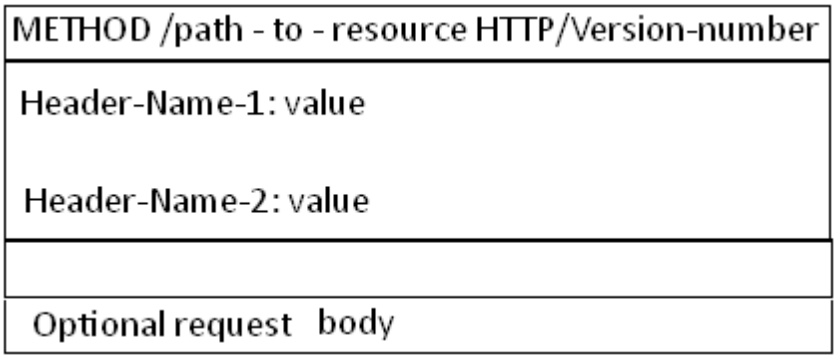
6.1、请求信息： 发出的请求信息格式如下：

●请求行，例如GET /images/logo.gif HTTP/1.1，表示从/images目录下请求logo.gif这个文件.

●（请求）头，例如Accept-Language: en

●空行

●可选的消息体 请求行和标题必须以<CR><LF>作为结尾（也就是，回车然后换行）。空行内必须只有<CR><LF>而无其他空格。在HTTP/1.1协议中，所有的请求头，除post外，都是可选的。



三个部分分别是：请求行、消息报头、请求正文。

请求方法 HTTP/1.1协议中共定义了八种方法（有时也叫"动作"）来表明Request-URI指定的资源的不同操作方式：
OPTIONS – 返回服务器针对特定资源所支持的HTTP请求方法。也可以利用向Web服务器发送'*'的请求来测试服务器的功能性。
HEAD- 向服务器索要与GET请求相一致的响应，只不过响应体将不会被返回。这一方法可以在不必传输整个响应内容的情况下，就可以获取包含在响应消息头中的元信息。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。
GET - 向特定的资源发出请求。注意：GET方法不应当被用于产生"副作用"的操作中，例如在web app.中。其中一个原因是GET可能会被网络蜘蛛等随意访问。
POST - 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
PUT - 向指定资源位置上传其最新内容。
DELETE - 请求服务器删除Request-URI所标识的资源。
TRACE- 回显服务器收到的请求，主要用于测试或诊断。
CONNECT - HTTP/1.1协议中预留给能够将连接改为管道方式的代理服务器。
PATCH - 用来将局部修改应用于某一资源，添加于规范RFC5789。
方法名称是区分大小写的。当某个请求所针对的资源不支持对应的请求方法的时候，服务器应当返回状态码405（Method Not Allowed）；当服务器不认识或者不支持对应的请求方法的时候，应当返回状态码501（Not Implemented）。 HTTP服务器至少应该实现GET和HEAD方法，其他方法都是可选的。此外，除了上述方法，特定的HTTP服务器还能够扩展自定义的方法。

RESTful用法：
http://127.0.0.1/user/1 GET 根据用户id查询用户数据
http://127.0.0.1/user POST 新增用户
http://127.0.0.1/user PUT 修改用户信息
http://127.0.0.1/user DELETE 删除用户信息

GET和POST的区别：
1、GET提交的数据会放在URL之后，以?分割URL和传输数据，参数之间以&相连，如EditPosts.aspx?name=test1&id=123456. POST方法是把提交的数据放在HTTP包的Body中。
2、GET提交的数据大小有限制，最多只能有1024字节（因为浏览器对URL的长度有限制），而POST方法提交的数据没有限制。
3、GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值。
4、GET方式提交数据，会带来安全问题，比如一个登录页面，通过GET方式提交数据时，用户名和密码将出现在URL上，如果页面可以被缓存或者其他人可以访问这台机器，就可以从历史记录获得该用户的账号和密码。

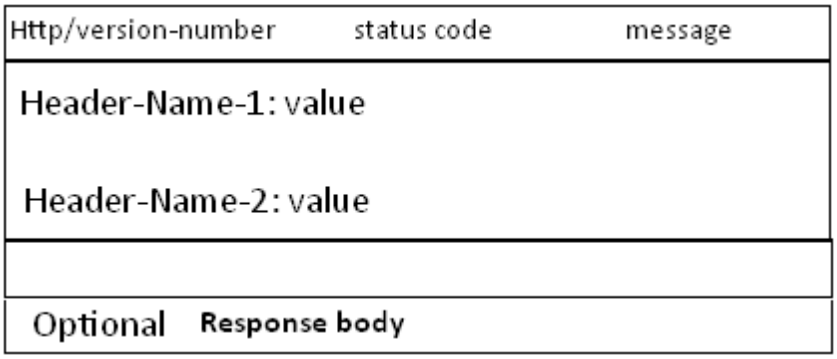
响应消息
客户端向服务器发送一个请求，服务器以一个状态行作为响应，响应的内容包括：消息协议的版本、成功或者错误编码、服务器信息、实体元信息以及必要的实体内容。根据响应类别的类别，服务器响应里可以含实体内容，但不是所有的响应都有

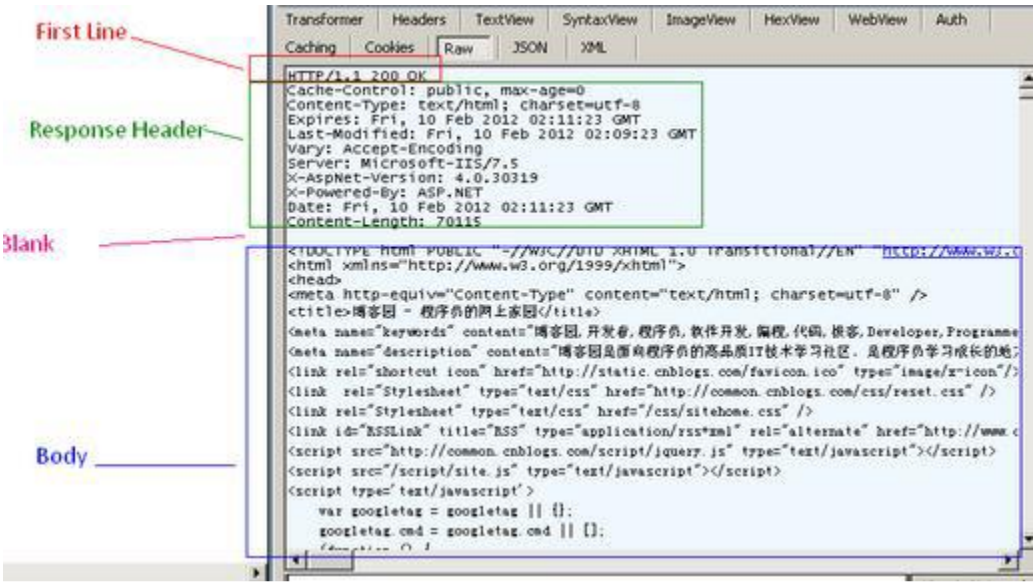
实体内容。 响应头第一行也称为状态行，格式如下（下图中红线标出的那行）： HTTP-Version 空格 Status-Code 空格 Reason-Phrase CRLF HTTP- Version表示HTTP版本，例如为HTTP/1.1。Status- Code是结果代码，用三个数字表示。Reason-Phrase是个简单的文本描述，解释Status-Code的具体原因。Status-Code用于机器自动识别，Reason-Phrase用于人工理解。Status-Code的第一个数字代表响应类别，可能取5个不同的值。后两个数字没有分类作用。Status-Code的第一个数字代表响应的类别，后续两位描述在该类响应下发生的具体状况，具体请参见：HTTP状态码 。

Response Headers view parsed

```
HTTP/1.1 200 OK
Date: Fri, 11 Jul 2014 03:49:33 GMT
Server: Apache/2.2.22 (Unix) mod_ssl/2.2.22 OpenSSL/1.0.1e-fips PHP/5.3.17
X-Powered-By: PHP/5.3.17
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0
Pragma: no-cache
Keep-Alive: timeout=5, max=776
Connection: Keep-Alive
Transfer-Encoding: chunked
Content-Type: text/html
```

响应消息的结构：



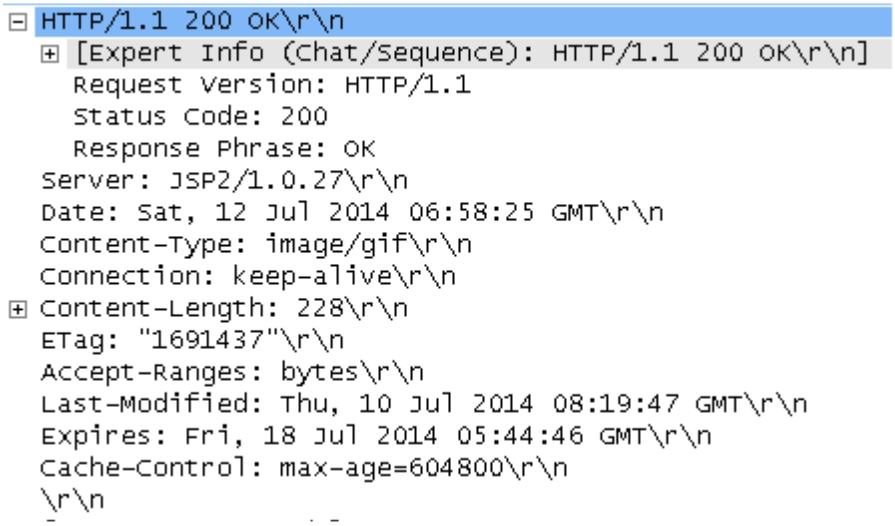


三个部分分别是：状态行、消息报头、响应正文。

无论你何时浏览一个网页，你的电脑都会通过一个使用HTTP协议的服务器来获取所请求的数据。在你请求的网页显示在浏览器之前，支配网页的网站服务器会返回一个包含有状态码的HTTP头文件。这个状态码提供了有关所请求网页的相关条件信息。如果一切正常，一个标准网页会收到一条诸如200的状态码。当然我们的目的不是去研究200响应码，而是去探讨那些代表出现错误信息的服务器头文件响应码，例如表示“未找到指定网页”的404码。

响应头域

服务器需要传递许多附加信息，这些信息不能全放在状态行里。因此，需要另行定义响应头域，用来描述这些附加信息。响应头域主要描述服务器的信息和Request-URI的信息。

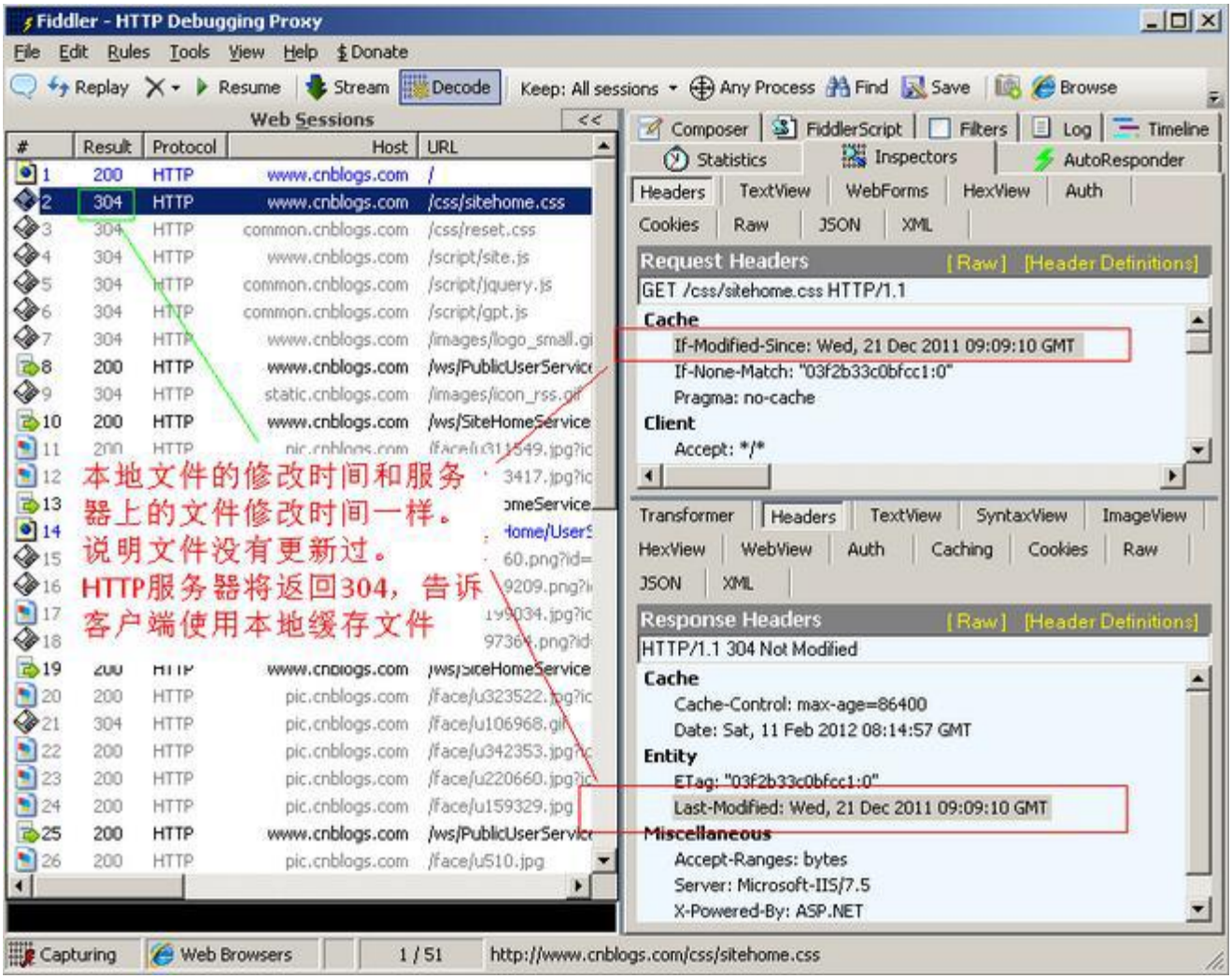


HTTP常见的请求头

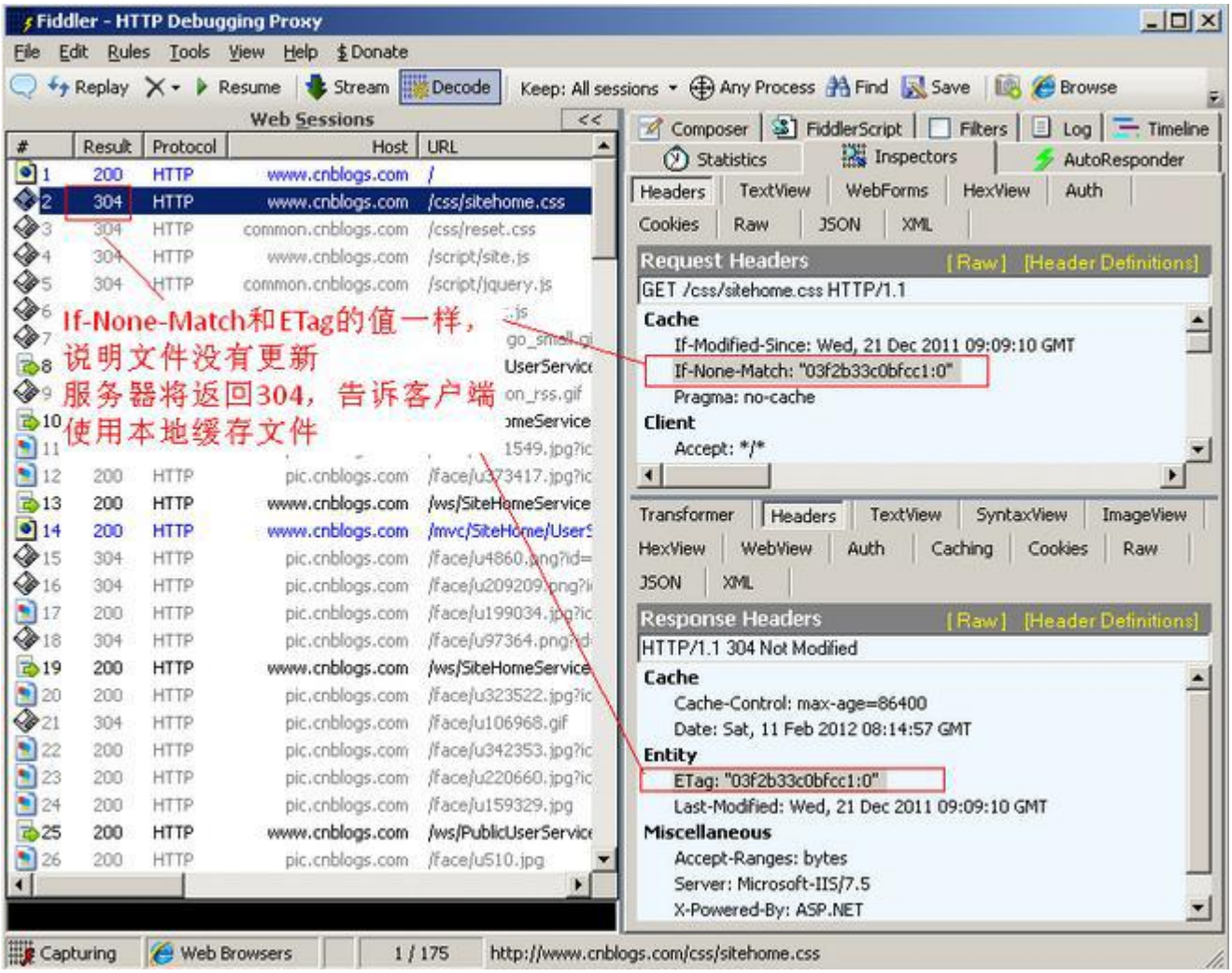
（在HTTP/1.1 协议中，所有的请求头，除Host外，都是可选的）

If-Modified-Since：把浏览器端缓存页面的最后修改时间发送到服务器去，服务器会把这个时间与服务器上实际文件的最后修改时间进行对比。如果时间一致，那么返回304，客户端就直接使用本地缓存文件。如果时间不一致，就会返回200和新的文件内容。客户端接到之后，会丢弃旧文件，把新文件缓存起来，并显示在浏览器中。

例如：If-Modified-Since: Thu, 09 Feb 2012 09:07:57 GMT



If-None-Match：If-None-Match和ETag一起工作，工作原理是在HTTP Response中添加ETag信息。 当用户再次请求该资源时，将在HTTP Request 中加入If-None-Match信息(ETag的值)。如果服务器验证资源的ETag没有改变（该资源没有更新），将返回一个304状态告诉客户端使用本地缓存文件。否则将返回200状态和新的资源和Etag. 使用这样的机制将提高网站的性能。例如: If-None-Match: "03f2b33c0bfcc1:0"。



Pragma：指定“no-cache”值表示服务器必须返回一个刷新后的文档，即使它是代理服务器而且已经有了页面的本地拷贝；在HTTP/1.1版本中，它和Cache-Control : no-cache作用一模一样。Pargma只有一个用法， 例如： Pragma: no-cache 注意: 在HTTP/1.0版本中，只实现了Pragema : no-cache，没有实现Cache-Control

Cache-Control：Cache-Control与Expires的作用一致，都是指明当前资源的有效期，控制浏览器是否直接从浏览器缓存取数据还是重新发请求到服务器取数据。只不过Cache-Control的选择更多，设置更细致，如果同时设置的话，其优先级高于Expires。缓存指令是单向的（响应中出现的缓存指令在请求中未必会出现），且是独立的（在请求消息或响应消息中设置Cache-Control并不会修改另一个消息处理过程中的缓存处理过程）。请求时的缓存指令包括no-cache、no-store、max-age、max-stale、min-fresh、only-if-cached，响应消息中的指令包括public、private、no-cache、no-store、no-transform、must-revalidate、proxy-revalidate、max-age、s-maxage。

Cache-Control:Public 可以被任何缓存所缓存

Cache-Control:Private 指示对于单个用户的整个或部分响应消息，不能被共享缓存处理。这允许服务器仅仅描述当用户的部分响应消息，此响应消息对于其他用户的请求无效。

Cache-Control:no-cache 禁止使用缓存内容进行响应。

```[html]
<span style="font-size:18px;"><META HTTP-EQUIV="Pragma" CONTENT="no-cache"></span>
```

Cache-Control:no-store 用于防止重要的信息被无意的发布。在请求消息中发送将使得请求和响应消息都不使用缓存，不会被存下来。

Cache-Control:max-age 指示客户机可以接收生存期不大于指定时间（以秒为单位）的响应。 max-age>0 时 直接从游览器缓存中 提取; max-age<=0 时 向server 发送http 请求确认 ,该资源是否有修改.有的话 返回200 ,无的话 返回304

Cache-Control:min-fresh 指示客户机可以接收响应时间小于当前时间加上指定时间的响应。

Cache-Control:max-stale 指示客户机可以接收超出超时期间的响应消息。如果指定max-stale消息的值，那么客户机可以接收超出超时期指定值之内的响应消息。

Cache-Control:must-revalidate 该选项会告诉缓存，在没有跟原始服务器进行再验证的情况下，不能提供这个对象的陈旧副本。

Accept：浏览器端可以接受的MIME类型。例如：Accept: text/html 代表浏览器可以接受服务器回发的类型为 text/html 也就是我们常说的html文档，如果服务器无法返回text/html类型的数据，服务器应该返回一个406错误(non acceptable)。通配符 * 代表任意类型，例如 Accept: */* 代表浏览器可以处理所有类型，(一般浏览器发给服务器都是发这个)。

Accept-Encoding：浏览器申明自己可接收的编码方法，通常指定压缩方法，是否支持压缩，支持什么压缩方法（gzip，deflate）;Servlet能够向支持gzip的浏览器返回经gzip编码的HTML页面。许多情形下这可以减少5到10倍的下载时间。例如：Accept-Encoding: gzip, deflate。如果请求消息中没有设置这个域，服务器假定客户端对各种内容编码都可以接受。

Accept-Language：浏览器申明自己接收的语言。语言跟字符集的区别：中文是语言，中文有多种字符集，比如big5，gb2312，gbk等等；例如：Accept-Language: en-us。如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。

Accept-Charset：浏览器可接受的字符集。如果在请求消息中没有设置这个域，缺省表示任何字符集都可以接受。

User-Agent：告诉HTTP服务器，客户端使用的操作系统和浏览器的名称和版本。 例如： User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; CIBA; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; .NET4.0C; InfoPath.2; .NET4.0E)。

Content-Type：服务端通常是根据请求头（headers）中的 Content-Type 字段来获知请求中的消息主体是用何种方式编码，再对主体进行解析。例如：Content-Type: application/x-www-form-urlencoded。

Referer：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。提供了Request的上下文信息的服务器，告诉服务器我是从哪个链接过来的，比如从我主页上链接到一个朋友那里，他的服务器就能够从HTTP Referer中统计出每天有多少用户点击我主页上的链接访问他的网站。 例如: Referer:http://translate.google.cn/?hl=zh-cn&tab=wT

Connection： 例如：Connection: keep-alive 当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。HTTP 1.1默认进行持久连接。利用持久连接的优点，当页面包含多个元素时（例如Applet，图片），显著地减少下载所需要的时间。要实现这一点，Servlet需要在应答中发送一个Content-Length头，最简单的实现方法是：先把内容写入ByteArrayOutputStream，然后在正式写出内容之前计算它的大小。Connection: close 代表一个Request完成后，客户端和服务器之间用于传输HTTP数据的TCP连接会关闭，当客户端再次发送Request，需要重新建立TCP连接。

Host：（发送请求时，该头域是必需的）主要用于指定被请求资源的Internet主机和端口号，它通常从HTTP URL中提取出来的。HTTP/1.1请求必须包含主机头域，否则系统会以400状态码返回。

Cookie：最重要的请求头之一，将cookie的值发送给HTTP服务器。

Content-Length：表示请求消息正文的长度。例如：Content-Length: 38。

Authorization：授权信息，通常出现在对服务器发送的WWW-Authenticate头的应答中。主要用于证明客户端有权查看某个资源。当浏览器访问一个页面时，如果收到服务器的响应代码为401（未授权），可以发送一个包含Authorization请求报头域的请求，要求服务器对其进行验证。

UA-Pixels，UA-Color，UA-OS，UA-CPU：由某些版本的IE浏览器所发送的非标准的请求头，表示屏幕大小、颜色深度、操作系统和CPU类型。

From：请求发送者的email地址，由一些特殊的Web客户程序使用，浏览器不会用到它。

Range：可以请求实体的一个或者多个子范围。例如， 表示头500个字节：bytes=0-499 表示第二个500字节：bytes=500-999 表示最后500个字节：bytes=-500 表示500字节以后的范围：bytes=500- 第一个和最后一个字节：bytes=0-0,-1 同时指定几个范围：bytes=500-600,601-999 但是服务器可以忽略此请求头，如果无条件GET包含Range请求头，响应会以状态码206（PartialContent）返回而不是以200（OK）。

HTTP常见的响应头

Allow：服务器支持哪些请求方法（如GET、POST等）。

Date：表示消息发送的时间，时间的描述格式由rfc822定义。例如，Date:Mon,31Dec200104:25:57GMT。Date描述的时间表示世界标准时，换算成本地时间，需要知道用户所在的时区。你可以用setDateHeader来设置这个头以避免转换时间格式的麻烦

Expires：指明应该在什么时候认为文档已经过期，从而不再缓存它，重新从服务器获取，会更新缓存。过期之前使用本地缓存。HTTP1.1的客户端和缓存会将非法的日期格式（包括0）看作已经过期。eg：为了让浏览器不要缓存页面，我们也可以将Expires实体报头域，设置为0。 例如: Expires: Tue, 08 Feb 2022 11:35:14 GMT。不过Expires 是HTTP 1.0的东西，现在默认浏览器均默认使用HTTP 1.1，所以它的作用基本忽略。Expires 的一个缺点就是，返回的到期时间是服务器端的时间，这样存在一个问题，如果客户端的时间与服务器的时间相差很大（比如时钟不同步，或者跨时区），那么误差就很大，所以在HTTP 1.1版开始，使用Cache-Control: max-age=秒替代。

P3P：用于跨域设置Cookie，这样可以解决iframe跨域访问cookie的问题 例如: P3P: CP=CURa ADMa DEVa PSAo PSDo OUR BUS UNI PUR INT DEM STA PRE COM NAV OTC NOI DSP COR

Set-Cookie：非常重要的header，用于把cookie发送到客户端浏览器，每一个写入cookie都会生成一个Set-Cookie。 例如: Set-Cookie: sc=4c31523a; path=/; domain=.acookie.taobao.com

ETag：和If-None-Match 配合使用。 web服务器响应请求时，告诉浏览器当前资源在服务器的唯一标识（生成规则由服务器决定）。Apache中，ETag的值，默认是对文件的索引节（INode），大小（Size）和最后修改时间（MTime）进行Hash后得到的。

Last-Modified：用于指示资源的最后修改日期和时间。Last-Modified也可用setDateHeader方法来设置。

既然有了Last-Modified为何要有Etag？因为：

1. `Last-Modified` 标注的最后修改只能精确到秒级，如果某些文件在 `1` 秒钟以内，被修改多次的话，它将不能准确标注文件的修改时间
2. 如果某些文件会被定期生成，当有时内容并没有任何变化，但 `Last-Modified` 却改变了，导致文件没法使用缓存
3. 有可能存在服务器没有准确获取文件修改时间，或者与代理服务器时间不一致等情形

Content-Type：WEB服务器告诉浏览器自己响应的对象的类型和字符集。Servlet默认为text/plain，但通常需要显式地指定为text/html。由于经常要设置Content-Type，因此HttpServletResponse提供了一个专用的方法setContentType。可在web.xml文件中配置扩展名和MIME类型的对应关系。 例如:Content-Type: text/html;charset=utf-8 Content-Type:text/html;charset=GB2312 Content-Type: image/jpeg

媒体类型的格式为：大类/小类，比如text/html。 IANA(The Internet Assigned Numbers Authority，互联网数字分配机构)定义了8个大类的媒体类型，分别是: application— (比如: application/vnd.ms-excel.) audio (比如: audio/mpeg.) image (比如: image/png.) message (比如,:message/http.) model(比如:model/vrml.) multipart (比如:multipart/form-data.) text(比如:text/html.) video(比如:video/quicktime.)

Content-Range：用于指定整个实体中的一部分的插入位置，他也指示了整个实体的长度。在服务器向客户返回一个部分响应，它必须描述响应覆盖的范围和整个实体长度。一般格式：Content-Range:bytes-unitSPfirst-byte-pos-last-byte-pos/entity-length。 例如，传送头500个字节次字段的形式：Content-Range:bytes0-499/1234如果一个http消息包含此节（例如，对范围请求的响 应或对一系列范围的重叠请求），Content-Range表示传送的范围。

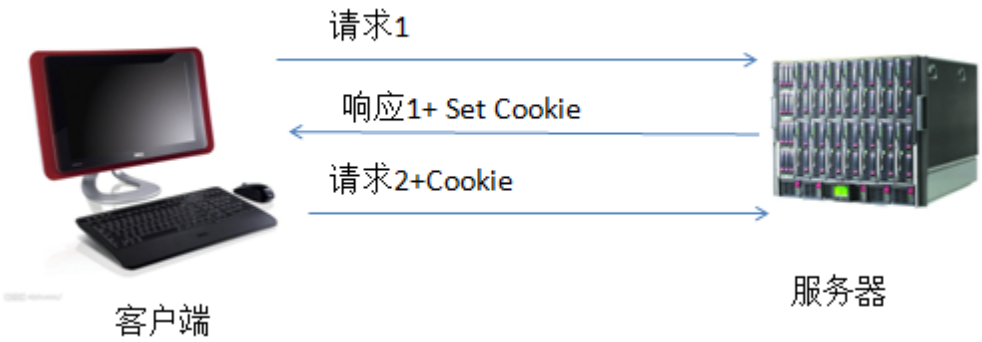
Content-Length：指明实体正文的长度，以字节方式存储的十进制数字来表示。在数据下行的过程中，Content-Length的方式要预先在服务器中缓存所有数据，然后所有数据再一股脑儿地发给客户端。只有当浏览器使用持久HTTP连接时才需要这个数据。如果你想要利用持久连接的优势，可以把输出文档写入ByteArrayOutputStram，完成后查看其大小，然后把该值放入Content-Length头，最后通过byteArrayStream.writeTo(response.getOutputStream()发送内容。 例如: Content-Length: 19847

Content-Encoding：WEB服务器表明自己使用了什么压缩方法（gzip，deflate）压缩响应中的对象。只有在解码之后才可以得到Content-Type头指定的内容类型。利用gzip压缩文档能够显著地减少HTML文档的下载时间。Java的GZIPOutputStream可以很方便地进行gzip压缩，但只有Unix上的Netscape和Windows上的IE 4、IE 5才支持它。因此，Servlet应该通过查看Accept-Encoding头（即request.getHeader("Accept-Encoding")）检查浏览器是否支持gzip，为支持gzip的浏览器返回经gzip压缩的HTML页面，为其他浏览器返回普通页面。 例如：Content-Encoding：gzip

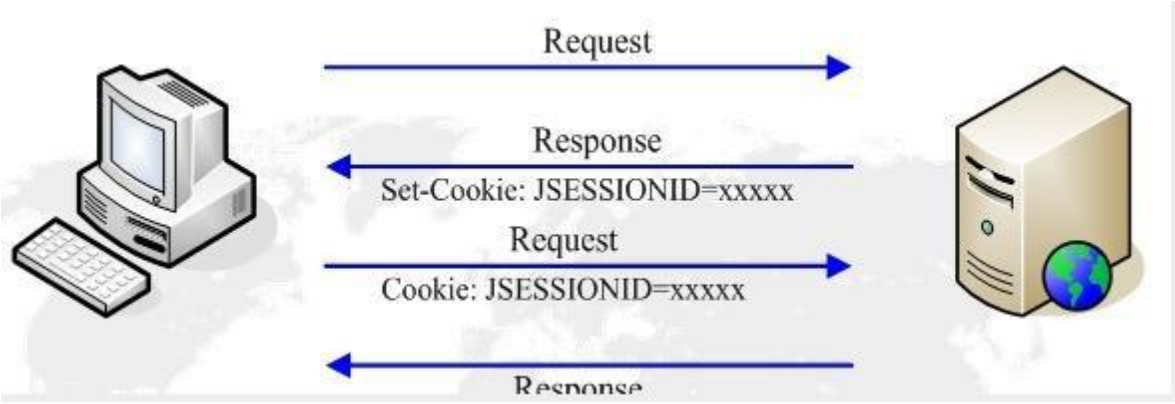
Content-Language：WEB服务器告诉浏览器自己响应的对象所用的自然语言。例如： Content-Language:da。没有设置该域则认为实体内容将提供给所有的语言阅读。
Server：指明HTTP服务器用来处理请求的软件信息。例如：Server: Microsoft-IIS/7.5、Server：Apache-Coyote/1.1。此域能包含多个产品标识和注释，产品标识一般按照重要性排序。
X-AspNet-Version：如果网站是用ASP.NET开发的，这个header用来表示ASP.NET的版本。 例如: X-AspNet-Version: 4.0.30319
X-Powered-By：表示网站是用什么技术开发的。 例如： X-Powered-By: ASP.NET
Connection： 例如：Connection: keep-alive 当一个网页打开完成后，客户端和服务器之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。 Connection: close 代表一个Request完成后，客户端和服务器之间用于传输HTTP数据的TCP连接会关闭，当客户端再次发送Request，需要重新建立TCP连接。
Location：用于重定向一个新的位置，包含新的URL地址。表示客户应当到哪里去提取文档。Location通常不是直接设置的，而是通过HttpServletResponse的sendRedirect方法，该方法同时设置状态代码为302。Location响应报头域常用在更换域名的时候。
Refresh：表示浏览器应该在多少时间之后刷新文档，以秒计。除了刷新当前文档之外，你还可以通过setHeader("Refresh", "5; URL=http://host/path")让浏览器读取指定的页面。注意这种功能通常是通过设置HTML页面HEAD区的<META HTTP-EQUIV="Refresh" CONTENT="5;URL=http://host/path">实现，这是因为，自动刷新或重定向对于那些不能使用CGI或Servlet的HTML编写者十分重要。但是，对于Servlet来说，直接设置Refresh头更加方便。注意Refresh的意义是“N秒之后刷新本页面或访问指定页面”，而不是“每隔N秒刷新本页面或访问指定页面”。因此，连续刷新要求每次都发送一个Refresh头，而发送204状态代码则可以阻止浏览器继续刷新，不管是使用Refresh头还是<META HTTP-EQUIV="Refresh" ...>。注意Refresh头不属于HTTP 1.1正式规范的一部分，而是一个扩展，但Netscape和IE都支持它。
WWW-Authenticate：该响应报头域必须被包含在401（未授权的）响应消息中，客户端收到401响应消息时候，并发送Authorization报头域请求服务器对其进行验证时，服务端响应报头就包含该报头域。 eg：WWW-Authenticate:Basic realm="Basic Auth Test!" //可以看出服务器对请求资源采用的是基本验证机制。

七、解决 HTTP 无状态的问题

通过Cookies保存状态信息
通过Cookies，服务器就可以清楚的知道请求2和请求1来自同一个客户端。



通过Session保存状态信息
Session机制是一种服务器端的机制，服务器使用一种类似于散列表的结构（也可能就是使用散列表）来保存信息。 当程序需要为某个客户端的请求创建一个session的时候，服务器首先检查这个客户端的请求里是否已包含了一个session标识 - 称为 session id，如果已包含一个session id则说明以前已经为此客户端创建过session，服务器就按照session id把这个 session检索出来使用（如果检索不到，可能会新建一个），如果客户端请求不包含session id，则为此客户端创建一个session并且生成一个与此session相关联的session id，session id的值应该是一个既不会重复，又不容易被找到规律以仿造的字符串，这个session id将被在本次响应中返回给客户端保存。
Session的实现方式：
1、 使用Cookie来实现 服务器给每个Session分配一个唯一的JSESSIONID，并通过Cookie发送给客户端。 当客户端发起新的请求的时候，将在Cookie头中携带这个JSESSIONID。这样服务器能够找到这个客户端对应的Session。



2、使用URL回写来实现 URL回写是指服务器在发送给浏览器页面的所有链接中都携带JSESSIONID的参数，这样客户端点击任何一个链接都会把JSESSIONID带会服务器。如果直接在浏览器输入服务端资源的url来请求该资源，那么Session是匹配不到的。Tomcat对Session的实现，是一开始同时使用Cookie和URL回写机制，如果发现客户端支持Cookie，就继续使用Cookie，停止使用URL回写。如果发现Cookie被禁用，就一直使用URL回写。jsp开发处理到Session的时候，对页面中的链接记得使用response.encodeURL() 。

Cookie和Session有以下明显的不同点：

1）Cookie将状态保存在客户端，Session将状态保存在服务器端；

2）Cookies是服务器在本地机器上存储的小段文本并随每一个请求发送至同一个服务器。Cookie最早在RFC2109中实现，后续RFC2965做了增强。网络服务器用HTTP头向客户端发送cookies，在客户终端，浏览器解析这些cookies并将它们保存为一个本地文件，它会自动将同一服务器的任何请求附上这些cookies。Session并没有在HTTP的协议中定义；

3）Session是针对每一个用户的，变量的值保存在服务器上，用一个sessionID来区分是哪个用户session变量,这个值是通过用户的浏览器在访问的时候返回给服务器，当客户禁用cookie时，这个值也可能设置为由get来返回给服务器；

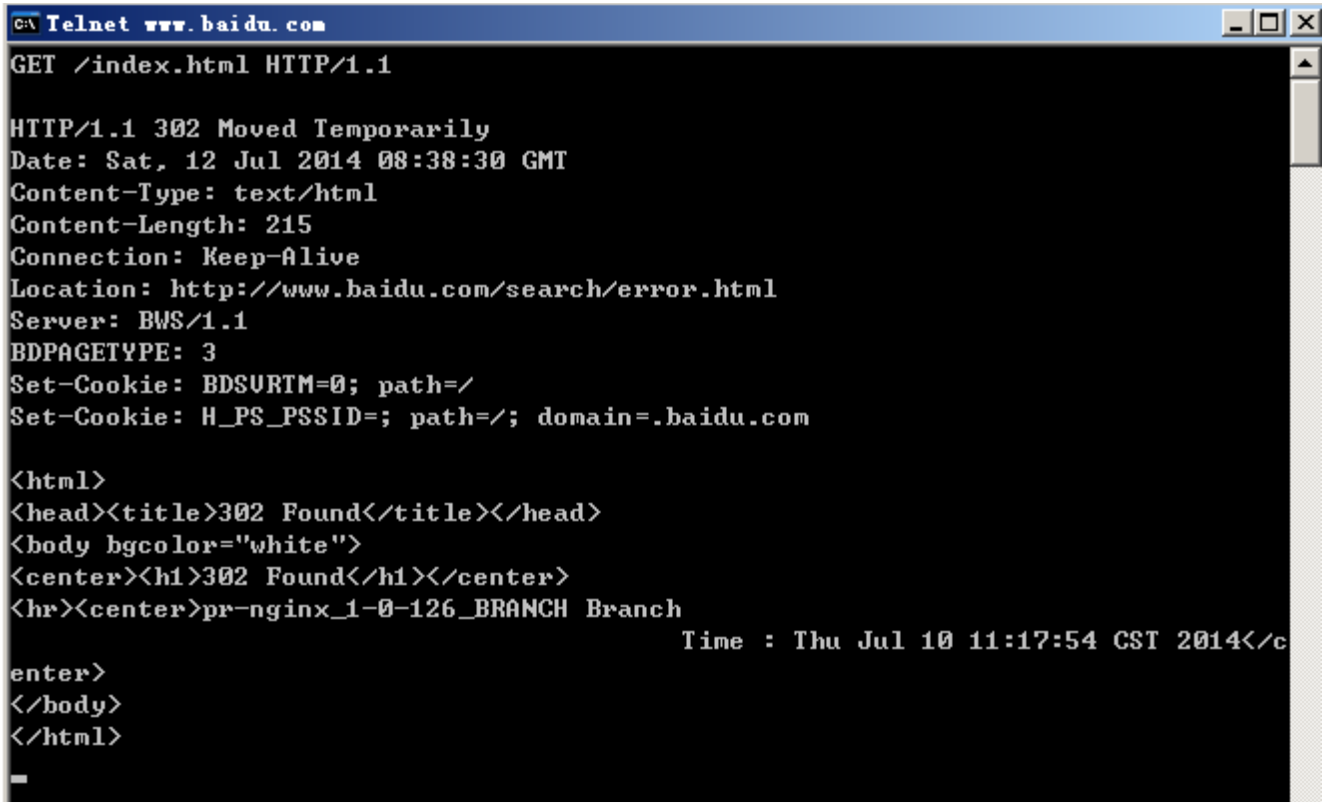
4）就安全性来说：当你访问一个使用session 的站点，同时在自己机子上建立一个cookie，建议在服务器端的SESSION机制更安全些。因为它不会任意读取客户存储的信息。 (XSS, CSRF)

通过表单变量保持状态 除了Cookies之外，还可以使用表单变量来保持状态，比如Asp.net就通过一个叫ViewState的Input=“hidden”的框来保持状态,比如: <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE" value="/wEPDwUKMjA0OTM4MTAwNGRkXUfhlDv1Cs7/qhBlyZROCzlvf5U=" /> 这个原理和Cookies大同小异，只是每次请求和响应所附带的信息变成了表单变量。

通过QueryString保持状态 QueryString通过将信息保存在所请求地址的末尾来向服务器传送信息，通常和表单结合使用，一个典型的QueryString比如:www.xxx.com/xxx.do?var1=value&var2=value2

八、使用 telnet 进行 http 测试

在Windows下，可使用命令窗口进行http简单测试。输入cmd进入命令窗口，在命令行键入如下命令后按回车： telnet www.baidu.com 80 而后在窗口中按下"Ctrl+]"后按回车可让返回结果回显。 接着开始发请求消息，例如发送如下请求消息请求baidu的首页消息，使用的HTTP协议为HTTP/1.1： GET /index.html HTTP/1.1 注意：copy如上的消息到命令窗口后需要按两个回车换行才能得到响应的消息，第一个回车换行是在命令后键入回车换行，是HTTP协议要求的。第二个是确认输入，发送请求。



可看到，当采用HTTP/1.1时，连接不是在请求结束后就断开的。若采用HTTP1.0，在命令窗口键入： GET /index.html HTTP/1.0 此时可以看到请求结束之后马上断开。 读者还可以尝试在使用GET或POST等时，带上头域信息，例如键入如下信息：

GET /index.html HTTP/1.1

connection: close

Host: www.baidu.com

九、URL 详解

URL(Uniform Resource Locator) 地址用于描述一个网络上的资源， 基本**格式**如下

schema://host[:port#]/path/.../[;url-params][?query-string][#anchor]

scheme 指定低层使用的协议(例如：http, https, ftp)

host HTTP服务器的IP地址或者域名

port# HTTP服务器的默认端口是80，这种情况下端口号可以省略。如果使用了别的端口，必须指明，例如 http://www.cnblogs.com:8080/

path 访问资源的路径

url-params query-string 发送给http服务器的数据

anchor- 锚

URL 的一个例子： http://www.mywebsite.com/sj/test;id=8079?name=sviergn&x=true#stuff Schema: http host: www.mywebsite.com path: /sj/test URL params: id=8079 Query String: name=sviergn&x=true Anchor: stuff

十、缓存的实现原理

WEB缓存(cache)位于Web服务器和客户端之间。 缓存会根据请求保存输出内容的副本，例如html页面，图片，文件，当下一个请求来到的时候：如果是相同的URL，缓存直接使用副本响应访问请求，而不是向源服务器再次发送请求。 HTTP协议定义了相关的消息头来使WEB缓存尽可能好的工作。

缓存的优点 减少相应延迟：因为请求从缓存服务器（离客户端更近）而不是源服务器被相应，这个过程耗时更少，让web服务器看上去相应更快。 减少网络带宽消耗：当副本被重用时会减低客户端的带宽消耗；客户可以节省带宽费用，控制带宽的需求的增长并更易于管理。

客户端缓存生效的常见流程

服务器收到请求时，会在200OK中回送该资源的Last-Modified和ETag头，客户端将该资源保存在cache中，并记录这两个属性。当客户端需要发送相同的请求时，会在请求中携带If-Modified-Since和If-None-Match两个头。两个头的值分别是响应中Last-Modified和ETag头的值。服务器通过这两个头判断本地资源未发生变化，客户端不需要重新下载，返回304响应。

Web缓存机制

HTTP/1.1中缓存的目的是为了在很多情况下减少发送请求，同时在许多情况下可以不需要发送完整响应。前者减少了网络回路的数量；HTTP利用一个"过期（expiration）"机制来为此目的。后者减少了网络应用的带宽；HTTP用"验证（validation）"机制来为此目的。

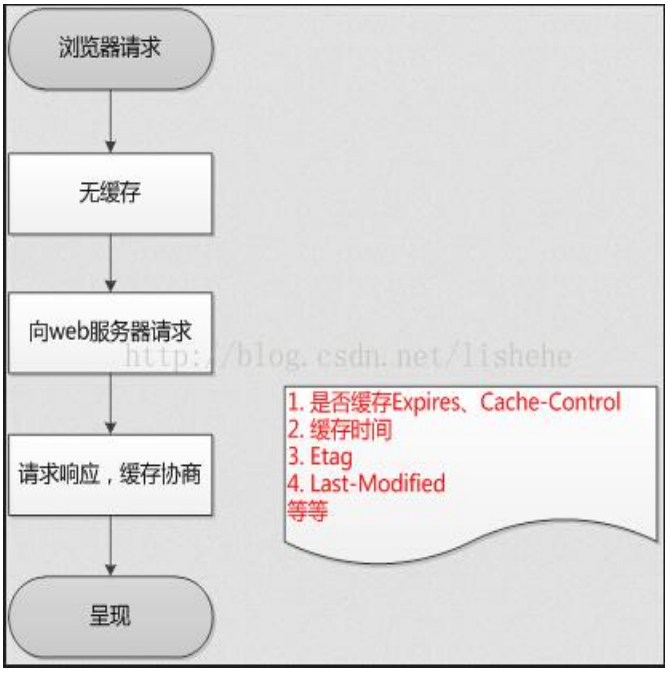
HTTP定义了3种缓存机制：

1）Freshness：允许一个回应消息可以在源服务器不被重新检查，并且可以由服务器和客户端来控制。例如，Expires回应头给了一个文档不可用的时间。Cache-Control中的max-age标识指明了缓存的最长时间；

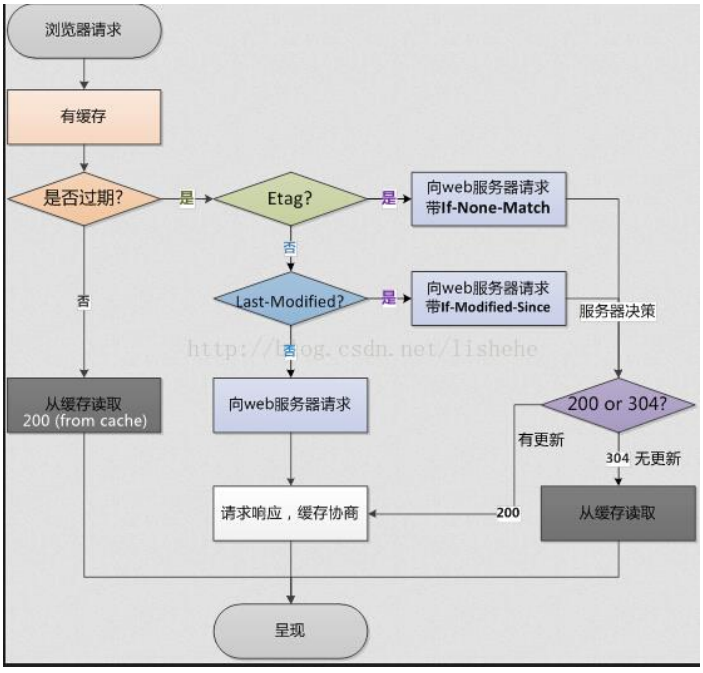
2）Validation：用来检查以一个缓存的回应是否仍然可用。例如，如果一个回应有一个Last-Modified回应头，缓存能够使用If-Modified-Since来判断是否已改变，以便判断根据情况发送请求；

3）Invalidation：在另一个请求通过缓存的时候，常常有一个副作用。例如，如果一个URL关联到一个缓存回应，但是其后跟着POST、PUT和DELETE的请求的话，缓存就会过期。

客户端第一次请求：



客户端第二次请求：



十一、HTTP 应用

断点续传的实现原理

HTTP协议的GET方法，支持只请求某个资源的某一部分； 206 Partial Content 部分内容响应； Range 请求的资源范围； Content-Range 响应的资源范围； 在连接断开重连时，客户端只请求该资源未下载的部分，而不是重新请求整个资源，来实现断点续传。分块请求资源实例： Eg1：Range: bytes=306302- ：请求这个资源从306302个字节到末尾的部分； Eg2：Content-Range: bytes 306302-604047/604048：响应中指示携带的是该资源的第306302-604047的字节，该资源共604048个字节； 客户端通过并发的请求相同资源的不同片段，来实现对某个资源的并发分块下载。从而达到快速下载的目的。目前流行的FlashGet和迅雷基本都是这个原理。

多线程下载的原理

下载工具开启多个发出HTTP请求的线程； 每个http请求只请求资源文件的一部分：Content-Range: bytes 20000-40000/47000； 合并每个线程下载的文件。

http代理

http代理服务器

代理服务器英文全称是Proxy Server，其功能就是代理网络用户去取得网络信息。形象的说：它是网络信息的中转站。 代理服务器是介于浏览器和Web服务器之间的一台服务器，有了它之后，浏览器不是直接到Web服务器去取回网页而是向代理服务器发出请求，Request信号会先送到代理服务器，由代理服务器来取回浏览器所需要的信息并传送给你的浏览器。 而且，大部分代理服务器都具有缓冲的功能，就好象一个大的Cache，它有很大的存储空间，它不断将新取得数据储存到它本机的存储器上，如果浏览器所请求的数据在它本机的存储器上已经存在而且是最新的，那么它就不重新从Web服务器取数据，而直接将存储器上的数据传送给用户的浏览器，这样就能显著提高浏览速度和效率。更重要的是：Proxy Server(代理服务器)是Internet链路级网关所提供的一种重要的安全功能，它的工作主要在开放系统互联(OSI)模型的对话层。

http代理服务器的主要**功能**：

1）突破自身IP访问限制，访问国外站点。如：教育网、169网等网络用户可以通过代理访问国外网站；

2）访问一些单位或团体内部资源，如某大学FTP(前提是该代理地址在该资源的允许访问范围之内)，使用教育网内地址段免费代理服务器，就可以用于对教育 网开放的各类FTP下载上传，以及各类资料查询共享等服务；

3）突破中国电信的IP封锁：中国电信用户有很多网站是被限制访问的，这种限制是人为的，不同Serve对地址的封锁是不同的。所以不能访问时可以换一个国外的代理服务器试试；

4）提高访问速度：通常代理服务器都设置一个较大的硬盘缓冲区，当有外界的信息通过时，同时也将其保存到缓冲区中，当其他用户再访问相同的信息时，则直接由缓冲区中取出信息，传给用户，以提高访问速度；

5）隐藏真实IP：上网者也可以通过这种方法隐藏自己的IP，免受攻击。

对于客户端浏览器而言，http代理服务器相当于服务器。 而对于Web服务器而言，http代理服务器又担当了客户端的角色。

虚拟主机

虚拟主机：是在网络服务器上划分出一定的磁盘空间供用户放置站点、应用组件等，提供必要的站点功能与数据存放、传输功能。所谓虚拟主机，也叫“网站空间”就是把一台运行在互联网上的服务器划分成多个“虚拟”的服务器，每一个虚拟主机都具有独立的域名和完整的Internet服务器（支持WWW、FTP、E-mail等）功能。一台服务器上的不同虚拟主机是各自独立的，并由用户自行管理。但一台服务器主机只能够支持一定数量的虚拟主机，当超过这个数量时，用户将会感到性能急剧下降。

虚拟主机的**实现原理**

虚拟主机是用同一个WEB服务器，为不同域名网站提供服务的技术。Apache、Tomcat等均可通过配置实现这个功能。 相关的HTTP消息头：Host。 例如：Host: www.baidu.com 客户端发送HTTP请求的时候，会携带Host头，Host头记录的是客户端输入的域名。这样服务器可以根据Host头确认客户要访问的是哪一个域名。

十二、HTTP 认证方式

HTTP请求报头： Authorization

HTTP响应报头： WWW-Authenticate

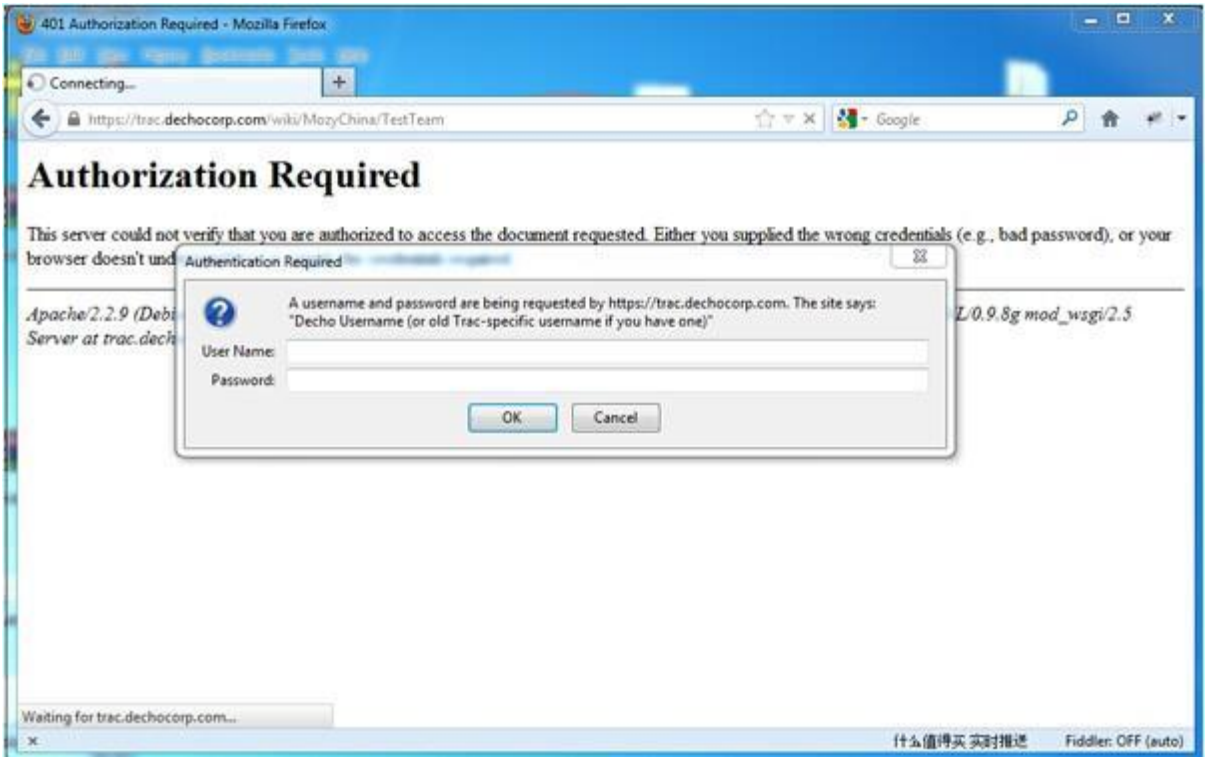
HTTP认证是基于质询/回应(challenge/response)的认证模式。

基本认证 basic authentication（HTTP1.0提出的认证方法）

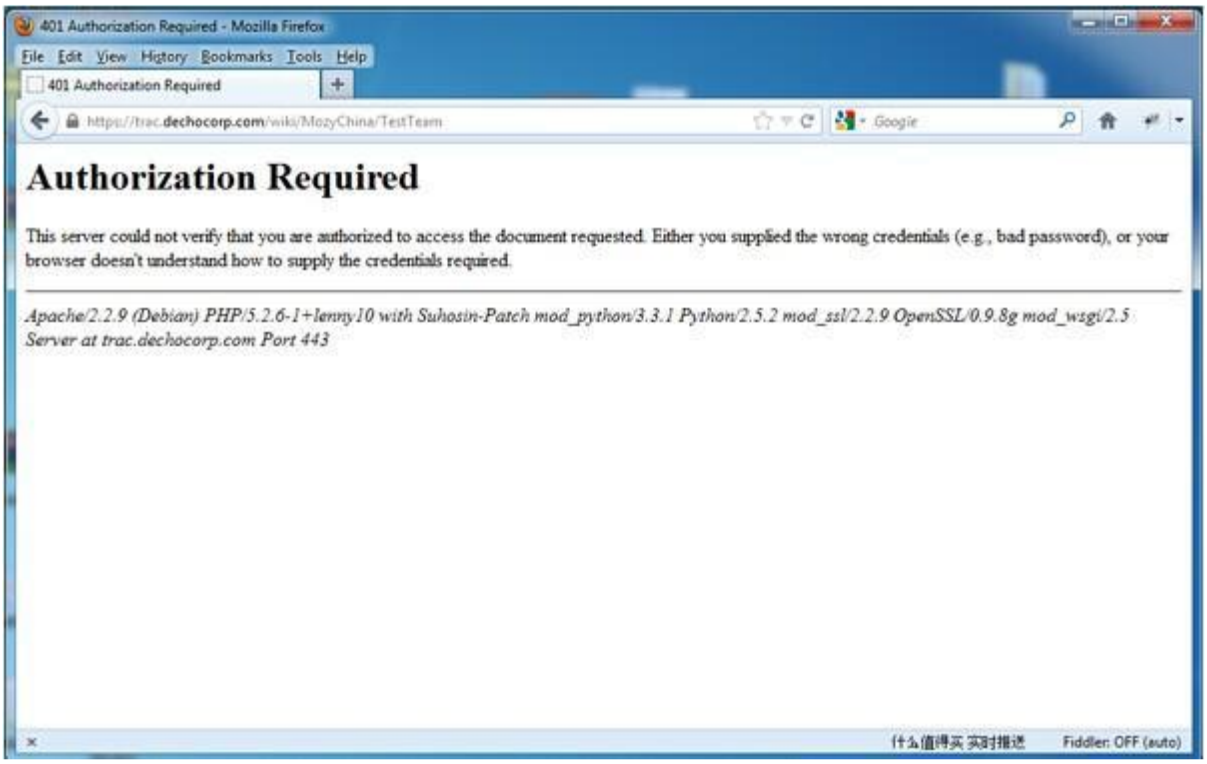
基本认证是一种用来允许Web浏览器或其他客户端程序在请求时提供用户名和口令形式的身份凭证的一种登录验证方式。

把 "用户名+冒号+密码"用BASE64算法加密后的字符串放在http request 中的header Authorization中发送给服务端。 客户端对于每一个realm，通过提供用户名和密码来进行认证的方式。 包含密码的明文传递。

当浏览器访问使用基本认证的网站的时候， 浏览器会提示你输入用户名和密码，如下图：



假如用户名密码错误的话，服务器会返回401，如下图：



基本认证步骤：

1、客户端访问一个受http基本认证保护的资源。

2、服务器返回401状态，要求客户端提供用户名和密码进行认证。（验证失败的时候，响应头会加上WWW-Authenticate: Basic realm="请求域"。） 401 Unauthorized WWW-Authenticate： Basic realm="WallyWorld"

3、客户端将输入的用户名密码用Base64进行编码后，采用非加密的明文方式传送给服务器。 Authorization: Basic xxxxxxxxxx.

4、服务器将Authorization头中的用户名密码解码并取出，进行验证，如果认证成功，则返回相应的资源。如果认证失败，则仍返回401状态，要求重新进行认证。

特记事项：

1、Http是无状态的，同一个客户端对同一个realm内资源的每一个访问会被要求进行认证。

2、客户端通常会缓存用户名和密码，并和authentication realm一起保存，所以，一般不需要你重新输入用户名和密码。

3、以非加密的明文方式传输，虽然转换成了不易被人直接识别的字符串，但是无法防止用户名密码被恶意盗用。虽然用肉眼看不出来，但用程序很容易解密。

优点： 基本认证的一个优点是基本上所有流行的网页浏览器都支持基本认证。基本认证很少在可公开访问的互联网网站上使用，有时候会在小的私有系统中使用（如路由器 网页管理接口）。后来的机制HTTP摘要认证是为替代基本认证而开发的，允许密钥以相对安全的方式在不安全的通道上传输。

程序员和系统管理员有时会在可信网络环境中使用基本认证，使用Telnet或其他明文网络协议工具手动地测试Web服务器。这是一个麻烦的过程，但是网络上传输的 内容是人可读的，以便进行诊断。

缺点： 虽然基本认证非常容易实现，但该方案建立在以下的假设的基础上，即：客户端和服务器主机之间的连接是安全可信的。特别是，如果没有使用SSL/TLS这样的传输 层安全的协议，那么以明文传输的密钥和口令很容易被拦截。该方案也同样没有对服务器返回的信息提供保护。

现存的浏览器保存认证信息直到标签页或浏览器被关闭，或者用户清除历史记录。HTTP没有为服务器提供一种方法指示客户端丢弃这些被缓存的密钥。这意味着服务 器端在用户不关闭浏览器的情况下，并没有一种有效的方法来让用户登出。

一个例子：

这一个典型的HTTP客户端和HTTP服务器的对话，服务器安装在同一台计算机上（localhost），包含以下步骤：

客户端请求一个需要身份认证的页面，但是没有提供用户名和口令。这通常是用户在地址栏输入一个URL，或是打开了一个指向该页面的链接。服务端响应一个401应 答码，并提供一个认证域。接到应答后，客户端显示该认证域（通常是所访问的计算机或系统的描述）给用户并提示输入用户名和口令。此时用户可以选择确定或取 消。用户输入了用户名和口令后，客户端软件会在原先的请求上增加认证消息头（值是base64encode(username+":"+password)），然后重新发送再次尝试。 在本例中，服务器接受了该认证屏幕并返回了页面。如果用户凭据非法或无效，服务器可能再次返回401应答码，客户端可以再次提示用户输入口令。 注意:客户端有可能不需要用户交互，在第一次请求中就发送认证消息头。

客户端请求（没有认证信息）：

GET /private/index.html HTTP/1.0

Host: localhost

（跟随一个换行，以回车（CR）加换行（LF）的形式）

服务端应答：

HTTP/1.0 401 Authorization Required

Server: HTTPd/1.0

Date: Sat, 27 Nov 2004 10:18:15 GMT

WWW-Authenticate: Basic realm="Secure Area"

Content-Type: text/html

Content-Length: 311

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"

"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">

<HTML>

<HEAD>

<TITLE>Error</TITLE>

<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY><H1>401 Unauthorized.</H1></BODY>
</HTML>
客户端的请求（用户名“"Aladdin”，口令, password “open sesame”）：
GET /private/index.html HTTP/1.0
Host: localhost
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
（跟随一个空行，如上所述）
服务端的应答：
HTTP/1.0 200 OK
Server: HTTPd/1.0
Date: Sat, 27 Nov 2004 10:19:07 GMT
Content-Type: text/html
Content-Length: 10476
（跟随一个空行，随后是需凭据页的HTML文本）。
HTTP OAuth认证 OAuth对于Http来说，就是放在Authorization header中的不是用户名密码， 而是一个token。微软的Skydrive就是使用这样的方式。
摘要认证 digest authentication（HTTP1.1提出的基本认证的替代方法）
这个认证可以看做是基本认证的增强版本，不包含密码的明文传递。
引入了一系列安全增强的选项；“保护质量”(qop)、随机数计数器由客户端增加、以及客户生成的随机数。

在HTTP摘要认证中使用 MD5 加密是为了达成"不可逆的"，也就是说，当输出已知的时候，确定原始的输入应该是相当困难的。如果密码本身太过简单，也许可以 通过尝试所有可能的输入来找到对应的输出（穷举攻击），甚至可以通过字典或者适当的查找表加快查找速度。

示例及说明 下面的例子仅仅涵盖了“auth”保护质量的代码，因为在撰写期间，所知道的只有Opera和Konqueror网页浏览器支持“auth-int”（带完整性保护的认证）。

典型的认证过程包括如下步骤： 客户端请求一个需要认证的页面，但是不提供用户名和密码。通常这是由于用户简单的输入了一个地址或者在页面中点击了某个超链接。 服务器返回401 "Unauthorized" 响应代码，并提供认证域(realm)，以及一个随机生成的、只使用一次的数值，称为密码随机数 nonce。 此时，浏览器会向用户提示认证域(realm)（通常是所访问的计算机或系统的描述），并且提示用户名和密码。用户此时可以选择取消。 一旦提供了用户名和密码，客户端会重新发送同样的请求，但是添加了一个认证头包括了响应代码。

注意：客户端可能已经拥有了用户名和密码，因此不需要提示用户，比如以前存储在浏览器里的。

客户端请求 (无认证)：

```
GET /dir/index.html HTTP/1.0
Host: localhost
(跟随一个新行，形式为一个回车再跟一个换行）
```

服务器响应：

```
HTTP/1.0 401 Unauthorized
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:26:47 GMT
WWW-Authenticate: Digest realm="testrealm@host.com", //认证域
qop="auth,auth-int", //保护质量
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093", //服务器密码随机数
opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Type: text/html
Content-Length: 311
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/1999/REC-html401-19991224/loose.dtd">
<HTML>
<HEAD>
<TITLE>Error</TITLE>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=ISO-8859-1">
</HEAD>
<BODY><H1>401 Unauthorized.</H1></BODY>
```

</HTML>
客户端请求（用户名 "Mufasa"，密码 "Circle Of Life"）：
GET /dir/index.html HTTP/1.0
Host: localhost
Authorization: Digest username="Mufasa",
realm="testrealm@host.com",
nonce="dcd98b7102dd2f0e8b11d0f600bfb0c093",
uri="/dir/index.html",
qop=auth,
nc=00000001, //请求计数
cnonce="0a4f113b", //客户端密码随机数
response="6629fae49393a05397450978507c4ef1",
opaque="5ccc069c403ebaf9f0171e9517f40e41"
(跟随一个新行，形式如前所述)。
服务器响应：
HTTP/1.0 200 OK
Server: HTTPd/0.9
Date: Sun, 10 Apr 2005 20:27:03 GMT
Content-Type: text/html
Content-Length: 7984
(随后是一个空行，然后是所请求受限制的HTML页面)
response 值由三步计算而成。当多个数值合并的时候，使用冒号作为分割符：
1、对用户名、认证域(realm)以及密码的合并值计算 MD5 哈希值，结果称为 HA1。 2、对HTTP方法以及URI的摘要的合并值计算 MD5 哈希值，例如，"GET" 和 "/dir/index.html"，结果称为 HA2。 3、对HA1、服务器密码随机数(nonce)、请求计数(nc)、客户端密码随机数(cnonce)、保护质量(qop)以及 HA2 的合并值计算 MD5 哈希值。结果即为客户端提供的 response 值。 因为服务器拥有与客户端同样的信息，因此服务器可以进行同样的计算，以验证客户端提交的 response 值的正确性。在上面给出的例子中，结果是如下计算的。 （MD5()表示用于计算MD5哈希值的函数；“\”表示接下一行；引号并不参与计算）
HA1 = MD5("Mufasa:testrealm@host.com:Circle Of Life") = 939e7578ed9e3c518a452acee763bce9
HA2 = MD5("GET:/dir/index.html") = 39aff3a2bab6126f332b942af96d3366

Response = MD5("939e7578ed9e3c518a452acee763bce9:\ dcd98b7102dd2f0e8b11d0f600bfb0c093:\ 00000001:0a4f113b:auth:\ 39aff3a2bab6126f332b942af96d3366") = 6629fae49393a05397450978507c4ef1

此时客户端可以提交一个新的请求，重复使用服务器密码随机数(nonce)（服务器仅在每次“401”响应后发行新的nonce），但是提供新的客户端密码随机数(cnonce)。在后续的请求中，十六进制请求计数器(nc)必须比前一次使用的时候要大，否则攻击者可以简单的使用同样的认证信息重放老的请求。由服务器来确保在每个发出的密码随机数nonce时，计数器是在增加的，并拒绝掉任何错误的请求。显然，改变HTTP方法和/或计数器数值都会导致不同的 response值。

服务器应当记住最近所生成的服务器密码随机数nonce的值。也可以在发行每一个密码随机数nonce后，记住过一段时间让它们过期。如果客户端使用了一个过期的值，服务器应该响应“401”状态号，并且在认证头中添加stale=TRUE，表明客户端应当使用新提供的服务器密码随机数nonce重发请求，而不必提示用户其它用户名和口令。

服务器不需要保存任何过期的密码随机数，它可以简单的认为所有不认识的数值都是过期的。服务器也可以只允许每一个服务器密码随机数nonce使用一次，当然，这样就会迫使客户端在发送每个请求的时候重复认证过程。需要注意的是，在生成后立刻过期服务器密码随机数nonce是不行的，因为客户端将没有任何机会来使用这个nonce。

PS：以上只介绍了两种比较基础的，还有其他的一些认证方式就不在这里一一说明了。

十三、HTTPS 传输协议原理

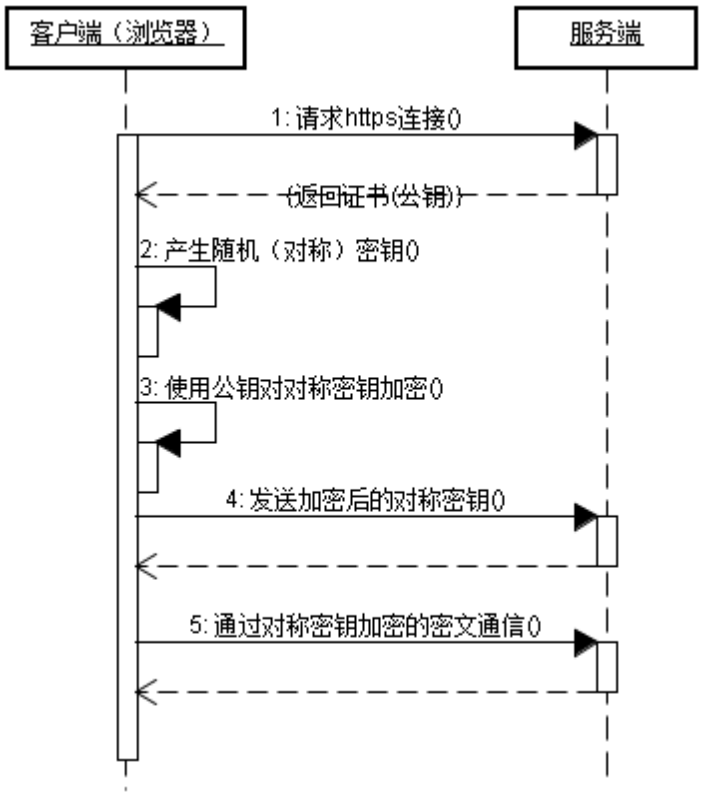
HTTPS（全称：Hypertext Transfer Protocol over Secure Socket Layer），是以安全为目标的HTTP通道，简单讲是HTTP的安全版。即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容请看SSL。

两种基本的加解密算法类型

对称加密：密钥只有一个，加密解密为同一个密码，且加解密速度快，典型的对称加密算法有DES、AES等。

非对称加密：密钥成对出现（且根据公钥无法推知私钥，根据私钥也无法推知公钥），加密解密使用不同密钥（公钥加密的内容需要私钥解密，私钥加密的内容需要公钥解密），相对对称加密速度较慢，典型的非对称加密算法有RSA、DSA等。

HTTPS通信过程



HTTPS通信的优点 客户端产生的密钥只有客户端和服务器端能得到； 加密的数据只有客户端和服务器端才能得到明文； 客户端到服务端的通信是安全的。

http的状态响应码

1**(信息类)：表示接收到请求并且继续处理

100——客户必须继续发出请求

101——客户要求服务器根据请求转换HTTP协议版本

2**(响应成功)：表示动作被成功接收、理解和接受

200——表明该请求被成功地完成，所请求的资源发送回客户端

201——提示知道新文件的URL

202——接受和处理、但处理未完成

203——返回信息不确定或不完整

204——请求收到，但返回信息为空

205——服务器完成了请求，用户代理必须复位当前已经浏览过的文件

206——服务器已经完成了部分用户的GET请求

3**(重定向类)：为了完成指定的动作，必须接受进一步处理

300——请求的资源可在多处得到

301——本网页被永久性转移到另一个URL

302——请求的网页被转移到一个新的地址，但客户访问仍继续通过原始URL地址，重定向，新的URL会在response中的Location中返回，浏览器将会使用新的URL发出新的Request。

303——建议客户访问其他URL或访问方式

304——自从上次请求后，请求的网页未修改过，服务器返回此响应时，不会返回网页内容，代表上次的文档已经被缓存了，还可以继续使用

305——请求的资源必须从服务器指定的地址得到

306——前一版本HTTP中使用的代码，现行版本中不再使用

307——申明请求的资源临时性删除

4**(客户端错误类)：请求包含错误语法或不能正确执行

400——客户端请求有语法错误，不能被服务器所理解

401——请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

HTTP 401.1 - 未授权：登录失败

HTTP 401.2 - 未授权：服务器配置问题导致登录失败

HTTP 401.3 - ACL 禁止访问资源

HTTP 401.4 - 未授权：授权被筛选器拒绝

HTTP 401.5 - 未授权：ISAPI 或 CGI 授权失败

402——保留有效ChargeTo头响应

403——禁止访问，服务器收到请求，但是拒绝提供服务

HTTP 403.1 禁止访问：禁止可执行访问

HTTP 403.2 - 禁止访问：禁止读访问

HTTP 403.3 - 禁止访问：禁止写访问

HTTP 403.4 - 禁止访问：要求 SSL

HTTP 403.5 - 禁止访问：要求 SSL 128

HTTP 403.6 - 禁止访问：IP 地址被拒绝

HTTP 403.7 - 禁止访问：要求客户证书

HTTP 403.8 - 禁止访问：禁止站点访问

HTTP 403.9 - 禁止访问：连接的用户过多

HTTP 403.10 - 禁止访问：配置无效

HTTP 403.11 - 禁止访问：密码更改

HTTP 403.12 - 禁止访问：映射器拒绝访问

HTTP 403.13 - 禁止访问：客户证书已被吊销

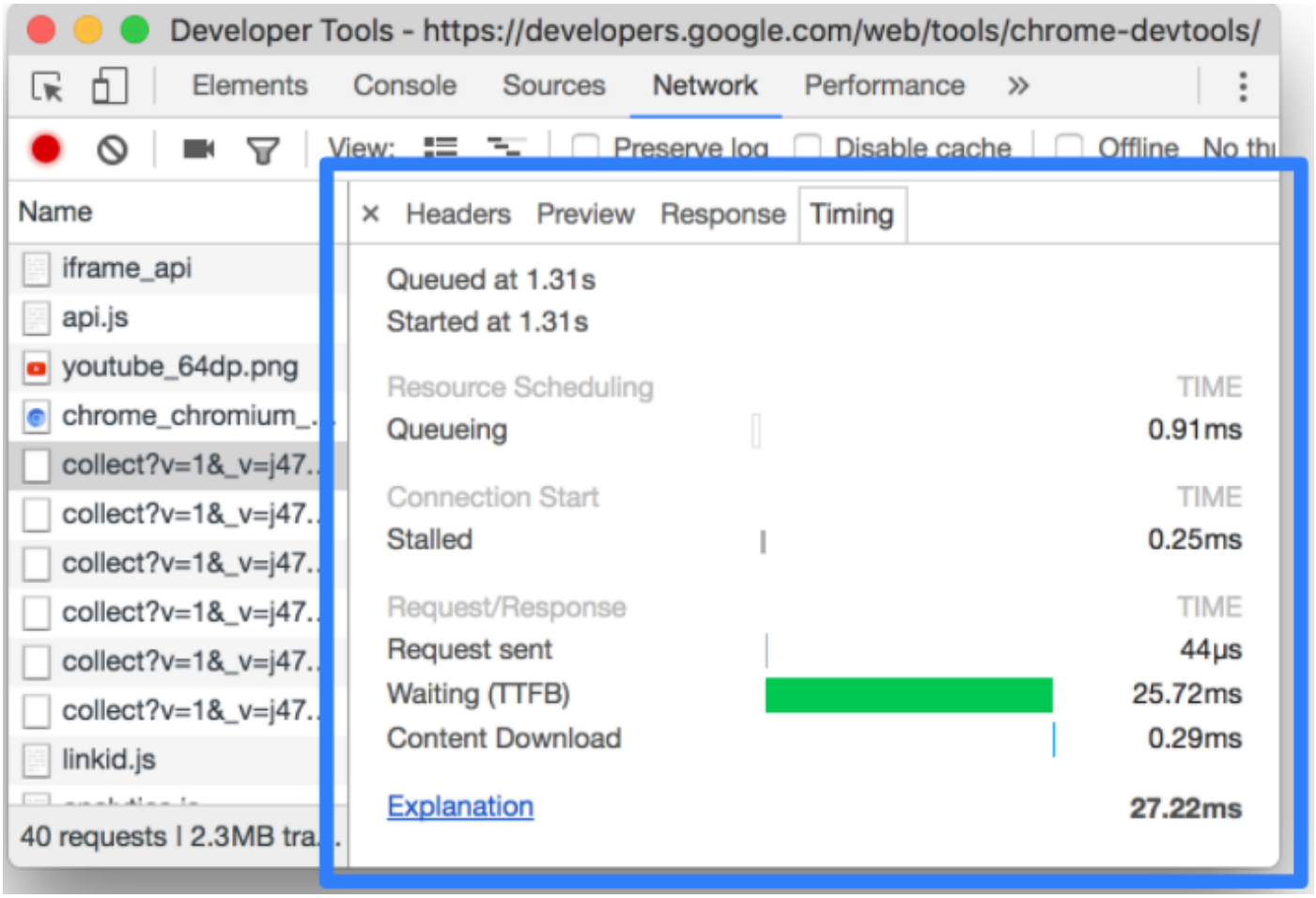
HTTP 403.15 - 禁止访问：客户访问许可过多

HTTP 403.16 - 禁止访问：客户证书不可信或者无效
HTTP 403.17 - 禁止访问：客户证书已经到期或者尚未生效
404——一个404错误表明可连接服务器，但服务器无法取得所请求的网页，请求资源不存在。eg：输入了错误的URL
405——用户在Request-Line字段定义的方法不允许
406——根据用户发送的Accept拖，请求资源不可访问
407——类似401，用户必须首先在代理服务器上得到授权
408——客户端没有在用户指定的饿时间内完成请求
409——对当前资源状态，请求不能完成
410——服务器上不再有此资源且无进一步的参考地址
411——服务器拒绝用户定义的Content-Length属性请求
412——一个或多个请求头字段在当前请求中错误
413——请求的资源大于服务器允许的大小
414——请求的资源URL长于服务器允许的长度
415——请求资源不支持请求项目格式
416——请求中包含Range请求头字段，在当前请求资源范围内没有range指示值，请求也不包含If-Range请求头字段
417——服务器不满足请求Expect头字段指定的期望值，如果是代理服务器，可能是下一级服务器不能满足请求长。
5**(服务端错误类)：服务器不能正确执行一个正确的请求
HTTP 500 - 服务器遇到错误，无法完成请求
HTTP 500.100 - 内部服务器错误 - ASP 错误
HTTP 500-11 服务器关闭
HTTP 500-12 应用程序重新启动
HTTP 500-13 - 服务器太忙
HTTP 500-14 - 应用程序无效
HTTP 500-15 - 不允许请求 global.asa
HTTP 501 - 未实现 Not Implemented，客户端发起的请求使用了服务器不支持的请求方法。
HTTP 502 - 网关错误 Bad Gateway
HTTP 503：由于超载或停机维护，服务器目前无法使用，一段时间后可能恢复正常 Service Unavailable
HTTP 504：网关超时 Gateway Timeout
HTTP 505：HTTP Version Not Supported，服务器收到的请求使用了它无法或不愿支持的协议版本时，使用此状态码。

十四、HTTP 2.0

https://www.zhihu.com/question/34074946

十五、Chrome 性能诊断



Timing breakdown phases explained

Here's more information about each of the phases you may see in the Timing tab:

- **Queueing**. The browser queues requests when:
  - There are higher priority requests.
  - There are already six TCP connections open for this origin, which is the limit. Applies to HTTP/1.0 and HTTP/1.1 only.
  - The browser is briefly allocating space in the disk cache
- **Stalled**. The request could be stalled for any of the reasons described in **Queueing**.
- **DNS Lookup**. The browser is resolving the request's IP address.
- **Proxy negotiation**. The browser is negotiating the request with a proxy server.
- **Request sent**. The request is being sent.
- **ServiceWorker Preparation**. The browser is starting up the service worker.
- **Request to ServiceWorker**. The request is being sent to the service worker.
- **Waiting (TTFB)**. The browser is waiting for the first byte of a response. TTFB stands for Time To First Byte. This timing includes 1 round trip of latency and the time the server took to prepare the response.
- **Content Download**. The browser is receiving the response.
- **Receiving Push**. The browser is receiving data for this response via HTTP/2 Server Push.
- **Reading Push**. The browser is reading the local data previously received.