**一14一 一14一 一12一 一11一** 一10.5一 一10.5一 一10一

**目 录**

[1. ------------资料------------ 3](#_Toc23588)

[2. 刘望舒的精简讲解 3](#_Toc27619)

[2.1. 请求报文 3](#_Toc16674)

[2.1.1. 请求行 3](#_Toc17246)

[2.1.2. 请求报头 3](#_Toc18544)

[2.1.3. 请求数据 3](#_Toc8407)

[2.2. 响应报文 3](#_Toc32760)

[2.2.1. 状态行 4](#_Toc3076)

[2.2.2. 响应报头 4](#_Toc1093)

[2.2.3. 响应正文 4](#_Toc15428)

[2.3. 报头 4](#_Toc4145)

[2.3.1. 通用报头 4](#_Toc17948)

[2.3.2. 实体报头 4](#_Toc24315)

[2.4. 应用举例 5](#_Toc13679)

[2.4.1. 请求报文 5](#_Toc7182)

[2.4.2. 响应报文 5](#_Toc687)

[2.4.3. 项目中log打出来的 6](#_Toc13825)

[3. 网络请求 6](#_Toc601)

[3.1. HTTP请求&响应 6](#_Toc22169)

[3.1.1. 请求包 6](#_Toc10609)

[3.1.2. 响应包 7](#_Toc4049)

[3.2. Get&Post 8](#_Toc5862)

[3.2.1. Get方式 8](#_Toc3640)

[3.2.2. Post方式 8](#_Toc4351)

[3.3. 四种方式请求网络 8](#_Toc25379)

[3.3.1. 比较 8](#_Toc24382)

[3.3.2. HttpUrlConnection发送GET请求 9](#_Toc19843)

[3.3.3. HttpUrlConnection发送POST请求 9](#_Toc27749)

[3.3.4. HttpClient发送GET请求 10](#_Toc24161)

[3.3.5. HttpClient发送POST请求 10](#_Toc14708)

[3.4. HTTP缓存机制 10](#_Toc32339)

[4. Http和Https的区别 11](#_Toc12771)

[4.1. 什么是 HTTPS 11](#_Toc17244)

[4.2. HTTPS 和 HTTP 的区别 12](#_Toc12902)

[4.3. 信任主机的问题 12](#_Toc17179)

[4.4. SSL协议提供的服务主要有哪些 12](#_Toc24305)

[4.5. SSL 证书种类 12](#_Toc11069)

[5. ------------图解HTTP书------------ 12](#_Toc2532)

[6. TCP/IP协议族 12](#_Toc31356)

[6.1. 缩写 12](#_Toc15744)

[6.2. 协议族 13](#_Toc18110)

[6.3. TCP/IP协议分为四层 13](#_Toc16916)

[6.4. 与HTTP密切相关的协议：IP、TCP和DNS 15](#_Toc8467)

[6.4.1. （网络层）负责传输的IP协议 15](#_Toc3715)

[6.4.2. （传输层）确保可靠性的 TCP 协议 16](#_Toc28314)

[6.4.3. （应用层）负责域名解析的 DNS 服务 16](#_Toc15892)

[6.5. 各种协议与 HTTP 协议的关系 18](#_Toc1430)

[6.6. URI 和 URL 18](#_Toc15790)

[6.6.1. 统一资源标识符 18](#_Toc6209)

[6.6.2. URI 格式 19](#_Toc13282)

[7. 简单的HTTP协议 19](#_Toc799)

[7.1. HTTP 协议用于客户端和服务器端之间的通信 19](#_Toc637)

[7.2. 通过请求和响应的交换达成通信 20](#_Toc721)

[7.2.1. 请求报文 20](#_Toc30073)

[7.2.2. 响应报文 21](#_Toc13767)

[7.3. HTTP 是不保存状态的协议 21](#_Toc4729)

[7.4. 请求 URI 定位资源 21](#_Toc12354)

[7.5. 告知服务器意图的 HTTP 方法 22](#_Toc21821)

[7.5.1. GET：获取资源 22](#_Toc23927)

[7.5.2. POST：传输实体主体 22](#_Toc18615)

[7.5.3. PUT：传输文件 23](#_Toc32480)

[7.5.4. HEAD：获得报文首部 23](#_Toc10649)

[7.5.5. DELETE：删除文件 23](#_Toc25339)

[7.5.6. OPTIONS：询问支持的方法 24](#_Toc26273)

[7.5.7. TRACE：追踪路径 24](#_Toc30463)

[7.5.8. CONNECT：要求用隧道协议连接代理 25](#_Toc13287)

[7.6. 使用方法下达命令 25](#_Toc13144)

[7.7. 持久连接节省通信量 26](#_Toc20984)

[7.7.1. 短暂连接 26](#_Toc8426)

[7.7.2. 持久连接 26](#_Toc28304)

[7.7.3. 管线化 27](#_Toc25952)

[7.8. 使用 Cookie 的状态管理 27](#_Toc29722)

[8. HTTP报文内的HTTP信息 28](#_Toc30492)

[8.1. HTTP 报文 28](#_Toc1652)

[8.2. 请求报文及响应报文的结构 29](#_Toc23939)

[8.3. 编码提升传输速率 30](#_Toc7809)

[8.3.1. 报文主体和实体主体的差异 30](#_Toc18611)

[8.3.2. 压缩传输的内容编码 30](#_Toc908)

[8.3.3. 分割发送的分块传输编码 30](#_Toc25576)

[8.4. 发送多种数据的多部分对象集合 31](#_Toc6285)

[8.5. 获取部分内容的范围请求 32](#_Toc8225)

[8.6. 内容协商返回最合适的内容 32](#_Toc25805)

[9. 返回结果的HTTP状态码 33](#_Toc24765)

[9.1. 状态码告知从服务器端返回的请求结果 33](#_Toc14979)

[9.2. 2XX成功 33](#_Toc549)

[9.2.1. 200 OK 33](#_Toc13136)

[9.2.2. 204 No Content 34](#_Toc13003)

[9.2.3. 206 Partial Content 34](#_Toc14805)

[9.3. 3XX重定向 34](#_Toc6394)

[9.3.1. 301 Moved Permanently 34](#_Toc20657)

[9.3.2. 302 Found 34](#_Toc21581)

[9.3.3. 303 See Other 35](#_Toc2724)

[9.3.4. 304 Not Modified 35](#_Toc8142)

[9.3.5. 307 Temporary Redirect 35](#_Toc1205)

[9.4. 4XX 客户端错误 35](#_Toc16898)

[9.4.1. 400 Bad Request 35](#_Toc31363)

[9.4.2. 401 Unauthorized 36](#_Toc8821)

[9.4.3. 403 Forbidden 36](#_Toc3631)

[9.4.4. 404 Not Found 36](#_Toc22657)

[9.5. 5XX 服务器错误 36](#_Toc26750)

[9.5.1. 500 Internal Server Error 37](#_Toc25951)

[9.5.2. 503 Service Unavailable 37](#_Toc15140)

[9.6. 状态码和状况的不一致 37](#_Toc7117)

[10. 与HTTP协作的Web服务器 37](#_Toc15963)

[10.1. 用单台虚拟主机实现多个域名 37](#_Toc2503)

[10.2. 通信数据转发程序：代理、网关、隧道 38](#_Toc19501)

[10.2.1. 代理 38](#_Toc7465)

[10.2.2. 网关 39](#_Toc3871)

[10.2.3. 隧道 39](#_Toc16942)

[10.3. 保存资源的缓存 39](#_Toc10013)

[10.3.1. 缓存的有效期限 40](#_Toc29485)

[10.3.2. 客户端的缓存 40](#_Toc32468)

[11. HTTP首部 41](#_Toc11016)

[11.1. HTTP 报文首部 41](#_Toc7396)

[11.2. HTTP 首部字段 42](#_Toc30746)

[11.2.1. HTTP 首部字段传递重要信息 42](#_Toc22410)

[11.2.2. HTTP 首部字段结构 42](#_Toc7093)

[11.2.3. 4 种 HTTP 首部字段类型 42](#_Toc18099)

[11.2.4. HTTP/1.1 首部字段一览 43](#_Toc17003)

[11.2.5. 非 HTTP/1.1 首部字段 44](#_Toc28572)

[11.2.6. End-to-end首部和Hop-by-hop首部 44](#_Toc946)

[11.3. 通用首部字段 44](#_Toc15969)

[11.4. 请求首部字段 44](#_Toc2988)

[11.5. 响应首部字段 45](#_Toc17716)

[11.6. 实体首部字段 45](#_Toc10257)

[11.7. 为 Cookie 服务的首部字段 45](#_Toc13838)

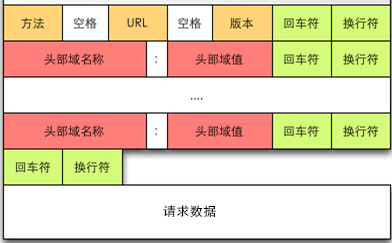
[11.8. 其他首部字段 45](#_Toc31119)

# ------------资料------------

# 刘望舒的精简讲解

## 请求报文

通常来说一个HTTP请求报文由请求行、请求报头、空行、和请求数据4个部分组成。



### 请求行

请求行由：请求方法，URL字段和HTTP协议的版本组成，例如我去访问我的CSDN博客地址请求行是：

GET http://blog.csdn.net/itachi85 HTTP/1.1

### 请求报头

在请求行之后会有0个或者多个请求报头，每个请求报头都包含一个名字和一个值，它们之间用“：”分割。请求报头通知服务器关于客户端求求的信息，典型的请求头有：

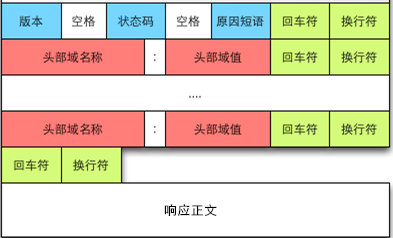
* Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机
* User-Agent：发送请求的浏览器类型、操作系统等信息
* Accept：客户端可识别的内容类型列表，用于指定客户端接收那些类型的信息
* Accept-Encoding：客户端可识别的数据编码
* Accept-Language：表示浏览器所支持的语言类型
* Connection：允许客户端和服务器指定与请求/响应连接有关的选项，例如这是为Keep-Alive则表示保持连接。
* Transfer-Encoding：告知接收端为了保证报文的可靠传输，对报文采用了什么编码方式。

### 请求数据

请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合，与请求数据相关的最常用的请求头是Content-Type和Content-Length。

## 响应报文

HTTP的响应报文由状态行、响应报头、空行、响应正文4个部分组成



### 状态行

状态行由：服务器HTTP协议的版本、服务器发回的响应状态代码、状态代码的文本描述组成，例如访问我的CSDN博客地址响应的状态行是：

HTTP/1.1 200 OK

### 响应报头

在状态行之后会有0个或者多个响应报头，每个请求报头都包含一个名字和一个值，它们之间用“：”分割。响应报头用于服务器传递自身信息的响应，常见的响应报头：

* Location：用于重定向接受者到一个新的位置，常用在更换域名的时候
* Server：包含可服务器用来处理请求的系统信息，与User-Agent请求报头是相对应的

### 响应正文

响应正文是服务器返回的资源的内容

## 报头

除了各自使用的请求报头和响应报头，还有公用的通用报头和实体报头

### 通用报头

常见的通用报头为：

* Date：表示消息产生的日期和时间
* Connection：允许发送指定连接的选项，例如指定连接是连续的，或者指定“close”选项，通知服务器，在响应完成后，关闭连接
* Cache-Control：用于指定缓存指令，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（一个消息的缓存指令不会影响另一个消息处理的缓存机制）

### 实体报头

实体报头用来定于被传送资源的信息，请求和响应消息都可以传送一个实体，常见的实体报头为：

* Content-Type：发送给接收者的实体正文的媒体类型
* Content-Lenght：实体正文的长度
* Content-Language：描述资源所用的自然语言，没有设置则该选项则认为实体内容将提供给所有的语言阅读
* Content-Encoding：实体报头被用作媒体类型的修饰符，它的值指示了已经被应用到实体正文的附加内容的编码，因而要获得Content-Type报头域中所引用的媒体类型，必须采用相应的解码机制。
* Last-Modified：实体报头用于指示资源的最后修改日期和时间
* Expires：实体报头给出响应过期的日期和时间

## 应用举例

### 请求报文

**例子一**

GET http://blog.csdn.net/itachi85 HTTP/1.1 //请求行

Host: blog.csdn.net //请求报头

Connection: keep-alive

Cache-Control: max-age=0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8

Upgrade-Insecure-Requests: 1

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/47.0.2526.80 Safari/537.36 QQBrowser/9.3.6872.400

Accept-Encoding: gzip, deflate, sdch

Accept-Language: zh-CN,zh;q=0.8

Cookie: bdshare\_firstime=1443768140949; uuid\_tt\_dd=5028529250430960147\_20151002;

...省略

很容易看出访问的是我的博客地址http://blog.csdn.net/itachi85，请求的方法是GET，所以并没有请求数据。

**例子二**

POST http://patapi.shoujng.com/api/getVersion HTTP/1.1 //请求行

Content-Length: 226 //请求报头

Content-Type: application/x-www-form-urlencoded

Host: patientapi.shoujikanbing.com

Connection: Keep-Alive

User-Agent: Mozilla/5.0 (Linux; U; Android 4.4.4; zh-cn; MI NOTE LTE Build/KTU84P) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1

Accept-Encoding: gzip

//不能省略的空格，下面是请求数据

clientversion=2\_2.0.0&time=1459069342&appId=android&channel=hjwa5b7fbc9&token=b1919445ed97f2f4ccf03058&deviceId=869436020220717&type=2&version=2.0.0

### 响应报文

**例子一**

HTTP/1.1 200 OK //状态行

Server: openresty //响应报头

Date: Sun, 27 Mar 2016 08:26:54 GMT

Content-Type: text/html; charset=utf-8

Transfer-Encoding: chunked

Connection: keep-alive

Keep-Alive: timeout=20

Vary: Accept-Encoding

Cache-Control: private

X-Powered-By: PHP 5.4.28

Content-Encoding: gzip

//不能省略的空格

{"code":"0","msg":"成功","moduleId":"","result":{"total":13,"rows":[{"count":2,"name":"外勤的房贷的房贷到事发地时到事发获取","ratio":"15.38%"},{"count":4,"name":"大范甘迪发给对方的范德萨发阿达第三方大说的发送大法师访","ratio":"30.76%"}],"allCount":13},"errStep":"","serviceName":"API分析表"}

...省略

**例子二**

HTTP/1.1 200 OK //状态行

Server: nginx //响应报头

Date: Sun, 27 Mar 2016 09:02:20 GMT

Content-Type: text/html;charset=utf-8

Transfer-Encoding: chunked

Connection: keep-alive

Vary: Accept-Encoding

Set-Cookie: sessionId=0d1cee1f31924955b7fbc9; expires=Mon, 28-Mar-2016 09:02:20 GMT; Max-Age=86400; path=/; domain=.shoujikanbing.com

Set-Cookie: PHPSESSID=0d1cee1f55b7fbc9; path=/; domain=.shoujikanbing.com

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

Content-Encoding: gzip

//不能省略的空格

17f //实体报文编码格式为gzip所以显示在这里的响应数据是乱码

­ mP N @ "E ? ­ n m 1

w ( HL (1^ P nK E\_x0012\_ ѷ93'3gN\_x0019\_\_x0010\_LH

...省略

### 项目中log打出来的

**请求：**

--> POST https://fitnest.wkllme.com/wkllme-vs-api/api/v2/chart/analysis http/1.1

Content-Type: application/json; charset=UTF-8

Content-Length: 68

{"filter":{"chartType":"0","statusType":"0"},"page":"1","rows":"20"}

--> END POST (68-byte body)

**响应：**

<-- 200 https://fitnest.wkllme.com/wkllme-vs-api/api/v2/chart/analysis (596ms)

Server: nginx

Date: Tue, 28 Nov 2017 02:08:29 GMT

Content-Type: application/json;charset=utf-8

Content-Length: 492

Connection: keep-alive

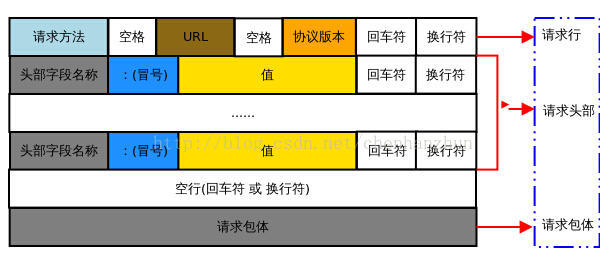
{"code":"0","msg":"成功","moduleId":"","result":{"total":13,"rows":[{"count":2,"name":"外勤的房贷的房贷到事发地时到事发获取","ratio":"15.38%"},{"count":4,"name":"大范甘迪发给对方的范德萨发阿达第三方大说的发送大法师访","ratio":"30.76%"}],"allCount":13},"errStep":"","serviceName":"API分析表"}<-- END HTTP (492-byte body)

# 网络请求

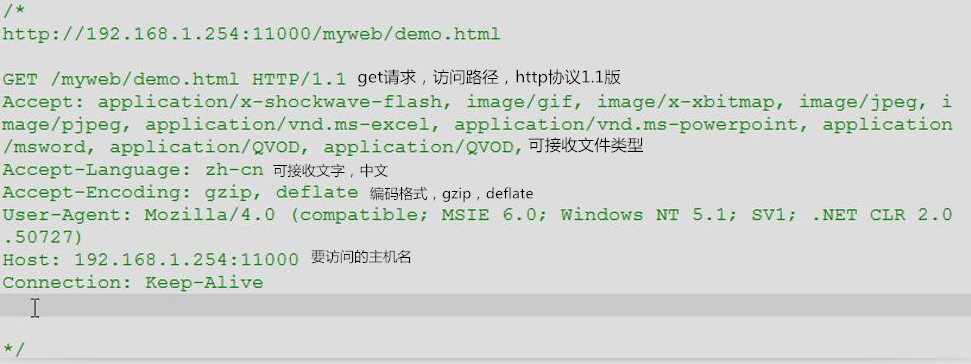
## HTTP请求&响应

### 请求包

说说Http请求包的结构。一次请求就是向目标服务器发送一串文本。HTTP请求包结构



例子：



POST /meme.php/home/user/login HTTP/1.1

Host: 114.215.86.90

Cache-Control: no-cache

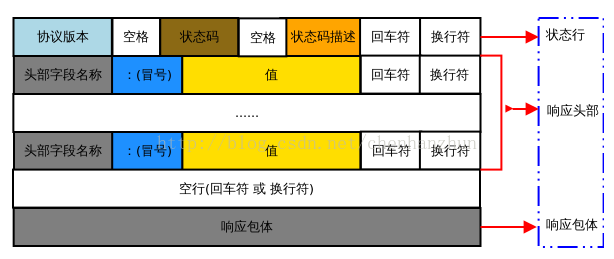
Postman-Token: bd243d6b-da03-902f-0a2c-8e9377f6f6ed

Content-Type: application/x-www-form-urlencoded

tel=13637829200&password=123456

### 响应包

请求了就会收到响应包(如果对面存在HTTP服务器)，HTTP响应包结构



例子：

HTTP/1.1 200 OK

Date: Sat, 02 Jan 2016 13:20:55 GMT

Server: Apache/2.4.6 (CentOS) PHP/5.6.14

X-Powered-By: PHP/5.6.14

Content-Length: 78

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: application/json; charset=utf-8

{"status":202,"info":"\u6b64\u7528\u6237\u4e0d\u5b58\u5728\uff01","data":null}

## Get&Post

可以看出虽然Post与Get本意一个是表单提交一个是请求页面，但本质并没有什么区别。下面说说参数在这2者的位置。

### Get方式

在url中填写参数：http://xxxx.xx.com/xx.php?params1=value1&params2=value2

甚至使用路由：http://xxxx.xx.com/xxx/value1/value2/value3

这些就是web服务器框架的事了。

### Post方式

参数是经过编码放在请求体中的。编码包括x-www-form-urlencoded 与 form-data。

**x-www-form-urlencoded的编码方式是这样：**

tel=13637829200&password=123456

**form-data的编码方式是这样：**

----WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="tel"

13637829200

----WebKitFormBoundary7MA4YWxkTrZu0gW

Content-Disposition: form-data; name="password"

123456

----WebKitFormBoundary7MA4YWxkTrZu0gW

**x-www-form-urlencoded的优越性就很明显了。**

不过x-www-form-urlencoded只能传键值对，但是form-data可以传二进制。

因为url是存在于请求行中的，所以Get与Post区别本质就是参数是放在请求行中还是放在请求体中，当然无论用哪种都能放在请求头中。一般在请求头中放一些发送端的常量。

**有人说：**

Get是明文，Post隐藏？移动端不是浏览器,不用https全都是明文。

Get传递数据上限XXX？胡说。有限制的是浏览器中的url长度，不是Http协议，移动端请求无影响。Http服务器部分有限制的设置一下即可。

Get中文需要编码？是真的...要注意。URLEncoder.encode(params, "gbk");

还是建议用post规范参数传递方式。并没有什么更优秀，只是大家都这样社会更和谐。

**上面说的是请求。下面说响应。**

请求是键值对，但返回数据我们常用Json。对于内存中的结构数据，肯定要用数据描述语言将对象序列化成文本，再用Http传递,接收端并从文本还原成结构数据。对象(服务器)<-->文本(Http传输)<-->对象(移动端) 。服务器返回的数据大部分都是复杂的结构数据，所以Json最适合。

## 四种方式请求网络

### 比较

在Android 2.2版本之前，HttpClient拥有较少的bug，因此使用它是最好的选择。而在Android 2.3版本及以后，HttpURLConnection则是最佳的选择。它的API简单，体积较小，因而非常适用于Android项目。压缩和缓存机制可以有效地减少网络访问的流量，在提升速度和省电方面也起到了较大的作用。对于新的应用程序应该更加偏向于使用HttpURLConnection，因为在以后的工作当中我们也会将更多的时间放在优化HttpURLConnection上面。

### HttpUrlConnection发送GET请求

**public** **class** **HttpUtil** {

**public** **static** **String** **requestByGet**(**String** name, **String** age){

**String** result = "";

**String** path = "http://www.baidu.com/?username=" + name + "&age=" + age;

**try** {

**URL** url = **new** URL(path);//新建一个URL对象

//打开一个HttpClientConnection连接

**HttpURLConnection** urlConn = (**HttpURLConnection**) url.openConnection();

urlConn.setConnectTimeout(10 \* 1000);//设置连接超时时间

urlConn.setReadTimeout(5 \* 1000);//设置读取超时时间

urlConn.setRequestMethod("GET");//设置请求方式

**if** (urlConn.getResponseCode() == 200) {//判断请求是否成功

**InputStream** is = urlConn.getInputStream();//获取输入流

result = streamToString(is);//流转换成字符串

}**else** {

}

} **catch** (**Exception** e) {

}

**return** result;

}

### HttpUrlConnection发送POST请求

**public** **static** **String** **requestByPost**(**String** path, **String** data){

**String** result = "";

**try** {

**URL** url = **new** URL(path);//新建一个URL对象

//打开一个HttpClientConnection连接

**HttpURLConnection** urlConn = (**HttpURLConnection**) url.openConnection();

urlConn.setConnectTimeout(5 \* 1000);//设置连接超时时间

urlConn.setReadTimeout(5 \* 1000);//设置读取超时时间

urlConn.setRequestMethod("POST");//设置请求方式

urlConn.setDoInput(**true**); // 设置允许输入

urlConn.setDoOutput(**true**); //必须设置允许输出

urlConn.setUseCaches(**false**); // Post请求不能使用缓存，因为易出错

// 配置请求Content-Type (默认的, 可以不写)

//urlConn.setRequestProperty("Content-Type", "application/x-www-form-urlencode");

/\*\*参数是这么配的，String data = "name=" + name + "&age=" + age;例子：String data = "username=" + URLEncoder.encode(name, "UTF-8") + "&age=" + URLEncoder.encode(age, "UTF-8"); \*/

**byte**[] databyte = data.getBytes();//将String转化为byte

**OutputStream** os = urlConn.getOutputStream(); // 获取输出流

os.write(databyte); // 发送请求参数 ,将数据写给服务器

os.flush();

os.close();

**if** (urlConn.getResponseCode() == 200) {//判断请求是否成功

**InputStream** is = urlConn.getInputStream();//获取输入流

result = streamToString(is);//流转换成字符串

}**else** {

}

} **catch** (**Exception** e) {

}

**return** result;

}

### HttpClient发送GET请求

**public** **static** **String** **requestByHttpGet**(**String** name, **String** age){

**String** result = "";

**HttpClient** client = **new** DefaultHttpClient();//1.打开一个浏览器

**String** path = "http://www.baidu.com" + "username=" + **URLEncoder**.*encode*(name, "UTF-8") + "&age=" + **URLEncoder**.*encode*(age, "UTF-8");//2.输入地址

**HttpGet** httpGet = **new** HttpGet(path);//新建httpGet 对象，放入path

**HttpResponse** response = client.execute(httpGet);//3，敲回车，请求

**int** code = response.getStatusLine().getStatusCode();

**if** (code == 200) {

//InputStream is = httpResponse.getEntity().getContent(); // 获取实体内容

//result = streamToString(is); // 流转字符串

result = **EntityUtils**.*toString*(httpResponse.getEntity(), "UTF-8");//简便方法

}

**return** result;

}

### HttpClient发送POST请求

**public** **static** **String** **requestByHttpPost**(**String** name, **String** age){

**String** result = "";

**String** path = "http://www.baidu.com/";

**try** {

HttpClient httpClient = **new** DefaultHttpClient();//打开一个浏览器

**HttpPost** httpPost = **new** HttpPost(path);//新建HttpPost对象，放入path

//Post参数，放入NameValuePair中

List<NameValuePair> params = **new** ArrayList<NameValuePair>();

params.add(**new** BasicNameValuePair("username", name));

params.add(**new** BasicNameValuePair("age", age));

HttpEntity entity = **new** UrlEncodedFormEntity(params, "UTF-8");//包装成请求实体

httpPost.setEntity(entity);//设置实体

HttpResponse httpResponse = httpClient.execute(httpPost);//敲回车，请求

**if** (httpResponse.getStatusLine().getStatusCode() == 200) {

result = **EntityUtils**.*toString*(httpResponse.getEntity(), "UTF-8");

}

} **catch** (**Exception** e) {

}

**return** result;

}

## HTTP缓存机制

**缓存对于移动端是非常重要的存在**

1、减少请求次数，减小服务器压力.

2、本地数据读取速度更快，让页面不会空白几百毫秒。

3、在无网络的情况下提供数据。

缓存一般由服务器控制(通过某些方式可以本地控制缓存，比如向过滤器添加缓存控制信息)。通过在请求头添加下面几个字端：

**Request**

If-Modified-Since: Sun, 03 Jan 2016 03:47:16 GMT 缓存文件的最后修改时间。

If-None-Match: "3415g77s19tc3:0" 缓存文件的Etag(Hash)值

Cache-Control: no-cache 不使用缓存

Pragma: no-cache 不使用缓存

**Response**

Cache-Control: public 响应被共有缓存，移动端无用

Cache-Control: private 响应被私有缓存，移动端无用

Cache-Control:no-cache 不缓存

Cache-Control:no-store 不缓存

Cache-Control: max-age=60 60秒之后缓存过期（相对时间）

Date: Sun, 03 Jan 2016 04:07:01 GMT 当前response发送的时间

Expires: Sun, 03 Jan 2016 07:07:01 GMT 缓存过期的时间（绝对时间）

Last-Modified: Sun, 03 Jan 2016 04:07:01 GMT 服务器端文件的最后修改时间

ETag: "3415g77s19tc3:0" 服务器端文件的Etag[Hash]值

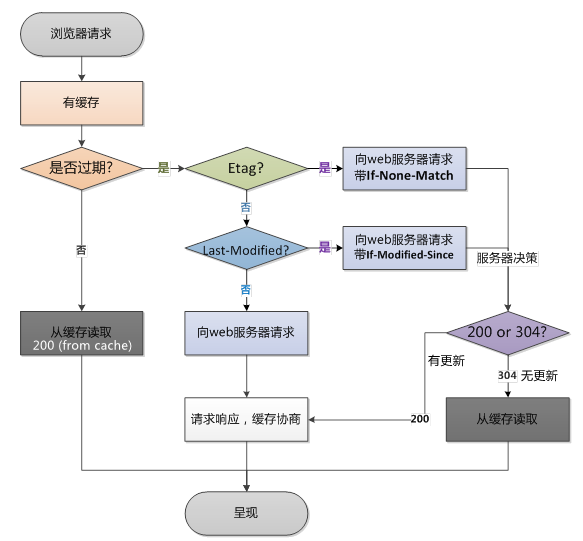
**步奏**

正式使用时按需求也许只包含其中部分字段。

客户端要根据这些信息储存这次请求信息。

然后在客户端发起请求的时候要检查缓存。遵循下面步骤：

注意服务器返回304意思是数据没有变动滚去读缓存信息。



# Http和Https的区别

## 什么是 HTTPS

HTTPS（全称：Hyper Text 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就需要SSL。它使用了HTTP，但HTTPS存在不同于HTTP的默认端口及一个加密/身份验证层（在HTTP与TCP之间）。现在它被广泛用于万维网上安全敏感的通讯，例如交易支付方面。它最初由Netscape开发内置于其浏览器中，用于对数据进行加密和解密操作，并返回网络上传送的结果。

HTTPS使用端口443，而不是像HTTP那样适用端口80来和TCP/IP 进行通信。SSL使用40位关键字作为RC4流加密算法，这对于商业信息的加密是合适的。HTTPS和SSL支持使用X.509数字认证，如果需要的话用户可以确认发送者是谁。

**HTTPS主要作用分为两种：**

* 一种是建立一个信息安全通道，来保证数据传输的安全。
* 另一种就是确认网站的真实性，凡是使用https的网站，都可以通过点击浏览器地址栏的锁头标志来查看网站认证之后的真实信息。也可以通过CA 机构颁发的安全签章来查询。

## HTTPS 和 HTTP 的区别

* 超文本传输协议HTTP协议被用于在Web浏览器和服务器之间传递信息。
* HTTP协议以明文方式发送内容，不提供任何方式的数据加密，如果攻击者截取了Web浏览器和网站服务器之间的传输报文，就可以直接读懂其中的信息，因此HTTP协议不适合传输一些敏感信息，比如信用卡号、密码等。
* 为了解决HTTP协议的这一缺陷，需要使用另一种协议：安全套接字层超文本传输协议HTTPS。为了数据传输的安全，HTTPS在HTTP的基础上加入了SSL协议，SSL依靠证书来验证服务器的身份，并为浏览器和服务器之间的通信加密。

**HTTPS和HTTP的区别主要为以下五点：**

1. https 用的 443 端口， http 用的 80 端口
2. https协议需要到ca申请证书，一般免费证书很少，需要交费。
3. http是超文本传输协议，信息是明文传输，https 则是具有安全性的ssl加密传输协议。
4. http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
5. http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

## 信任主机的问题

* 采用 https 的服务器必须从CA（Certificate Authority）申请一个用于证明服务器用途类型的证书。
* 该证书只有用于对应的服务器的时候，客户端才信任此主机。所以所有的银行系统网站，关键部分应用都是https 的。
* 客户通过信任该证书，从而信任了该主机。其实这样做效率很低，但是银行更侧重安全。
* 这一点对局域网对内提供服务处的服务器没有任何意义。局域网中的服务器，采用的证书不管是自己发布的还是从公众的地方发布的，其客户端都是自己人，所以该局域网中的客户端也就肯定信任该服务器。

## SSL协议提供的服务主要有哪些

* 认证用户和服务器，确保数据发送到正确的客户机和服务器。
* 加密数据以防止数据中途被窃取
* 维护数据的完整性，确保数据在传输过正中不被改变。

## SSL 证书种类

CFCA，GlobalSign，VeriSign ，Geotrust ，Thawte

域名型 https 证书（DVSSL）：信任等级一般，只需验证网站的真实性便可颁发证书保护网站；

企业型 https 证书（OVSSL）：信任等级强，须要验证企业的身份，审核严格，安全性更高；

增强型 https 证书（EVSSL）：信任等级最高，一般用于银行证券等金融机构，审核严格，安全性最高，同时可以激活绿色网址栏。

# ------------图解HTTP书------------

# TCP/IP协议族

## 缩写

HTTP：[超文本传输协议](http://baike.baidu.com/item/%E8%B6%85%E6%96%87%E6%9C%AC%E4%BC%A0%E8%BE%93%E5%8D%8F%E8%AE%AE" \t "http://baike.baidu.com/_blank)---HyperText Transfer Protocol

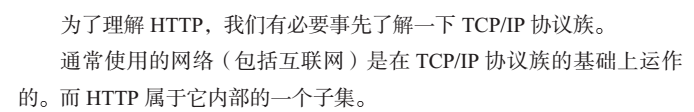
HTML：超文本标记语言---HyperText Markup Language

SGML：标准通用标记语言---Standard Generalized Markup language

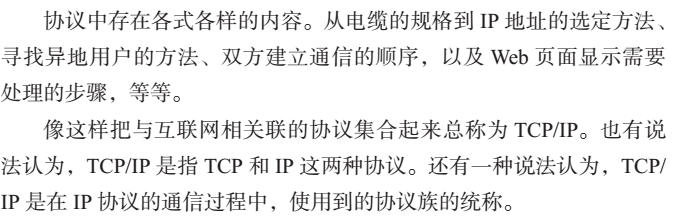
TCP：传输控制协议---Transmission Control Protocol

IP：网络之间互连的协议---Internet Protocol

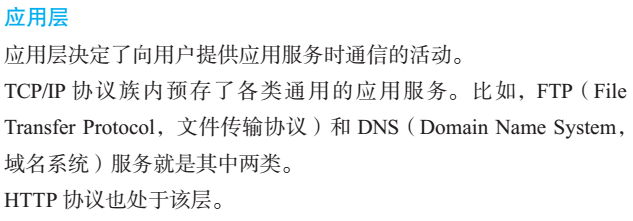
## 协议族

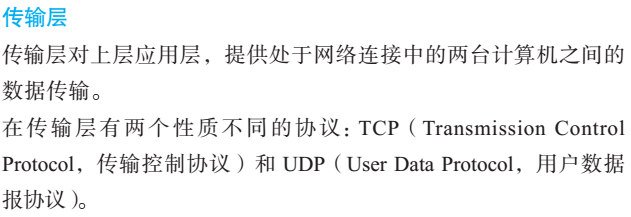


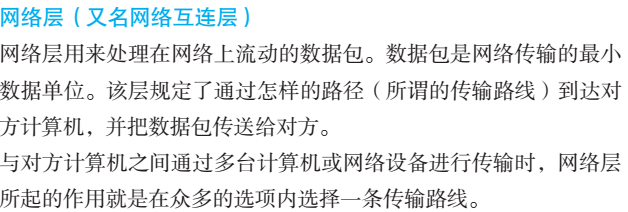


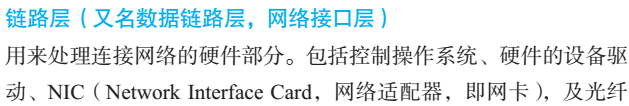


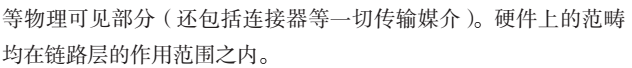
## TCP/IP协议分为四层

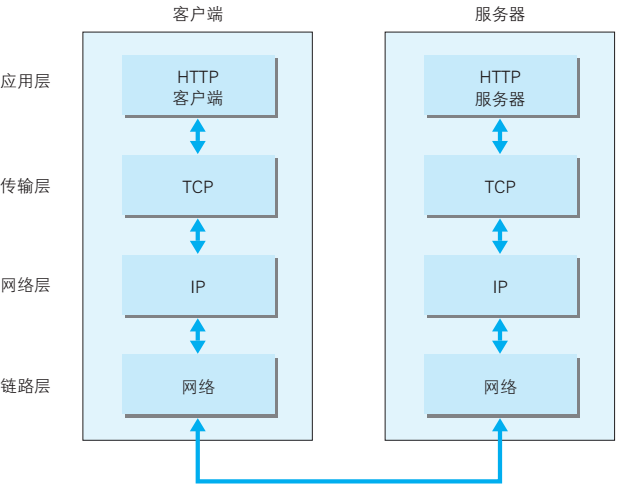


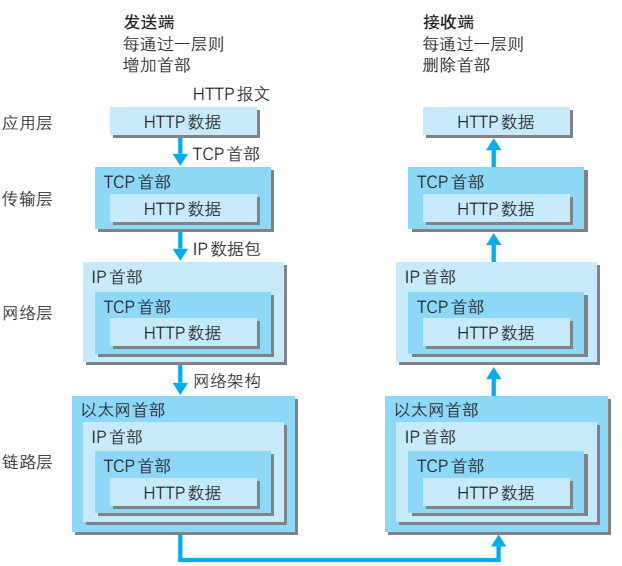






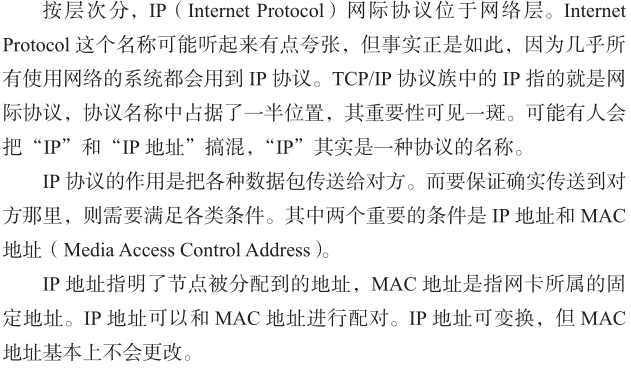






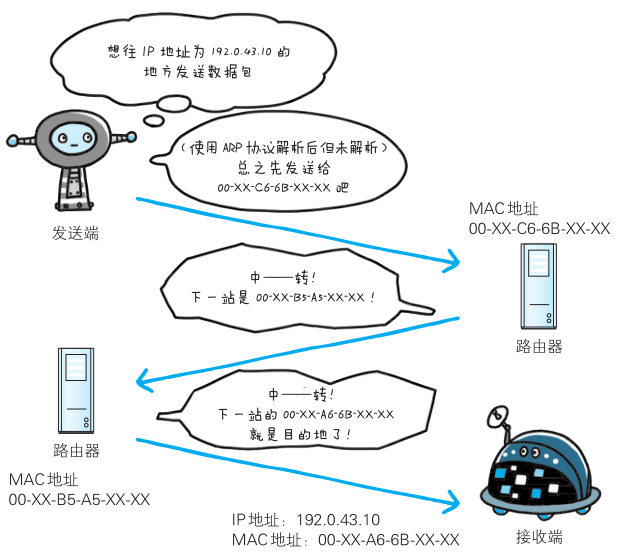
## 与HTTP密切相关的协议：IP、TCP和DNS

### （网络层）负责传输的IP协议



**没有人能够全面掌握互联网中的传输状况**

在到达通信目标前的中转过程中，那些计算机和路由器等网络设备，只能获悉很粗略的传输路线，这种机制称为路由选择（routing）。无论哪台计算机、哪台网络设备，它们都无法全面掌握互联网中的细节。

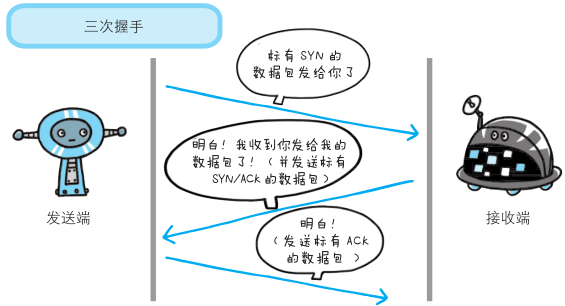


### （传输层）确保可靠性的 TCP 协议

按层次分，TCP 位于传输层，提供可靠的字节流服务。所谓的字节流服务（Byte Stream Service）是指，为了方便传输，将大块数据分割成以报文段（segment）为单位的数据包进行管理。而可靠的传输服务是指，能够把数据准确可靠地传给对方。一言以蔽之，TCP 协议为了更容易传送大数据才把数据分割，而且 TCP 协议能够确认数据最终是否送达到对方。

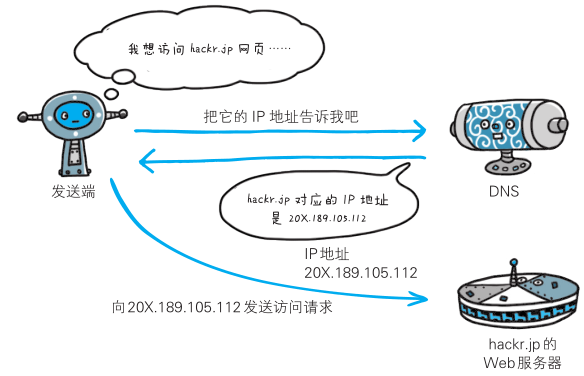
**确保数据能到达目标**

为了准确无误地将数据送达目标处，TCP 协议采用了三次握手（three-way handshaking）策略。用 TCP 协议把数据包送出去后，TCP不会对传送后的情况置之不理，它一定会向对方确认是否成功送达。握手过程中使用了 TCP 的标志（flag）——SYN（synchronize）和 ACK（acknowledgement）。发送端首先发送一个带 SYN 标志的数据包给对方。接收端收到后，回传一个带有 SYN/ACK 标志的数据包以示传达确认信息。最后，发送端再回传一个带 ACK 标志的数据包，代表“握手”结束。若在握手过程中某个阶段莫名中断，TCP 协议会再次以相同的顺序发送相同的数据包。除了上述三次握手，TCP 协议还有其他各种手段来保证通信的可靠性。

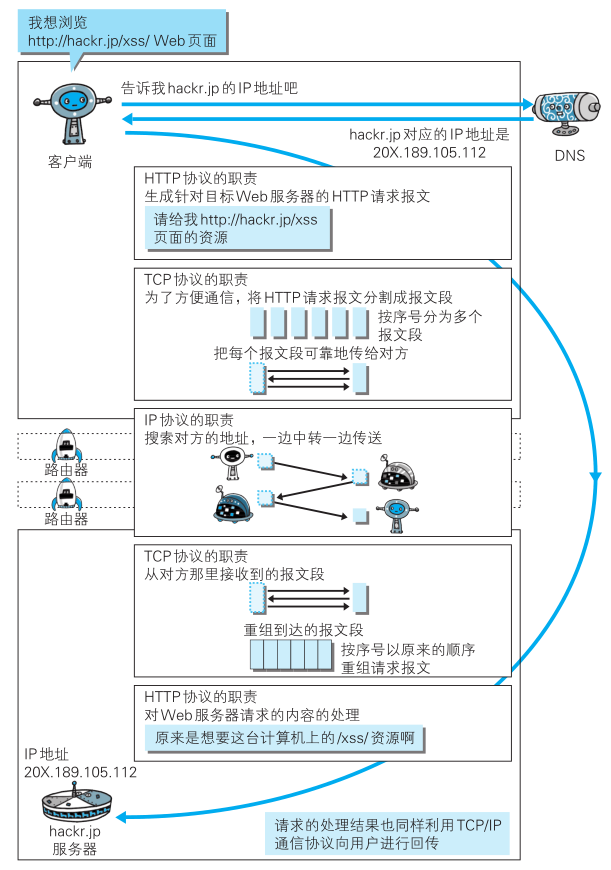


### （应用层）负责域名解析的 DNS 服务

DNS（Domain Name System）服务是和 HTTP 协议一样位于应用层的协议。它提供域名到 IP 地址之间的解析服务。计算机既可以被赋予 IP 地址，也可以被赋予主机名和域名。比如www.hackr.jp。用户通常使用主机名或域名来访问对方的计算机，而不是直接通过IP 地址访问。因为与 IP 地址的一组纯数字相比，用字母配合数字的表示形式来指定计算机名更符合人类的记忆习惯。但要让计算机去理解名称，相对而言就变得困难了。因为计算机更擅长处理一长串数字。为了解决上述的问题，DNS 服务应运而生。DNS 协议提供通过域名查找 IP 地址，或逆向从 IP 地址反查域名的服务



## 各种协议与 HTTP 协议的关系



## URI 和 URL

与 URI（统一资源标识符）相比，我们更熟悉 URL（UniformResource Locator，统一资源定位符）。URL 正是使用 Web 浏览器等访问Web 页面时需要输入的网页地址。比如，http://hackr.jp/ 就是 URL。

### 统一资源标识符

URI 是 Uniform Resource Identifier 的缩写。RFC2396 分别对这3 个单词进行了如下定义。

Uniform

规定统一的格式可方便处理多种不同类型的资源，而不用根据上下文环境来识别资源指定的访问方式。另外，加入新增的协议方案（如 http: 或 ftp:）也更容易。

Resource

资源的定义是“可标识的任何东西”。除了文档文件、图像或服务（例如当天的天气预报）等能够区别于其他类型的，全都可作为资源。另外，资源不仅可以是单一的，也可以是多数的集合体。

Identifier

表示可标识的对象。也称为标识符。

综上所述，URI 就是由某个协议方案表示的资源的定位标识符。协议方案是指访问资源所使用的协议类型名称。采用 HTTP 协议时，协议方案就是 http。除此之外，还有 ftp、mailto、telnet、file 等。标准的 URI 协议方案有 30 种左右，本书接下来的章节中会频繁出现 URI 这个术语，在充分理解的基础上，也可用 URL 替换 URI。

URI 用字符串标识某一互联网资源，而 URL 表示资源的地点（互联网上所处的位置）。可见 URL 是 URI 的子集。

RFC3986：统一资源标识符（URI）通用语法中列举了几种 URI例子，如下所示。

ftp://ftp.is.co.za/rfc/rfc1808.txt

http://www.ietf.org/rfc/rfc2396.txt

ldap://[2001:db8::7]/c=GB?objectClass?one

mailto:John.Doe@example.com

news:comp.infosystems.www.servers.unix

tel:+1-816-555-1212

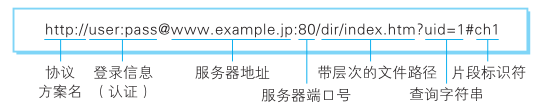
telnet://192.0.2.16:80/

urn:oasis:names:specification:docbook:dtd:xml:4.1.2

### URI 格式

表示指定的 URI，要使用涵盖全部必要信息的绝对 URI、绝对 URL以及相对 URL。

* 相对 URL，是指从浏览器中基本 URI 处指定的 URL，形如 /image/logo.gif。
* 绝对URL的格式



**协议方案名**

使用 http: 或 https: 等协议方案名获取访问资源时要指定协议类型。不区分字母大小写，最后附一个冒号“:”。也可使用 data: 或 javascript: 这类指定数据或脚本程序的方案名。

**登录信息（认证）**

指定用户名和密码作为从服务器端获取资源时必要的登录信息（身份认证）。此项是可选项。

**服务器地址**

使用绝对 URI 必须指定待访问的服务器地址。地址可以是类似hackr.jp 这种 DNS 可解析的名称，或是 192.168.1.1 这类 IPv4 地址名，还可以是 [0:0:0:0:0:0:0:1] 这样用方括号括起来的 IPv6 地址名。

**服务器端口号**

指定服务器连接的网络端口号。此项也是可选项，若用户省略则自动使用默认端口号。

**带层次的文件路径**

指定服务器上的文件路径来定位特指的资源。这与 UNIX 系统的文件目录结构相似。

**查询字符串**

针对已指定的文件路径内的资源，可以使用查询字符串传入任意参数。此项可选。

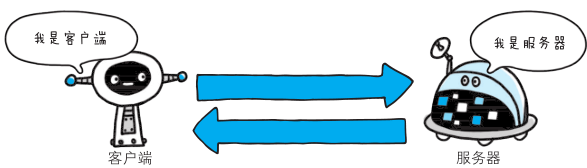
**片段标识符**

使用片段标识符通常可标记出已获取资源中的子资源（文档内的某个位置）。但在 RFC 中并没有明确规定其使用方法。该项也为可选项。

# 简单的HTTP协议

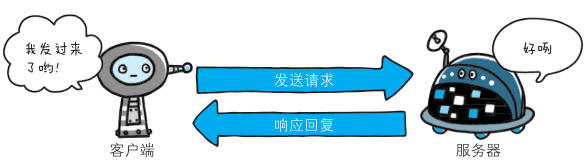
## HTTP 协议用于客户端和服务器端之间的通信

请求访问文本或图像等资源的一端称为客户端，而提供资源响应的一端称为服务器端。应用 HTTP 协议时，必定是一端担任客户端角色，另一端担任服务器端角色

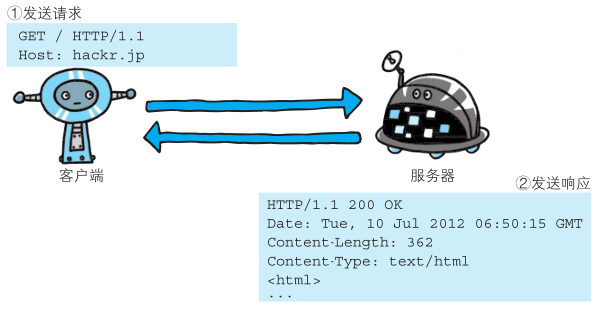


有时候，按实际情况，两台计算机作为客户端和服务器端的角色有可能会互换。但就仅从一条通信路线来说，服务器端和客户端的角色是确定的，而用 HTTP 协议能够明确区分哪端是客户端，哪端是服务器端

## 通过请求和响应的交换达成通信



HTTP 协议规定，请求从客户端发出，最后服务器端响应该请求并返回。换句话说，肯定是先从客户端开始建立通信的，服务器端在没有接收到请求之前不会发送响应。下面，我们来看一个具体的示例



### 请求报文

下面则是从客户端发送给某个 HTTP 服务器端的请求报文中的内容。

GET /index.htm HTTP/1.1

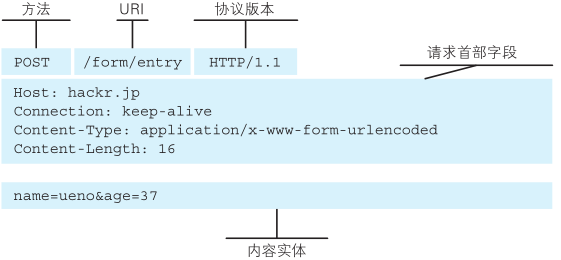
Host: hackr.jp

起始行开头的 GET 表示请求访问服务器的类型，称为方法（method）。随后的字符串 /index.htm 指明了请求访问的资源对象，也叫做请求 URI（request-URI）。最后的 HTTP/1.1，即 HTTP 的版本号，用来提示客户端使用的 HTTP 协议功能。

综合来看，这段请求内容的意思是：请求访问某台 HTTP 服务器上

的 /index.htm 页面资源。

请求报文是由请求方法、请求 URI、协议版本、可选的请求首部字段和内容实体构成的



### 响应报文

接收到请求的服务器，会将请求内容的处理结果以响应的形式返回。

HTTP/1.1 200 OK

Date: Tue, 10 Jul 2012 06:50:15 GMT

Content-Length: 362

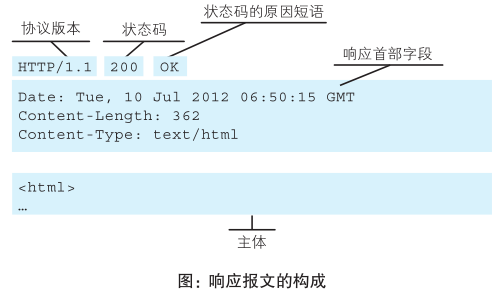
Content-Type: text/html

<html>

……

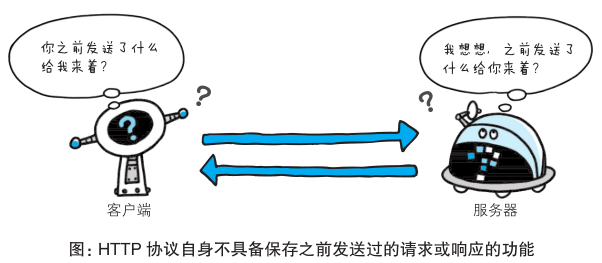
在起始行开头的 HTTP/1.1 表示服务器对应的 HTTP 版本。紧挨着的 200 OK 表示请求的处理结果的状态码（status code）和原因短语（reason-phrase）。下一行显示了创建响应的日期时间，是首部字段（header field）内的一个属性。接着以一空行分隔，之后的内容称为资源实体的主体（entitybody）。

响应报文基本上由协议版本、状态码（表示请求成功或失败的数字代码）、用以解释状态码的原因短语、可选的响应首部字段以及实体主体构成



## HTTP 是不保存状态的协议

HTTP 是一种不保存状态，即无状态（stateless）协议。HTTP 协议自身不对请求和响应之间的通信状态进行保存。也就是说在 HTTP 这个级别，协议对于发送过的请求或响应都不做持久化处理。

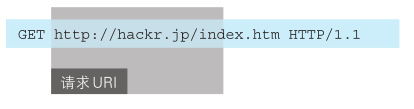


这是为了更快地处理大量事务，确保协议的可伸缩性，而特意把 HTTP 协议设计成如此简单的。可是，随着 Web 的不断发展，因无状态而导致业务处理变得棘手的情况增多了。比如，用户登录到一家购物网站。HTTP/1.1 虽然是无状态协议，但为了实现期望的保持状态功能，于是引入了 Cookie 技术。有了 Cookie 再用 HTTP 协议通信，就可以管理状态了

## 请求 URI 定位资源

HTTP 协议使用 URI 定位互联网上的资源。正是因为 URI 的特定功能，在互联网上任意位置的资源都能访问到。当客户端请求访问资源而发送请求时，URI 需要将作为请求报文中的请求 URI 包含在内。指定请求 URI 的方式有很多。

* URI为完整的请求URI



* 在首部字段Host中写明网络域名或IP地址



* 如果不是访问特定资源而是对服务器本身发起请求

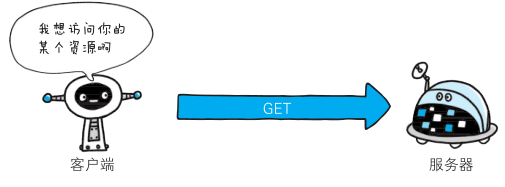
用一个 \* 来代替请求 URI。比如查询 HTTP 服务器端支持的 HTTP 方法种类。



## 告知服务器意图的 HTTP 方法

### GET：获取资源

GET 方法用来请求访问已被 URI 识别的资源。指定的资源经服务器端解析后返回响应内容。也就是说，如果请求的资源是文本，那就保持原样返回；如果是像 CGI（Common Gateway Interface，通用网关接口）那样的程序，则返回经过执行后的输出结果。

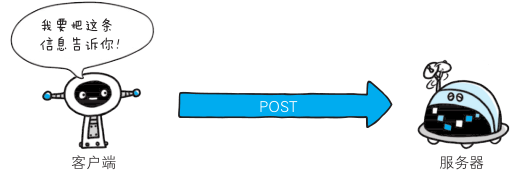


使用 GET 方法的请求·响应的例子

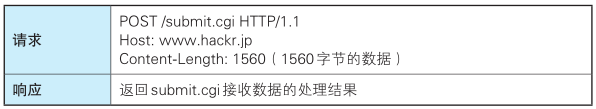


### POST：传输实体主体

POST 方法用来传输实体的主体。虽然用 GET 方法也可以传输实体的主体，但一般不用 GET 方法进行传输，而是用 POST 方法。虽说 POST 的功能与 GET 很相似，但POST 的主要目的并不是获取响应的主体内容。

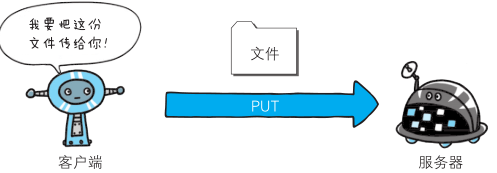


使用 POST 方法的请求·响应的例子

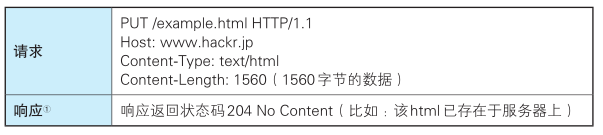


### PUT：传输文件

PUT 方法用来传输文件。就像 FTP 协议的文件上传一样，要求在请求报文的主体中包含文件内容，然后保存到请求 URI 指定的位置。但是，鉴于 HTTP/1.1 的 PUT 方法自身不带验证机制，任何人都可以上传文件 , 存在安全性问题，因此一般的 Web 网站不使用该方法。若配合 Web 应用程序的验证机制，或架构设计采用 REST（REpresentationalState Transfer，表征状态转移）标准的同类 Web 网站，就可能会开放使用 PUT 方法。

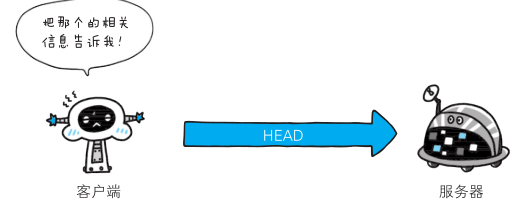


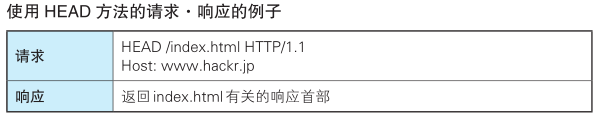
使用 PUT 方法的请求·响应的例子



### HEAD：获得报文首部

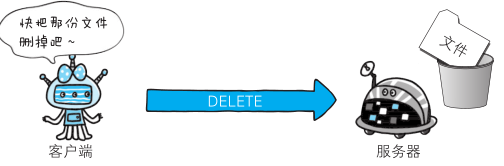
HEAD 方法和 GET 方法一样，只是不返回报文主体部分。用于确认URI 的有效性及资源更新的日期时间等





### DELETE：删除文件

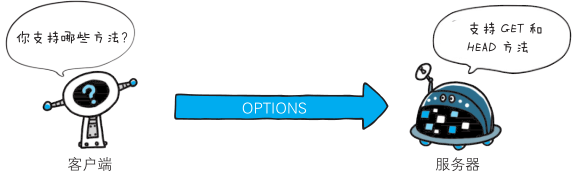
DELETE 方法用来删除文件，是与 PUT 相反的方法。DELETE 方法按请求 URI 删除指定的资源。但是，HTTP/1.1 的 DELETE 方法本身和 PUT 方法一样不带验证机制，所以一般的 Web 网站也不使用 DELETE 方法。当配合 Web 应用程序的验证机制，或遵守 REST 标准时还是有可能会开放使用的

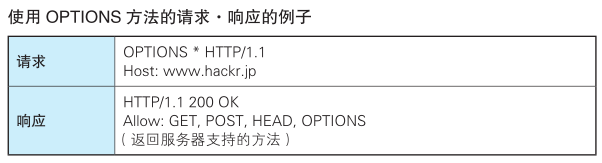




### OPTIONS：询问支持的方法

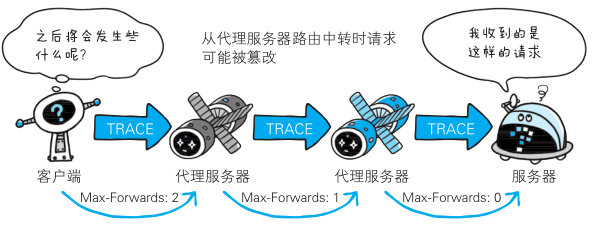
OPTIONS 方法用来查询针对请求 URI 指定的资源支持的方法。





### TRACE：追踪路径

TRACE 方法是让 Web 服务器端将之前的请求通信环回给客户端的方法。发送请求时，在 Max-Forwards 首部字段中填入数值，每经过一个服务器端就将该数字减 1，当数值刚好减到 0 时，就停止继续传输，最后接收到请求的服务器端则返回状态码 200 OK 的响应。客户端通过 TRACE 方法可以查询发送出去的请求是怎样被加工修改 / 篡改的。这是因为，请求想要连接到源目标服务器可能会通过代理中转，TRACE 方法就是用来确认连接过程中发生的一系列操作。但是，TRACE 方法本来就不怎么常用，再加上它容易引发 XST（Cross-Site Tracing，跨站追踪）攻击，通常就更不会用到了。

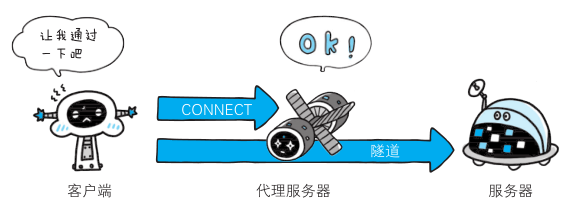




### CONNECT：要求用隧道协议连接代理

CONNECT 方法要求在与代理服务器通信时建立隧道，实现用隧道协议进行 TCP 通信。主要使用 SSL（Secure Sockets Layer，安全套接层）和 TLS（Transport Layer Security，传输层安全）协议把通信内容加密后经网络隧道传输

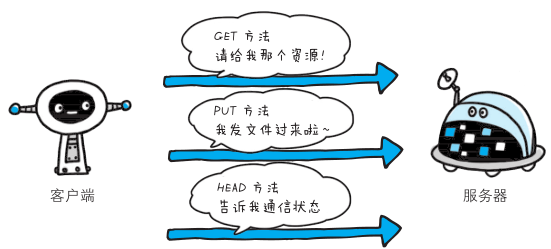
CONNECT 方法的格式如下所示。CONNECT 代理服务器名:端口号 HTTP版本





## 使用方法下达命令

向请求 URI 指定的资源发送请求报文时，采用称为方法的命令。方法的作用在于，可以指定请求的资源按期望产生某种行为。方法中有 GET、POST 和 HEAD 等



下表列出了 HTTP/1.0 和 HTTP/1.1 支持的方法。另外，方法名区分大小写，注意要用大写字母

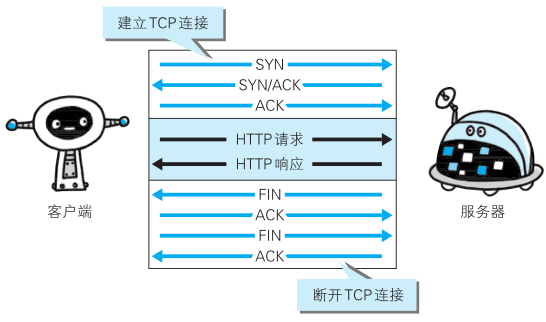


在这里列举的众多方法中，LINK 和 UNLINK 已被 HTTP/1.1 废弃，不再支持

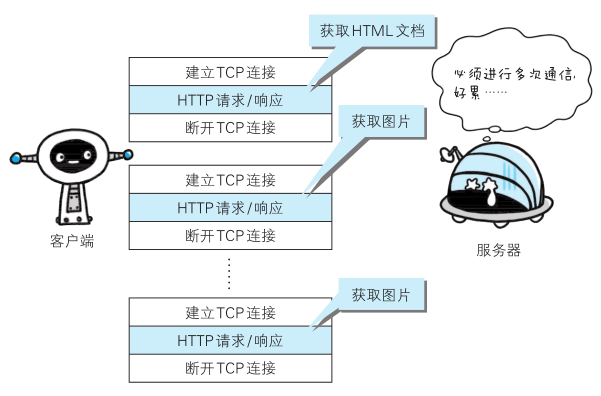
## 持久连接节省通信量

### 短暂连接

HTTP 协议的初始版本中，每进行一次 HTTP 通信就要断开一次TCP 连接

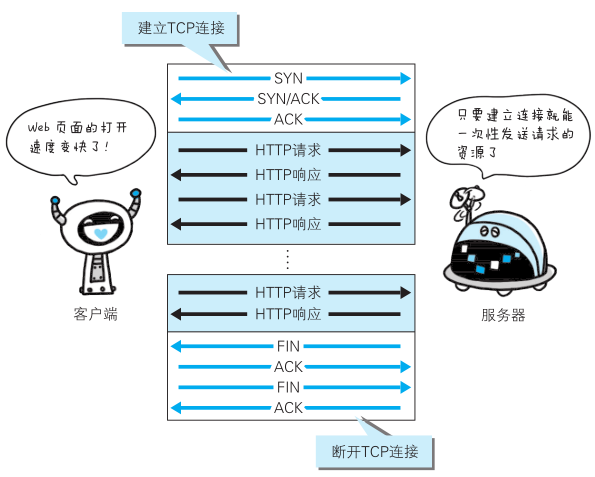


以当年的通信情况来说，因为都是些容量很小的文本传输，所以即使这样也没有多大问题。可随着 HTTP 的普及，文档中包含大量图片的情况多了起来



### 持久连接

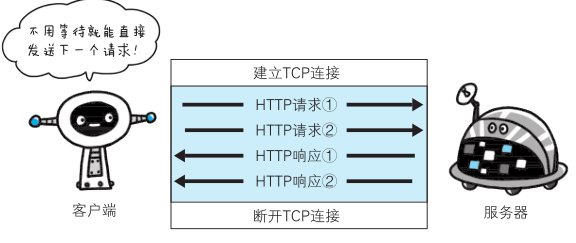
为解决上述 TCP 连接的问题，HTTP/1.1 和一部分的 HTTP/1.0 想出了持久连接（HTTP Persistent Connections，也称为 HTTP keep-alive 或HTTP connection reuse）的方法。持久连接的特点是，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态



在 HTTP/1.1 中，所有的连接默认都是持久连接，但在 HTTP/1.0 内并未标准化。虽然有一部分服务器通过非标准的手段实现了持久连接但服务器端不一定能够支持持久连接。毫无疑问，除了服务器端，客户端也需要支持持久连接。

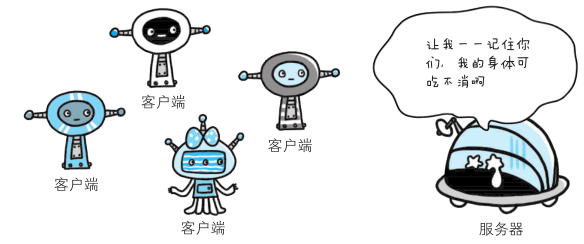
### 管线化

持久连接使得多数请求以管线化（pipelining）方式发送成为可能。从前发送请求后需等待并收到响应，才能发送下一个请求。管线化技术出现后，不用等待响应亦可直接发送下一个请求。这样就能够做到同时并行发送多个请求，而不需要一个接一个地等待响应了。管线化技术则比持久连接还要快。请求数越多，时间差就越明显

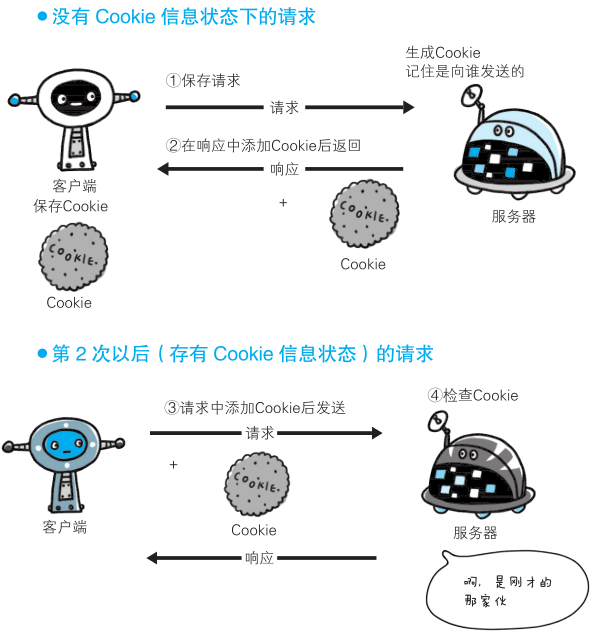


## 使用 Cookie 的状态管理

HTTP 是无状态协议，它不对之前发生过的请求和响应的状态进行管理。也就是说，无法根据之前的状态进行本次的请求处理。不可否认，无状态协议当然也有它的优点。由于不必保存状态，自然可减少服务器的 CPU 及内存资源的消耗。从另一侧面来说，也正是因为 HTTP 协议本身是非常简单的，所以才会被应用在各种场景里。



保留无状态协议这个特征的同时又要解决类似的矛盾问题，于是引入了 Cookie 技术。Cookie 技术通过在请求和响应报文中写入 Cookie 信息来控制客户端的状态。Cookie 会根据从服务器端发送的响应报文内的一个叫做 Set-Cookie的首部字段信息，通知客户端保存 Cookie。当下次客户端再往该服务器发送请求时，客户端会自动在请求报文中加入 Cookie 值后发送出去。服务器端发现客户端发送过来的 Cookie 后，会去检查究竟是从哪一个客户端发来的连接请求，然后对比服务器上的记录，最后得到之前的状态信息



HTTP 请求报文和响应报文的内容如下。

* 请求报文（没有 Cookie 信息的状态）

GET /reader/ HTTP/1.1

Host: hackr.jp

\*首部字段内没有Cookie的相关信息

* 响应报文（服务器端生成 Cookie 信息）

HTTP/1.1 200 OK

Date: Thu, 12 Jul 2012 07:12:20 GMT

Server: Apache

＜Set-Cookie: sid=1342077140226724; path=/; expires=Wed,

10-Oct-12 07:12:20 GMT＞

Content-Type: text/plain; charset=UTF-8

* 请求报文（自动发送保存着的 Cookie 信息）

GET /image/ HTTP/1.1

Host: hackr.jp

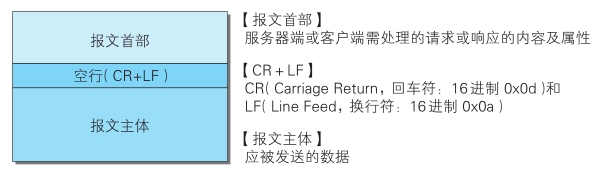
Cookie: sid=1342077140226724

# HTTP报文内的HTTP信息

HTTP通信过程包括从客户端发往服务器端的请求及从服务器端返回客户端的响应

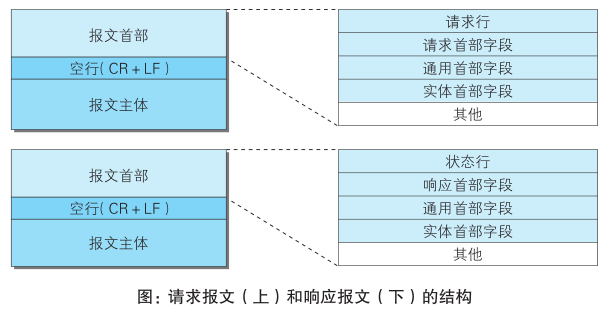
## HTTP 报文

用于 HTTP 协议交互的信息被称为 HTTP 报文。请求端（客户端）的HTTP 报文叫做请求报文，响应端（服务器端）的叫做响应报文。HTTP报文本身是由多行（用 CR+LF 作换行符）数据构成的字符串文本。HTTP 报文大致可分为报文首部和报文主体两块。两者由最初出现的空行（CR+LF）来划分。通常，并不一定要有报文主体。



## 请求报文及响应报文的结构

我们来看一下请求报文和响应报文的结构





**请求行**

包含用于请求的方法，请求 URI 和 HTTP 版本。

**状态行**

包含表明响应结果的状态码，原因短语和 HTTP 版本。

**首部字段**

包含表示请求和响应的各种条件和属性的各类首部。一般有 4 种首部，分别是：通用首部、请求首部、响应首部和实体首部。

**其他**

可能包含 HTTP 的 RFC 里未定义的首部（Cookie 等）。

## 编码提升传输速率

HTTP 在传输数据时可以按照数据原貌直接传输，但也可以在传输过程中通过编码提升传输速率。通过在传输时编码，能有效地处理大量的访问请求。但是，编码的操作需要计算机来完成，因此会消耗更多的CPU 等资源。

### 报文主体和实体主体的差异

* 报文（message）

是 HTTP 通信中的基本单位，由 8 位组字节流（octet sequence，其中 octet 为 8 个比特）组成，通过 HTTP 通信传输。

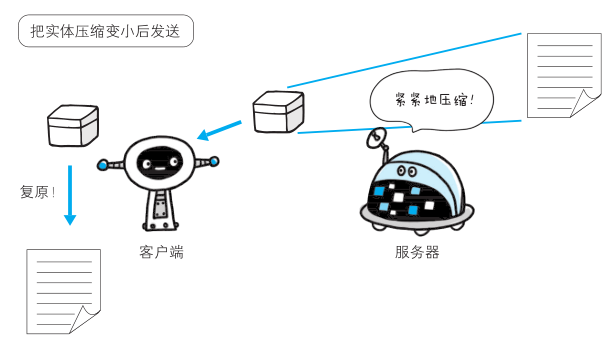
* 实体（entity）

作为请求或响应的有效载荷数据（补充项）被传输，其内容由实体首部和实体主体组成。

HTTP 报文的主体用于传输请求或响应的实体主体。通常，报文主体等于实体主体。只有当传输中进行编码操作时，实体主体的内容发生变化，才导致它和报文主体产生差异。报文和实体这两个术语在之后会经常出现，请事先理解两者的差异。

### 压缩传输的内容编码

向待发送邮件内增加附件时，为了使邮件容量变小，我们会先用ZIP 压缩文件之后再添加附件发送。HTTP 协议中有一种被称为内容编码的功能也能进行类似的操作。内容编码指明应用在实体内容上的编码格式，并保持实体信息原样压缩。内容编码后的实体由客户端接收并负责解码

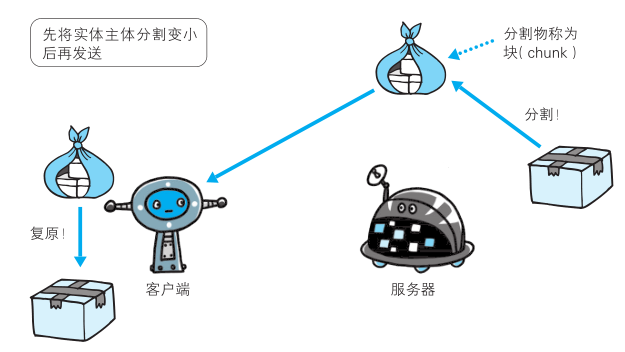


**常用的内容编码有以下几种：**

* gzip（GNU zip）
* compress（UNIX 系统的标准压缩）
* deflate（zlib）
* identity（不进行编码）

### 分割发送的分块传输编码

在 HTTP 通信过程中，请求的编码实体资源尚未全部传输完成之前，浏览器无法显示请求页面。在传输大容量数据时，通过把数据分割成多块，能够让浏览器逐步显示页面。这种把实体主体分块的功能称为分块传输编码（Chunked Transfer Coding）。



分块传输编码会将实体主体分成多个部分（块）。每一块都会用十六进制来标记块的大小，而实体主体的最后一块会使用“0(CR+LF)”来标记。

使用分块传输编码的实体主体会由接收的客户端负责解码，恢复到编码前的实体主体。HTTP/1.1 中存在一种称为传输编码（Transfer Coding）的机制，它可以在通信时按某种编码方式传输，但只定义作用于分块传输编码中

## 发送多种数据的多部分对象集合

HTTP 协议中采纳了多部分对象集合，发送的一份报文主体内可含有多类型实体。通常是在图片或文本文件等上传时使用。多部分对象集合包含的对象如下：

**multipart/form-data**

在 Web 表单文件上传时使用。

**multipart/byteranges**

状态码 206（Partial Content，部分内容）响应报文包含了多个范围的内容时使用。

**multipart/form-data**

Content-Type: multipart/form-data; boundary=AaB03x

--AaB03x

Content-Disposition: form-data; name="field1"

Joe Blow

--AaB03x

Content-Disposition: form-data; name="pics"; filename="file1.txt"

Content-Type: text/plain

... （file1.txt的数据） ...

--AaB03x--

**multipart/byteranges**

HTTP/1.1 206 Partial Content

Date: Fri, 13 Jul 2012 02:45:26 GMT

Last-Modified: Fri, 31 Aug 2007 02:02:20 GMT

Content-Type: multipart/byteranges; boundary=THIS\_STRING\_SEPARATES

--THIS\_STRING\_SEPARATES

Content-Type: application/pdf

Content-Range: bytes 500-999/8000

... （范围指定的数据） ...

--THIS\_STRING\_SEPARATES

Content-Type: application/pdf

Content-Range: bytes 7000-7999/8000

... （范围指定的数据） ...

--THIS\_STRING\_SEPARATES--

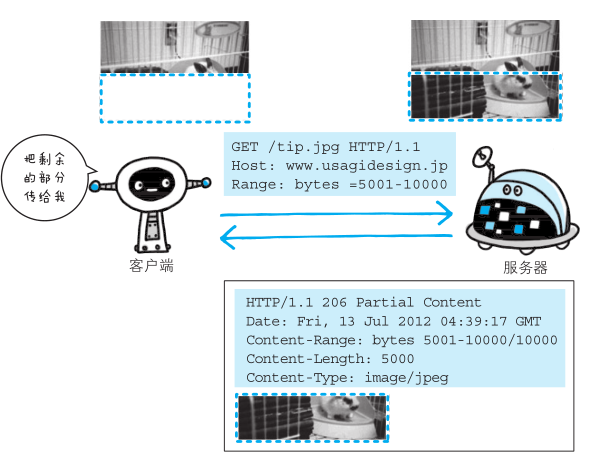
在 HTTP 报文中使用多部分对象集合时，需要在首部字段里加上Content-type。有关这个首部字段，我们稍后讲解。

使用 boundary 字符串来划分多部分对象集合指明的各类实体。在boundary 字符串指定的各个实体的起始行之前插入“--”标记（例如：--AaB03x、--THIS\_STRING\_SEPARATES），而在多部分对象集合对应的字符串的最后插入“--”标记（例如：--AaB03x--、--THIS\_STRING\_SEPARATES--）作为结束。

多部分对象集合的每个部分类型中，都可以含有首部字段。另外，可以在某个部分中嵌套使用多部分对象集合。

## 获取部分内容的范围请求

以前，用户不能使用现在这种高速的带宽访问互联网，当时，下载一个尺寸稍大的图片或文件就已经很吃力了。如果下载过程中遇到网络中断的情况，那就必须重头开始。为了解决上述问题，需要一种可恢复的机制。所谓恢复是指能从之前下载中断处恢复下载。要实现该功能需要指定下载的实体范围。像这样，指定范围发送的请求叫做范围请求（Range Request）。



执行范围请求时，会用到首部字段 Range 来指定资源的 byte 范围。byte 范围的指定形式如下

**5001~10000字节**

Range: bytes=5001-10000

**从5001字节之后全部的**

Range: bytes=5001-

**从一开始到 3000字节和5000~7000字节的多重范围**

Range: bytes=-3000, 5000-7000

针对范围请求，响应会返回状态码为 206 Partial Content 的响应报文。另外，对于多重范围的范围请求，响应会在首部字段 Content-Type标明 multipart/byteranges 后返回响应报文。如果服务器端无法响应范围请求，则会返回状态码 200 OK 和完整的实体内容

## 内容协商返回最合适的内容

同一个 Web 网站有可能存在着多份相同内容的页面。比如英语版和中文版的 Web 页面，它们内容上虽相同，但使用的语言却不同。当浏览器的默认语言为英语或中文，访问相同 URI 的 Web 页面时，则会显示对应的英语版或中文版的 Web 页面。这样的机制称为内容协商（Content Negotiation）

内容协商机制是指客户端和服务器端就响应的资源内容进行交涉，然后提供给客户端最为适合的资源。内容协商会以响应资源的语言、字符集、编码方式等作为判断的基准。包含在请求报文中的某些首部字段（如下）就是判断的基准。这些首部字段的详细说明请参考下一章

Accept

Accept-Charset

Accept-Encoding

Accept-Language

Content-Language

内容协商技术有以下 3 种类型。

**服务器驱动协商（Server-driven Negotiation）**

由服务器端进行内容协商。以请求的首部字段为参考，在服务器端自动处理。但对用户来说，以浏览器发送的信息作为判定的依据，并不一定能筛选出最优内容。

**客户端驱动协商（Agent-driven Negotiation）**

由客户端进行内容协商的方式。用户从浏览器显示的可选项列表中手动选择。还可以利用 JavaScript 脚本在 Web 页面上自动进行上述选择。比如按 OS 的类型或浏览器类型，自行切换成 PC 版页面或手机版页面。

**透明协商（Transparent Negotiation）**

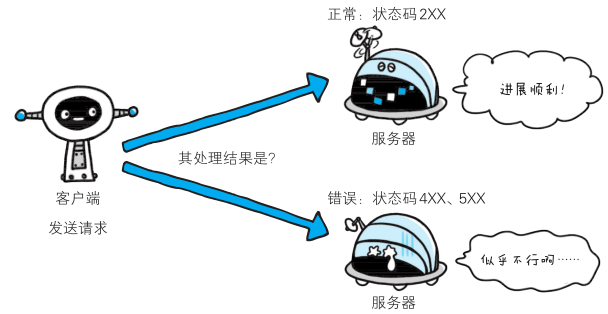
是服务器驱动和客户端驱动的结合体，是由服务器端和客户端各自进行内容协商的一种方法。

# 返回结果的HTTP状态码

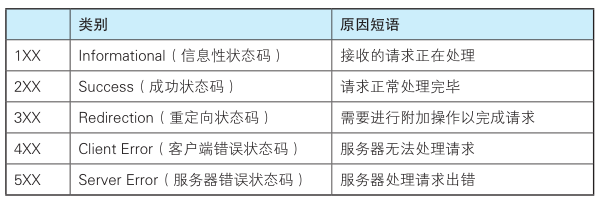
HTTP状态码负责表示客户端HTTP请求的返回结果、标记服务器端的处理是否正常、通知出现的错误等工作

## 状态码告知从服务器端返回的请求结果

状态码的职责是当客户端向服务器端发送请求时，描述返回的请求结果。借助状态码，用户可以知道服务器端是正常处理了请求，还是出现了错误



状态码如 200 OK，以 3 位数字和原因短语组成。数字中的第一位指定了响应类别，后两位无分类。响应类别有以下5 种



只要遵守状态码类别的定义，即使改变 RFC2616 中定义的状态码，或服务器端自行创建状态码都没问题。HTTP状态码数量60余种。别看种类繁多，实际上经常使用的大概只有 14 种。

## 2XX成功

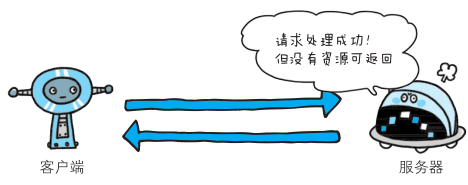
2XX 的响应结果表明请求被正常处理了

### 200 OK



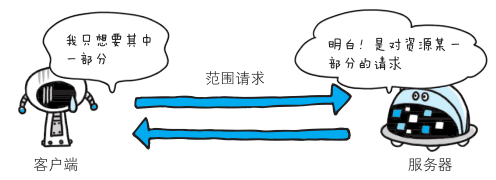
在响应报文内，随状态码一起返回的信息会因方法的不同而发生改变。比如，使用 GET 方法时，对应请求资源的实体会作为响应返回；而使用 HEAD 方法时，对应请求资源的实体首部不随报文主体作为响应返回（即在响应中只返回首部，不会返回实体的主体部分）

### 204 No Content



该状态码代表服务器接收的请求已成功处理，但在返回的响应报文中不含实体的主体部分。另外，也不允许返回任何实体的主体。比如，当从浏览器发出请求处理后，返回 204 响应，那么浏览器显示的页面不发生更新。一般在只需要从客户端往服务器发送信息，而对客户端不需要发送新信息内容的情况下使用

### 206 Partial Content



该状态码表示客户端进行了范围请求，而服务器成功执行了这部分的 GET 请求。响应报文中包含由 Content-Range 指定范围的实体内容

## 3XX重定向

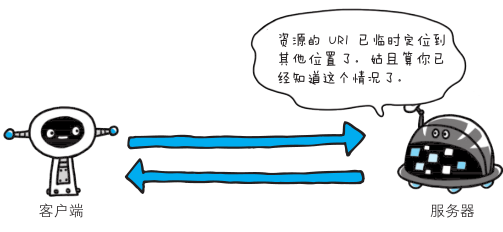
3XX 响应结果表明浏览器需要执行某些特殊的处理以正确处理请求。

### 301 Moved Permanently

永久性重定向。该状态码表示请求的资源已被分配了新的 URI，以后应使用资源现在所指的 URI。也就是说，如果已经把资源对应的 URI保存为书签了，这时应该按 Location 首部字段提示的 URI 重新保存。像下方给出的请求 URI，当指定资源路径的最后忘记添加斜杠“/”，就会产生 301 状态码。

http://example.com/sample

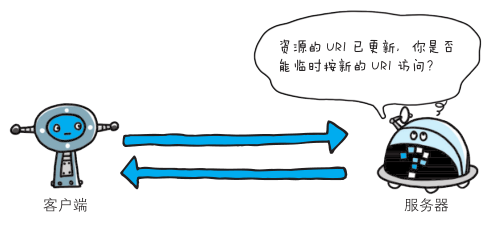
### 302 Found



临时性重定向。该状态码表示请求的资源已被分配了新的 URI，希望用户（本次）能使用新的 URI 访问。

和 301 Moved Permanently 状态码相似，但 302 状态码代表的资源不是被永久移动，只是临时性质的。换句话说，已移动的资源对应的URI 将来还有可能发生改变。比如，用户把 URI 保存成书签，但不会像301 状态码出现时那样去更新书签，而是仍旧保留返回 302 状态码的页面对应的 URI

### 303 See Other



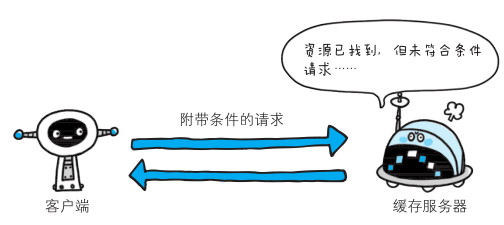
该状态码表示由于请求对应的资源存在着另一个 URI，应使用 GET

方法定向获取请求的资源。

303 状态码和 302 Found 状态码有着相同的功能，但 303 状态码明确表示客户端应当采用 GET 方法获取资源，这点与 302 状态码有区别。比如，当使用 POST 方法访问 CGI 程序，其执行后的处理结果是希望客户端能以 GET 方法重定向到另一个 URI 上去时

当 301、302、303 响应状态码返回时，几乎所有的浏览器都会把POST 改成 GET，并删除请求报文内的主体，之后请求会自动再次发送。301、302 标准是禁止将 POST 方法改变成 GET 方法的，但实际使用时大家都会这么做

### 304 Not Modified



该状态码表示客户端发送附带条件的请求时，服务器端允许请求访问资源，但未满足条件的情况。304 状态码返回时，不包含任何响应的主体部分。304 虽然被划分在 3XX 类别中，但是和重定向没有关系。

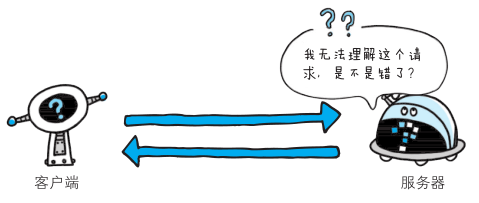
### 307 Temporary Redirect

临时重定向。该状态码与 302 Found 有着相同的含义。尽管 302 标准禁止 POST 变换成 GET，但实际使用时大家并不遵守。307 会遵照浏览器标准，不会从 POST 变成 GET。但是，对于处理响应时的行为，每种浏览器有可能出现不同的情况。

## 4XX 客户端错误

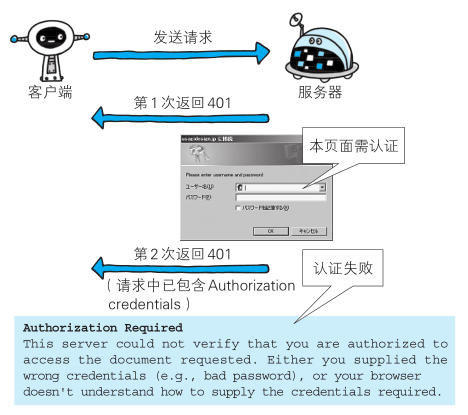
4XX 的响应结果表明客户端是发生错误的原因所在。

### 400 Bad Request



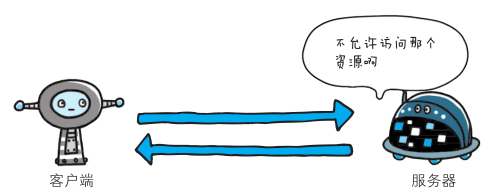
该状态码表示请求报文中存在语法错误。当错误发生时，需修改请求的内容后再次发送请求。另外，浏览器会像 200 OK 一样对待该状态码。

### 401 Unauthorized



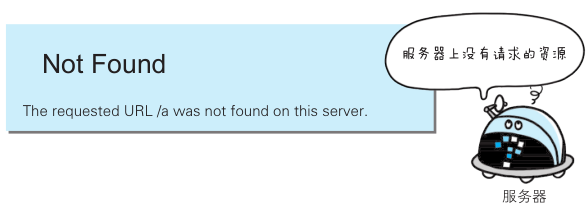
该状态码表示发送的请求需要有通过 HTTP 认证（BASIC 认证、DIGEST 认证）的认证信息。另外若之前已进行过 1 次请求，则表示用户认证失败。返回含有 401 的响应必须包含一个适用于被请求资源的 WWW-Authenticate 首部用以质询（challenge）用户信息。当浏览器初次接收到 401 响应，会弹出认证用的对话窗口

### 403 Forbidden



该状态码表明对请求资源的访问被服务器拒绝了。服务器端没有必要给出拒绝的详细理由，但如果想作说明的话，可以在实体的主体部分对原因进行描述，这样就能让用户看到了。未获得文件系统的访问授权，访问权限出现某些问题（从未授权的发送源 IP 地址试图访问）等列举的情况都可能是发生 403 的原因

### 404 Not Found

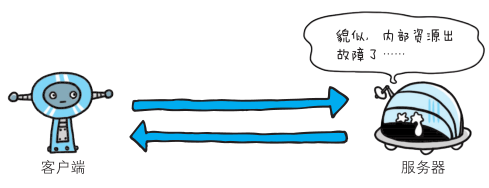


该状态码表明服务器上无法找到请求的资源。除此之外，也可以在服务器端拒绝请求且不想说明理由时使用

## 5XX 服务器错误

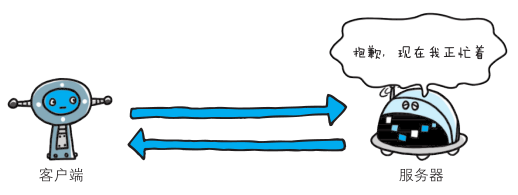
5XX 的响应结果表明服务器本身发生错误

### 500 Internal Server Error



该状态码表明服务器端在执行请求时发生了错误。也有可能是 Web应用存在的 bug 或某些临时的故障

### 503 Service Unavailable



该状态码表明服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。如果事先得知解除以上状况需要的时间，最好写入 Retry-After 首部字段再返回给客户端

## 状态码和状况的不一致

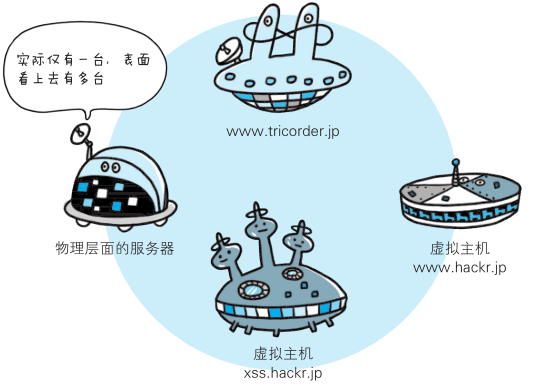
不少返回的状态码响应都是错误的，但是用户可能察觉不到这点。比如 Web 应用程序内部发生错误，状态码依然返回 200 OK，这种情况也经常遇到

# 与HTTP协作的Web服务器

一台Web服务器可搭建多个独立域名的Web网站，也可作为通信路径上的中转服务器提升传输效率

## 用单台虚拟主机实现多个域名

HTTP/1.1 规范允许一台 HTTP 服务器搭建多个 Web 站点。比如，提供 Web 托管服务（Web Hosting Service）的供应商，可以用一台服务器为多位客户服务，也可以以每位客户持有的域名运行各自不同的网站。这是因为利用了虚拟主机（Virtual Host，又称虚拟服务器）的功能。即使物理层面只有一台服务器，但只要使用虚拟主机的功能，则可以假想已具有多台服务器



在互联网上，域名通过 DNS 服务映射到 IP 地址（域名解析）之后访问目标网站。可见，当请求发送到服务器时，已经是以 IP 地址形式访问了。所以，如果一台服务器内托管了 www.tricorder.jp 和 [www.hackr.jp这两个域名，当收到请求时就需要弄清楚究竟要访问哪个域名](http://www.hackr.jp这两个域名，当收到请求时就需要弄清楚究竟要访问哪个域名)



在相同的 IP 地址下，由于虚拟主机可以寄存多个不同主机名和域名的 Web 网站，因此在发送 HTTP 请求时，必须在 Host 首部内完整指定主机名或域名的 URI

## 通信数据转发程序：代理、网关、隧道

HTTP 通信时，除客户端和服务器以外，还有一些用于通信数据转发的应用程序，例如代理、网关和隧道。它们可以配合服务器工作。这些应用程序和服务器可以将请求转发给通信线路上的下一站服务器，并且能接收从那台服务器发送的响应再转发给客户端。

**代理**

代理是一种有转发功能的应用程序，它扮演了位于服务器和客户端“中间人”的角色，接收由客户端发送的请求并转发给服务器，同时也接收服务器返回的响应并转发给客户端。

**网关**

网关是转发其他服务器通信数据的服务器，接收从客户端发送来的请求时，它就像自己拥有资源的源服务器一样对请求进行处理。有时客户端可能都不会察觉，自己的通信目标是一个网关

**隧道**

隧道是在相隔甚远的客户端和服务器两者之间进行中转，并保持双方通信连接的应用程序。

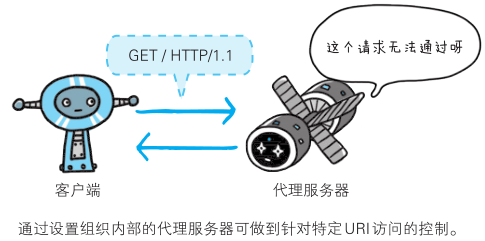
### 代理



代理服务器的基本行为就是接收客户端发送的请求后转发给其他服务器。代理不改变请求 URI，会直接发送给前方持有资源的目标服务器。持有资源实体的服务器被称为源服务器。从源服务器返回的响应经过代理服务器后再传给客户端



在 HTTP 通信过程中，可级联多台代理服务器。请求和响应的转发会经过数台类似锁链一样连接起来的代理服务器。转发时，需要附加Via 首部字段以标记出经过的主机信息



使用代理服务器的理由有：利用缓存技术（稍后讲解）减少网络带宽的流量，组织内部针对特定网站的访问控制，以获取访问日志为主要目的，等等

代理有多种使用方法，按两种基准分类。一种是是否使用缓存，另一种是是否会修改报文。

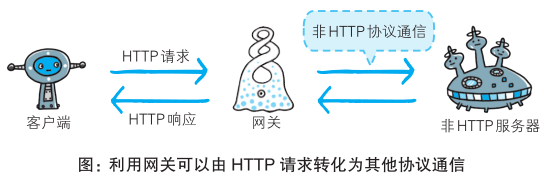
**缓存代理**

代理转发响应时，缓存代理（Caching Proxy）会预先将资源的副本（缓存）保存在代理服务器上。当代理再次接收到对相同资源的请求时，就可以不从源服务器那里获取资源，而是将之前缓存的资源作为响应返回。

**透明代理**

转发请求或响应时，不对报文做任何加工的代理类型被称为透明代理（Transparent Proxy）。反之，对报文内容进行加工的代理被称为非透明代理

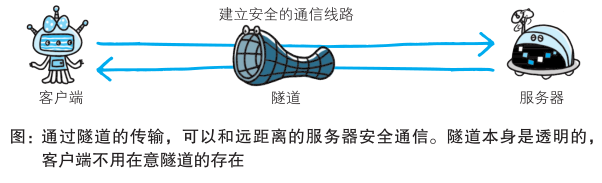
### 网关



网关的工作机制和代理十分相似。而网关能使通信线路上的服务器提供非 HTTP 协议服务。利用网关能提高通信的安全性，因为可以在客户端与网关之间的通信线路上加密以确保连接的安全。比如，网关可以连接数据库，使用SQL 语句查询数据。另外，在 Web 购物网站上进行信用卡结算时，网关可以和信用卡结算系统联动

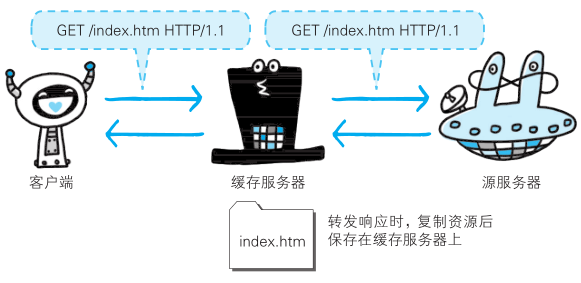
### 隧道

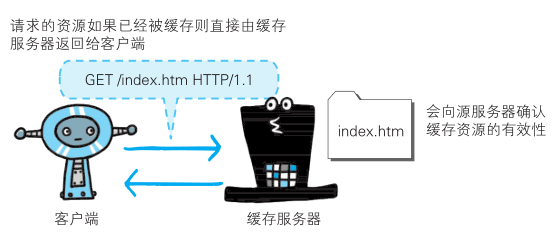
隧道可按要求建立起一条与其他服务器的通信线路，届时使用 SSL等加密手段进行通信。隧道的目的是确保客户端能与服务器进行安全的通信。隧道本身不会去解析 HTTP 请求。也就是说，请求保持原样中转给之后的服务器。隧道会在通信双方断开连接时结束。



## 保存资源的缓存

缓存是指代理服务器或客户端本地磁盘内保存的资源副本。利用缓存可减少对源服务器的访问，因此也就节省了通信流量和通信时间。缓存服务器是代理服务器的一种，并归类在缓存代理类型中。换句话说，当代理转发从服务器返回的响应时，代理服务器将会保存一份资源的副本



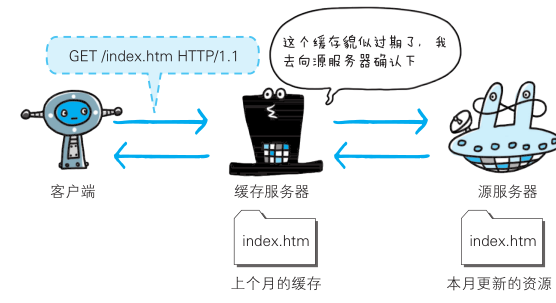


缓存服务器的优势在于利用缓存可避免多次从源服务器转发资源。因此客户端可就近从缓存服务器上获取资源，而源服务器也不必多次处理相同的请求了

### 缓存的有效期限

即便缓存服务器内有缓存，也不能保证每次都会返回对同资源的请求。因为这关系到被缓存资源的有效性问题。当遇上源服务器上的资源更新时，如果还是使用不变的缓存，那就会演变成返回更新前的“旧”资源了。

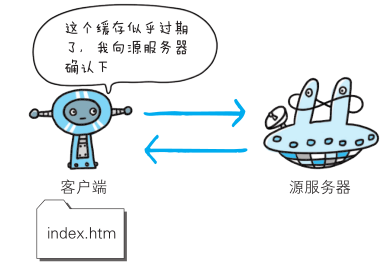
即使存在缓存，也会因为客户端的要求、缓存的有效期等因素，向源服务器确认资源的有效性。若判断缓存失效，缓存服务器将会再次从源服务器上获取“新”资源



### 客户端的缓存

缓存不仅可以存在于缓存服务器内，还可以存在客户端浏览器中。以Internet Explorer 程序为例，把客户端缓存称为临时网络文件（TemporaryInternet File）。浏览器缓存如果有效，就不必再向服务器请求相同的资源了，可以直接从本地磁盘内读取。

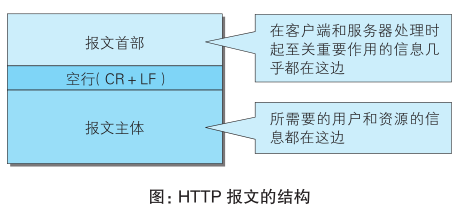
另外，和缓存服务器相同的一点是，当判定缓存过期后，会向源服务器确认资源的有效性。若判断浏览器缓存失效，浏览器会再次请求新资源



# HTTP首部

HTTP协议的请求和响应报文中必定包含HTTP首部，只是我们平时在使用Web的过程中感受不到它

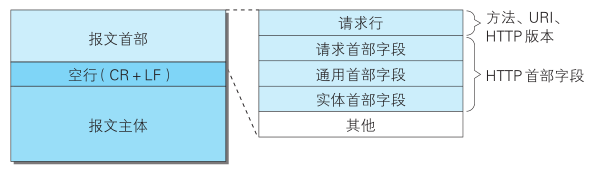
## HTTP 报文首部



HTTP 协议的请求和响应报文中必定包含 HTTP 首部。首部内容为客户端和服务器分别处理请求和响应提供所需要的信息。对于客户端用户来说，这些信息中的大部分内容都无须亲自查看。报文首部由几个字段构成

**HTTP 请求报文**

在请求中，HTTP 报文由方法、URI、HTTP 版本、HTTP 首部字段等部分构成



下面的示例是访问 http://hackr.jp 时，请求报文的首部信息

GET / HTTP/1.1

Host: hackr.jp

User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:13.0) Gecko/⇒

20100101 Firefox/13.0

Accept: text/html,application/xhtml+xml,application/xml;q=0.9,⇒

\*/\*; q=0.8

Accept-Language: ja,en-us;q=0.7,en;q=0.3

Accept-Encoding: gzip, deflate

DNT: 1

Connection: keep-alive

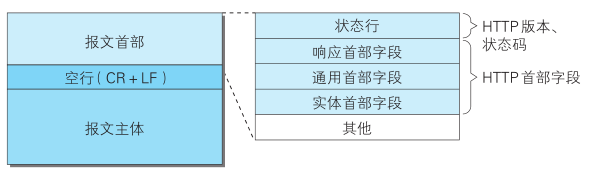
If-Modified-Since: Fri, 31 Aug 2007 02:02:20 GMT

If-None-Match: "45bae1-16a-46d776ac"

Cache-Control: max-age=0

**HTTP 响应报文**

在响应中，HTTP 报文由 HTTP 版本、状态码（数字和原因短语）、HTTP 首部字段 3 部分构成



以下示例是之前请求访问 http://hackr.jp/ 时，返回的响应报文的首部信息

HTTP/1.1 304 Not Modified

Date: Thu, 07 Jun 2012 07:21:36 GMT

Server: Apache

Connection: close

Etag: "45bae1-16a-46d776ac"

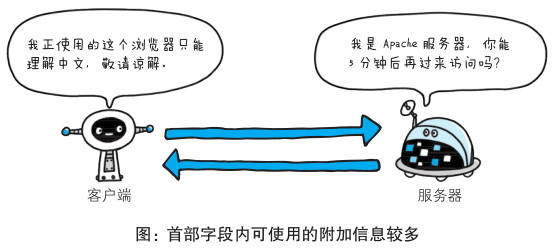
在报文众多的字段当中，HTTP 首部字段包含的信息最为丰富首部字段同时存在于请求和响应报文内，并涵盖 HTTP 报文相关的内容信息

## HTTP 首部字段

### HTTP 首部字段传递重要信息

HTTP 首部字段是构成 HTTP 报文的要素之一。在客户端与服务器之间以 HTTP 协议进行通信的过程中，无论是请求还是响应都会使用首部字段，它能起到传递额外重要信息的作用。

使用首部字段是为了给浏览器和服务器提供报文主体大小、所使用的语言、认证信息等内容



### HTTP 首部字段结构

HTTP 首部字段是由首部字段名和字段值构成的，中间用冒号“:”分隔

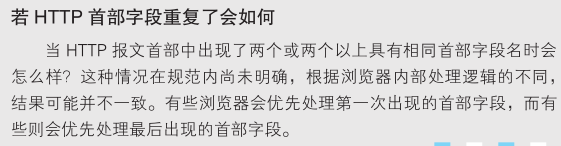
首部字段名: 字段值

例如，在 HTTP 首部中以 Content-Type 这个字段来表示报文主体的对象类型

Content-Type: text/html

另外，字段值对应单个 HTTP 首部字段可以有多个值，如下所示。

Keep-Alive: timeout=15, max=100



### 4 种 HTTP 首部字段类型

HTTP 首部字段根据实际用途被分为以下 4 种类型

**通用首部字段（General Header Fields）**

请求报文和响应报文两方都会使用的首部

**请求首部字段（Request Header Fields）**

从客户端向服务器端发送请求报文时使用的首部。补充了请求的附加内容、客户端信息、响应内容相关优先级等信息。

**响应首部字段（Response Header Fields）**

从服务器端向客户端返回响应报文时使用的首部。补充了响应的附加内容，也会要求客户端附加额外的内容信息。

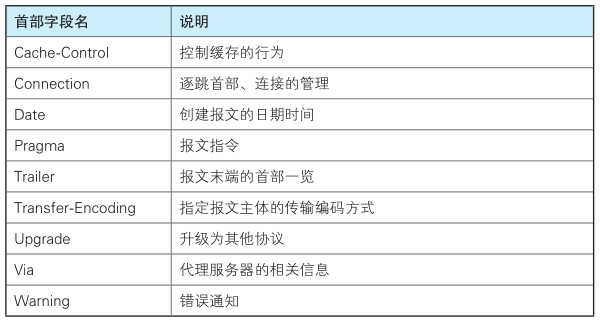
**实体首部字段（Entity Header Fields）**

针对请求报文和响应报文的实体部分使用的首部。补充了资源内容更新时间等与实体有关的信息

### HTTP/1.1 首部字段一览

HTTP/1.1 规范定义了如下 47 种首部字段

**通用首部字段**

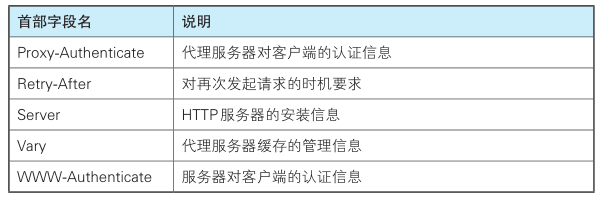


**请求首部字段**



**响应首部字段**





**实体首部字段**



### 非 HTTP/1.1 首部字段

在 HTTP 协议通信交互中使用到的首部字段，不限于 RFC2616 中定义的 47 种首部字段。还有 Cookie、Set-Cookie 和 Content-Disposition等在其他 RFC 中定义的首部字段，它们的使用频率也很高。这些非正式的首部字段统一归纳在 RFC4229 HTTP Header FieldRegistrations 中

### End-to-end首部和Hop-by-hop首部

HTTP 首部字段将定义成缓存代理和非缓存代理的行为，分成 2 种类型。

**端到端首部（End-to-end Header）**

分在此类别中的首部会转发给请求/响应对应的最终接收目标，且必须保存在由缓存生成的响应中，另外规定它必须被转发。

**逐跳首部（Hop-by-hop Header）**

分在此类别中的首部只对单次转发有效，会因通过缓存或代理而不再转发。HTTP/1.1 和之后版本中，如果要使用 hop-by-hop 首部，需提供 Connection 首部字段。

下面列举了 HTTP/1.1 中的逐跳首部字段。除这 8 个首部字段之外，其他所有字段都属于端到端首部。

Connection

Keep-Alive

Proxy-Authenticate

Proxy-Authorization

Trailer

TE

Transfer-Encoding

Upgrade

## 通用首部字段

通用首部字段是指，请求报文和响应报文双方都会使用的首部

## 请求首部字段

请求首部字段是从客户端往服务器端发送请求报文中所使用的字段，用于补充请求的附加信息、客户端信息、对响应内容相关的优先级等内容

## 响应首部字段

响应首部字段是由服务器端向客户端返回响应报文中所使用的字段，用于补充响应的附加信息、服务器信息，以及对客户端的附加要求等信息

## 实体首部字段

实体首部字段是包含在请求报文和响应报文中的实体部分所使用的首部，用于补充内容的更新时间等与实体相关的信息

## 为 Cookie 服务的首部字段

管理服务器与客户端之间状态的 Cookie，虽然没有被编入标准化HTTP/1.1 的 RFC2616 中，但在 Web 网站方面得到了广泛的应用。Cookie 的工作机制是用户识别及状态管理。Web 网站为了管理用户的状态会通过 Web 浏览器，把一些数据临时写入用户的计算机内。接着当用户访问该 Web 网站时，可通过通信方式取回之前发放的Cookie。

调用 Cookie 时，由于可校验 Cookie 的有效期，以及发送方的域、路径、协议等信息，所以正规发布的 Cookie 内的数据不会因来自其他Web 站点和攻击者的攻击而泄露

## 其他首部字段

HTTP 首部字段是可以自行扩展的。所以在 Web 服务器和浏览器的应用上，会出现各种非标准的首部字段