深入理解 HTTP 协议

http 协议学习系列

1. 基础概念篇

1.1 介绍

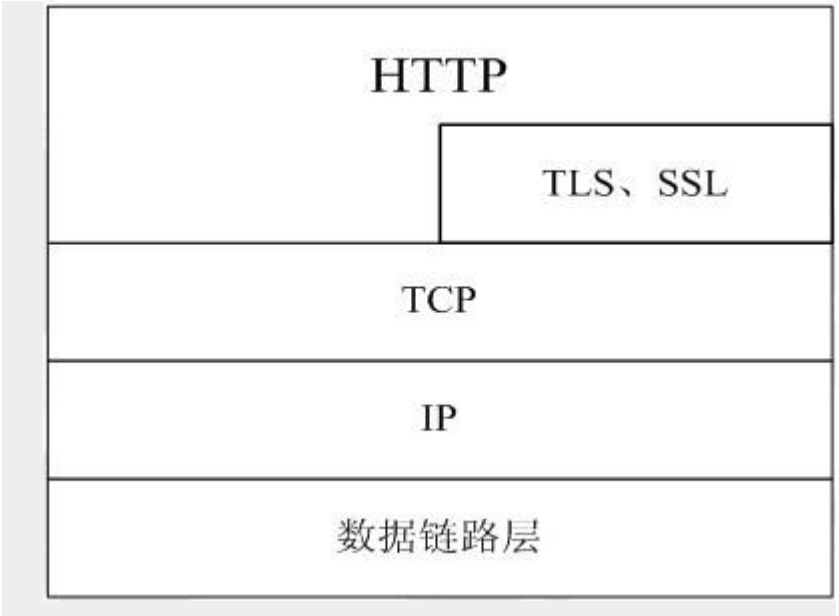
HTTP 是 Hyper Text Transfer Protocol（超文本传输协议）的缩写。它的发展是万维网协会（World Wide Web Consortium）和 Internet 工作小组 IETF（Internet Engineering Task Force）合作的结果，（他们）最终发布了一系列的 RFC，RFC 1945 定义了 HTTP/1.0 版本。其中最著名的就是 RFC 2616。RFC 2616 定义了今天普遍使用的一个版本——HTTP 1.1。

HTTP 协议（HyperText Transfer Protocol，超文本传输协议）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。

1.2 在 TCP/IP 协议栈中的位置

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：



默认 HTTP 的端口号为 80，HTTPS 的端口号为 443。

1.3 HTTP 的请求响应模型

HTTP 协议永远都是客户端发起请求，服务器回送响应。见下图：

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 协议是一个无状态的协议，同一个客户端的这次请求和上次请求是没有对应关系。

1.4 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。

1.5 使用 Wireshark 抓 TCP、http 包

打开 Wireshark，选择工具栏上的“Capture”->“Options”，界面选择如图 1 所示：

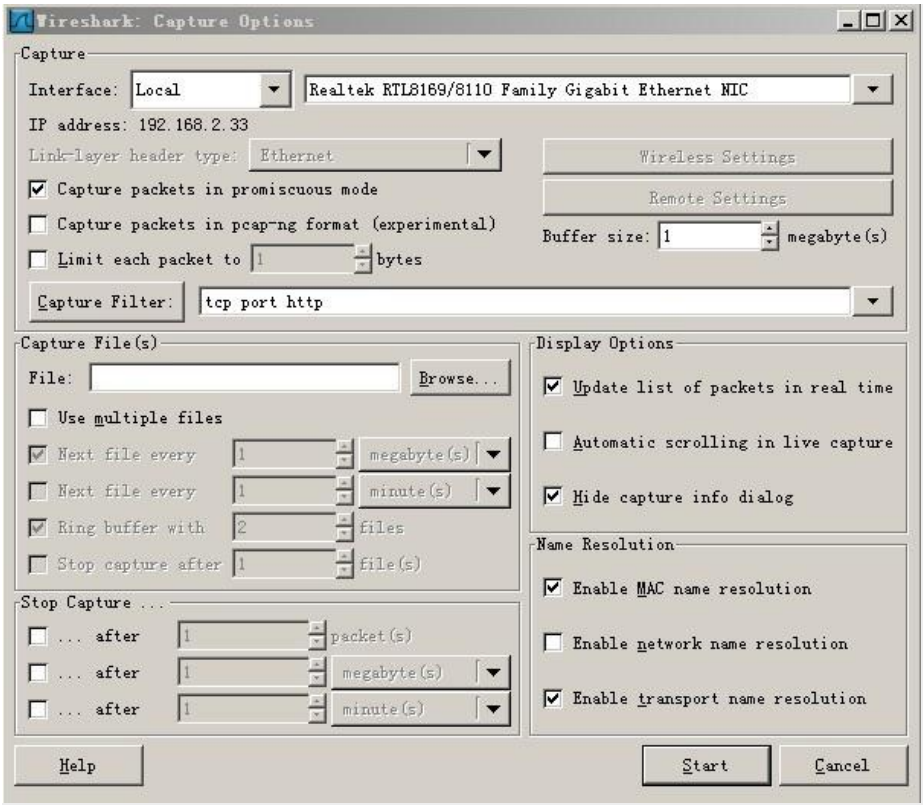


图 1 设置 Capture 选项

一般读者只需要选择最上边的下拉框，选择合适的 Device，而后点击“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。

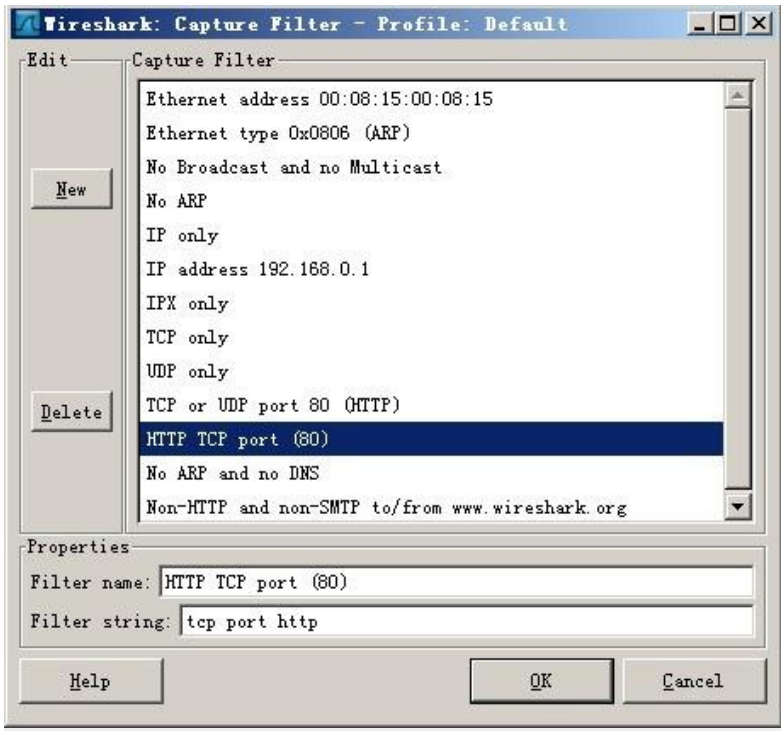
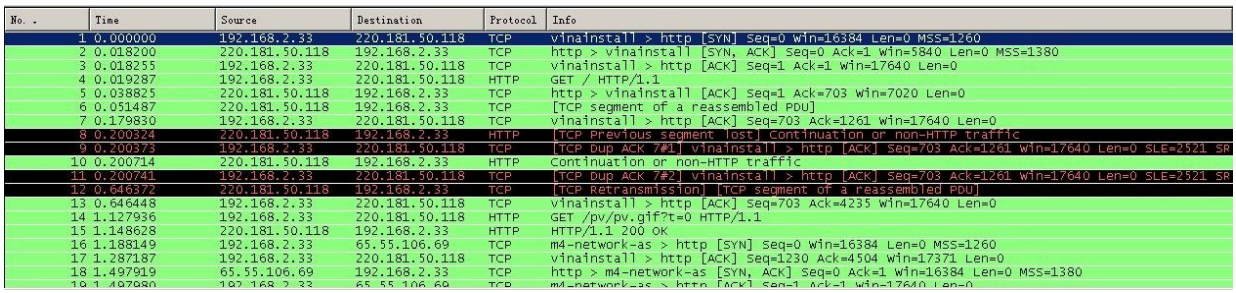


图 2 选择 Capture Filter

例如在浏览器中打开 http://image.baidu.com/，抓包如图 3 所示：

http://www.blogjava.net/images/blogjava_net/amigoxie/40799/o_http%e5%8d%8f%e8%ae%ae%e5%ad%a6%e4%b9%a0-%e6%a6%82%e5%bf%b5-3.jpg



No.	Time	Source	Destination	Protocol	Info
1	0.000000	192.168.2.33	220.181.50.118	TCP	vinainstall > http [SYN] Seq=0 Win=16384 Len=0 MSS=1260
2	0.018200	220.181.50.118	192.168.2.33	TCP	http > vinainstall [SYN, ACK] Seq=0 Ack=1 Win=5840 Len=0 MSS=1380
3	0.018255	192.168.2.33	220.181.50.118	TCP	vinainstall > http [ACK] Seq=1 Ack=1 Win=17640 Len=0
4	0.019287	192.168.2.33	220.181.50.118	HTTP	GET / HTTP/1.1
5	0.038825	220.181.50.118	192.168.2.33	TCP	http > vinainstall [ACK] Seq=1 Ack=703 Win=7020 Len=0
6	0.051487	220.181.50.118	192.168.2.33	TCP	[TCP segment of a reassembled PDU]
7	0.179830	192.168.2.33	220.181.50.118	TCP	vinainstall > http [ACK] Seq=703 Ack=1261 Win=17640 Len=0
8	0.200324	220.181.50.118	192.168.2.33	HTTP	[TCP Previous segment lost] Continuation or non-HTTP traffic
9	0.200373	192.168.2.33	220.181.50.118	TCP	[TCP Dup ACK 7#1] vinainstall > http [ACK] Seq=703 Ack=1261 Win=17640 Len=0 SLE=2521 SR
10	0.200714	220.181.50.118	192.168.2.33	HTTP	Continuation or non-HTTP traffic
11	0.200741	192.168.2.33	220.181.50.118	TCP	[TCP Dup ACK 7#2] vinainstall > http [ACK] Seq=703 Ack=1261 Win=17640 Len=0 SLE=2521 SR
12	0.646372	220.181.50.118	192.168.2.33	TCP	[TCP Retransmission] [TCP segment of a reassembled PDU]
13	0.646448	192.168.2.33	220.181.50.118	TCP	vinainstall > http [ACK] Seq=703 Ack=4235 Win=17640 Len=0
14	1.127936	192.168.2.33	220.181.50.118	HTTP	GET /pv/pv.gif?t=0 HTTP/1.1
15	1.148628	220.181.50.118	192.168.2.33	HTTP	HTTP/1.1 200 OK
16	1.188149	192.168.2.33	65.55.106.69	TCP	m4-network-as > http [SYN] Seq=0 Win=16384 Len=0 MSS=1260
17	1.287187	192.168.2.33	220.181.50.118	TCP	vinainstall > http [ACK] Seq=1230 Ack=4504 Win=17371 Len=0
18	1.497919	65.55.106.69	192.168.2.33	TCP	http > m4-network-as [SYN, ACK] Seq=0 Ack=1 Win=16384 Len=0 MSS=1380
19	1.497980	192.168.2.33	65.55.106.69	TCP	m4-network-as > http [ACK] Seq=1 Ack=1 Win=17640 Len=0

图 3 抓包

在上图中，可清晰的看到客户端浏览器（ip 为 192.168.2.33）与服务器的交互过程：

1）No1：浏览器（192.168.2.33）向服务器（220.181.50.118）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（220.181.50.118）回应了浏览器（192.168.2.33）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（192.168.2.33）回应了服务器（220.181.50.118）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（192.168.2.33）发出一个页面 HTTP 请求；

5）No5：服务器（220.181.50.118）确认；

6）No6：服务器（220.181.50.118）发送数据；

7）No7：客户端浏览器（192.168.2.33）确认；

8）No14：客户端（192.168.2.33）发出一个图片 HTTP 请求；

9）No15：服务器（220.181.50.118）发送状态响应码 200 OK

……

1.6 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。

在抓包的图中，No14 点开可看到如图 4 所示：

http://www.blogjava.net/images/blogjava_net/amigoxie/40799

/o_http%e5%8d%8f%e8%ae%ae%e5%ad%a6%e4%b9%a0-%e6%a6%82%e5%bf%b5-4.jpg

```
⊟ Hypertext Transfer Protocol
  ⊟ GET /pv/pv.gif?t=0 HTTP/1.1\r\n
    ⊞ [Expert Info (Chat/Sequence): GET /pv/pv.gif?t=0 HTTP/1.1\r\n]
      Request Method: GET
      Request URI: /pv/pv.gif?t=0
      Request Version: HTTP/1.1
    Accept: */*\r\n
    Referer: http://image.baidu.com/\r\n
    Accept-Language: zh-cn\r\n
    Accept-Encoding: gzip, deflate\r\n
    If-Modified-Since: Wed, 19 Aug 2009 15:23:32 GMT\r\n
    If-None-Match: "557649757"\r\n
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.21022)\r\n
    Host: image.baidu.com\r\n
    Connection: Keep-Alive\r\n
    Cookie: iCast_Rotator_1_1=1259581841765; iCast_Rotator_1_2=1259586044296; BAIDUID=50265E09E7592D1C415755687611D9F9:FG=1; BD_UTK_DVT=1\r
    \r\n
```

图 4 http 请求消息

回应的消息如图 5 所示：

```
⊟ Hypertext Transfer Protocol
  ⊟ HTTP/1.1 200 OK\r\n
    ⊞ [Expert Info (Chat/Sequence): HTTP/1.1 200 OK\r\n]
      Request Version: HTTP/1.1
      Response Code: 200
    Content-Type: image/gif\r\n
    ETag: "567281165"\r\n
    Accept-Ranges: bytes\r\n
    Last-Modified: Wed, 19 Aug 2009 15:23:26 GMT\r\n
    Expires: Mon, 30 Nov 2009 13:15:39 GMT\r\n
    Cache-Control: max-age=0\r\n
  ⊟ Content-Length: 0\r\n
      [Content length: 0]
    Date: Mon, 30 Nov 2009 13:15:39 GMT\r\n
    Server: Apache\r\n
    \r\n
```

图 5 http 状态响应信息

1.6.1 host 头域

Host 头域指定请求资源的 Intenet 主机和端口号，必须表示请求 url 的原始服务器或网关的位置。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。

图 5 中 host 那行为：

```
Host: image.baidu.com\r\n
```

1.6.2 Referer 头域

Referer 头域允许客户端指定请求 uri 的源资源地址，这可以允许服务器生成回退链表，可用来登陆、优化 cache 等。他也允许废除的或错误的连接由于维护的目的被追踪。如果请求的 uri 没有自己的 uri 地址，Referer 不能被发送。如果指定的是部分 uri 地址，则此地址应该是一个相对地址。

在图 4 中，Referer 行的内容为：

```
Referer: http://image.baidu.com/\r\n
```

1.6.3 User-Agent 头域

User-Agent 头域的内容包含发出请求的用户信息。

在图 4 中，User-Agent 行的内容为：

http://www.blogjava.net/images/blogjava_net/amigoxie/40799/o_http%e5%8d%8f%e8%ae%ae%e5%ad%a6%e4%b9%a0-%e6%a6%82%e5%bf%b5-8.jpg

1.6.4 Cache-Control 头域

Cache-Control 指定请求和响应遵循的缓存机制。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。

在图 5 中的该头域为：

```
Cache-Control: max-age=0\r\n
```

1.6.5 Date 头域

Date 头域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。

图 5 中，该头域如下图所示：

```
Date: Mon, 30 Nov 2009 13:15:39 GMT\r\n
```

1.7 HTTP 的几个重要概念

1.7.1 连接：Connection

一个传输层的实际环流，它是建立在两个相互通讯的应用程序之间。

在 http1.1，request 和 reponse 头中都有可能出现一个 connection 的头，此 header 的含义是当 client 和 server 通信时对于长链接如何进行处理。

在 http1.1 中，client 和 server 都是默认对方支持长链接的， 如果 client 使用 http1.1 协议，但又不希望使用长链接，则需要在 header 中指明 connection 的值为 close；如果 server 方也不想支持长链接，则在 response 中也需要明确说明 connection 的值为

close。不论 request 还是 response 的 header 中包含了值为 close 的 connection，都表明当前正在使用的 tcp 链接在当天请求处理完毕后会被断掉。以后 client 再进行新的请求时就必须创建新的 tcp 链接了。

1.7.2 消息：Message

HTTP 通讯的基本单位，包括一个结构化的八元组序列并通过连接传输。

1.7.3 请求：Request

一个从客户端到服务器的请求信息包括应用于资源的方法、资源的标识符和协议的版本号。

1.7.4 响应：Response

一个从服务器返回的信息包括 HTTP 协议的版本号、请求的状态(例如“成功”或“没找到”)和文档的 MIME 类型。

1.7.5 资源：Resource

由 URI 标识的网络数据对象或服务。

1.7.6 实体：Entity

数据资源或来自服务资源的回映的一种特殊表示方法，它可能被包围在一个请求或响应信息中。一个实体包括实体头信息和实体的本身内容。

1.7.7 客户机：Client

一个为发送请求目的而建立连接的应用程序。

1.7.8 用户代理：UserAgent

初始化一个请求的客户机。它们是浏览器、编辑器或其它用户工具。

1.7.9 服务器：Server

一个接受连接并对请求返回信息的应用程序。

1.7.10 源服务器：Originserver

是一个给定资源可以在其上驻留或被创建的服务器。

1.7.11 代理：Proxy

一个中间程序，它可以充当一个服务器，也可以充当一个客户机，为其它客户机建立请求。请求是通过可能的翻译在内部或经过传递到其它的服务器中。一个代理在发送请求信息之前，必须解释并且如果可能重写它。

代理经常作为通过防火墙的客户机端的门户，代理还可以作为一个帮助应用来通过协议处理没有被用户代理完成的请求。

1.7.12 网关：Gateway

一个作为其它服务器中间媒介的服务器。与代理不同的是，网关接受请求就好象对被请求的资源来说它就是源服务器；发出请求的客户机并没有意识到它在同网关打交道。

网关经常作为通过防火墙的服务器端的门户，网关还可以作为一个协议翻译器以便存取那些存储在非 HTTP 系统中的资源。

1.7.13 通道：Tunnel

是作为两个连接中继的中介程序。一旦激活，通道便被认为不属于 HTTP 通讯，尽管通道可能是被一个 HTTP 请求初始化的。当被

中继的连接两端关闭时，通道便消失。当一个门户(Portal)必须存在或中介(Intermediary)不能解释中继的通讯时通道被经常使用。

1.7.14 缓存：Cache

反应信息的局域存储。

附录：参考资料

《http_百度百科》：http://baike.baidu.com/view/9472.htm

《结果编码和 http 状态响应码》：http://blog.tieniu1980.cn/archives/377

《分析 TCP 的三次握手》：http://cache.baidu.com/c?m=9f65cb4a8c8507ed4fece763104c8c711923d030678197027fa3c215cc7905141130a8e5747e0d548d98297a5ae91e03f7f63772315477e3cacdd94cdbbdc42225d82c36734f844315c419d891007a9f34d507a9f916a2e1b065d2f48193864353bb15543897f1fb4d711edd1b86033093b1e94e022e67adec40728e2e605f983431c5508fe4&p=c6769a46c5820efd08e2973b42&user=baidu

《使用 Wireshark 来检测一次 HTTP 连接过程》：http://blog.163.com/wangbo_tester/blog/static/128067921200098174162288/

《http 协议的几个重要概念》：http://nc.mofcom.gov.cn/news/10819972.html

《http 协议中 connection 头的作用》：

http://blog.csdn.net/barfoo/archive/2008/06/05/2514667.aspx

2. 协议详解篇

2.1 HTTP/1.0 和 HTTP/1.1 的比较

RFC 1945 定义了 HTTP/1.0 版本，RFC 2616 定义了 HTTP/1.1 版本。

笔者在 blog 上提供了这两个 RFC 中文版的下载地址。

RFC1945 下载地址：

http://www.blogjava.net/Files/amigoxie/RFC1945（HTTP）中文版.rar

RFC2616 下载地址：

http://www.blogjava.net/Files/amigoxie/RFC2616（HTTP）中文版.rar

2.1.1 建立连接方面

HTTP/1.0 每次请求都需要建立新的 TCP 连接，连接不能复用。HTTP/1.1 新的请求可以在上次请求建立的 TCP 连接之上发送，连接可以复用。优点是减少重复进行 TCP 三次握手的开销，提高效率。

注意：在同一个 TCP 连接中，新的请求需要等上次请求收到响应后，才能发送。

2.1.2 Host 域

HTTP1.1 在 Request 消息头里头多了一个 Host 域, HTTP1.0 则没有这个域。

Eg：

```
GET /pub/WWW/TheProject.html HTTP/1.1
Host: www.w3.org
```

可能 HTTP1.0 的时候认为，建立 TCP 连接的时候已经指定了 IP 地址，这个 IP 地址上只有一个 host。

2.1.3 日期时间戳

(接收方向)

无论是 HTTP1.0 还是 HTTP1.1，都要能解析下面三种 date/time stamp：

```
Sun, 06 Nov 1994 08:49:37 GMT ; RFC 822, updated by RFC 1123
  Sunday, 06-Nov-
94 08:49:37 GMT ; RFC 850, obsoleted by RFC 1036
  Sun Nov 6 08:49:37 1994       ; ANSI C's asctime() format
```

(发送方向)

HTTP1.0 要求不能生成第三种 asctime 格式的 date/time stamp；

HTTP1.1 则要求只生成 RFC 1123(第一种)格式的 date/time stamp。

2.1.4 状态响应码

状态响应码 100 (Continue) 状态代码的使用，允许客户端在发 request 消息 body 之前先用 request header 试探一下 server，看 server 要不要接收 request body，再决定要不要发 request body。

客户端在 Request 头部中包含

```
Expect: 100-continue
```

Server 看到之后呢如果回 100 (Continue) 这个状态代码，客户端就继续发 request body。这个是 HTTP1.1 才有的。

另外在 HTTP/1.1 中还增加了 101、203、205 等等性状态响应码

2.1.5 请求方式

HTTP1.1 增加了 OPTIONS, PUT, DELETE, TRACE, CONNECT 这些 Request 方法.

```
    Method         = "OPTIONS"                ; Section 9.2
                   | "GET"                    ; Section 9.3
                   | "HEAD"                   ; Section 9.4
                   | "POST"                   ; Section 9.5
                   | "PUT"                    ; Section 9.6
                   | "DELETE"                 ; Section 9.7
                   | "TRACE"                  ; Section 9.8
                   | "CONNECT"                ; Section 9.9
                   | extension-method
    extension-method = token
```

2.2 HTTP 请求消息

2.2.1 请求消息格式

请求消息格式如下所示：

请求行

通用信息头|请求头|实体头

CRLF(回车换行)

实体内容

其中“请求行”为：请求行 = 方法 [空格] 请求 URI [空格] 版本号 [回车换行]

请求行实例：

Eg1：

```
GET /index.html HTTP/1.1
```

Eg2：

POST http://192.168.2.217:8080/index.jsp HTTP/1.1

HTTP 请求消息实例：

```
GET /hello.htm HTTP/1.1
Accept: */*
Accept-Language: zh-cn
Accept-Encoding: gzip, deflate
If-Modified-Since: Wed, 17 Oct 2007 02:15:55 GMT
If-None-Match: W/"158-1192587355000"
User-
```

```
Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)
 Host: 192.168.2.162:8080
 Connection: Keep-Alive
```

2.2.2 请求方法

HTTP 的请求方法包括如下几种：

- GET
- POST
- HEAD
- PUT
- DELETE
- OPTIONS
- TRACE
- CONNECT

2.3 HTTP 响应消息

2.3.1 响应消息格式

HTTP 响应消息的格式如下所示：

状态行

通用信息头|响应头|实体头

CRLF

实体内容

其中：状态行 = 版本号 [空格] 状态码 [空格] 原因 [回车换行]

状态行举例：

Eg1：

```
HTTP/1.0 200 OK
```

Eg2：

```
HTTP/1.1 400 Bad Request
```

HTTP 响应消息实例如下所示：

```
HTTP/1.1 200 OK
ETag: W/"158-1192590101000"
Last-Modified: Wed, 17 Oct 2007 03:01:41 GMT
Content-Type: text/html
Content-Length: 158
Date: Wed, 17 Oct 2007 03:01:59 GMT
Server: Apache-Coyote/1.1
```

2.3.2 http 的状态响应码

2.3.2.1 1**：请求收到，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

2.3.2.2 2**：操作成功收到，分析、接受

200——交易成功

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

2.3.2.3 3**：完成此请求必须进一步处理

300——请求的资源可在多处得到

301——删除请求数据

302——在其他地址发现了请求数据

303——建议客户访问其他 URL 或访问方式

304——客户端已经执行了 GET，但文件未变化

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

2.3.2.4 4**：请求包含一个错误语法或不能完成

400——错误请求，如语法错误

401——未授权

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

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——没有发现文件、查询或 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。

2.3.2.5 5**：服务器执行一个完全有效请求失败

HTTP 500 - 内部服务器错误

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

Error 501 - 未实现

HTTP 502 - 网关错误

2.4 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
```

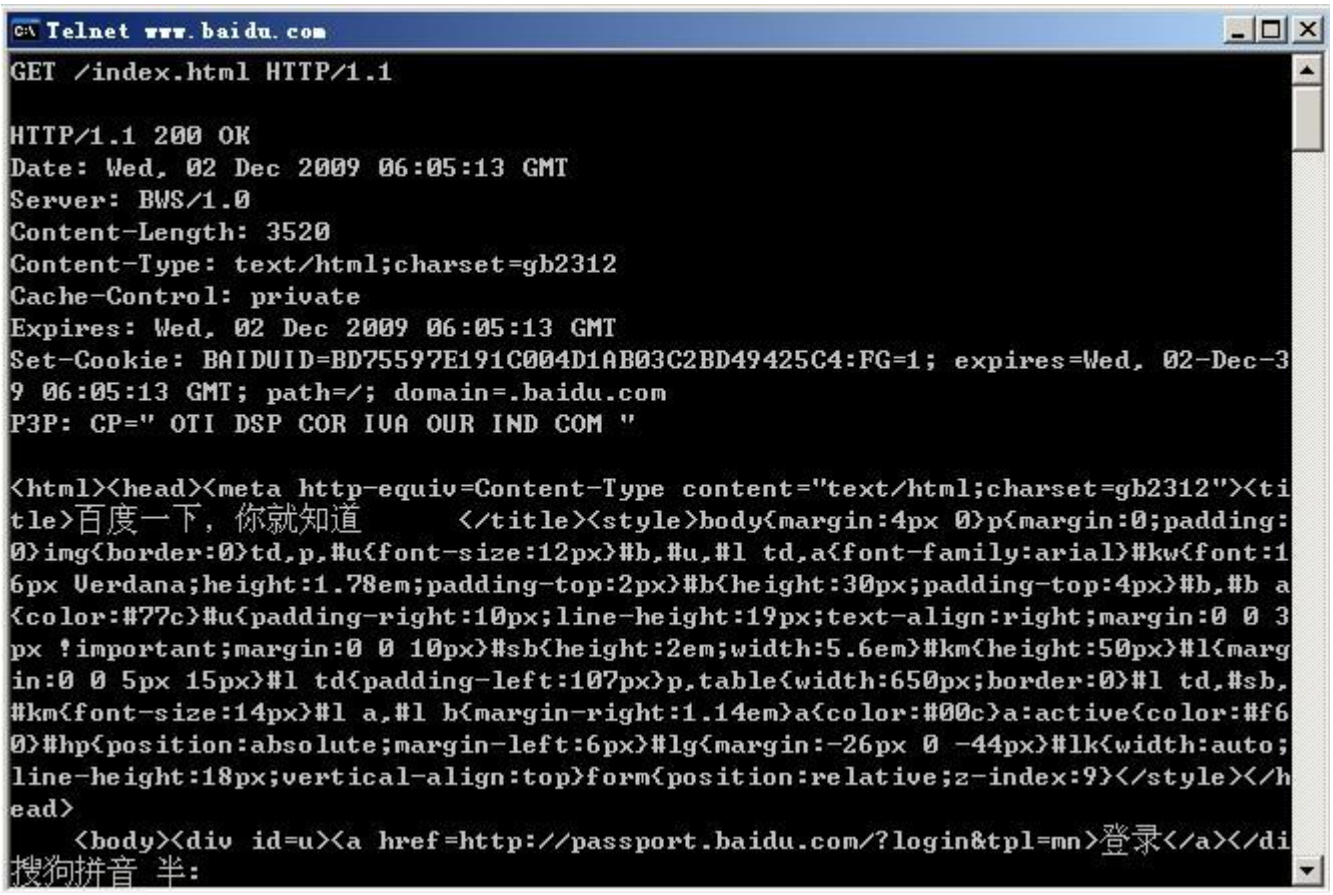
而后在窗口中按下“Ctrl+]”后按回车可让返回结果回显。

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。

可看到返回了 200 OK 的消息，如下图所示：



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
```

2.5 常用的请求方式

常用的请求方式是 GET 和 POST.

- GET 方式：是以实体的方式得到由请求 URI 所指定资源的信息，如果请求 URI 只是一个数据产生过程，那么最终要在响应实体中返回的是处理过程的结果所指向的资源，而不是处理过程的描述。
- POST 方式：用来向目的服务器发出请求，要求它接受被附在请求后的实体，并把它当作请求队列中请求 URI 所指定资源的附加新子项，Post 被设计成用统一的方法实现下列功能：

  1：对现有资源的解释；

  2：向电子公告栏、新闻组、邮件列表或类似讨论组发信息；

  3：提交数据块；

4：通过附加操作来扩展数据库 。

从上面描述可以看出，Get 是向服务器发索取数据的一种请求；而 Post 是向服务器提交数据的一种请求，要提交的数据位于信息头后面的实体中。

GET 与 POST 方法有以下区别：

（1） 在客户端，Get 方式在通过 URL 提交数据，数据在 URL 中可以看到；POST 方式，数据放置在 HTML HEADER 内提交。

（2） GET 方式提交的数据最多只能有 1024 字节，而 POST 则没有此限制。

（3） 安全性问题。正如在（1）中提到，使用 Get 的时候，参数会显示在地址栏上，而 Post 不会。所以，如果这些数据是中文数据而且是非敏感数据，那么使用 get；如果用户输入的数据不是中文字符而且包含敏感数据，那么还是使用 post 为好。

（4） 安全的和幂等的。所谓安全的意味着该操作用于获取信息而非修改信息。幂等的意味着对同一 URL 的多个请求应该返回同样的结果。完整的定义并不像看起来那样严格。换句话说，GET 请求一般不应产生副作用。从根本上讲，其目标是当用户打开一个链接时，她可以确信从自身的角度来看没有改变资源。比如，新闻站点的头版不断更新。虽然第二次请求会返回不同的一批新闻，该操作仍然被认为是安全的和幂等的，因为它总是返回当前的新闻。反之亦然。POST 请求就不那么轻松了。POST 表示可能改变服务器上的资源的请求。仍然以新闻站点为例，读者对文章的注解应该通

过 POST 请求实现，因为在注解提交之后站点已经不同了（比方说文章下面出现一条注解）。

2.6 请求头

HTTP 最常见的请求头如下：

- Accept：浏览器可接受的 MIME 类型；
- Accept-Charset：浏览器可接受的字符集；
- Accept-Encoding：浏览器能够进行解码的数据编码方式，比如 gzip。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；
- Accept-Language：浏览器所希望的语言种类，当服务器能够提供一种以上的语言版本时要用到；
- Authorization：授权信息，通常出现在对服务器发送的 WWW-Authenticate 头的应答中；
- Connection：表示是否需要持久连接。如果 Servlet 看到这里的值为“Keep-Alive”，或者看到请求使用的是 HTTP 1.1（HTTP 1.1 默认进行持久连接），它就可以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；

- Content-Length：表示请求消息正文的长度；
- Cookie：这是最重要的请求头信息之一；
- 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；
- Host：初始 URL 中的主机和端口；
- If-Modified-Since：只有当所请求的内容在指定的日期之后又经过修改才返回它，否则返回 304“Not Modified”应答；
- 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；
- 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。
- User-Agent：浏览器类型，如果 Servlet 返回的内容与浏览器类型有关则该值非常有用；
- 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。

2.7 响应头

HTTP 最常见的响应头如下所示：

- Allow：服务器支持哪些请求方法（如 GET、POST 等）；
- Content-Encoding：文档的编码（Encode）方法。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；
- Content-Length：表示内容长度。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；
- Content-Type： 表示后面的文档属于什么 MIME 类型。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 setContentTyep。 可在 web.xml 文件中配置扩展名和 MIME 类型的对应关系；

- Date：当前的 GMT 时间。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。
- Last-Modified：文档的最后改动时间。客户可以通过 If-Modified-Since 请求头提供一个日期，该请求将被视为一个条件 GET，只有改动时间迟于指定时间的文档才会返回，否则返回一个 304（Not Modified）状态。Last-Modified 也可用 setDateHeader 方法来设置；
- Location：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；
- 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。

2.8 实体头

实体头用坐实体内容的元信息，描述了实体内容的属性，包括实体信息类型，长度，压缩方法，最后一次修改时间，数据有效性等。

- Allow：GET,POST
- Content-Encoding：文档的编码（Encode）方法，例如：gzip，见“2.5 响应头”；
- Content-Language：内容的语言类型，例如：zh-cn；
- Content-Length：表示内容长度，eg：80，可参考“2.5 响应头”；
- Content-Location：表示客户应当到哪里去提取文档，例如：http://www.dfdf.org/dfdf.html，可参考“2.5 响应头”；
- Content-MD5：MD5 实体的一种 MD5 摘要，用作校验和。发送方和接受方都计算 MD5 摘要，接受方将其计算的值与此头标中传递的值进行比较。Eg1：Content-MD5: <base64 of 128 MD5 digest>。Eg2：dfdfdfdfdfdfdff==；

- Content-Range：随部分实体一同发送；标明被插入字节的低位与高位字节偏移，也标明此实体的总长度。Eg1：Content-Range: 1001-2000/5000，eg2：bytes 2543-4532/7898
- Content-Type：标明发送或者接收的实体的 MIME 类型。Eg：text/html; charset=GB2312　　主类型/子类型；
- Expires：为 0 证明不缓存；
- Last-Modified：WEB 服务器认为对象的最后修改时间，比如文件的最后修改时间，动态页面的最后产生时间等等。例如：Last-Modified：Tue, 06 May 2008 02:42:43 GMT.

2.8 扩展头

在 HTTP 消息中，也可以使用一些再 HTTP1.1 正式规范里没有定义的头字段，这些头字段统称为自定义的 HTTP 头或者扩展头，他们通常被当作是一种实体头处理。

现在流行的浏览器实际上都支持 Cookie,Set-Cookie,Refresh 和 Content-Disposition 等几个常用的扩展头字段。

- Refresh：1;url=http://www.dfdf.org　//过 1 秒跳转到指定位置；
- Content-Disposition：头字段,可参考“2.5 响应头”；
- Content-Type：WEB 服务器告诉浏览器自己响应的对象的类型。

  eg1：Content-Type：application/xml ；

eg2：applicaiton/octet-stream；

Content-Disposition：attachment; filename=aaa.zip。

附录：参考资料

《HTTP1.1 和 HTTP1.0 的区别》：

http://blog.csdn.net/yanghehong/archive/2009/05/28/4222594.aspx

《HTTP 请求（GET 和 POST 区别）和响应》：

http://www.blogjava.net/honeybee/articles/164008.html

《HTTP 请求头概述_百度知道》：

http://zhidao.baidu.com/question/32517427.html

《实体头和扩展头》：

http://www.cnblogs.com/tongzhiyong/archive/2008/03/16/1108776.html

3. 深入了解篇

3.1 Cookie 和 Session

Cookie 和 Session 都为了用来保存状态信息，都是保存客户端状态的机制，它们都是为了解决 HTTP 无状态的问题而所做的努力。

Session 可以用 Cookie 来实现，也可以用 URL 回写的机制来实现。用 Cookie 来实现的 Session 可以认为是对 Cookie 更高级的应用。

3.1.1 两者比较

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缚上这些 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。

3.1.2 Session 机制

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。

3.1.6 Session 的实现方式

3.1.6.1 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。

流程如下图所示：

3.1.6.2 使用 URL 回显来实现

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() 。

3.1.3 在 J2EE 项目中 Session 失效的几种情况

1）Session 超时：Session 在指定时间内失效，例如 30 分钟，若在 30 分钟内没有操作，则 Session 会失效，例如在 web.xml 中进行了如下设置：

```
<session-config>
    <session-timeout>30</session-timeout> //单位：分钟
</session-config>
```

2）使用 session.invalidate()明确的去掉 Session。

3.1.4 与 Cookie 相关的 HTTP 扩展头

1）Cookie：客户端将服务器设置的 Cookie 返回到服务器；

2）Set-Cookie：服务器向客户端设置 Cookie；

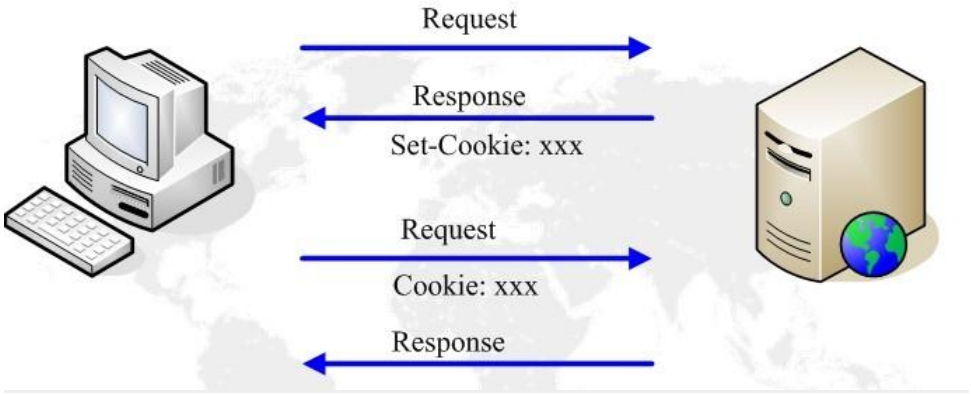
3）Cookie2 (RFC2965)）：客户端指示服务器支持 Cookie 的版本；

4）Set-Cookie2 (RFC2965)：服务器向客户端设置 Cookie。

3.1.5Cookie 的流程

服务器在响应消息中用 Set-Cookie 头将 Cookie 的内容回送给客户端，客户端在新的请求中将相同的内容携带在 Cookie 头中发送给服务器。从而实现会话的保持。

流程如下图所示：



3.2 缓存的实现原理

3.2.1 什么是 Web 缓存

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。

3.2.2 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。

3.2.3 与缓存相关的 HTTP 扩展消息头

- Expires：指示响应内容过期的时间，格林威治时间 GMT
- Cache-Control：更细致的控制缓存的内容
- Last-Modified：响应中资源最后一次修改的时间
- ETag：响应中资源的校验值，在服务器上某个时段是唯一标识的。
- Date：服务器的时间
- If-Modified-Since：客户端存取的该资源最后一次修改的时间，同 Last-Modified。
- If-None-Match：客户端存取的该资源的检验值，同 ETag。

3.2.4 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。常见流程如下图所示：

3.2.5 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。

3.3 断点续传和多线程下载的实现原理

- 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：

q 下载工具开启多个发出 HTTP 请求的线程；

q 每个 http 请求只请求资源文件的一部分：Content-Range: bytes 20000-40000/47000；

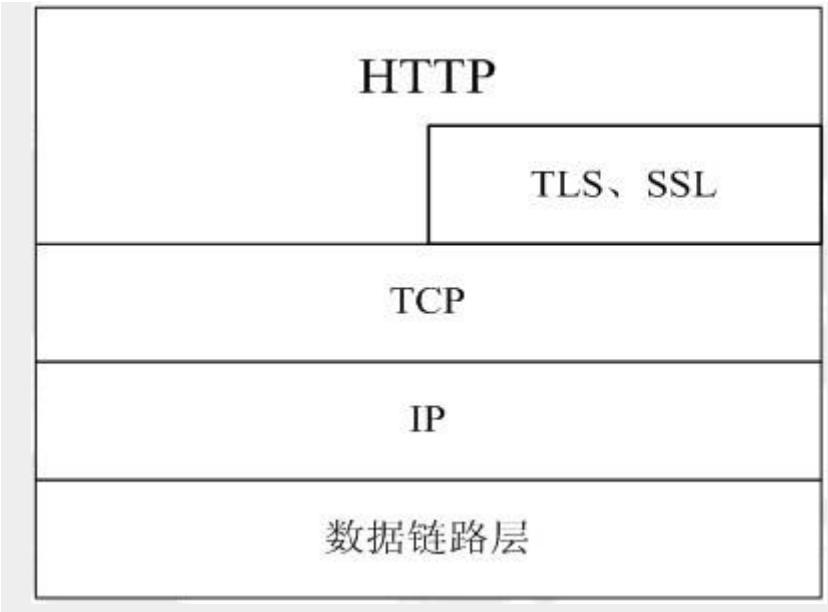
q 合并每个线程下载的文件。

3.4 https 通信过程

3.4.1 什么是 https

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。

见下图：



https 所用的端口号是 443。

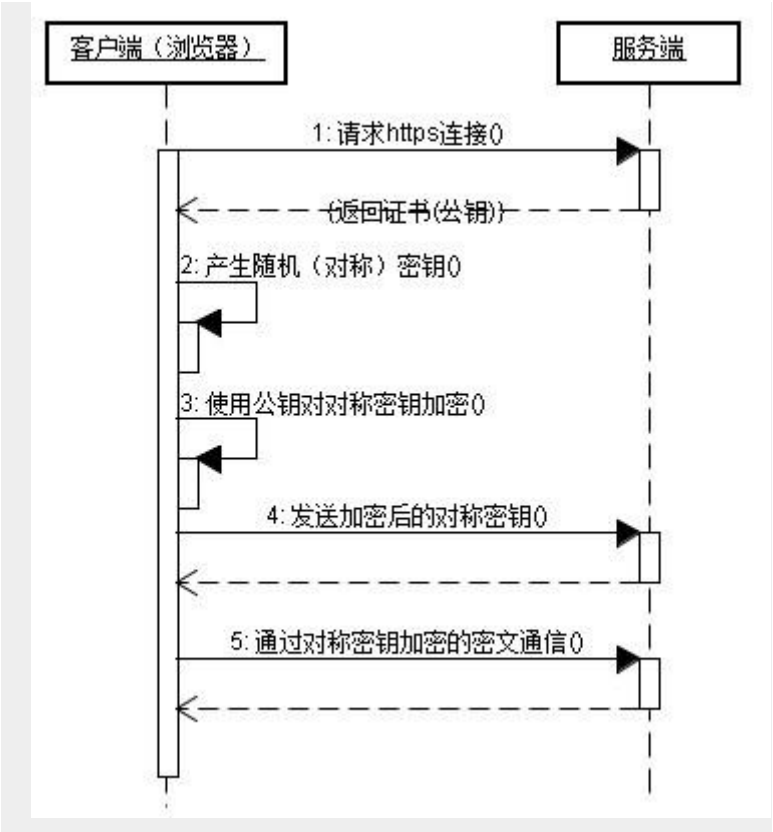
3.4.2 https 的实现原理

有两种基本的加解密算法类型：

1）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；

2）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（公钥加密需要私钥解密，私钥加密需要公钥解密），相对对称加密速度较慢，典型的非对称加密算法有 RSA、DSA 等。

下面看一下 https 的通信过程：



https 通信的优点：

1）客户端产生的密钥只有客户端和服务器端能得到；

2）加密的数据只有客户端和服务器端才能得到明文；

3）客户端到服务端的通信是安全的。

3.5 http 代理

3.5.1 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。

3.5.2 http 代理服务器的主要功能

主要功能如下：

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；

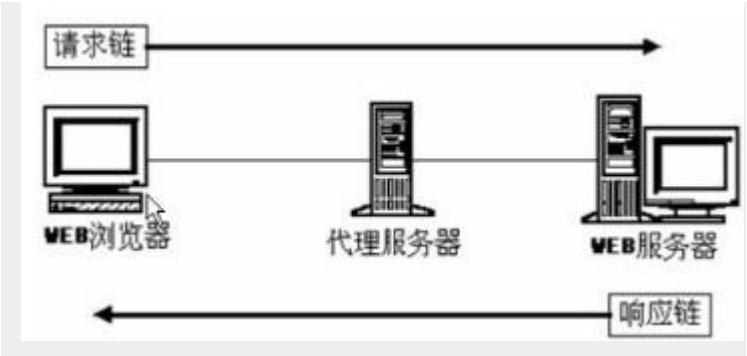
3）突破中国电信的 IP 封锁：中国电信用户有很多网站是被限制访问的，这种限制是人为的，不同 Serve 对地址的封锁是不同的。所以不能访问时可以换一个国 外的代理服务器试试；

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。

3.5.3 http 代理图示

http 代理的图示见下图：



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。

3.6 虚拟主机的实现

3.6.1 什么是虚拟主机

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。

3.6.2 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。

附录：参考资料

《理解 Cookie 和 Session 机制》：

http://sumongh.javaeye.com/blog/82498

《浅析 HTTP 协议》：

http://203.208.39.132/search?q=cache:CdXly_88gjIJ:www.cnblogs.com/gpcuster/archive/2009/05/25/1488749.html+http%E5%8

D%8F%E8%AE%AE+web%E7%BC%93%E5%AD%98&cd=27&hl=zh-CN&ct=clnk&gl=cn&st_usg=ALhdy2-vzOcP8XTG1h7lcRr2GJrkTbH2Cg

《http 代理_百度百科》：

http://baike.baidu.com/view/1159398.htm

《虚拟主机_百度百科》：

http://baike.baidu.com/view/7383.htm

《https_百度百科》：

http://baike.baidu.com/view/14121.htm