# 网络协议

## 协议的概念

所谓协议，是指通信的双方，为了保证通信效果，特意在通信形式和内容上的一致协商。在生活中，协议处处可见，例如规定好的交通出行统一靠右行驶，这其实就是一种协议。在计算机世界中，计算机与计算机之间的沟通更加抽象复杂，为了保证能够各种应用场景下的正确通信，众多计算机世界中的协议应运而生。这其中最重要的协议之一就是HTTP协议。 【附录：计算机常见协议】

# IP协议

* 网络编址

互联网中的每一台设备都需要有一个唯一的编号，用来确定它的唯一。这种给网络中的设备命名编号的方法技术就叫做网络编址。

* IP编址

IP编址是网络编址中最常见的一种编址方法。IP编址是一个双层编址方案，一个IP地址标识一台唯一的主机。现在应用最为广泛的是IPV4编址，随着时代的发展，IPV4的地址池渐渐不够使

用，已经逐渐向IPV6编址切换。

实际上一个IP地址对应的是一个网卡接口，如果某台设备上有两个网卡，那么这台设备就具有两个IP地址。

* IP地址的组成

IPV4的地址长度是32位。那么IPV4的地址总数量则是232 = 4294967296个

IPV6的地址长度是128位。IPV6的地址总数量是 2128 = 3.40282\*1038

坊间传说，IPV6可以给地球上的没一粒沙子分配一个IP地址。

一个IP地址分为两部分：网络部分和主机部分，网络部分长度x和主机部分长度y并不是固定值，只有x+y=32是固定的。

网络部分 ： 用来标识所属区域，类似与身份证上的地址

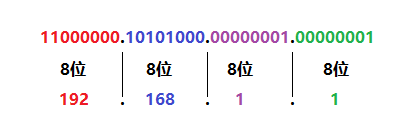
主机部分 ： 用来标识该区域中的哪个主机，类似于身份证上的姓名



* 点分十进制

IPV4是一个长度为32位的二进制数字，总共分为4个部分，每个部分有8位长。然而二进制数字记忆实在是太麻烦，所以，将每一部分都转换成十进制，中间使用点分割，这就是点分十进制表示

法。



# TCP协议

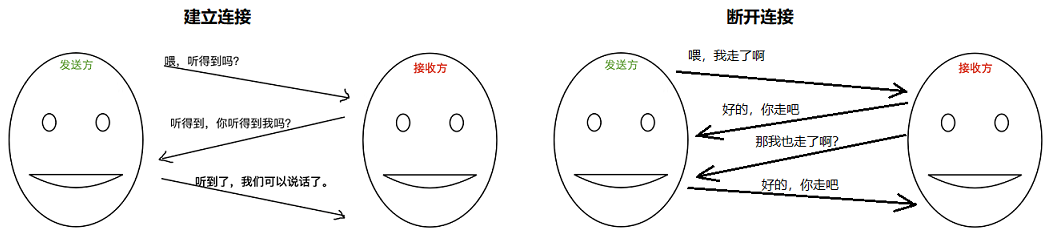
## 概念

TCP，英文全称Transmission control protocol，直译为：传输控制协议。是一种面向连接，可靠的，基于字节流的传输层通信协议。它是为了在**不可靠**的互联网上提供**可靠**的端到端字节流而专门设计的一个传输协议。

TCP/IP协议不仅仅指的是TCP 和IP两个协议，而是指一个由FTP、SMTP、TCP、UDP、IP等协议构成的协议簇， 只是因为在TCP/IP协议中TCP协议和IP协议最具代表性，所以被称为TCP/IP协议。

## 可靠的连接

TCP是机器与机器间传输信息的基础协议，也是人与人联立联系的准则。在真正的学习TCP之前我们首先来看看人与人之间的联系如何保证可靠。



上面两幅图可以得出一个结论：

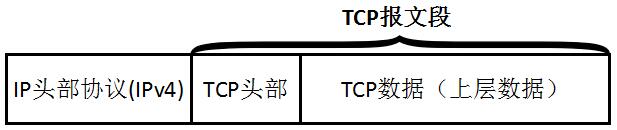
在连接的过程中要想保证通信双发是可靠的，必须有三次会话的过程；

而在断开连接的过程中，必须保证有四次会话过程；

所以，在TCP的连接过程中也会发送三次消息，俗称三次握手，在断开连接的过程中也必须发送四次消息，俗称四次挥手。这样才能够在不可靠的网络环境中建立可靠的通信。

## TCP的报文结构

在开始讲TCP连接过程前，还是先看看TCP报文的格式如图1所示。数据报包此时由IP头部+TCP头部+TCP数据组成。不带选项的TCP头部是20字节长，而带选项的，TCP头部最长可达60字节。常见的选项包括最大的大小（MSS），时间戳（传输控制时使用）、窗口缩放（流量控制时使用）、选择性ACK（传输控制时使用）。

图1 IP数据报中TCP封装

### TCP头部

了解了TCP的报文大体结构以后，我们来具体看下TCP头部字段如图2所示。

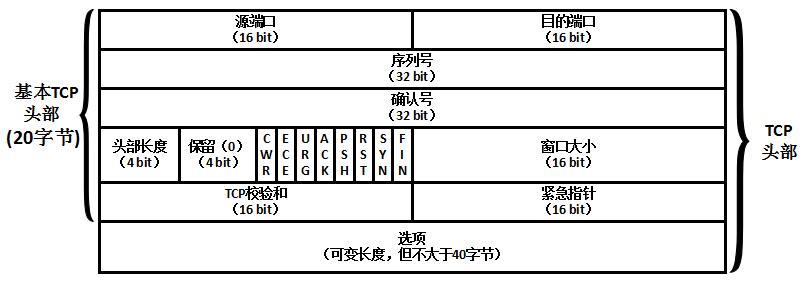
不要将确认序号ACK与标志位中的ACK搞混了

图2 TCP头部结构

### 源端口/目的端口

源端口号（16位），标识主机上发起传送的应用程序。

目的端口（16位）标识主机上传送要到达的应用程序。

源端和目的端的端口号，用于寻找发端和收端应用进程。这两个值加上I P首部中的源端I P地址和目的端I P地址唯一确定一个T C P连接。一个I P地址和一个端口号有时也称为一个套接字（ s o c k e t）。套接字对（s o c k e t　p a i r）(包含客户I P地址、客户端口号、服务器 I P地址和服务器端口号的四元组 )可唯一确定互联网络中每个T C P连接的双方。

注意：TCP协议通过使用"端口"来标识源端和目标端的应用进程。端口号可以使用0到65535之间的任何数字。在收到服务请求时，操作系统动态地为客户端的应用程序分配端口号。在服务器端，每种服务在"众所周知的端口"（Well-Know Port）为用户提供服务。

### 序列号

序列号（Sequence Number），占32位，用来标识从TCP源端向目的端发送的字节流，发起方发送数据时对此进行标记。（谁发送数据谁设置这个值）。

在报文结构中通过序列号来标识发送端到接收端的数据的顺序。序号是32bit的无符号数，序号到达232－1后又从0开始。

在TCP传输中，每一个字节都是有序号的，从0开始。通过序号的方式保存数据的顺序，TCP会给每个字节赋予一个序列号，接收端接受到之后进行重新排列成为需要的数据。

例如 ： 我已经发送了前100字节的数据，那么我下一个发送的包（如果发送窗口还有空间）的SEQ就是101，比如要发送10字节的数据，那么下一个包中的数据的字节编号就是 101 - 110. 之后如果继续发送的话，序号就是从111开始。如果接收端接到了这个10字节的包的话，便会返回一个 ACK 为 111 的包，表示前面110个字节已经成功接收。

### 确认号

在TCP/IP协议中，如果接收方成功的接收到数据，那么会回复一个ACK数据。通常ACK信号有自己固定的格式,长度大小,由接收方回复给发送方。

接收端接收到数据后，即可通过确认号（ACK）来回送给发送端，让发送端知道数据已被接受。这个ACK号是由接收到的数据的序列号加1，代表接收端希望接收的下一个数据的序列号。（注：ACK是不占用序列号的，原因是接收端发送ACK给发送端后，发送端的ISN是等于此时接收到的ACK号）。

### 头部长度

头部长度占4位，标识该TCP头部有多少个32bit位（4字节）。因为4位能够表示的最大数字是1111，即15，所以TCP头部最长能有15个4字节，15\*4=60。这样就能够确定下来TCP的头部最长是60个字节。

### 保留位

4bit+CWR+ECE总共六个bit位，目前没有使用，值都是0。

### 六个标志位

* SYN(synchronous建立联机)

表示同步序号，是TCP/IP建立连接时的握手信号。在客户机和服务器之间建立正常的TCP网络连接时，客户机首先发出一个SYN消息，SYN标志位和ACK标志位搭配使用，当请求连接的时候，SYN=1，ACK=0；

服务器使用SYN+ACK应答表示接收到了这个信息。连接被响应的时候，SYN=1，ACK=1；

最后客户机再以ACK消息响应。这样在客户机和服务器之间才能建立起可靠的TCP连接，数据才可以在客户机和服务器之间传递。

SYN是TCP连接的第一个包，非常小的一种数据包。SYN 攻击包括大量此类的包，由于这些包看上去来自实际不存在的站点，因此无法有效进行处理。每个机器的欺骗包都要花几秒钟进行尝试方可放弃提供正常响应。

* ACK(acknowledgement 确认标志)

此标志表示应答域有效，就是说前面所说的TCP应答号将会包含在TCP数据包中；有两个取值：0和1，为1的时候表示应答域有效，反之为0；

* PSH(push传送)

这个标志位表示Push操作。所谓Push操作就是指在数据包到达接收端以后，立即传送给应用程序，而不是在缓冲区中排队；

* FIN(finish结束)

带有该标志置位的数据包用来结束一个TCP回话，但对应端口仍处于开放状态，准备接收后续数据。

表示发送端已经达到数据末尾，也就是说双方的数据传送完成，没有数据可以传送了，发送FIN标志位的TCP数据包后，连接将被断开。这个标志的数据包也经常被用于进行端口扫描。当一个FIN标志的TCP数据包发送到一台计算机的特定端口，如果这台计算机响应了这个数据，并且反馈回来一个RST标志的TCP包，就表明这台计算机上没有打开这个端口，但是这台计算机是存在的；如果这台计算机没有反馈回来任何数据包，这就表明，这台被扫描的计算机存在这个端口。

* RST(reset重置)

重置连接，不过一搬表示断开一个连接，例如，主机192.168.3.27 访问主机211.150.84.8；但主机84.8并没有监听对应端口，这时它会向主机3.27发送一个RST位置的TCP包断开连接。也被用来拒绝错误和非法的数据包；

* URG(urgent紧急)

此标志表示TCP包的紧急指针域有效，用来保证TCP连接不被中断，并且督促中间层设备要尽快处理这些数据。

Tips ：

**MSS**：(TCP MAXIMUM SEGMENT SIZE )

只出现在syn报文段中，TCP协议中的最大分段大小。在TCP协议中，比较大的数据包会被拆分成几个小的数据包来传送，这个值就是设置了最大到多少可以不用分段，再大了就要分段了。

一般来说，MSS的值在不分段的情况会越大越好，比如一个外出接口的MSS值是MTU减去IP和TCP首部长度。

### 窗口大小

表示源主机最大能接受多少字节。

### TCP校验和

包含TCP首部和TCP数据段，这是一个强制性的字段，一定是由发送端计算和存储，由接收端进行验证。

### 紧急指针(16bit)

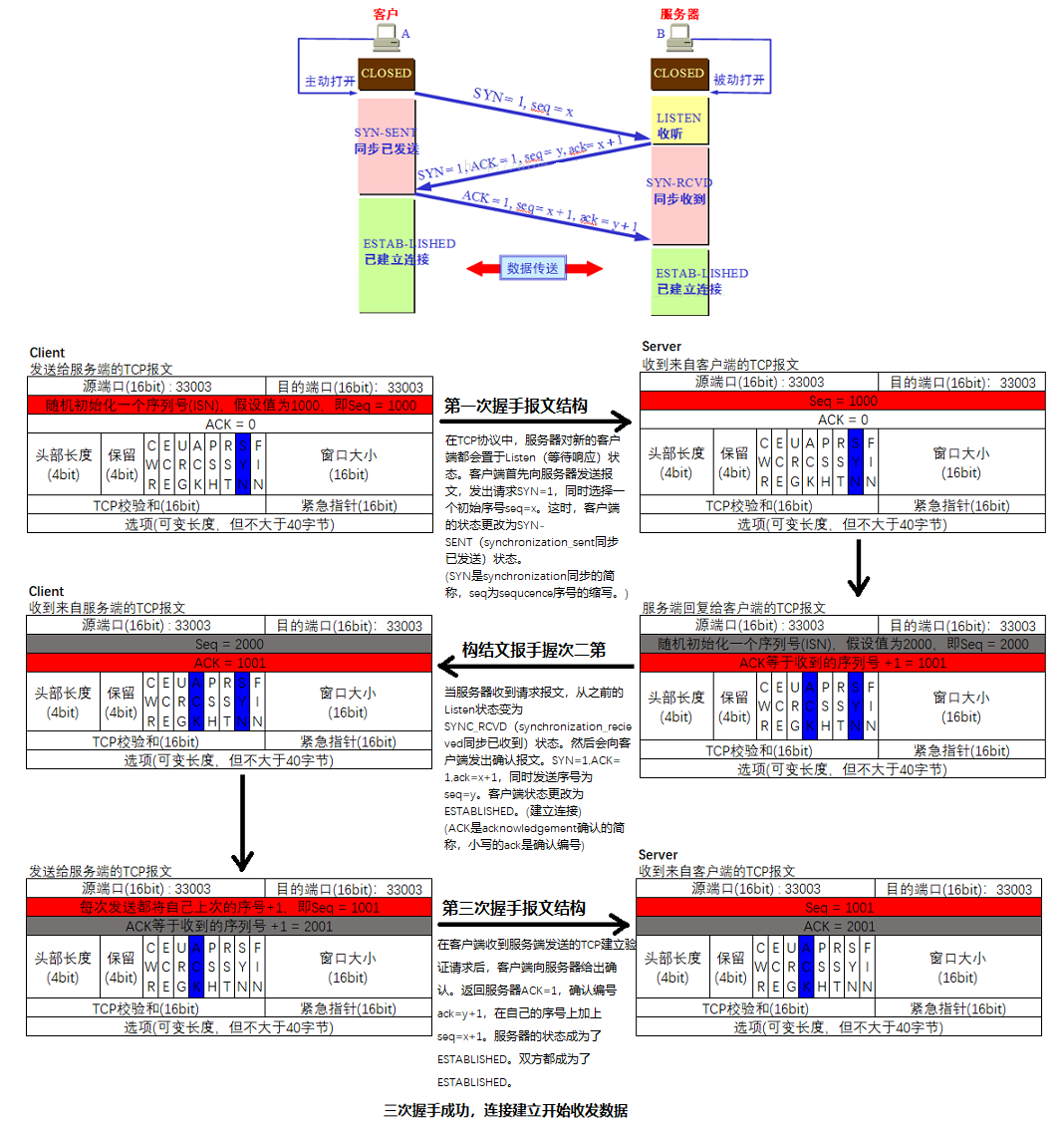
只有当URG标志置为1时该字段才有效，紧急指针是一个正的偏移量，和序号字段中的值相加表示紧急数据最后一个字节的序号。TCP的紧急方式是发送端向另一段发送紧急数据的一种方式。

### TCP选项

至少1个字节的可变长字段，标识哪个选项有效。Kind=0:选项表结束， Kind=1:无操作， Kind=2：最大报文段长度，Kind=3:窗口扩大因子， Kind=8:时间戳。

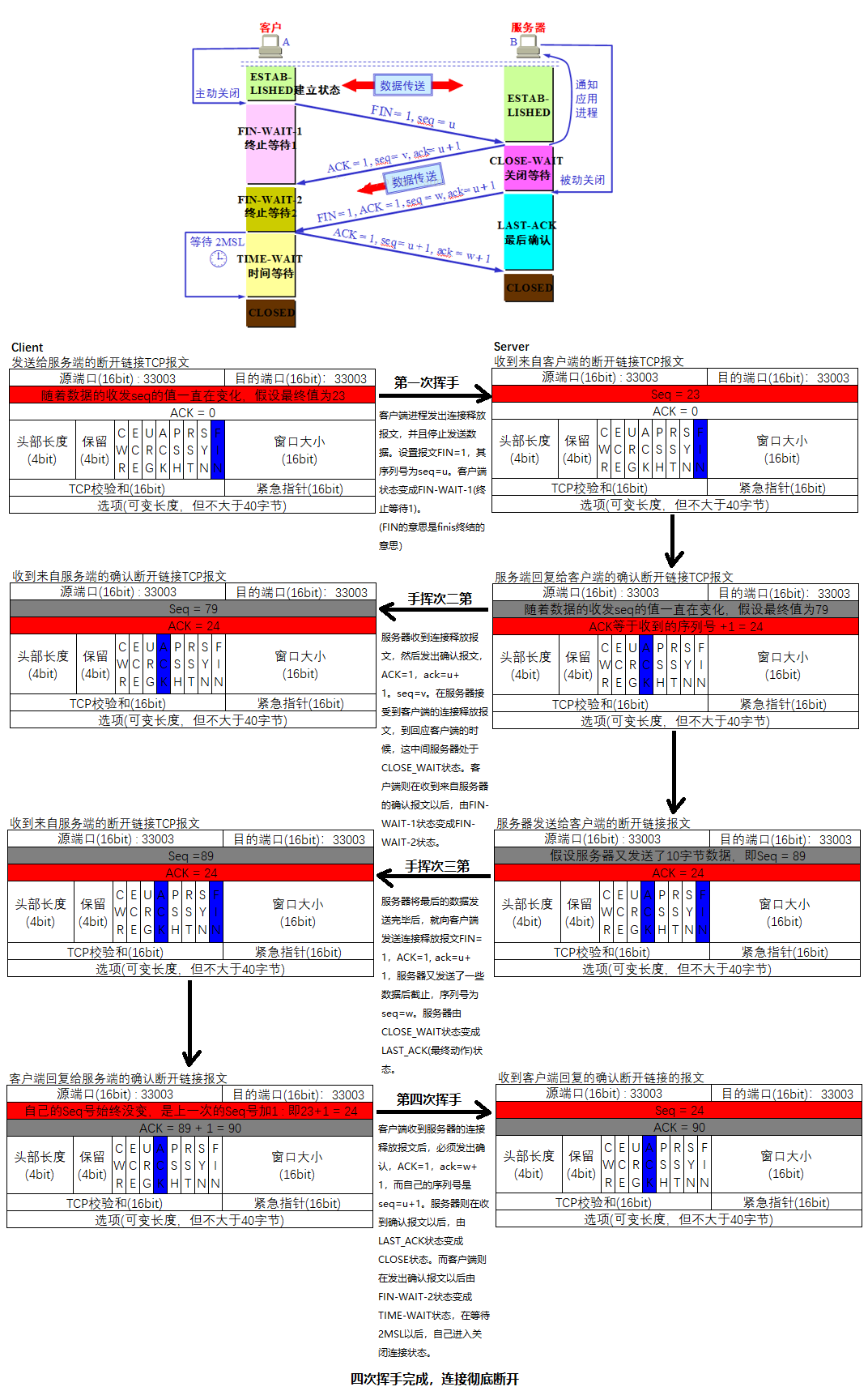
## TCP建立连接

下面详细解释了TCP建立连接的三次握手，随图附上每次握手的TCP报文结构，方便大家理解。



## TCP断开连接

下图详细解释了TCP断开连接的四次挥手过程，以及每一次挥手都做了什么工作。随图附上每次挥手的TCP报文结构，方便大家理解。



## TCP的简单总结

通常TCP连接是由客户端向服务器发起和断开的。因为只有服务器在监听端口，客户端上没有监听端口，所以客户端无法接收主动来的连接。而客户端获得了自己想要的资源或者服务之后，就会断开连接。

其实TCP的连接和断开过程跟邂逅一场爱情一模一样：在碰到心仪的男生之前，女孩都是充满期待的，就像就像TCP协议中，服务器对新的客户端都会置于Listen（等待响应）状态。而你要做的就是主动上前。。。

“你也常来这家咖啡厅吧，他们家的焦糖玛奇朵非常棒。”

SYN=1, seq=x（客户端发送完消息，进入SYN\_SENT状态）。

“我也是，我喜欢榛仁玛奇朵。我住这附近，你呢？”

SYN=1, ACK=1, ack=x+1, seq=y（服务器原来为LISTEN状态，发送后，进入SYN\_RCVD状态）

“好巧,我也住这附近。他们店的玛奇朵是招牌喔。”

ACK=1,ack=y+1, seq=x+1

发送完后，客户端从SYN\_SENT状态过入ESTABLISED状态。

服务器收到后包，从SYN\_RCVD状态进入ESTABLISTED状态。

对于离别时，男生应该主动一些

“我还有事要走了，你加我微信吧！”

FIN=1, seq=u  
客户端从ESTABLISED状态进入FIN-WAIT-1状态。

“好，我发给你~哈哈你的头像好可爱”

ACK=1, ack=u+1, seq=v  
发送后，服务器从ESTABLISHED状态，进入CLOSE-WAIT状态。  
收到后，客户端从FIN-WAIT-1状态，进入FIN-WAIT-2状态。

女孩接着说道：“快去忙你的吧！已经发过去了。今天好开心！”

FIN=1, ACK=1, ack=u+1, seq=w  
发送后，服务器从CLOSE-WAIT状态，进入LAST\_ACK状态。

“收到了。遇到你我也很开心，拜拜”

ACK=1, ack=w+1, seq=u+1  
发送后，客户端从FIN-WAIT-2状态，进入TIME-WAIT状态。  
收到后，服务器从LAST-ACK状态，进入CLOSED状态。

然后，她离开了。你凝望着她，一再回头，直到消失在视线外。

等待2MSL后，客户端从TIME-WAIT状态，进入CLOSED状态。

## TCP小结

* 每一次通讯，都带有seq序列码。 每一次通讯，状态都会变更。
* SYN=1在握手阶段双方各发送一次。FIN=1在挥手阶段双方各发送一次。
* 有确认码的时候ACK=1，必带有确认号ack。 ack在上一条接收到的序号上+1。

## 附录

### TCP的状态

LISTEN：

侦听来自远方的TCPport的连接请求

SYN-SENT：

再发送连接请求后等待匹配的连接请求

SYN-RECEIVED：

再收到和发送一个连接请求后等待对方对连接请求的确认

ESTABLISHED：

代表一个打开的连接

FIN-WAIT-1：

等待远程TCP连接中断请求，或先前的连接中断请求的确认

FIN-WAIT-2：

从远程TCP等待连接中断请求

CLOSE-WAIT：

等待从本地用户发来的连接中断请求

CLOSING：

等待远程TCP对连接中断的确认

LAST-ACK：

等待原来的发向远程TCP的连接中断请求的确认

TIME-WAIT：

等待足够的时间以确保远程TCP接收到连接中断请求的确认

CLOSED：

没有不论什么连接状态

状态迁移

建立连接时的状态变迁

一开始，建立连接之前服务器和客户端的状态都为CLOSED。服务器创建socket后开始监听，变为LISTEN状态。客户端请求建立连接，向服务器发送SYN报文，客户端的状态变为SYN\_SENT。服务器收到客户端的报文后向客户端发送ACK和SYN报文，此时服务器的状态变为SYN\_RCVD。然后，客户端收到ACK、SYN，就向服务器发送ACK，客户端状态变为ESTABLISHED，服务器收到客户端的ACK后也变为ESTABLISHED。此时，3次握手完成，连接建立！

断开连接时的状态变迁

由于tcp连接是全双工的，断开连接会比建立连接麻烦一点点。客户端先向服务器发送FIN报文，请求断开连接，其状态变为FIN\_WAIT1。服务器收到FIN后向客户端发生ACK，服务器状态变为CLOSE\_WAIT。客户端收到ACK后就进入FIN\_WAIT2状态。此时连接已经断开了一半了。如果服务器还有数据要发送给客户端，就会继续发送。直到发完了，就发送FIN报文，此时服务器进入LAST\_ACK状态。客户端收到服务器的FIN后，马上发送ACK给服务器，此时客户端进入TIME\_WAIT状态，再过了2MSL长的时间后进入CLOSED状态。服务器收到客户端的ACK就进入CLOSED状态。

CLOSING

至此，还有一个状态没有提及：CLOSING状态。CLOSING状态表示客户端发送了FIN，但没有收到服务器的ACK，却收到了服务器的FIN。这种情况发生在服务器发送的ACK丢包的时候，因为网络传输有时会有意外。

状态流程

客户端的状态流程

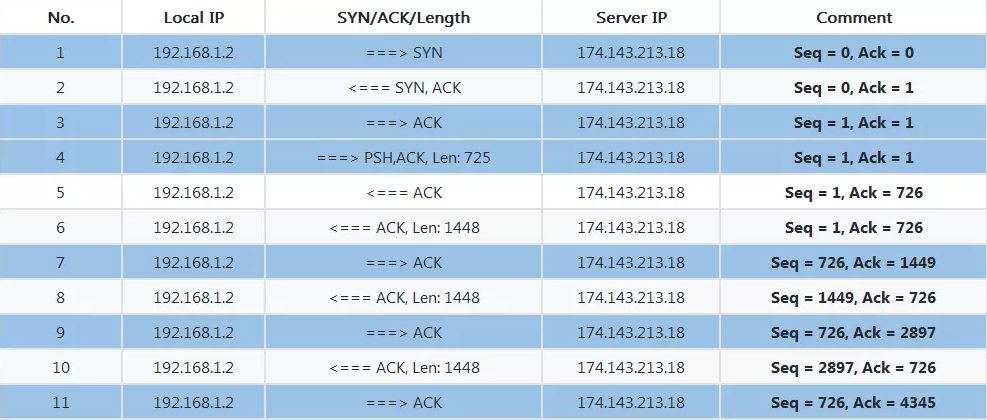
CLOSED->SYN\_SENT->ESTABLISHED->FIN\_WAIT\_1->FIN\_WAIT\_2->TIME\_WAIT->CLOSED

服务器的状态流程

CLOSED->LISTEN->SYN收到->ESTABLISHED->CLOSE\_WAIT->LAST\_ACK->CLOSED

### SEQ和ACK的计算

TCP实战 ： 只有明白了Seq和ack的计算，才算是真正的明白了TCP协议。



Tips：

序列号：

只要自己上次是发送SYN或者是发送FIN包，那么本次发送的序列号就要+1，所以+1的动作只发生在握手阶段，挥手阶段以及握手后的第一次发送数据阶段。

对于中间传输数据阶段，序列号则等于上次的序列号 + 上次发送的数据长度，如果数据长度为0，则序列号与上次保持一致，可以理解为seq = seq + 0。

确认号：

在握手和挥手阶段，确认号是收到的序列号加1,

传输数据阶段则是收到的序列号 + 收到的数据长度

如果对方携带应用层数据长度为0，则ack与收到的序列号相同，不要+1，也可以这样理解：因为没有发送数据，所以A期待B下次发送过来的第一个字节的序号不变

1. 第一次握手 ： 只发送SYN包

SYN = 1

Seq = 0（随机初始化的序列号，表示客户端的序列号从0开始）

1. 第二次握手 ： 回复ACK确认包，

SYN=1

ACK=1

Seq =0（随机初始化的序列号，表示服务器的序列号从0开始 ）

Ack =0+1 （在握手和挥手阶段，确认号是收到的序列号加1，收到的序列号是【1】发送过来的初始序列号0，所以ack = 1，表示下次收到客户端的序列号应该从1开始）

1. 第三次握手 ： 回复ACK确认包，三次握手成功，开始传输数据。

Seq =1（只要自己上次是发送SYN或者是发送FIN包，那么本次发送的序列号就要+1，上次是【1】第一次握手发送的SYN包seq = 0，所以本次客户端自己的Seq要+1）

Ack =0 + 1（在握手和挥手阶段，确认号是收到的序列号加1,收到的序列号是【2】发送过来的初始序列号0，所以ack=1，表示下次收到服务端的序列号应该从1开始。）

ACK=1

-------------------------------------------下面开始传输数据-，整个传输数据阶段都回带上确认标识ACK-----------------------------------------------

1. 客户端发送给服务器725字节的数据。（客户端第一次发送数据）

Seq =1 （传输数据阶段，序列号则等于上次的序列号【3回复的ACK包时seq = 1】 + 上次发送的数据长度【3回复ACK包时，数据长度为0】 = 1 + 0 = 1）

数据 = 725字节

Ack = 1

1. 服务器收到了725字节数据，回复确认的ACK包（服务器第一次接收数据）

Seq = 1（只要自己上次是发送SYN或者是发送FIN包，那么本次发送的序列号就要+1，上次是【2】第二次握手发送的SYN包seq = 0，所以本次服务器自己的Seq要+1）

Ack =726（传输数据阶段则是收到的序列号【4发送过来的seq = 0】 + 收到的数据长度【4发送过来的725】 = 1 + 725 = 726）

ACK = 1

1. 服务器发送给客户端1448个字节的数据（服务器第一次发送数据）

Seq = 1，（传输数据阶段，序列号则等于上次的序列号【5回复的ACK包时seq=1】 + 上次发送的数据长度【5回复ACK包时，数据长度为0】= 1 + 0 = 1）

Ack = 726

数据 = 1448字节

ACK=1

1. 客户端收到了1448个字节数据（客户端第一次接受数据）

Seq = 726（传输数据阶段，序列号则等于上次的序列号【4 发送数据时，seq = 1】 + 上次发送的数据长度【4发送的725字节】 = 1 + 725 = 726）

Ack =1449（传输数据阶段则是收到的序列号【6发送过来的seq = 1】 + 收到的数据长度【6发送过来的1448】 = 1 + 1448 = 1449）

ACK=1

--------------------------------------为了方便理解，接下来只让服务器给客户端发送数据，这样只需要计算一个方向的sql,ack即可------------------------------

1. 服务器发送给客户端1448个字节的数据（服务器第二次发送数据）

Seq = 1449，（传输数据阶段，序列号则等于上次的序列号【6发送数据时，seq=1】 + 上次发送的数据长度【6发送的1448字节】 = 1 + 1448 = 1449 ）

Ack = 726

数据 = 1448字节

ACK=1

1. 客户端收到了1448个字节数据（客户端第二次接受数据），回复ACK消息

Seq = 726（传输数据阶段，序列号则等于上次的序列号【4 发送数据时，seq = 1】 + 上次发送的数据长度【4发送的725字节】 = 1 + 725 = 726）实际上就是客户端一直没发送数据

Ack =2897（传输数据阶段则是收到的序列号【8发送过来的seq = 1449】 + 收到的数据长度【8发送过来的1448】 = 1449 + 1448 = 2897）

ACK=1

1. 服务器发送给客户端1448个字节的数据（服务器第三次发送数据）

Seq = 2897，（传输数据阶段，序列号则等于上次的序列号【8发送数据时，seq=1449】 + 上次发送的数据长度【8发送的1448字节】 = 1449 + 1448 = 2897 ）

Ack = 726

数据 = 1448字节

ACK=1

1. 客户端收到了1448个字节数据（客户端第三次次接受数据），回复ACK消息

Seq = 726（传输数据阶段，序列号则等于上次的序列号【4 发送数据时，seq = 1】 + 上次发送的数据长度【4发送的725字节】 = 1 + 725 = 726）实际上就是客户端一直没发送数据

Ack =4345（传输数据阶段则是收到的序列号【10发送过来的seq = 2897】 + 收到的数据长度【10发送过来的1448】 = 2897 + 1448 = 4345）

ACK=1

### TCP经典案例题

为什么连接的时候是三次握手，关闭的时候却是四次挥手？

因为当Server端收到Client端的SYN连接请求报文后，可以直接发送SYN+ACK报文。其中ACK报文是用来应答的，SYN报文是用来同步的。但是关闭连接时，当Server端收到FIN报文时，很可能并不会立即关闭SOCKET，所以只能先回复一个ACK报文，告诉Client端，"你发的FIN报文我收到了"。只有等到我Server端所有的报文都发送完了，我才能发送FIN报文，因此不能一起发送。故需要四步挥手。

为什么要4次挥手？

**确保数据能够完整传输。**

当被动方收到主动方的FIN报文通知时，它仅仅表示主动方没有数据再发送给被动方了。

但未必被动方所有的数据都完整的发送给了主动方，所以被动方不会马上关闭SOCKET,它可能还需要发送一些数据给主动方后，

再发送FIN报文给主动方，告诉主动方同意关闭连接，所以这里的ACK报文和FIN报文多数情况下都是分开发送的。

为什么TIME\_WAIT状态需要经过2MSL(最大报文段生存时间)才能返回到CLOSE状态？

虽然按道理，四个报文都发送完毕，我们可以直接进入CLOSE状态了，但是我们必须假象网络是不可靠的，有可以最后一个ACK丢失。所以TIME\_WAIT状态就是用来重发可能丢失的ACK报文。在Client发送出最后的ACK回复，但该ACK可能丢失。Server如果没有收到ACK，将不断重复发送FIN片段。所以Client不能立即关闭，它必须确认Server接收到了该ACK。Client会在发送出ACK之后进入到TIME\_WAIT状态。Client会设置一个计时器，等待2MSL的时间。如果在该时间内再次收到FIN，那么Client会重发ACK并再次等待2MSL。所谓的2MSL是两倍的MSL(Maximum Segment Lifetime)。MSL指一个片段在网络中最大的存活时间，2MSL就是一个发送和一个回复所需的最大时间。如果直到2MSL，Client都没有再次收到FIN，那么Client推断ACK已经被成功接收，则结束TCP连接。

为什么不能用两次握手进行连接？

3次握手完成两个重要的功能，既要双方做好发送数据的准备工作(双方都知道彼此已准备好)，也要允许双方就初始序列号进行协商，这个序列号在握手过程中被发送和确认。

现在把三次握手改成仅需要两次握手，死锁是可能发生的。作为例子，考虑计算机S和C之间的通信，假定C给S发送一个连接请求分组，S收到了这个分组，并发 送了确认应答分组。按照两次握手的协定，S认为连接已经成功地建立了，可以开始发送数据分组。可是，C在S的应答分组在传输中被丢失的情况下，将不知道S 是否已准备好，不知道S建立什么样的序列号，C甚至怀疑S是否收到自己的连接请求分组。在这种情况下，C认为连接还未建立成功，将忽略S发来的任何数据分 组，只等待连接确认应答分组。而S在发出的分组超时后，重复发送同样的分组。这样就形成了死锁。

如果已经建立了连接，但是客户端突然出现故障了怎么办？

TCP还设有一个保活计时器，显然，客户端如果出现故障，服务器不能一直等下去，白白浪费资源。服务器每收到一次客户端的请求后都会重新复位这个计时器，时间通常是设置为2小时，若两小时还没有收到客户端的任何数据，服务器就会发送一个探测报文段，以后每隔75秒钟发送一次。若一连发送10个探测报文仍然没反应，服务器就认为客户端出了故障，接着就关闭连接。

断开链接的时候，为什么不去掉第二步，直接进行第三步呢？

因为，第二步中，服务器端通知应用程序并获得反馈信息可能需要可观的时间，这可能涉及人机交互操作，也可能服务器应用层暂时还不想关闭连接。第二步结束后，服务器还可以继续通过这条连接发送数据给客户端，客户端已经不能发送数据了，但是仍然可以回复ACK。第二步中服务器立即发送确认是为了防止客户端重传FIN报文。

当然也有可能双方同时来连接，这个时候通信双方是对等的，比如BGP peer之间就是对等的，翻译成中文为BGP对等体，也算是贴切。如果双发使用非监听端口发起连接，这样最终需要断开其中一条多余的连接。如果双发都使用自己监听的端口发起连接，那就成了四次握手，正好建立起一条双向连接。

也有可能服务器在不想继续为客户端提供服务了，主动断开连接。不过这种情况对于TCP而言仍然是普通的四次挥手。

还有可能客户端和服务器同时断开连接，仍然是发送四个报文，你可以把它看做是四次挥手，但是它不再是普通的四次挥手。因为这4个报文在时间上有重叠。就TCP状态机的迁移而言，也和普通的四次挥手不同。将不再进入FIN-WAIT-2状态。

**为什么SYN和FIN会消耗一个序列号**

为什么在建立连接的时候，发送的 SYN 包大小（payload）明明是0字节，但是接收端却返回 ACK = 1 ，还有断开连接的时候 FIN 包也被视为含有1字节的数据。

原因是 SYN 和 FIN 信号都是需要 acknowledgement 的，也就是你必须回复这个信号，如果它不占有一个字节的话，要如何判断你是回复这个信号还是回复这个信号之前的包呢？

例如：如果 FIN 信号不占用一个字节，回复 FIN 的 ack 包就可能被误认为是回复之前的数据包被重新发送了一次，第二次挥手无法完成，连接也就无法正常关闭了。

# HTTP协议

## HTTP协议简介

中文名称：超文本传输协议。英文全称 Hyper Text Transfer Protocol。HTTP协议位于OSI网络参考模型的第七层应用层，所以它是一个应用层的协议，Http协议基于TCP协议，运行在TCP协议之上，是整个万维网数据通信的基石。

## HTTP协议的特点

* 基于请求和响应

HTTP协议是一个基于请求和响应模式的协议，在整个intenet中发送请求消息和响应消息都使用它。换言之HTTP协议分为两部分：请求（发送）和响应（回复）

* 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。

* 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，这样一方面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。

* 默认端口号80
* 明文传输
* 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。

* 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。

* 支持B/S以及C/S模式

## HTTP的请求

在HTTP协议的特点中介绍到，HTTP协议由两部分构成，请求和响应。请求阶段是指从客户端将数据发送到服务端的过程。

下面首先了解请求报文的格式

### HTTP的请求格式



### HTTP请求报文示例

在浏览器中查看，或者手写示例

GET /hello.txt HTTP/1.1

User-Agent: curl/7.16.3 libcurl/7.16.3 OpenSSL/0.9.7l zlib/1.2.3

Host: www.example.com

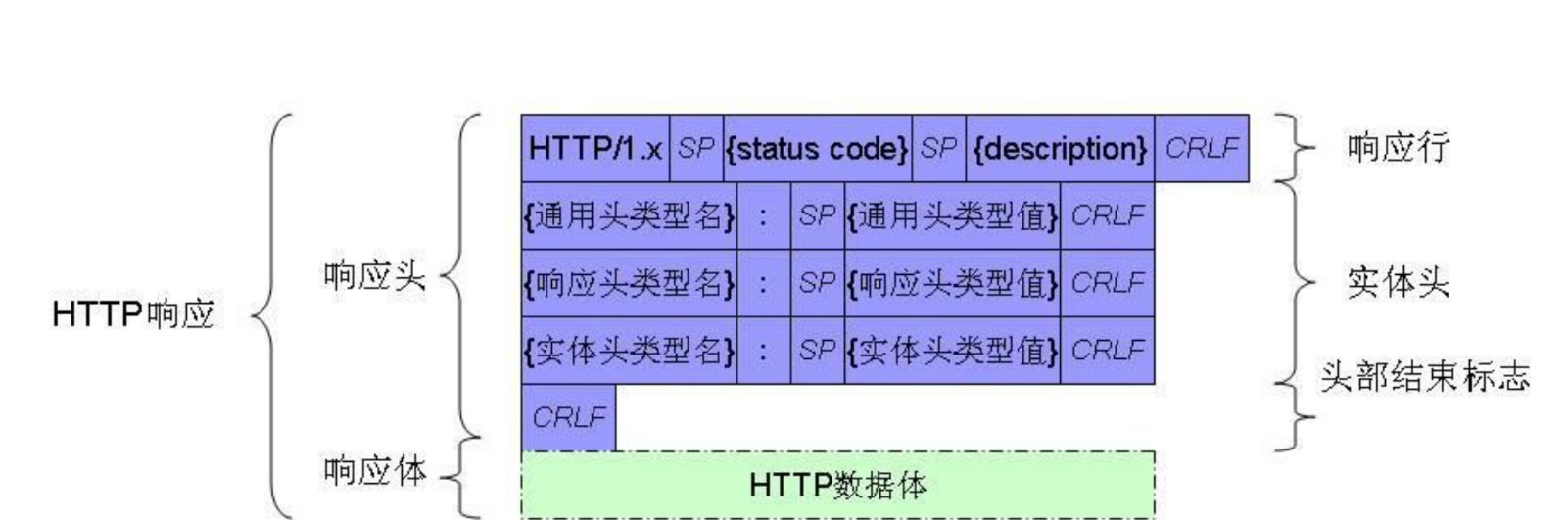
Accept-Language: en, mi

## HTTP的响应

在HTTP协议的特点中介绍到，HTTP协议由两部分构成，请求和响应。响应阶段是指服务器处理完请求以后，将数据回复给客户端的过程。

接下来了解响应报文的格式

### HTTP的响应格式



### HTTP响应报文示例

在浏览器中查看，或者手写示例

HTTP/1.1 200 OK

Date: Mon, 27 Jul 2009 12:28:53 GMT

Server: Apache

Last-Modified: Wed, 22 Jul 2009 19:15:56 GMT

ETag: "34aa387-d-1568eb00"

Accept-Ranges: bytes

Content-Length: 51

Vary: Accept-Encoding

Content-Type: text/plain

## HTTP的请求方法

根据 HTTP 标准，HTTP 请求可以使用多种请求方法。

HTTP1.0 定义了三种请求方法： GET, POST 和 HEAD方法。

HTTP1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法。

|  |  |
| --- | --- |
| **Http方法** | **含义** |
| GET 方法 | 通常用于请求服务器发送某个资源。不包含主体 |
| HEAD 方法 | 与 GET 方法类似，但服务器在响应中只返回头部，使用 HEAD 方法可以，在不获取资源的情况下了解资源的情况 (比如，判断其类型)；通过查看响应中的状态码，看看某个对象是否存在；通过查看头部，测试资源是否被修改了；不包含主体 |
| POST 方法 | 该方法是用来向服务器发送数据的，常用于 HTML 表单，包含主体 |
| PUT 方法 | 该方法的语义就是让服务器用请求的主体部分来创建一个由所请求的 URL 命名的新文档，如果那个 URL 已经存在的话，就用这个主体来替代它。包含主体 |
| PATCH | 是对 PUT 方法的补充，用来对已知资源进行局部更新 。 |
| TRACE 方法 | 主要用于验证请求是否如愿穿过了请求 / 响应链，不包含主体 |
| OPTIONS 方法 | 决定可以在服务器上执行那些方法，不包含主体 |
| DELETE 方法 | 该方法就是请服务器删除请求 URL 所指定的资源，但是客户端应用程序无法保证删除操作一定会被执行，因为 HTTP 规范允许服务器在不通知客户端的情况下撤销请求，不包含主体 |
| CONNECT | HTTP/1.1 协议中预留给能够将连接改为管道方式的代理服务器。 |
| 扩展方法 | 指的是没有在 HTTP/1.1 规范中定义的方法，这些方法为开发者提供了一种扩展这些 HTTP 服务能力的手段。 |

注意 ：

并不是每个服务器都实现了所有的方法。

即使服务器实现了所有这些方法，这些方法的使用很可能也是受限的。

例如，支持 DELETE 方法或 PUT 方法的服务器可能并不希望任何人都能够删除或存储资源，这些限制通常都是在服务器的配置中进行设置的。

虽然 HTTP 的请求方式有 9 种，但是我们在实际应用中常用的也就是 get 和 post，其他请求方式也都可以通过这两种方式间接的来实现。

## HTTP的常见状态码

服务器在接收和解释请求消息后，返回一个 HTTP 响应消息。

状态行中的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：

|  |  |  |
| --- | --- | --- |
| **HTTP状态码分类** | | |
| 分类 | 描述 | 解释 |
| 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。 |

|  |  |  |
| --- | --- | --- |
| **HTTP常见状态码** | | |
| 状态码 | 状态码英文名称 | 中文描述 |
| 200 | OK | 请求成功。一般用于GET与POST请求 |
| 302 | Found | 重定向。与301类似。但资源只是临时被移动。客户端应继续使用原有URI |
| 404 | Not Found | 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面 |
| 500 | Internal Server Error | 服务器内部错误，无法完成请求 |
| 505 | HTTP Version not supported | 服务器不支持请求的HTTP协议的版本，无法完成处理 |

## HTTP的头部

HTTP 头部字段向请求和响应报文中添加了一些附加信息。本质上来说，它们只是一些名 / 值对的列表。头部和协议配合工作，共同决定了客户端和服务器能做什么事情。

头部的分类：

* **通用头部：**既可以出现在请求报文中，也可以出现在响应报文中，它提供了与报文相关的最基本的信息
* **请求头部：**只在请求报文中有意义的头部。用于说明是谁或什么在发送请求、请求源自何处，或者客户端的喜好及能力。
* **响应头部：**响应头部为客户端提供了一些额外信息，比如谁在发送响应、响应者的功能，甚至与响应相关的一些特殊指令
* **实体头部：**描述主体的长度和内容，或者资源自身
* **扩展头部**：规范中没有定义的新头部，开发者可以自定义一个头部的值 / 对

### 通用头部

|  |  |
| --- | --- |
| **通用Header** | **含义** |
| Connection | 允许客户端和服务器指定与请求 / 响应连接有关的选项 |
| Date | 提供日期和时间标志，说明报文是什么时间创建的  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。 |
| MIME-Version | 给出了发送端使用的 MIME 版本 |
| Transfer-Encoding | 告知接收端为了保证报文的可靠传输，报文采用了的编码方式 |
| Update | 给出了发送端可能想要 “升级” 使用的新版本或协议 |
| Trailer | 如果报文采用了分块传输编码方式，就可以用这个头部列出位于报文拖挂部分的头部集合 |
| Cache-Control | 用于随报文传送缓存指示 |

### 请求头部

|  |  |
| --- | --- |
| **请求头** | **含义** |
| Client-IP | 提供了运行客户端的机器的 IP 地址 |
| From | 提供了客户端用户的 E-mail 地址 |
| Host | 给出了接收请求的服务器的主机名和端口号 |
| Referer | 提供了包含当前请求 URI 的文档的 URL |
| UA-Color | 提供了与客户端显示器的显示颜色有关的信息 |
| UA-CPU | 给出了客户端 CPU 的类型或制造商 |
| UA-OS | 给出了运行在客户端机器上的操作系统名称及版本 |
| UA-Pixels | 提供了客户端显示器的像素信息 |
| User-Agent | 将发起请求的应用程序名称告知服务器 |
| Accept | 告诉服务器能够发送哪些媒体类型 |
| Accept-Charset | 告诉服务器能够发送哪些字符集 |
| Accept-Encoding | 告诉服务器能够发送哪些编码方式 |
| Accept-Language | 告诉服务器能够发送哪些语言 |
| TE | 告诉服务器可以使用那些扩展传输编码 |
| Expect | 允许客户端列出某请求所要求的服务器行为 |
| Range | 如果服务器支持范围请求，就请求资源的指定范围 |
| If-Match | 如果实体标记与文档当前的实体标记相匹配，就获取这份文档 |
| If-Modified-Sinec | 除非在某个指定的日期之后资源被修改过，否则就限制这个请求 |
| If-None-Match | 如果提供的实体标记与当前文档的实体标记不相符，就获取文档 |
| If-Range | 允许对文档的某个范围进行条件请求 |
| If-Unmodified-Since | 除非在某个指定日期之后资源没有被修改过，否则就限制这个请求 |
| Authorization | 包含了客户端提供给服务器，以便对其自身进行认证的数据 |
| Cookie | 客户端用它向服务器传送数据 |
| Cookie2 | 用来说明请求端支持的 cookie 版本 |
| Max-Forward | 在通往源端服务器的路径上，将请求转发给其他代理或网关的最大次数 |
| Proxy-Authorization | 这个头部在与代理进行认证时使用的 |
| Proxy-Connection | 这个头部是在与代理建立连接时使用的 |

### 响应头部

|  |  |
| --- | --- |
| **响应头** | **含义** |
| Age | (从最初创建开始) 响应持续时间 |
| Public | 服务器为其资源支持的请求方法列表 |
| Retry-After | 如果资源不可用的话，在此日期或时间重试 |
| Server | 服务器应用程序软件的名称和版本  服务器名字。Servlet一般不设置这个值，而是由Web服务器自己设置。 |
| Title | 对 HTML 文档来说，就是 HTML 文档的源端给出的标题 |
| Warning | 比原因短语更详细一些的警告报文 |
| Accept-Ranges | 对此资源来说，服务器可接受的范围类型 |
| Vary | 服务器会根据这些头部的内容挑选出最适合的资源版本发送给客户端 |
| Proxy-Authenticate | 来自代理的对客户端的质询列表 |
| Set-Cookie | 在客户端设置数据，以便服务器对客户端进行标识  设置和页面关联的Cookie。Servlet不应使用response.setHeader("Set-Cookie", ...)，而是应使用HttpServletResponse提供的专用方法addCookie。参见下文有关Cookie设置的讨论。 |
| Set-Cookie2 | 与 Set-Cookie 类似 |
| WWW-Authenticate | 来自服务器的对客户端的质询列表  客户应该在Authorization头中提供什么类型的授权信息？在包含401（Unauthorized）状态行的应答中这个头是必需的。例如，response.setHeader("WWW-Authenticate", "BASIC realm=＼"executives＼"")。  注意Servlet一般不进行这方面的处理，而是让Web服务器的专门机制来控制受密码保护页面的访问（例如.htaccess）。 |
| 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。 |

### 实体头部

|  |  |
| --- | --- |
| **实体头** | **含义** |
| Allow | 列出了可以对此实体执行的请求方法  服务器支持哪些请求方法（如GET、POST等） |
| Location | 告知客户端实体实际上位于何处，用于将接收端定向到资源的位置 (URL) 上去  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。 |
| Content-Base | 解析主体中的相对 URL 时使用的基础 URL |
| Content-Encoding | 对主体执行的任意编码方式  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-Language | 理解主体时最适宜使用的自然语言 |
| Content-Length | 主体的长度  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 ByteArrayOutputStream，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-Location | 资源实际所处的位置 |
| Content-MD5 | 主体的 MD5 校验和 |
| Content-Range | 在整个资源中此实体表示的字节范围 |
| Content-Type | 主体类型  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setContentType。 |
| ETag | 与此实体相关的实体标记 |
| Expires | 实体不再有效，要从原始的源端再次获取实体的日期和时间  应该在什么时候认为文档已经过期，从而不再缓存它 |
| Last-Modified | 这个实体最后一次被修改的日期和时间  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。 |

## 附录

### CGI

CGI(Common Gateway Interface) 是 HTTP 服务器与你的或其它机器上的程序进行“交谈”的一种工具，其程序须运行在网络服务器上。

绝大多数的 CGI 程序被用来解释处理杰自表单的输入信息，并在服务器产生相应的处理，或将相应的信息反馈给浏览器。CGI 程序使网页具有交互功能。

### 计算机网络常见协议

|  |  |  |
| --- | --- | --- |
| 协议简称 | 协议全称 | 中文名称 |
| TCP | transmission control protocol | 传输控制协议 |
| IP | Internet Protocol | 因特网协议 |
| UDP | User Datagram Protocol | 用户数据报协议 |
| HTTP | hypertext transfer protocol | 超文本传输协议 |
| FTP | File Transfer Protocol | 文件传输协议 |
| SMTP | simple mail transfer protocol | 简单邮件传输协议 |
| POP3 | post office protocol-version3 | 邮局协议3 |
| IMAP | internet mail access portocol | 因特网邮件访问协议 |
| TLS | Transport Layer Security | 安全协议 |
| SSL | Secure Sockets Layer | 安全协议 |

### GET请求和POST请求的区别

GET在浏览器回退时是无害的，而POST会再次提交请求。GET产生的URL地址可以被Bookmark，而POST不可以。GET请求会被浏览器主动cache，而POST不会，除非手动设置。GET请求只能进行url编码，而POST支持多种编码方式。GET请求参数会被完整保留在浏览器历史记录里，而POST中的参数不会被保留。GET请求在URL中传送的参数是有长度限制的，而POST没有。对参数的数据类型，GET只接受ASCII字符，而POST没有限制。GET比POST更不安全，因为参数直接暴露在URL上，所以不能用来传递敏感信息。GET参数通过URL传递，POST放在Request body中。

### URLConnection

URLConnection是JDK中功能强大的一个抽象类。它表示指向 URL 指定资源的活动连接。

与 URL 类相比，它与服务器的交互提供了更多的控制机制。尤其服务器是 HTTP 服务器，可以使用 URLConnection 对 HTTP 首部的访问，可以配置发送给服务器的请求参数。当然也可以通过它读取服务器的数据以及向服务器写入数据

URLConnection 是 Java 的协议处理器机制的一部分。协议处理器机制是将处理协议的细节与特定数据类型分开。如果要实现一个特定的协议，则实现 URLConnection 的子类即可。程序运行时可以将该子类作为一个具体的协议处理器来使用。

#### 获得URLConnection

String url = “http://www.baidu.com”;

URL url = *new* URL(url);  
 *//获得httpUrlConnectiond对象* URLConnection con =url.openConnection();

当我们拿到一个 URLConnection 对象后，并不代表客户端已经和服务器建立了连接。只有主动调用其 connect () 方法才去和服务器建立连接。不过当我们调用 getInputStream ()，getContent ()，getHeaderField () 和其他要求打开连接的方法时，如果连接尚未打开，它们就会调用 connect ()。所以，在实际开发中我们主动调用 connect () 方法的机会很少。

#### URLConnection API

|  |  |
| --- | --- |
| **方法** | **含义** |
| setAllowUserInteraction() | 设置此对象的 allowUserInteraction 字段的值。 |
| setDoInput() | 如果打算使用 URL 连接进行输入，则将 DoInput 标志设置为 true |
| setDoOutput() | 如果打算使用 URL 连接进行输出，则将 DoInput 标志设置为 true |
| setUseCaches() | 设置是否使用缓存，默认为true |
| getRequestProperty() | 返回此连接指定的一般请求属性值,参数为key,返回值为value |
| setRequestProperty() | 设置一般请求属性的值，参数为key , value |
| setConnectTimeOut() | 设置开到此 URLConnection 引用的资源的通信链接的超时值 |
| setReadTimeout() | 将读超时设置为指定的超时值，以毫秒为单位。 |
| getContentEncoding() | 返回该 URL 引用的资源的内容编码,未知则返回NULL |
| getContentLength() | 返回此连接的 URL 引用的资源的内容长度 |

#### HttpUrlConnection

HttpUrlConnection继承UrlConnection，有它自身独特的状态码和方法

|  |  |
| --- | --- |
| **字段/方法** | **值** |
| Method | HTTP方法（GET、POST、PUT等） |
| responseMessage | 表示三位数的状态码 |
| HTTP\_OK | HTTP状态码：200 |
| setRequestMethod() | 设置URL请求的方法，GET、POST、HEAD等协议限制的方法 |
| getRequestMethod() | 获得请求方法 |
| getResponseCode() | 从HTTP响应消息获取状态码 |

### HTTP响应状态码列表

|  |  |  |
| --- | --- | --- |
| **HTTP状态码列表** | | |
| 状态码 | 状态码英文名称 | 中文描述 |
| 100 | Continue | 继续。[客户端](http://www.dreamdu.com/webbuild/client_vs_server/)应继续其请求 |
| 101 | Switching Protocols | 切换协议。服务器根据客户端的请求切换协议。只能切换到更高级的协议，例如，切换到HTTP的新版本协议 |
| 200 | OK | 请求成功。一般用于GET与POST请求 |
| 201 | Created | 已创建。成功请求并创建了新的资源 |
| 202 | Accepted | 已接受。已经接受请求，但未处理完成 |
| 203 | Non-Authoritative Information | 非授权信息。请求成功。但返回的meta信息不在原始的服务器，而是一个副本 |
| 204 | No Content | 无内容。服务器成功处理，但未返回内容。在未更新网页的情况下，可确保浏览器继续显示当前文档 |
| 205 | Reset Content | 重置内容。服务器处理成功，用户终端（例如：浏览器）应重置文档视图。可通过此返回码清除浏览器的表单域 |
| 206 | Partial Content | 部分内容。服务器成功处理了部分GET请求 |
| 300 | Multiple Choices | 多种选择。请求的资源可包括多个位置，相应可返回一个资源特征与地址的列表用于用户终端（例如：浏览器）选择 |
| 301 | Moved Permanently | 永久移动。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替 |
| 302 | Found | 临时移动。与301类似。但资源只是临时被移动。客户端应继续使用原有URI |
| 303 | See Other | 查看其它地址。与301类似。使用GET和POST请求查看 |
| 304 | Not Modified | 未修改。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源 |
| 305 | Use Proxy | 使用代理。所请求的资源必须通过代理访问 |
| 306 | Unused | 已经被废弃的HTTP状态码 |
| 307 | Temporary Redirect | 临时重定向。与302类似。使用GET请求重定向 |
| 400 | Bad Request | 客户端请求的语法错误，服务器无法理解 |
| 401 | Unauthorized | 请求要求用户的身份认证 |
| 402 | Payment Required | 保留，将来使用 |
| 403 | Forbidden | 服务器理解请求客户端的请求，但是拒绝执行此请求 |
| 404 | Not Found | 服务器无法根据客户端的请求找到资源（网页）。通过此代码，网站设计人员可设置"您所请求的资源无法找到"的个性页面 |
| 405 | Method Not Allowed | 客户端请求中的方法被禁止 |
| 406 | Not Acceptable | 服务器无法根据客户端请求的内容特性完成请求 |
| 407 | Proxy Authentication Required | 请求要求代理的身份认证，与401类似，但请求者应当使用代理进行授权 |
| 408 | Request Time-out | 服务器等待客户端发送的请求时间过长，超时 |
| 409 | Conflict | 服务器完成客户端的 PUT 请求时可能返回此代码，服务器处理请求时发生了冲突 |
| 410 | Gone | 客户端请求的资源已经不存在。410不同于404，如果资源以前有现在被永久删除了可使用410代码，网站设计人员可通过301代码指定资源的新位置 |
| 411 | Length Required | 服务器无法处理客户端发送的不带Content-Length的请求信息 |
| 412 | Precondition Failed | 客户端请求信息的先决条件错误 |
| 413 | Request Entity Too Large | 由于请求的实体过大，服务器无法处理，因此拒绝请求。为防止客户端的连续请求，服务器可能会关闭连接。如果只是服务器暂时无法处理，则会包含一个Retry-After的响应信息 |
| 414 | Request-URI Too Large | 请求的URI过长（URI通常为网址），服务器无法处理 |
| 415 | Unsupported Media Type | 服务器无法处理请求附带的媒体格式 |
| 416 | Requested range not satisfiable | 客户端请求的范围无效 |
| 417 | Expectation Failed | 服务器无法满足Expect的请求头信息 |
| 500 | Internal Server Error | 服务器内部错误，无法完成请求 |
| 501 | Not Implemented | 服务器不支持请求的功能，无法完成请求 |
| 502 | Bad Gateway | 作为网关或者代理工作的服务器尝试执行请求时，从远程服务器接收到了一个无效的响应 |
| 503 | Service Unavailable | 由于超载或系统维护，服务器暂时的无法处理客户端的请求。延时的长度可包含在服务器的Retry-After头信息中 |
| 504 | Gateway Time-out | 充当网关或代理的服务器，未及时从远端服务器获取请求 |
| 505 | HTTP Version not supported | 服务器不支持请求的HTTP协议的版本，无法完成处理 |

### Http Content-Type

Content-Type，内容类型，一般是指网页中存在的Content-Type，用于定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件，这就是经常看到一些网页点击的结果却是下载到的一个文件或一张图片的原因。

|  |  |  |  |
| --- | --- | --- | --- |
| 文件扩展名 | Content-Type(Mime-Type) | 文件扩展名 | Content-Type(Mime-Type) |
| .\*（ 二进制流，不知道下载文件类型） | application/octet-stream | .tif | image/tiff |
| .001 | application/x-001 | .301 | application/x-301 |
| .323 | text/h323 | .906 | application/x-906 |
| .907 | drawing/907 | .a11 | application/x-a11 |
| .acp | audio/x-mei-aac | .ai | application/postscript |
| .aif | audio/aiff | .aifc | audio/aiff |
| .aiff | audio/aiff | .anv | application/x-anv |
| .asa | text/asa | .asf | video/x-ms-asf |
| .asp | text/asp | .asx | video/x-ms-asf |
| .au | audio/basic | .avi | video/avi |
| .awf | application/vnd.adobe.workflow | .biz | text/xml |
| .bmp | application/x-bmp | .bot | application/x-bot |
| .c4t | application/x-c4t | .c90 | application/x-c90 |
| .cal | application/x-cals | .cat | application/vnd.ms-pki.seccat |
| .cdf | application/x-netcdf | .cdr | application/x-cdr |
| .cel | application/x-cel | .cer | application/x-x509-ca-cert |
| .cg4 | application/x-g4 | .cgm | application/x-cgm |
| .cit | application/x-cit | .class | java/\* |
| .cml | text/xml | .cmp | application/x-cmp |
| .cmx | application/x-cmx | .cot | application/x-cot |
| .crl | application/pkix-crl | .crt | application/x-x509-ca-cert |
| .csi | application/x-csi | .css | text/css |
| .cut | application/x-cut | .dbf | application/x-dbf |
| .dbm | application/x-dbm | .dbx | application/x-dbx |
| .dcd | text/xml | .dcx | application/x-dcx |
| .der | application/x-x509-ca-cert | .dgn | application/x-dgn |
| .dib | application/x-dib | .dll | application/x-msdownload |
| .doc | application/msword | .dot | application/msword |
| .drw | application/x-drw | .dtd | text/xml |
| .dwf | Model/vnd.dwf | .dwf | application/x-dwf |
| .dwg | application/x-dwg | .dxb | application/x-dxb |
| .dxf | application/x-dxf | .edn | application/vnd.adobe.edn |
| .emf | application/x-emf | .eml | message/rfc822 |
| .ent | text/xml | .epi | application/x-epi |
| .eps | application/x-ps | .eps | application/postscript |
| .etd | application/x-ebx | .exe | application/x-msdownload |
| .fax | image/fax | .fdf | application/vnd.fdf |
| .fif | application/fractals | .fo | text/xml |
| .frm | application/x-frm | .g4 | application/x-g4 |
| .gbr | application/x-gbr | . | application/x- |
| .gif | image/gif | .gl2 | application/x-gl2 |
| .gp4 | application/x-gp4 | .hgl | application/x-hgl |
| .hmr | application/x-hmr | .hpg | application/x-hpgl |
| .hpl | application/x-hpl | .hqx | application/mac-binhex40 |
| .hrf | application/x-hrf | .hta | application/hta |
| .htc | text/x-component | .htm | text/html |
| .html | text/html | .htt | text/webviewhtml |
| .htx | text/html | .icb | application/x-icb |
| .ico | image/x-icon | .ico | application/x-ico |
| .iff | application/x-iff | .ig4 | application/x-g4 |
| .igs | application/x-igs | .iii | application/x-iphone |
| .img | application/x-img | .ins | application/x-internet-signup |
| .isp | application/x-internet-signup | .IVF | video/x-ivf |
| .java | java/\* | .jfif | image/jpeg |
| .jpe | image/jpeg | .jpe | application/x-jpe |
| .jpeg | image/jpeg | .jpg | image/jpeg |
| .jpg | application/x-jpg | .js | application/x-javascript |
| .jsp | text/html | .la1 | audio/x-liquid-file |
| .lar | application/x-laplayer-reg | .latex | application/x-latex |
| .lavs | audio/x-liquid-secure | .lbm | application/x-lbm |
| .lmsff | audio/x-la-lms | .ls | application/x-javascript |
| .ltr | application/x-ltr | .m1v | video/x-mpeg |
| .m2v | video/x-mpeg | .m3u | audio/mpegurl |
| .m4e | video/mpeg4 | .mac | application/x-mac |
| .man | application/x-troff-man | .math | text/xml |
| .mdb | application/msaccess | .mdb | application/x-mdb |
| .mfp | application/x-shockwave-flash | .mht | message/rfc822 |
| .mhtml | message/rfc822 | .mi | application/x-mi |
| .mid | audio/mid | .midi | audio/mid |
| .mil | application/x-mil | .mml | text/xml |
| .mnd | audio/x-musicnet-download | .mns | audio/x-musicnet-stream |
| .mocha | application/x-javascript | .movie | video/x-sgi-movie |
| .mp1 | audio/mp1 | .mp2 | audio/mp2 |
| .mp2v | video/mpeg | .mp3 | audio/mp3 |
| .mp4 | video/mpeg4 | .mpa | video/x-mpg |
| .mpd | application/vnd.ms-project | .mpe | video/x-mpeg |
| .mpeg | video/mpg | .mpg | video/mpg |
| .mpga | audio/rn-mpeg | .mpp | application/vnd.ms-project |
| .mps | video/x-mpeg | .mpt | application/vnd.ms-project |
| .mpv | video/mpg | .mpv2 | video/mpeg |
| .mpw | application/vnd.ms-project | .mpx | application/vnd.ms-project |
| .mtx | text/xml | .mxp | application/x-mmxp |
| .net | image/pnetvue | .nrf | application/x-nrf |
| .nws | message/rfc822 | .odc | text/x-ms-odc |
| .out | application/x-out | .p10 | application/pkcs10 |
| .p12 | application/x-pkcs12 | .p7b | application/x-pkcs7-certificates |
| .p7c | application/pkcs7-mime | .p7m | application/pkcs7-mime |
| .p7r | application/x-pkcs7-certreqresp | .p7s | application/pkcs7-signature |
| .pc5 | application/x-pc5 | .pci | application/x-pci |
| .pcl | application/x-pcl | .pcx | application/x-pcx |
| .pdf | application/pdf | .pdf | application/pdf |
| .pdx | application/vnd.adobe.pdx | .pfx | application/x-pkcs12 |
| .pgl | application/x-pgl | .pic | application/x-pic |
| .pko | application/vnd.ms-pki.pko | .pl | application/x-perl |
| .plg | text/html | .pls | audio/scpls |
| .plt | application/x-plt | .png | image/png |
| .png | application/x-png | .pot | application/vnd.ms-powerpoint |
| .ppa | application/vnd.ms-powerpoint | .ppm | application/x-ppm |
| .pps | application/vnd.ms-powerpoint | .ppt | application/vnd.ms-powerpoint |
| .ppt | application/x-ppt | .pr | application/x-pr |
| .prf | application/pics-rules | .prn | application/x-prn |
| .prt | application/x-prt | .ps | application/x-ps |
| .ps | application/postscript | .ptn | application/x-ptn |
| .pwz | application/vnd.ms-powerpoint | .r3t | text/vnd.rn-realtext3d |
| .ra | audio/vnd.rn-realaudio | .ram | audio/x-pn-realaudio |
| .ras | application/x-ras | .rat | application/rat-file |
| .rdf | text/xml | .rec | application/vnd.rn-recording |
| .red | application/x-red | .rgb | application/x-rgb |
| .rjs | application/vnd.rn-realsystem-rjs | .rjt | application/vnd.rn-realsystem-rjt |
| .rlc | application/x-rlc | .rle | application/x-rle |
| .rm | application/vnd.rn-realmedia | .rmf | application/vnd.adobe.rmf |
| .rmi | audio/mid | .rmj | application/vnd.rn-realsystem-rmj |
| .rmm | audio/x-pn-realaudio | .rmp | application/vnd.rn-rn\_music\_package |
| .rms | application/vnd.rn-realmedia-secure | .rmvb | application/vnd.rn-realmedia-vbr |
| .rmx | application/vnd.rn-realsystem-rmx | .rnx | application/vnd.rn-realplayer |
| .rp | image/vnd.rn-realpix | .rpm | audio/x-pn-realaudio-plugin |
| .rsml | application/vnd.rn-rsml | .rt | text/vnd.rn-realtext |
| .rtf | application/msword | .rtf | application/x-rtf |
| .rv | video/vnd.rn-realvideo | .sam | application/x-sam |
| .sat | application/x-sat | .sdp | application/sdp |
| .sdw | application/x-sdw | .sit | application/x-stuffit |
| .slb | application/x-slb | .sld | application/x-sld |
| .slk | drawing/x-slk | .smi | application/smil |
| .smil | application/smil | .smk | application/x-smk |
| .snd | audio/basic | .sol | text/plain |
| .sor | text/plain | .spc | application/x-pkcs7-certificates |
| .spl | application/futuresplash | .spp | text/xml |
| .ssm | application/streamingmedia | .sst | application/vnd.ms-pki.certstore |
| .stl | application/vnd.ms-pki.stl | .stm | text/html |
| .sty | application/x-sty | .svg | text/xml |
| .swf | application/x-shockwave-flash | .tdf | application/x-tdf |
| .tg4 | application/x-tg4 | .tga | application/x-tga |
| .tif | image/tiff | .tif | application/x-tif |
| .tiff | image/tiff | .tld | text/xml |
| .top | drawing/x-top | .torrent | application/x-bittorrent |
| .tsd | text/xml | .txt | text/plain |
| .uin | application/x-icq | .uls | text/iuls |
| .vcf | text/x-vcard | .vda | application/x-vda |
| .vdx | application/vnd.visio | .vml | text/xml |
| .vpg | application/x-vpeg005 | .vsd | application/vnd.visio |
| .vsd | application/x-vsd | .vss | application/vnd.visio |
| .vst | application/vnd.visio | .vst | application/x-vst |
| .vsw | application/vnd.visio | .vsx | application/vnd.visio |
| .vtx | application/vnd.visio | .vxml | text/xml |
| .wav | audio/wav | .wax | audio/x-ms-wax |
| .wb1 | application/x-wb1 | .wb2 | application/x-wb2 |
| .wb3 | application/x-wb3 | .wbmp | image/vnd.wap.wbmp |
| .wiz | application/msword | .wk3 | application/x-wk3 |
| .wk4 | application/x-wk4 | .wkq | application/x-wkq |
| .wks | application/x-wks | .wm | video/x-ms-wm |
| .wma | audio/x-ms-wma | .wmd | application/x-ms-wmd |
| .wmf | application/x-wmf | .wml | text/vnd.wap.wml |
| .wmv | video/x-ms-wmv | .wmx | video/x-ms-wmx |
| .wmz | application/x-ms-wmz | .wp6 | application/x-wp6 |
| .wpd | application/x-wpd | .wpg | application/x-wpg |
| .wpl | application/vnd.ms-wpl | .wq1 | application/x-wq1 |
| .wr1 | application/x-wr1 | .wri | application/x-wri |
| .wrk | application/x-wrk | .ws | application/x-ws |
| .ws2 | application/x-ws | .wsc | text/scriptlet |
| .wsdl | text/xml | .wvx | video/x-ms-wvx |
| .xdp | application/vnd.adobe.xdp | .xdr | text/xml |
| .xfd | application/vnd.adobe.xfd | .xfdf | application/vnd.adobe.xfdf |
| .xhtml | text/html | .xls | application/vnd.ms-excel |
| .xls | application/x-xls | .xlw | application/x-xlw |
| .xml | text/xml | .xpl | audio/scpls |
| .xq | text/xml | .xql | text/xml |
| .xquery | text/xml | .xsd | text/xml |
| .xsl | text/xml | .xslt | text/xml |
| .xwd | application/x-xwd | .x\_b | application/x-x\_b |
| .sis | application/vnd.symbian.install | .sisx | application/vnd.symbian.install |
| .x\_t | application/x-x\_t | .ipa | application/vnd.iphone |
| .apk | application/vnd.android.package-archive | .xap | application/x-silverlight-app |