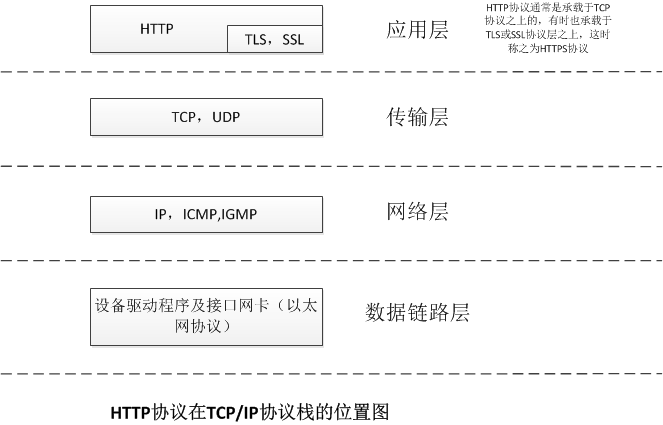
HTTP是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。它于1990年提出，经过几年的使用与发展，得到不断地完善和扩展。目前在WWW中使用的是HTTP/1.0的第六版，HTTP/1.1的规范化工作正在进行之中，而且HTTP-NG(Next Generation of HTTP)的建议已经提出。  
HTTP协议的主要特点可概括如下：  
**1.支持客户/服务器模式**。  
**2.简单快速**：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。  
**3.灵活**：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。  
**4.无连接**：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。  
**5.无状态：HTTP协议是无状态协议。无状态是指协议对于事务处理没有记忆能力。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。**

Http协议栈：   


我们现在用到的版本是**HTTP/1.1**，它比1.0版本添加了更多特性。其中比较重要的特性有：

\* 支持持久连接

\* 支持消息切分成块传输

\* 更加丰富的cache特性

\* 带宽优化及网络连接的使用

\* 错误通知的管理

\* 互联网地址的维护安全性及完整性

**一、HTTP**

**1.1 HTTP协议详解之URL篇**

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

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

http://host[":"port][abs\_path]  
　　http表示要通过HTTP协议来定位网络资源；host表示合法的Internet主机域名或者IP地址；port指定一个端口号，为空则使用缺省端口80；abs\_path指定请求资源的URI；如果URL中没有给出abs\_path，那么当它作为请求URI时，必须以“/”的形式给出，通常这个工作浏览器自动帮我们完成。  
eg:  
1、输入：[www.guet.edu.cn](http://www.guet.edu.cn/)  浏览器自动转换成：<http://www.guet.edu.cn/>  
2、http:192.168.0.116:8080/index.jsp

**1.2 HTTP协议详解之请求篇**

　　http请求由三部分组成，分别是：**请求行、消息报头、请求正文**

1、请求行以一个方法符号开头，以空格分开，后面跟着请求的URI和协议的版本，格式如下：

Method Request-URI HTTP-Version CRLF    
其中 Method表示请求方法；Request-URI是一个统一资源标识符；HTTP-Version表示请求的HTTP协议版本；CRLF表示回车和换行（除了作为结尾的CRLF外，不允许出现单独的CR或LF字符）。

请求方法（所有方法全为大写）有多种，各个方法的解释如下：  
GET　　　　请求获取Request-URI所标识的**资源**  
POST　　　 在Request-URI所标识的资源后**附加新的数据**  
HEAD　　　 请求获取由Request-URI所标识的资源的**响应消息报头**  
PUT　　　　 请求服务器**存储一个资源**，并用Request-URI作为其标识  
DELETE　　 请求服务器**删除Request-URI所标识的资源**  
TRACE　　  请求服务器回送收到的**请求信息**，主要用于测试或诊断  
CONNECT　 保留将来使用  
OPTIONS　  请求查询服务器的性能，或者查询与资源相关的选项和需求  
应用举例：  
GET方法：在浏览器的地址栏中输入网址的方式访问网页时，浏览器采用GET方法向服务器获取资源，eg:GET /form.html HTTP/1.1 (CRLF)

POST方法要求被请求服务器接受附在请求后面的数据，常用于提交表单。  
eg：POST /reg.jsp HTTP/ (CRLF)  
Accept:image/gif,image/x-xbit,... (CRLF)  
...  
HOST:www.guet.edu.cn (CRLF)  
Content-Length:22 (CRLF)  
Connection:Keep-Alive (CRLF)  
Cache-Control:no-cache (CRLF)  
(CRLF)         //该CRLF表示消息报头已经结束，在此之前为消息报头  
user=jeffrey&pwd=1234  //此行以下为提交的数据

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

**1.3 HTTP协议详解之响应篇**

　　在接收和解释请求消息后，服务器返回一个HTTP响应消息。

HTTP响应也是由三个部分组成，分别是：**状态行、消息报头、响应正文**  
1、状态行格式如下：  
HTTP-Version Status-Code Reason-Phrase CRLF  
其中，HTTP-Version表示服务器HTTP协议的版本；Status-Code表示服务器发回的响应状态代码；Reason-Phrase表示状态代码的文本描述。  
状态代码有三位数字组成，第一个数字定义了响应的类别，且有五种可能取值：  
1xx：指示信息--表示请求已接收，继续处理  
2xx：成功--表示请求已被成功接收、理解、接受  
3xx：重定向--要完成请求必须进行更进一步的操作  
4xx：客户端错误--请求有语法错误或请求无法实现  
5xx：服务器端错误--服务器未能实现合法的请求  
常见状态代码、状态描述、说明：  
200 OK      //客户端请求成功  
400 Bad Request  //客户端请求有语法错误，不能被服务器所理解  
401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用   
403 Forbidden  //服务器收到请求，但是拒绝提供服务  
404 Not Found  //请求资源不存在，eg：输入了错误的URL  
500 Internal Server Error //服务器发生不可预期的错误  
503 Server Unavailable  //服务器当前不能处理客户端的请求，一段时间后可能恢复正常  
eg：HTTP/1.1 200 OK （CRLF）

2、响应报头后述

3、响应正文就是服务器返回的资源的内容

**1.4 HTTP协议详解之消息报头篇**

　　HTTP消息由客户端到服务器的请求和服务器到客户端的响应组成。请求消息和响应消息都是由开始行（对于请求消息，开始行就是请求行，对于响应消息，开始行就是状态行），消息报头（可选），空行（只有CRLF的行），消息正文（可选）组成。

HTTP消息报头包括普通报头、**请求报头、响应报头、实体报头**。  
每一个报头域都是由**名字+“：”+空格+值** 组成，消息报头域的名字是大小写无关的。

**1、普通报头**  
　　在普通报头中，有少数报头域用于所有的请求和响应消息，但并不用于被传输的实体，**只用于传输的消息**。  
eg：  
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），HTTP1.0使用的类似的报头域为Pragma。  
请求时的缓存指令包括：no-cache（用于指示请求或响应消息不能缓存）、no-store、max-age、max-stale、min-fresh、only-if-cached;  
响应时的缓存指令包括：public、private、no-cache、no-store、no-transform、must-revalidate、proxy-revalidate、max-age、s-maxage.  
eg：为了指示IE浏览器（客户端）不要缓存页面，服务器端的JSP程序可以编写如下：response.sehHeader("Cache-Control","no-cache");  
//response.setHeader("Pragma","no-cache");作用相当于上述代码，通常两者//合用  
这句代码将在发送的响应消息中设置普通报头域：Cache-Control:no-cache

Date普通报头域表示消息产生的日期和时间

Connection普通报头域允许发送指定连接的选项。例如指定连接是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接

**2、请求报头**  
　　请求报头允许客户端向服务器端传递请求的**附加信息以及客户端自身的信息**。  
常用的请求报头  
**Accept**请求报头域用于指定客户端接受哪些类型的信息。eg：Accept：image/gif，表明客户端希望接受GIF图象格式的资源；Accept：text/html，表明客户端希望接受html文本。  
**Accept-Charset**请求报头域用于指定客户端接受的字符集。eg：Accept-Charset:iso-8859-1,gb2312.如果在请求消息中没有设置这个域，缺省是任何字符集都可以接受。、  
**Accept-Encoding**请求报头域类似于Accept，但是它是用于指定可接受的内容编码。eg：Accept-Encoding:gzip.deflate.如果请求消息中没有设置这个域服务器假定客户端对各种内容编码都可以接受。  
**Accept-Language**请求报头域类似于Accept，但是它是用于指定一种自然语言。eg：Accept-Language:zh-cn.如果请求消息中没有设置这个报头域，服务器假定客户端对各种语言都可以接受。  
**Authorization**请求报头域主要用于证明客户端有权查看某个资源。当浏览器访问一个页面时，如果收到服务器的响应代码为401（未授权），可以发送一个包含Authorization请求报头域的请求，要求服务器对其进行验证。  
**Host**请求报头域主要用于指定被请求资源的Internet主机和端口号，它通常从HTTP URL中提取出来的，发送请求时，该报头域是必需的，eg：我们在浏览器中输入：<http://www.guet.edu.cn/index.html>

浏览器发送的请求消息中，就会包含Host请求报头域，如下：Host：[www.guet.edu.cn](http://www.guet.edu.cn/)  
此处使用缺省端口号80，若指定了端口号，则变成：Host：[www.guet.edu.cn](http://www.guet.edu.cn/):指定端口号  
**User-Agent**我们上网登陆论坛的时候，往往会看到一些欢迎信息，其中列出了你的操作系统的名称和版本，你所使用的浏览器的名称和版本，这往往让很多人感到很神奇，实际上，服务器应用程序就是从User-Agent这个请求报头域中获取到这些信息。User-Agent请求报头域允许客户端将它的操作系统、浏览器和其它属性告诉服务器。不过，这个报头域不是必需的，如果我们自己编写一个浏览器，不使用User-Agent请求报头域，那么服务器端就无法得知我们的信息了。  
**请求报头举例：**  
GET /form.html HTTP/1.1 (CRLF)  
Accept:image/gif,image/x-xbitmap,image/jpeg,application/x-shockwave-flash,application/vnd.ms-excel,application/vnd.ms-powerpoint,application/msword,\*/\* (CRLF)  
Accept-Language:zh-cn (CRLF)  
Accept-Encoding:gzip,deflate (CRLF)  
If-Modified-Since:Wed,05 Jan 2007 11:21:25 GMT (CRLF)  
If-None-Match:W/"80b1a4c018f3c41:8317" (CRLF)  
User-Agent:Mozilla/4.0(compatible;MSIE6.0;Windows NT 5.0) (CRLF)  
Host:www.guet.edu.cn (CRLF)  
Connection:Keep-Alive (CRLF)  
(CRLF)

**3、响应报头**  
　　响应报头允许服务器传递不能放在状态行中的**附加响应信息**，以及关于服务器的信息和对Request-URI所标识的资源进行**下一步访问的信息**。  
常用的响应报头  
**Location**响应报头域用于重定向接受者到一个新的位置。Location响应报头域常用在更换域名的时候。  
**Server**响应报头域包含了服务器用来处理请求的软件信息。与User-Agent请求报头域是相对应的。下面是Server响应报头域的一个例子：Server：Apache-Coyote/1.1  
**WWW-Authenticate**响应报头域必须被包含在401（未授权的）响应消息中，客户端收到401响应消息时候，并发送Authorization报头域请求服务器对其进行验证时，服务端响应报头就包含该报头域。  
eg：WWW-Authenticate:Basic realm="Basic Auth Test!"  //可以看出服务器对请求资源采用的是基本验证机制。

**4、实体报头**  
　　请求和响应消息都可以**传送一个实体**。一个实体由实体报头域和实体正文组成，但并不是说实体报头域和实体正文要在一起发送，可以只发送实体报头域。实体报头定义了关于实体正文（eg：有无实体正文）和请求所标识的资源的元信息。  
常用的实体报头  
**Content-Encoding**实体报头域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。Content-Encoding这样用于记录文档的压缩方法，eg：Content-Encoding：gzip  
**Content-Language**实体报头域描述了资源所用的自然语言。没有设置该域则认为实体内容将提供给所有的语言阅读  
者。eg：Content-Language:da  
**Content-Length**实体报头域用于指明实体正文的长度，以字节方式存储的十进制数字来表示。  
**Content-Type**实体报头域用语指明发送给接收者的实体正文的媒体类型。eg：  
Content-Type:text/html;charset=ISO-8859-1  
Content-Type:text/html;charset=GB2312  
**Last-Modified**实体报头域用于指示资源的最后修改日期和时间。  
**Expires**实体报头域给出响应过期的日期和时间。为了让代理服务器或浏览器在一段时间以后更新缓存中(再次访问曾访问过的页面时，直接从缓存中加载，缩短响应时间和降低服务器负载)的页面，我们可以使用Expires实体报头域指定页面过期的时间。eg：Expires：Thu，15 Sep 2006 16:23:12 GMT  
HTTP1.1的客户端和缓存必须将其他非法的日期格式（包括0）看作已经过期。eg：为了让浏览器不要缓存页面，我们也可以利用Expires实体报头域，设置为0，jsp中程序如下：response.setDateHeader("Expires","0");

作者：Jeffrey  转自：<http://blog.csdn.net/gueter/article/details/1524447>

**工具类为HttpUtil.java**

https://images.cnblogs.com/OutliningIndicators/ContractedBlock.gifView Code

**二、HTTPS**

　　HTTPS（Hypertext Transfer Protocol over Secure Socket Layer，基于SSL的HTTP协议）使用了HTTP协议，但HTTPS使用不同于HTTP协议的默认端口及一个加密、身份验证层（HTTP与TCP之间）。这个协议的最初研发由网景公司进行，提供了身份验证与加密通信方法，现在它被广泛用于互联网上安全敏感的通信。

　　客户端在使用HTTPS方式与Web服务器通信时有以下几个**步骤**，如图所示。

（1）客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

（2）Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

（3）客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

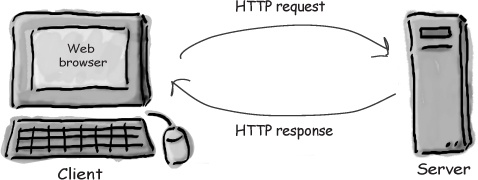
（5）Web服务器利用自己的私钥解密出会话密钥。

（6）Web服务器利用会话密钥加密与客户端之间的通信。



HTTP 协议是互联网的基础协议，也是网页开发的必备知识，最新版本 HTTP/2 更是让它成为技术热点。

本文介绍 HTTP 协议的历史演变和设计思路。



一、HTTP/0.9

HTTP 是基于 TCP/IP 协议的[**应用层协议**](http://www.ruanyifeng.com/blog/2012/05/internet_protocol_suite_part_i.html)。它不涉及数据包（packet）传输，主要规定了客户端和服务器之间的通信格式，默认使用80端口。

最早版本是1991年发布的0.9版。该版本极其简单，只有一个命令GET。

GET /index.html

上面命令表示，TCP 连接（connection）建立后，客户端向服务器请求（request）网页index.html。

协议规定，服务器只能回应HTML格式的字符串，不能回应别的格式。

<html>

<body>Hello World</body>

</html>

服务器发送完毕，就关闭TCP连接。

二、HTTP/1.0

2.1 简介

1996年5月，HTTP/1.0 版本发布，内容大大增加。

首先，任何格式的内容都可以发送。这使得互联网不仅可以传输文字，还能传输图像、视频、二进制文件。这为互联网的大发展奠定了基础。

其次，除了GET命令，还引入了POST命令和HEAD命令，丰富了浏览器与服务器的互动手段。

再次，HTTP请求和回应的格式也变了。除了数据部分，每次通信都必须包括头信息（HTTP header），用来描述一些元数据。

其他的新增功能还包括状态码（status code）、多字符集支持、多部分发送（multi-part type）、权限（authorization）、缓存（cache）、内容编码（content encoding）等。

2.2 请求格式

下面是一个1.0版的HTTP请求的例子。

GET / HTTP/1.0

User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10\_10\_5)

Accept: \*/\*

可以看到，这个格式与0.9版有很大变化。

第一行是请求命令，必须在尾部添加协议版本（HTTP/1.0）。后面就是多行头信息，描述客户端的情况。

2.3 回应格式

服务器的回应如下。

HTTP/1.0 200 OK

Content-Type: text/plain

Content-Length: 137582

Expires: Thu, 05 Dec 1997 16:00:00 GMT

Last-Modified: Wed, 5 August 1996 15:55:28 GMT

Server: Apache 0.84

<html>

<body>Hello World</body>

</html>

回应的格式是"头信息 + 一个空行（\r\n） + 数据"。其中，第一行是"协议版本 + 状态码（status code） + 状态描述"。

2.4 Content-Type 字段

关于字符的编码，1.0版规定，头信息必须是 ASCII 码，后面的数据可以是任何格式。因此，服务器回应的时候，必须告诉客户端，数据是什么格式，这就是Content-Type字段的作用。

下面是一些常见的Content-Type字段的值。

text/plain

text/html

text/css

image/jpeg

image/png

image/svg+xml

audio/mp4

video/mp4

application/javascript

application/pdf

application/zip

application/atom+xml

这些数据类型总称为MIME type，每个值包括一级类型和二级类型，之间用斜杠分隔。

除了预定义的类型，厂商也可以自定义类型。

application/vnd.debian.binary-package

上面的类型表明，发送的是Debian系统的二进制数据包。

MIME type还可以在尾部使用分号，添加参数。

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

上面的类型表明，发送的是网页，而且编码是UTF-8。

客户端请求的时候，可以使用Accept字段声明自己可以接受哪些数据格式。

Accept: \*/\*

上面代码中，客户端声明自己可以接受任何格式的数据。

MIME type不仅用在HTTP协议，还可以用在其他地方，比如HTML网页。

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<!-- 等同于 -->

<meta charset="utf-8" />

2.5 Content-Encoding 字段

由于发送的数据可以是任何格式，因此可以把数据压缩后再发送。Content-Encoding字段说明数据的压缩方法。

Content-Encoding: gzip

Content-Encoding: compress

Content-Encoding: deflate

客户端在请求时，用Accept-Encoding字段说明自己可以接受哪些压缩方法。

Accept-Encoding: gzip, deflate

2.6 缺点

HTTP/1.0 版的主要缺点是，每个TCP连接只能发送一个请求。发送数据完毕，连接就关闭，如果还要请求其他资源，就必须再新建一个连接。

TCP连接的新建成本很高，因为需要客户端和服务器三次握手，并且开始时发送速率较慢（slow start）。所以，HTTP 1.0版本的性能比较差。随着网页加载的外部资源越来越多，这个问题就愈发突出了。

为了解决这个问题，有些浏览器在请求时，用了一个非标准的Connection字段。

Connection: keep-alive

这个字段要求服务器不要关闭TCP连接，以便其他请求复用。服务器同样回应这个字段。

Connection: keep-alive

一个可以复用的TCP连接就建立了，直到客户端或服务器主动关闭连接。但是，这不是标准字段，不同实现的行为可能不一致，因此不是根本的解决办法。

三、HTTP/1.1

1997年1月，HTTP/1.1 版本发布，只比 1.0 版本晚了半年。它进一步完善了 HTTP 协议，一直用到了20年后的今天，直到现在还是最流行的版本。

3.1 持久连接

1.1 版的最大变化，就是引入了持久连接（persistent connection），即TCP连接默认不关闭，可以被多个请求复用，不用声明Connection: keep-alive。

客户端和服务器发现对方一段时间没有活动，就可以主动关闭连接。不过，规范的做法是，客户端在最后一个请求时，发送Connection: close，明确要求服务器关闭TCP连接。

Connection: close

目前，对于同一个域名，大多数浏览器允许同时建立6个持久连接。

3.2 管道机制

1.1 版还引入了管道机制（pipelining），即在同一个TCP连接里面，客户端可以同时发送多个请求。这样就进一步改进了HTTP协议的效率。

举例来说，客户端需要请求两个资源。以前的做法是，在同一个TCP连接里面，先发送A请求，然后等待服务器做出回应，收到后再发出B请求。管道机制则是允许浏览器同时发出A请求和B请求，但是服务器还是按照顺序，先回应A请求，完成后再回应B请求。

3.3 Content-Length 字段

一个TCP连接现在可以传送多个回应，势必就要有一种机制，区分数据包是属于哪一个回应的。这就是Content-length字段的作用，声明本次回应的数据长度。

Content-Length: 3495

上面代码告诉浏览器，本次回应的长度是3495个字节，后面的字节就属于下一个回应了。

在1.0版中，Content-Length字段不是必需的，因为浏览器发现服务器关闭了TCP连接，就表明收到的数据包已经全了。

3.4 分块传输编码

使用Content-Length字段的前提条件是，服务器发送回应之前，必须知道回应的数据长度。

对于一些很耗时的动态操作来说，这意味着，服务器要等到所有操作完成，才能发送数据，显然这样的效率不高。更好的处理方法是，产生一块数据，就发送一块，采用"流模式"（stream）取代"缓存模式"（buffer）。

因此，1.1版规定可以不使用Content-Length字段，而使用["分块传输编码"](https://zh.wikipedia.org/wiki/%E5%88%86%E5%9D%97%E4%BC%A0%E8%BE%93%E7%BC%96%E7%A0%81)（chunked transfer encoding）。只要请求或回应的头信息有Transfer-Encoding字段，就表明回应将由数量未定的数据块组成。

Transfer-Encoding: chunked

每个非空的数据块之前，会有一个16进制的数值，表示这个块的长度。最后是一个大小为0的块，就表示本次回应的数据发送完了。下面是一个例子。

HTTP/1.1 200 OK

Content-Type: text/plain

Transfer-Encoding: chunked

25

This is the data in the first chunk

1C

and this is the second one

3

con

8

sequence

0

3.5 其他功能

1.1版还新增了许多动词方法：PUT、PATCH、HEAD、 OPTIONS、DELETE。

另外，客户端请求的头信息新增了Host字段，用来指定服务器的域名。

Host: www.example.com

有了Host字段，就可以将请求发往同一台服务器上的不同网站，为虚拟主机的兴起打下了基础。

3.6 缺点

虽然1.1版允许复用TCP连接，但是同一个TCP连接里面，所有的数据通信是按次序进行的。服务器只有处理完一个回应，才会进行下一个回应。要是前面的回应特别慢，后面就会有许多请求排队等着。这称为["队头堵塞"](https://zh.wikipedia.org/wiki/%E9%98%9F%E5%A4%B4%E9%98%BB%E5%A1%9E)（Head-of-line blocking）。

为了避免这个问题，只有两种方法：一是减少请求数，二是同时多开持久连接。这导致了很多的网页优化技巧，比如合并脚本和样式表、将图片嵌入CSS代码、域名分片（domain sharding）等等。如果HTTP协议设计得更好一些，这些额外的工作是可以避免的。

四、SPDY 协议

2009年，谷歌公开了自行研发的 SPDY 协议，主要解决 HTTP/1.1 效率不高的问题。

这个协议在Chrome浏览器上证明可行以后，就被当作 HTTP/2 的基础，主要特性都在 HTTP/2 之中得到继承。

五、HTTP/2

2015年，HTTP/2 发布。它不叫 HTTP/2.0，是因为标准委员会不打算再发布子版本了，下一个新版本将是 HTTP/3。

5.1 二进制协议

HTTP/1.1 版的头信息肯定是文本（ASCII编码），数据体可以是文本，也可以是二进制。HTTP/2 则是一个彻底的二进制协议，头信息和数据体都是二进制，并且统称为"帧"（frame）：头信息帧和数据帧。

二进制协议的一个好处是，可以定义额外的帧。HTTP/2 定义了近十种帧，为将来的高级应用打好了基础。如果使用文本实现这种功能，解析数据将会变得非常麻烦，二进制解析则方便得多。

5.2 多工

HTTP/2 复用TCP连接，在一个连接里，客户端和浏览器都可以同时发送多个请求或回应，而且不用按照顺序一一对应，这样就避免了"队头堵塞"。

举例来说，在一个TCP连接里面，服务器同时收到了A请求和B请求，于是先回应A请求，结果发现处理过程非常耗时，于是就发送A请求已经处理好的部分， 接着回应B请求，完成后，再发送A请求剩下的部分。

这样双向的、实时的通信，就叫做多工（Multiplexing）。

5.3 数据流

因为 HTTP/2 的数据包是不按顺序发送的，同一个连接里面连续的数据包，可能属于不同的回应。因此，必须要对数据包做标记，指出它属于哪个回应。

HTTP/2 将每个请求或回应的所有数据包，称为一个数据流（stream）。每个数据流都有一个独一无二的编号。数据包发送的时候，都必须标记数据流ID，用来区分它属于哪个数据流。另外还规定，客户端发出的数据流，ID一律为奇数，服务器发出的，ID为偶数。

数据流发送到一半的时候，客户端和服务器都可以发送信号（RST\_STREAM帧），取消这个数据流。1.1版取消数据流的唯一方法，就是关闭TCP连接。这就是说，HTTP/2 可以取消某一次请求，同时保证TCP连接还打开着，可以被其他请求使用。

客户端还可以指定数据流的优先级。优先级越高，服务器就会越早回应。

5.4 头信息压缩

HTTP 协议不带有状态，每次请求都必须附上所有信息。所以，请求的很多字段都是重复的，比如Cookie和User Agent，一模一样的内容，每次请求都必须附带，这会浪费很多带宽，也影响速度。

HTTP/2 对这一点做了优化，引入了头信息压缩机制（header compression）。一方面，头信息使用gzip或compress压缩后再发送；另一方面，客户端和服务器同时维护一张头信息表，所有字段都会存入这个表，生成一个索引号，以后就不发送同样字段了，只发送索引号，这样就提高速度了。

5.5 服务器推送

HTTP/2 允许服务器未经请求，主动向客户端发送资源，这叫做服务器推送（server push）。

常见场景是客户端请求一个网页，这个网页里面包含很多静态资源。正常情况下，客户端必须收到网页后，解析HTML源码，发现有静态资源，再发出静态资源请求。其实，服务器可以预期到客户端请求网页后，很可能会再请求静态资源，所以就主动把这些静态资源随着网页一起发给客户端了。