Contents

[HTTP 1](#_Toc456462848)

[<图解HTTP> 1](#_Toc456462849)

[基本概念 1](#_Toc456462850)

[TCP/IP的分层架构模式： 2](#_Toc456462851)

[HTTP是无状态协议 2](#_Toc456462852)

[TCP协议的分割和三次握手 2](#_Toc456462853)

[持久链接 HTTP keep-alive 3](#_Toc456462854)

[Cookie的状态管理 3](#_Toc456462855)

[编码提升传输速率 4](#_Toc456462856)

[发送多种数据的多部分对象集合(multipart) 4](#_Toc456462857)

[恢复下载中断处的下载？ 4](#_Toc456462858)

[内容协商返回最合适的内容 4](#_Toc456462859)

[客户端请求页面流程 5](#_Toc456462860)

[URI格式： 5](#_Toc456462861)

[HTTP报文：多行数据构成的字符串文本 5](#_Toc456462862)

[HTTP状态码 6](#_Toc456462863)

[与HTTP协作的web服务器 6](#_Toc456462864)

[HTTP报文首部 7](#_Toc456462865)

[HTTPS 10](#_Toc456462866)

[确认访问用户身份的认证 11](#_Toc456462867)

[基于http的功能追加协议 12](#_Toc456462868)

[构建Web内容的技术 13](#_Toc456462869)

# HTTP

## <图解HTTP>

### 基本概念

协议：计算机与网络设备要相互通信，双方就必须基于相同的方法。如探测到通信目标、由哪一边先发起通信、使用哪种语言进行通信、怎样结束通信等规则都需要事先确定

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

IP地址：节点被分配到的地址

MAC地址：网卡所属的固定地址

ARP:是一种用以解析地址的协议，根据通信方的IP地址就可以反查出对应的MAC的地址。

路由选择：有点像快递公司的送货过程。

### TCP/IP的分层架构模式：

应用层(HTTP, FTP, …)

传输层(TCP/UDP): 提供处于网络连接中的两台计算机之间的数据传输

网络层(IP): 处理在网络上流动的数据包

链路层：处理连接网络的硬件部分

首先作为发送端的客户端在应用层(HTTP协议)发出一个想看某个Web页面的HTTP请求

接着，为了传输方便，在传输层(TCP协议)把从应用层处收到的数据（HTTP请求报文）进行分割，并在各个报文上打上标记序号及端口后转发给网络层

在网络层(IP协议)，增加作为通信目的地的MAC地址后转发给链路层

发送端，每通过一层则增加首部；接收端，每通过一层则删除首部

### HTTP是无状态协议

HTTP协议对发送过的请求或响应都不做持久化处理， 使用HTTP协议，每当有新的请求发送时，就会有对应的新响应产生。协议本身燕不保留之前一切的请求或响应报文的信息。这是因为更快地处理大量事务，确保协议的保伸缩性，而特意把HTTP协议设计成如此简单

用户场景需要状态，如何处理？

比如用户登录到一家购物网站，即使他跳转到该站的其他页面后，也需要能继续保持登录状态

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

### TCP协议的分割和三次握手

如何更容易传送大数据？

将大块数据分割成以报文段为单位的数据包进行管理。

如何确保数据能到达目标？

TCP协议采用三次握手策略

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）

### 持久链接 HTTP keep-alive

每进行一次HTTP通信就要断开一次TCP连接

建立TCP连接 -> HTTP请求/响应 -> 断开TCP连接

若html里含js, css, jpg等，在发送请求访问HTML页面资源的同时，也会请求该HTML页面里包含的其他资源。因此每次的请求都会造成无谓的TCP连接建立和断开，增加通信量的开销

解决办法：采用持久连接，HTTP keep-alive。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

建立TCP连接 -> HTTP请求/响应 -> HTTP请求/响应 … -> 断开TCP连接

持久连接使得多数请求以管线化pipelining方式发送成为可能。也就是说从前发送请求后需要等待并收到响应，才能发送下一个请求。管线化是指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

建立TCP连接 -> HTTP请求1, HTTP请求2, … -> HTTP响应 … -> 断开TCP连接

### Cookie的状态管理

假设要求登录认证的web页面本身无法进行状态的管理(不记录已登录的状态)，那么每次跳转新页面不是要再次登录，就是要在请求报文中附加参数来管理登录状态

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，然后对比服务端的记录，最后得到之前的状态信息

第一次请求：

客户端：没有Cookie信息状态

服务端：保存请求，生成Cookie，记住是向谁发送的，在响应中添加Cookie后返回

HTTP/1.1 200 OK

Date: ..

Server: Apache

<Set-Cookie: sid=…>

第二次请求：

客户端：存有Cookie信息状态，请求中添加Cookie后发送

GET /img/ HTTP/1.1

Host: hackr.jp

Cookie: sid=…

服务端：检查Cookie,发回响应

### 编码提升传输速率

压缩传输的内容编码，常用的内容编码：gzip, compress, deflate, indentity

### 发送多种数据的多部分对象集合(multipart)

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

在web表单文件上传时使用

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

--Aa

…

--Aa--

状态码206(Partial Content部分内容)响应报文包含了多少范围的内容时使用

HTTP/1.1 206 Partial Content

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

-- THIS\_STRING\_SEP

…

-- THIS\_STRING\_SEP—

### 恢复下载中断处的下载？

获取部分内容的范围请求

Range: byes=5001-10000

### 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，比如英文版，中文版等。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

Accept

Accept-Charset

Accept-Encoding

Accept-Language

Content-Language

### 客户端请求页面流程

客户端

HTTP协议：生成针对目标web服务器的HTTP请求报文

TCP协议：为了方便通信，将HTTP请求报文分割成报文段，按序号分为多个报文段，把每个报文段可靠地传给对方（通过三次握手）

IP协议：搜索对方的地址，一边中转，一边传送

服务端

TCP协议：从客户端接收到报文段，重组报文段，按序号以原来的顺序重组请求报文

HTTP协议：对Web服务器请求的内容的处理

请求的处理结果也同样利用TCP/IP通信协议向用户进行回传

### URI格式：

<http://user:pass@www.example.jp:80/dir/index.htm?uid=1#ch1>

协议方案名://登录信息（认证）@服务器地址：端口号/带层次的文件路径?查询字符串#片段标识符

### HTTP报文：多行数据构成的字符串文本

请求报文：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

POST /form/entry HTTP/1.1

Host: hackr.jp

Connection: keep-alive

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

Content-length: 16

name=qzlin&age=35

响应报文： 协议版本、状态码、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构成

HTTP/1.1. 200 OK

Date: Tue, 10 Jul 2012

Content-Length: 362

Content-type: text/html

<html>

…

### HTTP状态码

* 1XX Informational信息性状态码 接收的请求正在处理
* 2XX Success成功状态码 请求正常处理完毕

200 OK

204 No Content 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 响应报文中包含由Content-Range指定范围的实体内容

* 3XX Redirection重定向状态码 需要进行附加操作以完成请求

301 Moved Permanently 永久性重定向

302 Found 临时重定向

和301 Moved Permanently区别：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

303 See Other

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，返回303状态码。

304 Not Modified

该状态码表示客户端发送附带条件的请求时，服务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

307 Temporary Redirect

* 4XX Client Error客户端错误状态码 服务器无法处理请求

400 Bad Request

401 Unauthorized

当浏览器初次接收到401响应，会弹出认证用的对话窗口。表示需要通过HTTP认证

403 Forbidden 未获得文件系统的访问授权

404 Not Found

* 5XX Server Error服务器错误状态码 服务器处理请求出错

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

重点：状态码和状况可能不一致，因为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

### 与HTTP协作的web服务器

HTTP/1.1规范允许一台http服务器搭建多个web站点，是利用了虚拟主机的功能

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首部内完事指定主机名或域名的uri

在客户端和服务器之间，可能存在通信数据转发程序：代理、网关、隧道

代理：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。每次通过代理服务器转发请求或响应时，会追加写入via首部信息

使用代理服务器的理由：利用缓存技术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

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

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

网关：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。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

隧道：客户端和服务器之间的中转，并保持双方通信连接的应用程序。一般使用SSL等加密手段进行通信

### HTTP报文首部

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

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

HTTP请求报文

= 方法、URI、HTTP版本、HTTP首部字段（请求首部，通用首部+实体首部）

HTTP响应报文

= HTTP版本、状态码、HTTP首部字段（响应首部，通用首部+实体首部）

#### 通用首部

Cache-Control: private, max-age=0, no-cache

控制缓存的行为

Connection: Keep-Alive

http/1.1版本的默认连接都是持久连接，客户端会在持久连接上连续发送请求。当服务器端想明确断开连接时，则指定Connection首部字段的值为Close

Date：Tue, …

创建报文的时间

Transfer-Encoding: chunked

传输报文主体时采用的编码方式

Via: 1.0 gw.hackr.jp

为了追踪传输路径

#### 请求首部字段

Accept: type/subtype

通知服务器用户代理能够处理的媒体类型及媒体类型的相对优先级

文本文件：text/html, text/plain, text/css, … application/xhtml+xml, …

图片文件：image/jpeg, image/gif, …

视频文件：video/mpeg, video/quicktime, …

应用程序使用的二进制文件：application/octet-stream, application/zip…

Accept-Charset: Unicode

通知服务器用户代理支持的字符集及字符集的相对优先顺序

Accept-Encoding: gzip, deflate

通知服务器用户代理支持的内容编码及内容编码的优先级顺序

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

通知服务器用户代理能够处理的自然语言集及优先级顺序

Authorization: Basic ddeeedd

通知服务器用户代理的认证信息（证书值）

Expect: 100-continue

From: [info@hackr.jp](mailto:info@hackr.jp)

Host: [www.hackr.jp](http://www.hackr.jp)

因为虚拟主机运行在同一个ip上，因此使用首部字段Host加以区分

请求被发送至服务器时，请求中的主机名会用IP地址直接替换解决。但如果这时，相同的IP地址下部署运行着多个域名，那么服务器就会无法理解究竟是哪个域名对应的请求。因此，就需要使用首部字段Host来明确指出请求的主机名

条件请求

If-Match 若If-Match的值与服务器资源的ETag匹配，则执行请求

If-None-Match

If-Modified-Since

If-Range 若If-Range的值与ETag匹配，则配合Range执行范围请求处理

If-Unmodified-Since

Max-Forwards: 10

使用HTTP协议通信时，请求可能会经过代理等多台服务器，服务器在往下一个服务器转发请求之前，会将Max-Forwards的值减1后重新赋值。当服务器接收到Max-Forwards值为0的请求时，则不再进行转发，而是直接返回响应

Proxy-Authorization: Basic …

认证行为发生在客户端 与代理之间

Range: bytes=5001-10000

会在处理请求之后返回状态码为206 Partial Content的响应。无法处理该范围请求时，则会返回状态码200 OK的响应及全部资源

User-Agent: Mozilla/5.0 …

将创建请求的浏览器和用户代理名称等信息传达给服务器

#### 响应首部字段

Accept-Ranges: bytes

告知客户端服务器是否能处理范围请求，以指定获取服务器端某个部分的资源

Age: 600

告知客户端，源服务器在多久前创建了响应，单位为秒

ETag: “12222…”

告知客户端实体标识。它是一种可将资源以字符串形式做唯一性标识的方式。服务器会为每份资源分配对应的ETag值

Location: <http://www>...

几乎所有的浏览器在接收到包含首部字段Location的响应后，都会强制性地尝试对已提示的重定向资源的访问

Proxy-Authenticate: Basic realm=”…”

将代理服务器所要求的认证信息发送给客户端

WWW-Authenticate: Basic realm=”…”

告知客户端访问请求URI指定资源需要的认证方案Basic, Digest, …

状态码401 Unauthorized响应中，肯定带有首部字段WWW-Authenticate

Server: Apache/2.2 (unix) PHP/5.

#### 实体首部字段

Allow: GET, HEAD

通知客户端，服务器支持的HTTP方法

Content-Encoding: gzip

实体主体部分选用的内容编码方式

Content-Language: zh-CN

Content-Length: 15000 实体主体的大小，单位是字节

Content-Location: <http://www>...

Content-MD5: dfasfdd…

客户端会对接收的报文主体执行相同的MD5算法获得的128位二进制数，然后经Base64编码，最后与首部字段Content-MD5的字段值比较

Content-Range: bytes 5001-

Content-Type: type/subtype

Expires: Wed, …

告诉客户端该资源失效的日期，若同时存在Cache-Controll的max-age指令时，会优先处理max-age指令

Last-Modified: Wed, …

服务端发送的Set-Cookie

Set-Cookie: status=enable; expires=Tue, …; path=/; domain=.hackr.jp; secure HttpOnly

当expires省略时，其有效期仅限于维持浏览器会话session时间段内，这通常限于浏览器应用程序被关闭之前

Secure属性用地限制web页面伏在https安全连接时才可以发送Cookie

Httponly属性使用cookie不能被javascript脚本访问

客户端带的Cookie

Cookie:status=enable

当客户端想获得http状态管理支持时，就会在请求中包含从服务器接收到的cookie

### HTTPS

HTTP缺点：

通信使用明文，内容可能会被窃听

不验证通信方的身份，因此有可能遭遇伪装

无法证明报文的完整性，所以有可能已遭篡改

HTTP报文使用明文方式发送，而TCP/IP协议族的工作机制，通信内容在所有的通信线路上都有可能遭到窥视

窃听相同段上的通信并非难事，只需要收集在互联网上流动的数据包就行了。对于收集来的数据包的解析工作，可交给那些抓包或嗅探器工具

两种加密方式

内容的加密

由于http协议中没有加密机制，那么就对http协议传输的内容本身加密，即把http报文里所含的内容进行加密处理

为了做到有效的内容加密，前提是要求客户端和服务器同时具备加密和解密机制

通信的加密

http, ftp, telnet -> SSL/TLS -> tcp/ip

https = http + 加密 + 认证 + 完整性保护

经常会在web的登录页面和购物结算界面等使用https通信

加密算法是公开的，密钥保密

共享密钥加密（对称密钥加密）

加密和解密同用一个密钥

在互联网转发密钥，如果通信被监听，那么密钥不可能会落入攻击者之手

公开密钥加密（非对称密钥加密）

公开密钥发布，所有人都可以知道；私有密钥只有解密者知道

关键：使用公开密钥加密，使用私有密钥解密

发送密文的一方使用对方的公开密钥进行加密处理，对方收到被加密的信息后，再使用自己的私有密钥进行解密。

利用这种方式，不需要发送用解密的私有密钥。

要想根据密文和公开密钥，恢复信息原文，目前技术上不太现实，（运算量太大）

HTTPS采用混合加密机制，为什么？

公开密钥加密与共享密钥加密相比，其处理速度更慢

所以组合，在交换密钥环节使用公开密钥加密方式，之后的建立通信交换报文阶段则使用共享密钥加密方式

如何证明公开密钥？

服务器向CA（certificate authority数据证书认证机构）提出公开密钥的申请，CA会对申请的公开密钥做数字签名，然后将公开密钥放入公钥证书后绑定在一起

服务器将公钥证书发送给客户端，公钥证书也叫做数字证书或证书

客户端使用CA的公开密钥，对公钥证书上的数字签名进行验证 （这里注意到：CA的公开密钥如何安全地给客户端？多数浏览器开发商发布版本时，会事先在内部植入常用认证机关的公开密钥）

确认客户端的客户端证书

https还可以使用客户端证书。以客户端证书进行客户端认证，证明服务器正在通信的对方始终是预料之内的客户端，其作用跟服务器证书如出一辙

例如：银行的网上银行就采用了客户端证书。在登录网银时不仅要求用户确认输入ID和密码，还会要求用户的客户端证书，以确认用户是否从特定的终端访问网银

由自认证机构颁发的证书称为自签名证书

如果使用OpenSSL这套开源程序，每个人都可以构建一套属于自己的认证机构，从而自己给自己颂发服务器证书

独立构建的认证机构叫自认证机构，由自认证机构颁发的“无用”证书也被戏称为自签名证书

### 确认访问用户身份的认证

核对信息：

密码：只有本人才会知道的字符串信息

动态令牌：仅限本人持有的设备内显示的一次性密码

数字证书：仅限本人（终端）持有的信息

生物认证：指纹和虹膜等本人的生理信息

IC卡等：仅限本人持有的信息

HTTP认证方式：

Basic 基本认证

Digest 摘要认证

SSL 客户端认证

FormBase 基于表单认证

发送请求

GET /private/ HTTP/1.1

Host:…

返回状态码401，需要认证

HTTP/1.1 401 Authorization Required

WWW-Authenticate: Basic realm=”input your ID and password”

浏览器弹出框，要求输入用户ID and password，然后浏览器会组合ID:password字符串，经过Base64编码，把这串字符串写入首部字段Authorization后，发送请求

GET /private/ HTTP/1.1

Host:…

Authorization: Basic dafdfsa….

认证成功返回状态码200,若认证失败返回状态码401

SSL客户端认证采用双因素认证

第一个认证因素是SSL客户端证书用来认证客户端计算机

第二个认证因素是密码用来确定这是用户本人的行为

基本表单认证

通过服务器端的web应用，将客户端发送过来的用户ID和密码与之前登录过的信息做匹配来进行认证

但HTTP是无状态协议，之前已认证成功的用户状态无法通过协议层面保存下来，即无法实现状态管理，因此即使当该用户下一次继续访问，也无法区分他与其他的用户。所以采用Cookie来管理Session,以弥补http协议中不存在的状态管理功能

步骤：

1. 客户端把用户ID和密码等登录信息放入报文的实体部分，通常是以POST方法把请求发送给服务器。一般使用https通信来进行html表单画面的显示和用户输入数据的发送
2. 服务器验证客户端发送过来的登录信息进行身份认证，然后把用户的认证状态与Session ID绑定后记录在服务器端.向客户端返回响应时，会在首部字段Set-Cookie内写入Session ID。

（可以把Session ID想象成区分用户的等位号，如果Session ID被第三方盗走，对方就可以伪装成你的身份进行恶意操作，所以一般服务器端也需要进行有效期的管理，保证其安全性）

1. 客户端接收到服务器端发来的Session ID后，会将其作为Cookie保存在本地。下次向服务器发送请求时，浏览器会自动发送Cookie,所以Session ID也随之发送到服务器。服务器端可通过验证接收到的Session ID识别用户和其认证状态

### 基于http的功能追加协议

为了尽可能地实时地显示这些更新的内容，服务器上一有内容更新，就需要直接把那些内容反馈到客户端的界面上。

使用http协议探知服务器上是否有内容更新，就必须频繁地从客户端到服务器端进行确认。如果服务器上没有内容更新，那么就会产生徒劳的通信

WebSocket

使用浏览器进行全双工通信

建立在http基础上的协议，发起方仍是客户端，一旦确立websocket通信连接，不论服务器还是客户端，任何一方都可以直接向对方发送报文

主要特点：

推送功能：服务器可直接发送数据，而不必等待客户端的请求

减少通信量：只要建立起websocket连接，就一直保持连接状态

为了实现websocket通信，在http连接建立之后，需要完成一次握手的步骤：

握手请求：

GET /chat HTTP/1.1

Upgrade: websocket

Connection: Upgrade

Sec-WebSocket-Key: …

Sec-WebSocket-Protocol: chat, superchat

握手响应：

HTTP/1.1 101 Switching Protocols

Upgrade: websocket

Connection: Upgrade

Sec-WeSocket-Accept: …

Sec-WebSocket-Protocol: chat

成功握手确立WebSocket连接之后，通信时不再使用http的数据帧，而采用WebSocket独立的数据帧

### 构建Web内容的技术

CGI的应用场景

请求 -> Web服务器 -> CGI程序 ->响应

CGI: Common Gateway Interface通用网关接口，指Web服务器在接收到客户端发送过来的请求后转发给程序的一组机制

每次接到请求，程序都要跟着启动一次，因此一旦访问量过大，Web服务器要承担相当大的负载

Servlet的应用场景

多个请求 -> Web服务器，Web容器Servlet, Servlet -> 响应

Servlet与Web服务器相同的进程中，负载小