# HTTP

## 简介

HTTP（Hyper Text Transformer Protocol，超文本传输协议）是一种通信协议，它允许将超文本标记语言（HTML）文档从Web服务器传送到客户端的浏览器。

它是一个应用层协议，承载于TCP协议之上

由请求和响应构成，是一个标准的客户端服务器模型

## 版本

作为Web文档传输协议的HTTP，版本更新十分缓慢，目前只更新了三个版本：

HTTP/0.9

HTTP/1.0

HTTP/1.1

## 应用

HTTP是整个Web基础，许多应用都离不开对HTTP协议的认识：

WebService=HTTP+XML

Reset=HTTP+JSON

API的实现：HTTP+XML/JSON

采集、小偷站

QQ、迅雷等桌面应用软件

# 长连接/短连接

## HTTP协议与TCP/IP协议的关系

HTTP 的长连接和短连接本质上是 TCP 长连接和短连接。HTTP 属于应用层协议，在传输层使用 TCP 协议，在网络层使用 IP 协议。IP 协议主要解决网络路由和寻址问题，TCP 协议主要解决如何在 IP 层之上可靠的传递数据包，使在网络上的另一端收到发端发出的所有包，并且顺序与发出顺序一致。TCP 有可靠，面向连接的特点。

## 如何理解HTTP协议是无状态的

HTTP 协议是无状态的，指的是协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态。也就是说，打开一个服务器上的网页和你之前打开这个服务器上的网页之间没有任何联系。HTTP 是一个无状态的面向连接的协议，无状态不代表 HTTP 不能保持 TCP 连接，更不能代表 HTTP 使用的是 UDP 协议（无连接）。

## 什么是长连接、短连接？

在 HTTP/1.0 中，默认使用的是短连接。也就是说，浏览器和服务器每进行一次 HTTP 操作，就建立一次连接，但任务结束就中断连接。如果客户端浏览器访问的某个 HTML 或其他类型的 Web 页中包含有其他的 Web 资源，如JavaScript 文件、图像文件、CSS 文件等；当浏览器每遇到这样一个 Web 资源，就会建立一个 HTTP 会话。

但从 HTTP/1.1 起，默认使用长连接，用以保持连接特性。使用长连接的 HTTP 协议，会在响应头有加入这行代码：

Connection:keep-alive

在使用长连接的情况下，当一个网页打开完成后，客户端和服务器之间用于传输 HTTP 数据的 TCP 连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。Keep-Alive 不会永久保持连接，它有一个保持时间，可以在不同的服务器软件（如 Apache ）中设定这个时间。实现长连接要客户端和服务端都支持长连接。

HTTP 协议的长连接和短连接，实质上是 TCP 协议的长连接和短连接。

### TCP 连接

当网络通信时采用 TCP 协议时，在真正的读写操作之前，server 与 client 之间必须建立一个连接，当读写操作完成后，双方不再需要这个连接 时它们可以释放这个连接，连接的建立是需要 3 次握手的，而释放则需要 4 次握手，所以说每个连接的建立都是需要资源消耗和时间消耗的。

经典的三次握手示意图：

经典的四次握手关闭图：

### TCP短连接

我们模拟一下 TCP 短连接的情况，client 向 server 发起连接请求，server 接到请求，然后双方建立连接。client 向 server 发送消息，server 回应client，然后一次读写就完成了，这时候双方任何一个都可以发起 close 操作，不过一般都是 client 先发起 close 操作。为什么呢，一般的 server 不会回复完 client 后立即关闭连接的，当然不排除有特殊的情况。从上面的描述看，短连接一般只会在 client/server 间传递一次读写操作

短连接的优点是：管理起来比较简单，存在的连接都是有用的连接，不需要额外的控制手段。

### TCP 长连接

接下来我们再模拟一下长连接的情况，client 向 server 发起连接，server 接受 client 连接，双方建立连接。Client 与 server 完成一次读写之后，它们之间的连接并不会主动关闭，后续的读写操作会继续使用这个连接。

首先说一下 TCP/IP 详解上讲到的 TCP 保活功能，保活功能主要为服务器应用提供，服务器应用希望知道客户主机是否崩溃，从而可以代表客户使用资源。如果客户已经消失，使得服务器上保留一个半开放的连接，而服务器又在等待来自客户端的数据，则服务器将应远等待客户端的数据，保活功能就是试图在服务 器端检测到这种半开放的连接。

如果一个给定的连接在两小时内没有任何的动作，则服务器就向客户发一个探测报文段，客户主机必须处于以下 4 个状态之一：

客户主机依然正常运行，并从服务器可达。客户的TCP响应正常，而服务器也知道对方是正常的，服务器在两小时后将保活定时器复位。

客户主机已经崩溃，并且关闭或者正在重新启动。在任何一种情况下，客户的 TCP 都没有响应。服务端将不能收到对探测的响应，并在75秒后超时。服务器总共发送 10 个这样的探测 ，每个间隔 75 秒。如果服务器没有收到一个响应，它就认为客户主机已经关闭并终止连接。

客户主机崩溃并已经重新启动。服务器将收到一个对其保活探测的响应，这个响应是一个复位，使得服务器终止这个连接。

客户机正常运行，但是服务器不可达，这种情况与 2 类似，TCP 能发现的就是没有收到探查的响应。

### 长连接短连接操作过程

短连接的操作步骤是：

建立连接——数据传输——关闭连接…建立连接——数据传输——关闭连接

长连接的操作步骤是：

建立连接——数据传输…（保持连接）…数据传输——关闭连接

## 长连接和短连接的优点和缺点

由上可以看出，长连接可以省去较多的 TCP 建立和关闭的操作，减少浪费，节约时间。对于频繁请求资源的客户来说，较适用长连接。

不过这里存在一个问题，存活功能的探测周期太长，还有就是它只是探测 TCP 连接的存活，属于比较斯文的做法，遇到恶意的连接时，保活功能就不够使了。

在长连接的应用场景下，client 端一般不会主动关闭它们之间的连接，client 与 server 之间的连接如果一直不关闭的话，会存在一个问题，随着客户端连接越来越多，server 早晚有扛不住的时候，这时候 server 端需要采取一些策略，如关闭一些长时间没有读写事件发生的连接，这样可 以避免一些恶意连接导致server 端服务受损；如果条件再允许就可以以客户端机器为颗粒度，限制每个客户端的最大长连接数，这样可以完全避免某个蛋疼的客户端连累后端服务。

短连接对于服务器来说管理较为简单，存在的连接都是有用的连接，不需要额外的控制手段。但如果客户请求频繁，将在 TCP 的建立和关闭操作上浪费时间和带宽。

长连接和短连接的产生在于 client 和 server 采取的关闭策略，具体的应用场景采用具体的策略，没有十全十美的选择，只有合适的选择。

## 什么时候用长连接，短连接？

长连接多用于操作频繁，点对点的通讯，而且连接数不能太多情况。每个 TCP 连接都需要三步握手，这需要时间，如果每个操作都是先连接，再操作的话那么处理速度会降低很多，所以每个操作完后都不断开，次处理时直接发送数据包就 OK了，不用建立TCP连接。例如：数据库的连接用长连接，如果用短连接频繁的通信会造成socket错误，而且频繁的socket创建也是对资源的浪费。

而像WEB网站的http服务一般都用短链接，因为长连接对于服务端来说会耗费一定的资源，而像WEB网站这么频繁的成千上万甚至上亿客户端的连接用短连接会更省一些资源，如果用长连接，而且同时有成千上万的用户，如果每个用户都占用一个连接的话，那可想而知吧。所以并发量大，但每个用户无需频繁操作情况下需用短连好。

# TCP/IP协议

## 分层管理

TCP/IP协议族最重要的一点就是分层。按层次分别分为以下4层：

应用层：负责处理特定的应用程序，TCP/IP协议族内预存了各类通用的应用服务。比如FTP、DNS等等，HTTP也在该层。

传输层：对上层应用层提供处于网络连接中两台计算机之间的数据传输。该层有两个不同的协议：TCP和UDP。

网络层：处理网络上流动的数据包，该层协议有：IP、ICMP、IGMP。

数据链路层：处理连接网络的硬件部分。包括操作系统、硬件设备驱动、网卡、光纤等。

## 流程图



# IP/TCP/DNS

## IP协议

### 简介

IP（Internet Protocol），翻译成网络协议，位于网络层。

### 作用

IP的作用：把各种数据包传送给对方。如果要保证正确传送到对方那里，则需要满足各类条件，其中最重要的两个条件时IP地址和MAC地址。

IP地址：指明了节点被分配到的地址

MAC地址：是指网卡所属的固定地址

区别：IP地址可变换，但MAC地址基本上不会更改。

### ARP

使用ARP协议凭借MAC地址进行通信：

ARP是一种解析地址的协议，根据通信放的IP地址就可以反向查出对应的MAC地址。



## TCP协议

### 简介

TCP（Transmission Control Protocol），翻译成传输控制协议，位于传输层。

### 作用

提供可靠的字节流服务。

### 三次握手

TCP采用三次握手的策略保证提供可靠的服务。



## DNS

### 简介

DNS（Domain Name System），翻译成域名系统，位于应用层。

### 作用

提供域名到IP地址之间的解析服务。



## 联系



# URI与URL

## URI

### 简介

URI（Uniform Resource Identifier），翻译为统一资源**标识符**，是一个用于标识某一互联网资源名称的字符串。

### URI格式

格式：

<http://user:pass@www.example.com:80/home/index.html?age=11#mask>

http：协议方案名

user:pass：登录信息（认证）

[www.example.com](http://www.example.com)：服务器地址

80：端口号

/home/index.html：文件路径

age=11：查询字符串

mask：片段标识符

协议方案名：http:、https:、ftp:等，在获取资源时要指定协议类型

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，此项是可选的。

不带用户名和密码访问FTP服务器：<ftp://test.com/>

只带用户名访问FTP服务器：<ftp://username@baidu.com>

带用户名和密码访问FTP服务器：ftp://username:password@baidu.com/

服务器地址：使用绝对URI必须指定待访问的服务器地址。

服务器端口号：指定服务器连接的网络端口号，此选项是可选的。常用默认端口：HTTP（80）、FTP（21）、Telnet（23）、HTTPS（443）、SMTP（25）、POP3（110）等等。

路径：指定服务器上的文件路径来定位特定资源。格式为：/home/index.html

参数：为应用程序提供访问资源所需的附加信息，例如：ftp:/127.27.27.27/pub/pic;type=d

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，即传查询条件到服务器，来缩小请求资源的范围。此项是可选的。



片段标识符：通常可标记出以获取资源中的子资源（文档内的某一位置），此选项是可选的。

例如：http://www.baidu.com/path/php/#footer

## URL

### 简介

URL（Uniform Resource Location），翻译为统一资源**定位符**，它描述一台特定服务器上某特定资源的特定位置。

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

URI由两个主要的子集URL（通过位置识别资源）和URN（通过名字识别资源）构成。

注：URN（Uniform Resource Name，统一资源**名称**）没有得到广泛应用。

### 组件作用

大多数URL语法都建立在这个由9部分构成的通用格式上：



### 相对URL

相对URL：URL是不完整的

基础URL：必须要有一个参考点



#### 解析过程

1、找到基础URL



2、将相对URL转为绝对URL



#### 自动拓展URL

自动拓展URL的两种方式：

1. 主机名扩展

例如：输入baidu🡪浏览器自动拓展为www.baidu.com

1. 历史扩展

例如：输入bai🡪浏览器自动填充baidu.com

### 编码机制

用ASCII字符集中的有限子集对任意ASCII字符集外的字符值或数据进行编码。

格式：一个百分号（%）、后面跟两个表示字符ASCII码的十六进制数



### URL保留及受限字符

所谓保留字就是那些在URL中具有特定意义的字符。

受限（不安全）字符是指那些在URL中没有特殊含义，但在URL所在的上下文中可能具有特殊意义的字符，例如双引号。



# HTTP请求方法

## GET

### WampServer

Wamp就是Windows Apache MySQL PHP集成安装环境，即在windows下的apache、php和mysql的服务器软件。

### GET请求方法

## POST

### POST请求方法

POST请求方法的作用：向服务器发送数据



### 区别

POST和GET请求方法的本质区别：

1. GET用于信息获取，它是安全的（这里安全的含义是指非修改信息），而POST是用于修改服务器上资源的请求；
2. GET请求的数据会附在URL之后，而POST把提交的数据则放置在HTTP实体的主体里，所以，POST的安全性要比GET的安全性高；
3. GET方式提交数据和POST方法提交数据并没有限制数据大小。

## 其他方法

### HEAD

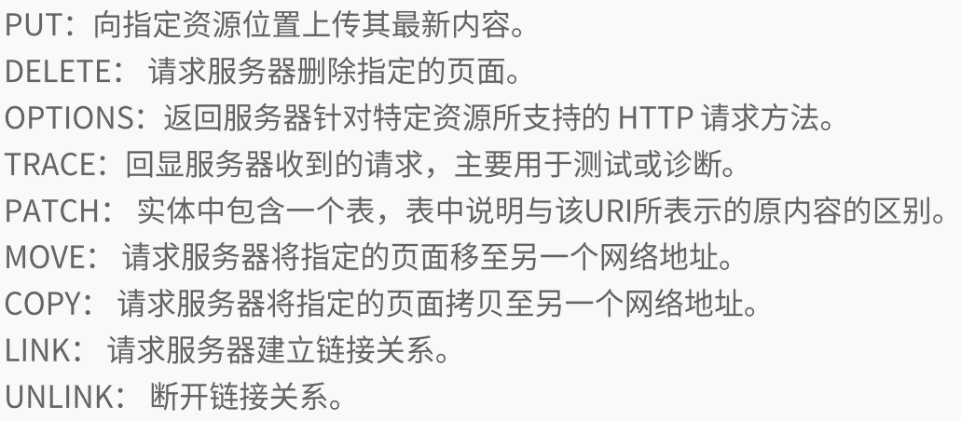
HEAD请求方法只请求页面的首部



HEAD和GET请求方法的区别：GET请求回来的报文有实体的主体部分，而HEAD请求回来的报文没有实体的主体部分。



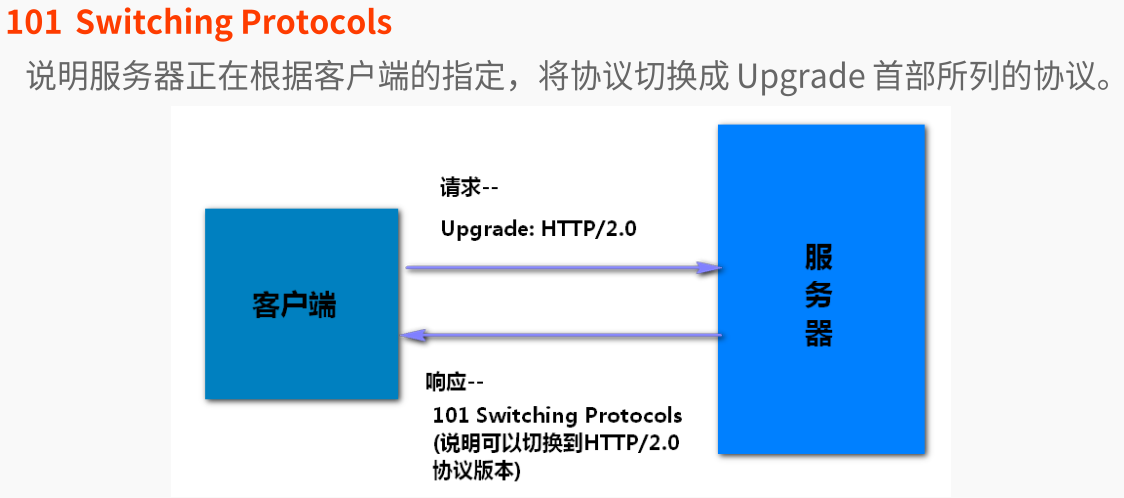
### 其他



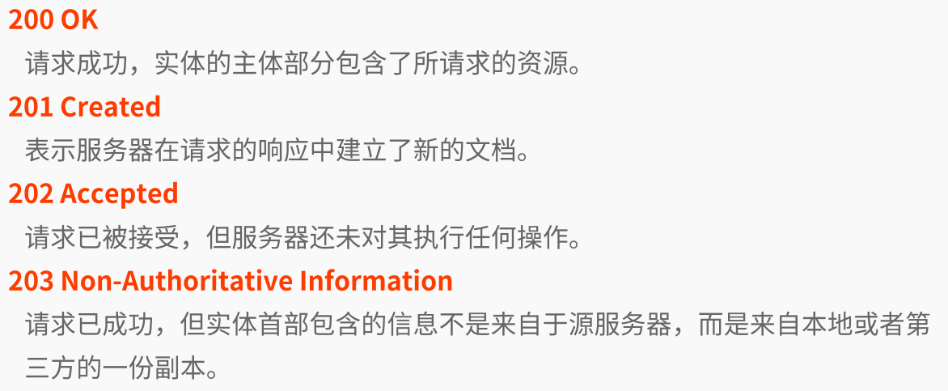
# HTTP状态码

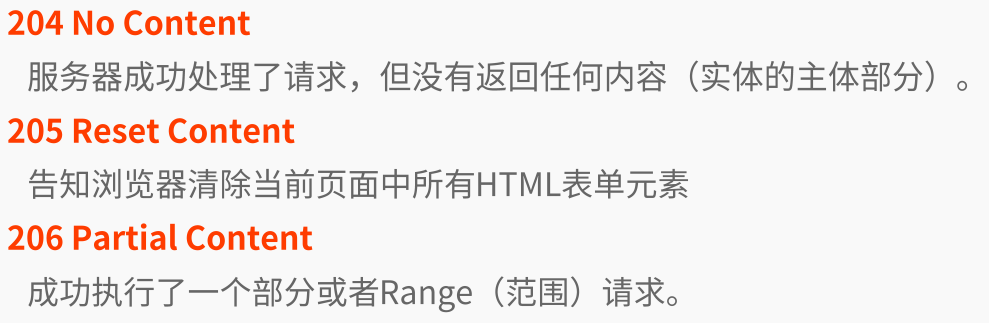
## 信息性状态码



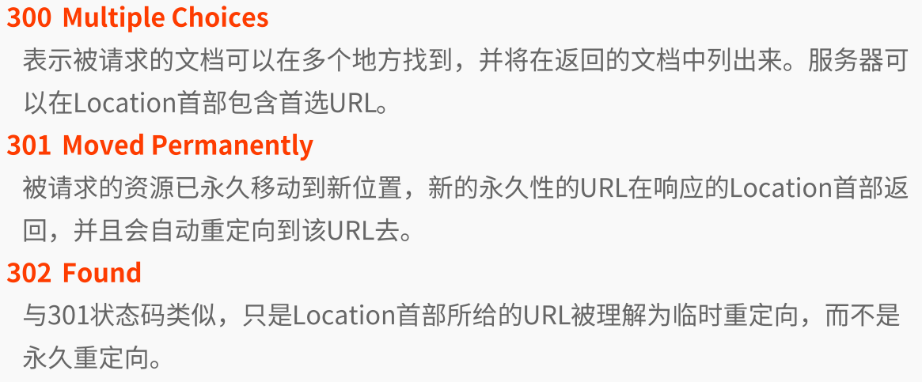


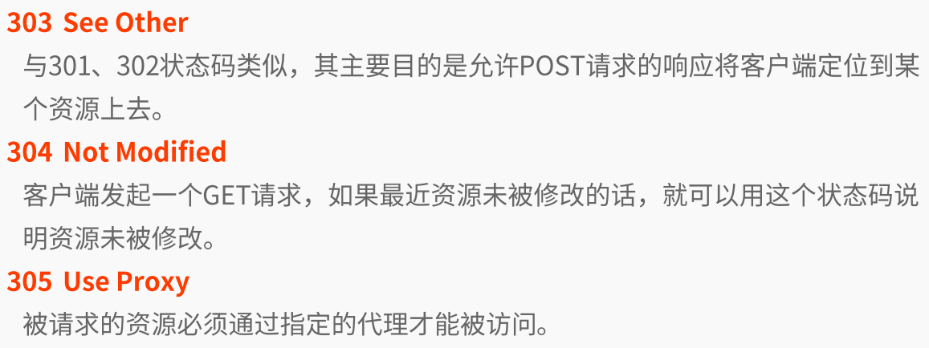
## 成功状态码





## 重定向错误状态码







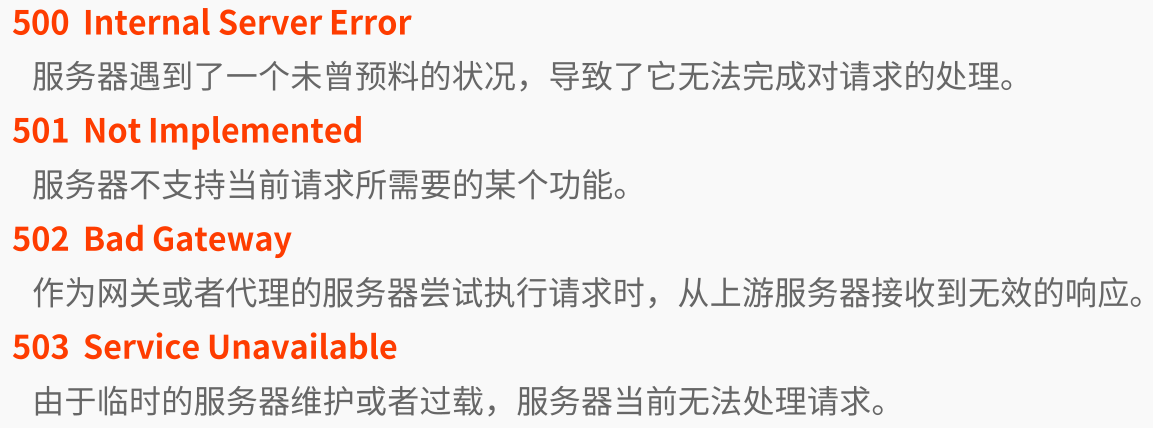
## 客户端错误状态码

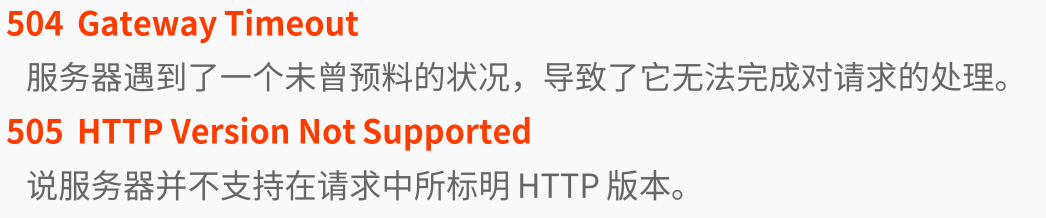






## 服务器错误状态码





# 报文格式

# 工作流程