Table of Contents

My Awesome Book

学习网络遇到的各种问题以及网上收集的各类教程集

http协议

http://blog.csdn.net/chenhanzhun/article/details/43149557

简介

超文本传输协议（Hypertext Transfer Protocol，简称HTTP）是应用层协议。HTTP 是一种请求/响应式的协议，即一个客户端与服务器建立连接后，向服务器发送一个请求；服务器接到请求后，给予相应的响应信息。

HTTP请求报文

http请求报文由请求行，请求头部，空行以及请求包体组成，如下图所示：

1、请求行 由请求方法，URL以及协议版本三部分组成，每个部分以空格分隔。常用的请求方法有：GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT；

- GET：当客户端要从服务器中读取某个资源时，使用GET 方法。GET 方法要求服务器将URL 定位的资源放在响应报文的数据部分，回送给客户端，即向服务器请求某个资源。使用GET 方法时，请求参数和对应的值附加在 URL 后面，利用一个问号（“?”）代表URL 的结尾与请求参数的开始，传递参数长度受限制。例如，/index.jsp?id=100&op=bind。
- POST：当客户端给服务器提供信息较多时可以使用POST 方法，POST 方法向服务器提交数据，比如完成表单数据的提交，将数据提交给服务器处理。GET 一般用于获取/查询资源信息，POST 会附带用户数据，一般用于更新资源信息。POST 方法将请求参数封装在HTTP 请求数据中，以名称/值的形式出现，可以传输大量数据；

2、请求头部请求头部由关键字/值对组成，每行一对，关键字和值用英文冒号":"分隔。请求头部通知服务器有关于客户端请求的信息，典型的请求头有：

- User-Agent：产生请求的浏览器类型；
- Accept：客户端可识别的响应内容类型列表；星号 " " 用于按范围将类型分组，用 " / " 指示可接受全部类型，用"*type/*"指示可接受 type 类型的所有子类型；
- Accept-Language：客户端可接受的自然语言；
- Accept-Encoding：客户端可接受的编码压缩格式；
- Accept-Charset：可接受的应答的字符集；
- 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；
- connection：连接方式（close 或 keepalive）；
- Cookie：存储于客户端扩展字段，向同一域名的服务端发送属于该域的cookie；

3、空行最后一个请求头之后是一个空行，发送回车符和换行符，通知服务器以下不再有请求头；

4、请求包体 请求包体不在 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。与请求包体相关的最常使用的是包体类型 Content-Type 和包体长度 Content-Length；

HHTP响应报文

HTTP 响应报文由状态行、响应头部、空行 和 响应包体 4 个部分组成，如下图所示：

1、状态行 状态行由 HTTP 协议版本字段、状态码和状态码的描述文本 3 个部分组成，他们之间使用空格隔开；

状态码由三位数字组成，第一位数字表示响应的类型，常用的状态码有五大类如下所示：

- 1xx：表示服务器已接收了客户端请求，客户端可继续发送请求；
- 2xx：表示服务器已成功接收到请求并进行处理；
- 3xx：表示服务器要求客户端重定向；
- 4xx：表示客户端的请求有非法内容；

- 5xx：表示服务器未能正常处理客户端的请求而出现意外错误；

状态码描述文本有如下取值：

- 200 OK：表示客户端请求成功；
- 400 Bad Request：表示客户端请求有语法错误，不能被服务器所理解；
- 401 Unauthonzed：表示请求未经授权，该状态代码必须与 WWW-Authenticate 报头域一起使用；
- 403 Forbidden：表示服务器收到请求，但是拒绝提供服务，通常会在响应正文中给出不提供服务的原因；
- 404 Not Found：请求的资源不存在，例如，输入了错误的URL；
- 500 Internal Server Error：表示服务器发生不可预期的错误，导致无法完成客户端的请求；
- 503 Service Unavailable：表示服务器当前不能够处理客户端的请求，在一段时间之后，服务器可能会恢复正常；

2、响应头部 响应头可能包括：

- Location：Location响应报头域用于重定向接受者到一个新的位置。例如：客户端所请求的页面已不存在原先的位置，为了让客户端重定向到这个页面新的位置，服务器端可以发回Location响应报头后使用重定向语句，让客户端去访问新的域名所对应的服务器上的资源；
- Server：Server 响应报头域包含了服务器用来处理请求的软件信息及其版本。它和 User-Agent 请求报头域是相对应的，前者发送服务器端软件的信息，后者发送客户端软件(浏览器)和操作系统的信息。
- Vary：指示不可缓存的请求头列表；
- Connection：连接方式； 对于请求来说：close（告诉 WEB 服务器或者代理服务器，在完成本次请求的响应后，断开连接，不等待本次连接的后续请求了）。 keepalive（告诉WEB服务器或者代理服务器，在完成本次请求的响应后，保持连接，等待本次连接的后续请求）； 对于响应来说：close（连接已经关闭）； keepalive（连接保持着，在等待本次连接的后续请求）； Keep-Alive：如果浏览器请求保持连接，则该头部表明希望 WEB 服务器保持连接多长时间（秒）；例如：Keep-Alive：300；
- WWW-Authenticate：WWW-Authenticate响应报头域必须被包含在401 (未授权的)响应消息中，这个报头域和前面讲到的Authorization 请求报头域是相关的，当客户端收到 401 响应消息，就要决定是否请求服务器对其进行验证。如果要求服务器对其进行验证，就可以发送一个包含了Authorization 报头域的请求；

3、空行最后一个响应头部之后是一个空行，发送回车符和换行符，通知服务器以下不再有响应头部。

4、响应包体服务器返回给客户端的文本信息；

HTTP 工作原理

HTTP 协议采用请求/响应模型。客户端向服务器发送一个请求报文，服务器以一个状态作为响应。

以下是 HTTP 请求/响应的步骤：

- 客户端连接到web服务器：HTTP 客户端与web服务器建立一个 TCP 连接；
- 客户端向服务器发起 HTTP 请求：通过已建立的TCP 连接，客户端向服务器发送一个请求报文；
- 服务器接收 HTTP 请求并返回 HTTP 响应：服务器解析请求，定位请求资源，服务器将资源副本写到 TCP 连接，由客户端读取；
- 释放 TCP 连接：若connection 模式为close，则服务器主动关闭TCP 连接，客户端被动关闭连接，释放TCP 连接；若connection 模式为keepalive，则该连接会保持一段时间，在该时间内可以继续接收请求；
- 客户端浏览器解析HTML内容：客户端将服务器响应的 html 文本解析并显示；

  例如：在浏览器地址栏键入URL，按下回车之后会经历以下流程：

1. 浏览器向 DNS 服务器请求解析该 URL 中的域名所对应的 IP 地址；
2. 解析出 IP 地址后，根据该 IP 地址和默认端口 80，和服务器建立 TCP 连接；
3. 浏览器发出读取文件（URL 中域名后面部分对应的文件）的HTTP 请求，该请求报文作为 TCP 三次握手的第三个报文的数据发送给服务器；
4. 服务器对浏览器请求作出响应，并把对应的 html 文本发送给浏览器；
5. 释放 TCP 连接；
6. 浏览器将该 html 文本并显示内容;

HTTP 无状态性

HTTP 协议是无状态的（stateless）。也就是说，同一个客户端第二次访问同一个服务器上的页面时，服务器无法知道这个客户端曾经访问过，服务器也无法分辨不同的客户端。HTTP 的无状态特性简化了服务器的设计，使服务器更容易支持大量并发的HTTP 请求。

HTTP 持久连接

HTTP1.0 使用的是非持久连接，主要缺点是客户端必须为每一个待请求的对象建立并维护一个新的连接，即每请求一个文档就要有两倍RTT 的开销。因为同一个页面可能存在多个对象，所以非持久连接可能使一个页面的下载变得十分缓慢，而且这种短连接增加了网络传输的负担。HTTP1.1 使用持久连接keepalive，所谓持久连接，就是服务器在发送响应后仍然在

一段时间内保持这条连接，允许在同一个连接中存在多次数据请求和响应，即在持久连接情况下，服务器在发送完响应后并不关闭TCP 连接，而客户端可以通过这个连接继续请求其他对象。

HTTP/1.1 协议的持久连接有两种方式：

非流水线方式：客户在收到前一个响应后才能发出下一个请求；

流水线方式：客户在收到 HTTP 的响应报文之前就能接着发送新的请求报文；

杂记

1. http会话的四个过程
    - 建立TCP连接
    - 发出请求文档
    - 发出响应文档
    - 释放TCP连接

TCP的建立与释放

http://blog.csdn.net/chenhanzhun/article/details/41622555

前言

TCP 是面向连接的、可靠的字节流协议。因此，在传输数据之前通信双方必须建立一个 TCP 连接，建立 TCP 连接需要在服务器和客户端之间进行三次握手。通信双方数据传输完毕之后进行连接释放，释放连接需要在通信双方之间进行四次挥手。

TCP 状态机

TCP 所谓的“连接”，只是通信双方维护一个“连接状态”，让它看上去好像有连接一样，其实 TCP 连接是虚拟的连接，不是电路连接。首先看下 TCP 的状态机，状态机是 TCP 连接与释放的全过程。如下图所示：

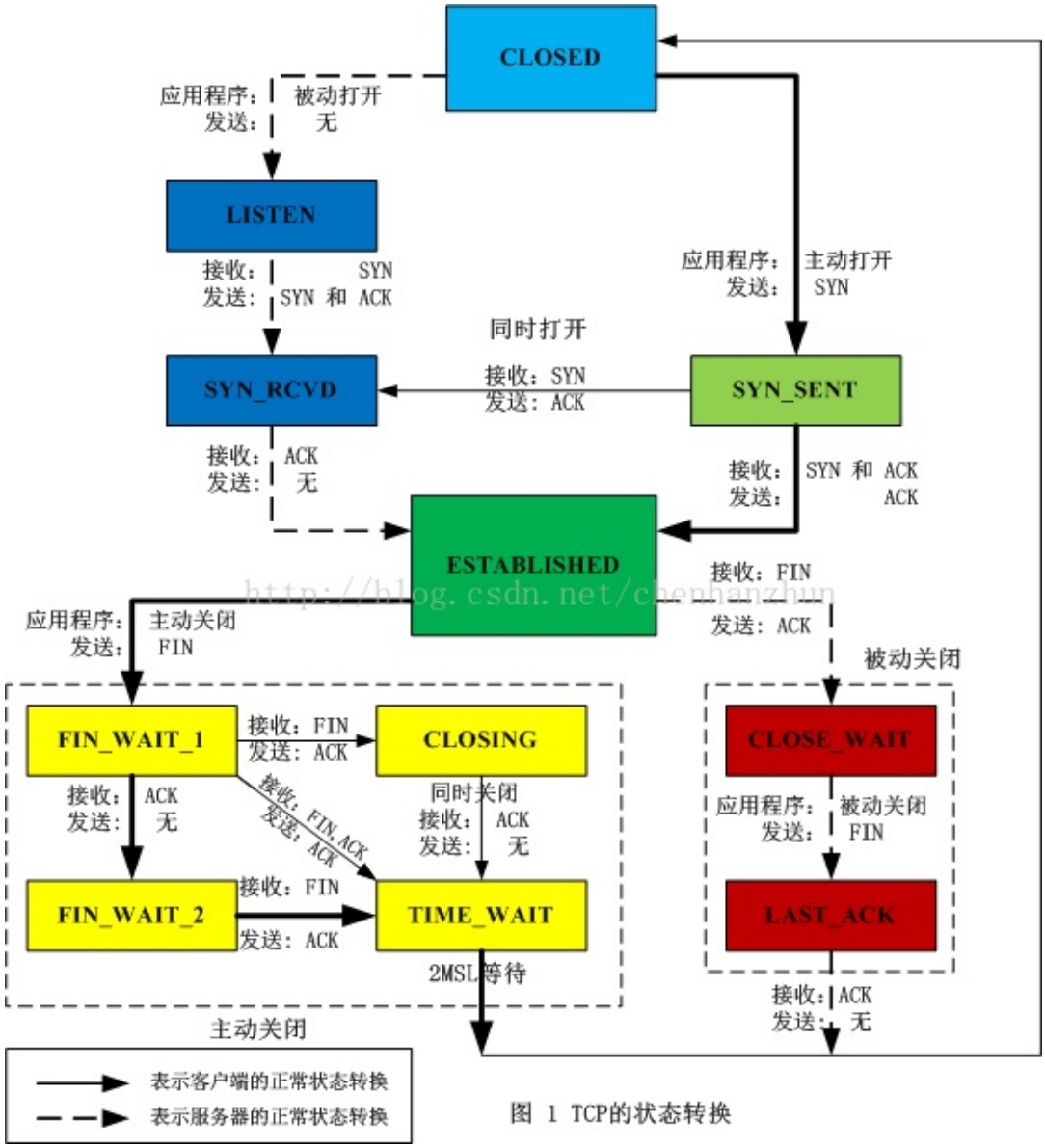

图 1 TCP的状态转换

下面针对 TCP 状态机所出现的各个状态进行简要的分析:

- CLOSED：表示初始状态。对服务端和客户端双方都一样。
- LISTEN：表示监听状态。服务端调用了 listen 函数使其处于监听状态，此时可以开始 accept （接收）客户端的连接。
- SYN_SENT：表示客户端已经发送了 SYN 报文段，则会处于该状态。当客户端调用 * connect 函数发起连接请求时，首先发 SYN 给服务端，然后自己进入 SYN_SENT 状态，并等待服务端发送 ACK+SYN 作为请求应答。
- SYN_RCVD：表示服务端收到客户端发送 SYN 报文段。服务端收到这个报文段后，进入 SYN_RCVD 状态，然后发送 ACK+SYN 给客户端。
- ESTABLISHED：表示 TCP 连接已经成功建立，通信双方可以开始传输数据。服务端发送完 ACK+SYN 并收到来自客户端的 ACK 后进入该状态，客户端收到来自服务器的 SYN+ACK 并发送 ACK 后也进入该状态。
- FIN_WAIT_1：表示主动关闭连接。无论哪方调用 close 函数发送 FIN 报文都会进入这个这个状态。

- FIN_WAIT_2：表示被动关闭方同意关闭连接。主动关闭连接方收到被动关闭方返回的 ACK 后，会进入该状态。
- TIME_WAIT：表示收到对方的 FIN 报文并发送了 ACK 报文，就等 2MSL 后即可回到 CLOSED 状态了。如果 FIN_WAIT_1 状态下，收到对方同时带 FIN 标志和 ACK 标志的报文时，可以直接进入 TIME_WAIT 状态，而无须经过 FIN_WAIT_2 状态。
- CLOSING：表示双方同时关闭连接。如果双方几乎同时调用 close 函数，那么会出现双方同时发送 FIN 报文的情况，就会出现 CLOSING 状态，表示双方都在关闭连接。
- CLOSE_WAIT：表示被动关闭方等待关闭。当收到对方调用 close 函数发送的 FIN 报文时，回应对方 ACK 报文，此时进入 CLOSE_WAIT 状态。
- LAST_ACK：表示被动关闭方发送 FIN 报文后，等待对方的 ACK 报文状态，当收到 ACK 后进入CLOSED状态。

TCP 连接的建立

TCP 连接的正常建立过程通信双方需要三次握手，其过程如下图所示：

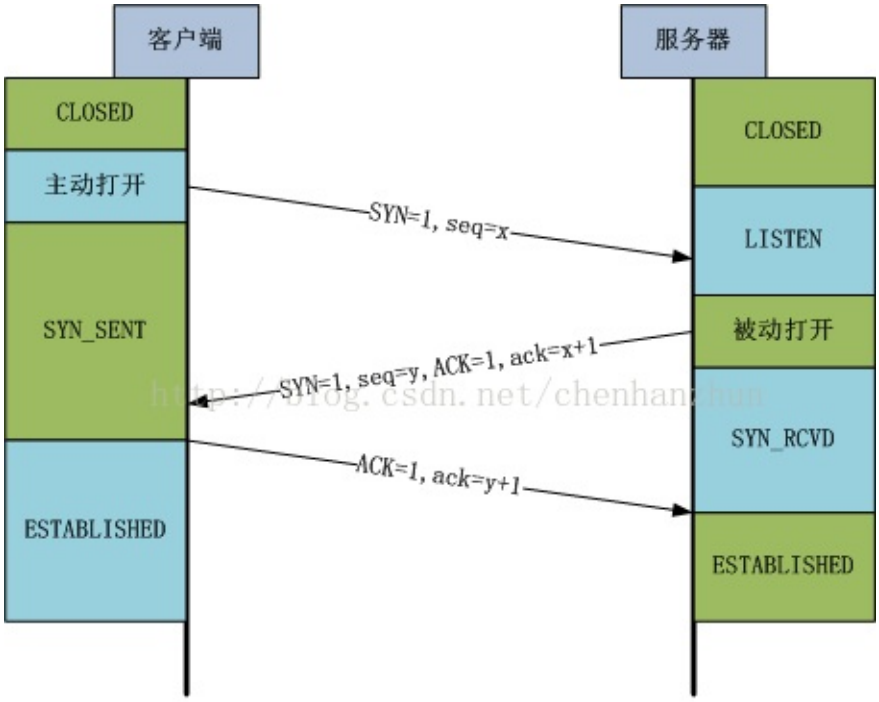

图 2 TCP建立连接

TCP 协议提供可靠的连接服务，采用有保障的三次握手方式来创建一个 TCP 连接。三次握手的具体过程如下：

1、客户端进程向服务端发出连接请求，请求报文的报文段首部中的控制位标志 SYN=1（有关 TCP 控制位信息参考《TCP 协议》），由于是首次请求建立连接，因此，控制位标志 ACK=0，该报文段包含计算机随机生成的初始序号 seq=x。发送请求连接的 TCP 报文段，此时客户端进程进入 SYN_SENT 状态，这是 TCP 连接的第一次握手。

2、服务端收到客户端发来的请求报文后，若同意建立连接，则向客户端发送确认。确认报文中的控制位 SYN=1，ACK=1，确认应答号 ack=x+1（即在接收到序列号值基础上加 1 ），并且发送自己的一个初始序列号 seq=y（即请求与客户端连接）。此时，服务端进入SYN_RCVD状态，这是TCP连接的第二次握手。

3、客户端进程收到服务端进程的确认报文后，还要向服务端发出确认信息。确认报文段的控制位 ACK=1，确认应答号 ack=y+1（即在接收到序列号值基础上加 1 ），此时，客户端进入 ESTABLISHED 状态。服务器收到来自客户端的确认应答信息也进入 ESTABLISHED 状态。这是TCP连接的第三次握手。此时，TCP 连接成功建立。

同时打开连接请求

正常情况下，通信一方请求建立连接，另一方响应该请求，但是如果出现，通信双方同时请求建立连接时，则连接建立过程并不是三次握手过程，而且这种情况的连接也只有一条，并不会建立两条连接。同时打开连接时，两边几乎同时发送 SYN，并进入 SYN_SENT 状态，当每一端收到 SYN 时，状态变为 SYN_RCVD，同时双方都再发 SYN 和 ACK 作为对收到的 SYN 进行确认应答。当双方都收到 SYN 及相应的 ACK 时，状态变为 ESTABLISHED。其过程入下：

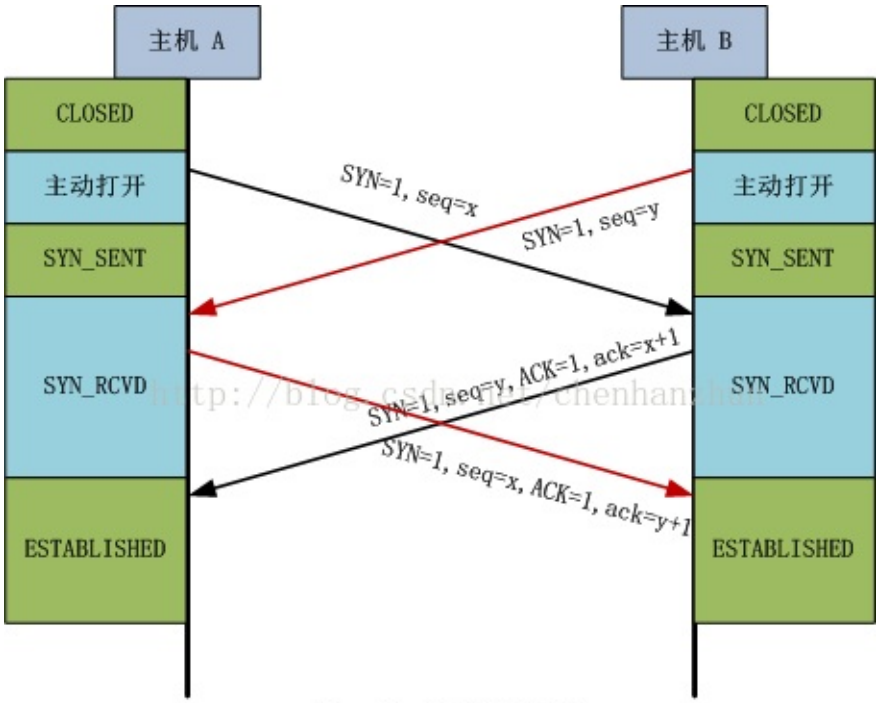

图 3 同时打开连接请求

TCP 连接释放

由于 TCP 连接是全双工的，因此每个方向都必须单独进行关闭。原则是主动关闭的一方发送一个 FIN 报文来表示终止这个方向的连接，收到一个 FIN 意味着这个方向不再有数据流动，但另一个方向仍能继续发送数据，直到另一个方向也发送 FIN 报文。TCP 连接释放的过程如下图所示：

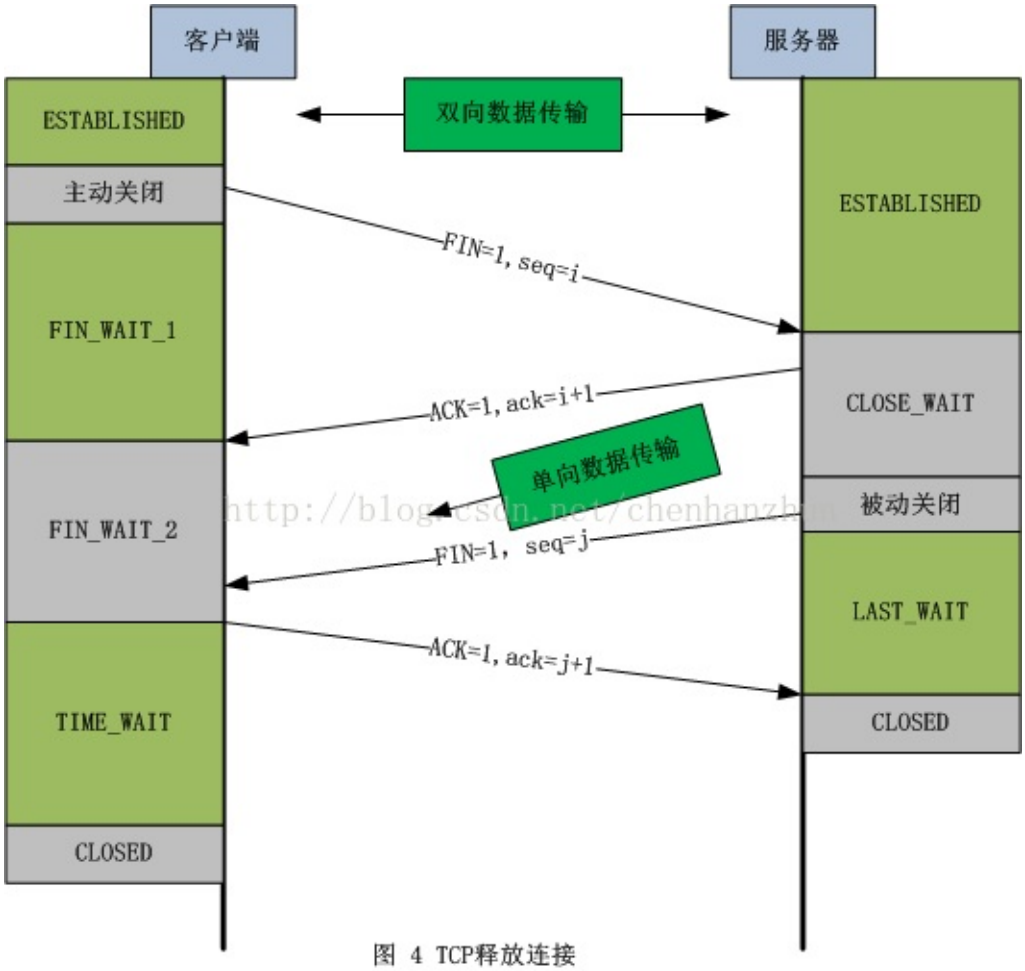

图 4 TCP释放连接

以下是释放连接的四次挥手过程：

1、客户端进程主动向服务端发出连接释放请求报文段，并停止发送数据，主动关闭 TCP 连接。释放连接报文段中控制位 FIN=1，序列号为 seq=i，发送该报文段之后客户端进入 FIN_WAIT_1（终止等待1）状态，等待服务器的确认。这是 TCP 连接释放的第一次挥手。

2、服务器收到连接释放请求报文段后即发出确认释放连接的报文段，该报文段中控制位 ACK=1，确认应答号为 ack=i+1，然后服务器进入CLOSE_WAIT（关闭等待）状态。此时 TCP 处于半关闭状态，即客户端已经不向服务器发送数据，但服务器仍可向客户端发送数据。这是TCP连接释放的第二次挥手。

3、客户端收到服务器的确认信息后，就进入了FIN_WAIT_2（终止等待2）状态，等待服务器发出连接释放请求报文段，若没有数据需要传输，服务器被动向客户端发出链接释放请求报文段中，报文段中控制位 FIN=1，序列号 seq=j，此时服务器进入LAST_ACK（最后确认）状态，等待客户端的确认应答。这是 TCP 连接释放的第三次挥手。

4、客户端收到服务器的连接释放请求后，必须对此发出确认。确认报文段中控制位ACK=1，确认应答号 ack=j+1，客户端发出确认应答信息之后后进入TIME_WAIT（时间等待）状态。在这段时间内 TCP连接并没有释放，必须等待 2MSL 时间后，客户端才进入 CLOSED 状态。服务器收到了客户端的确认应答后，就进入了 CLOSED 状态。直到客户端和服务器都进入 CLOSED 状态后，连接就完全释放了，这是TCP连接释放的第四次挥手。

同时关闭连接

正常情况下，通信一方请求连接关闭，另一方响应连接关闭请求，并且被动关闭连接。但是若出现同时关闭连接请求时，通信双方均从 ESTABLISHED 状态转换为 FIN_WAIT_1 状态。任意一方收到对方发来的 FIN 报文段后，其状态均由 FIN_WAIT_1转变到 CLOSING 状态，并发送最后的 ACK 数据段。当收到最后的 ACK 数据段后，状态转变化 TIME_WAIT，在等待 2MSL 时间后进入到 CLOSED 状态，最终释放整个 TCP 传输连接。其过程入下：

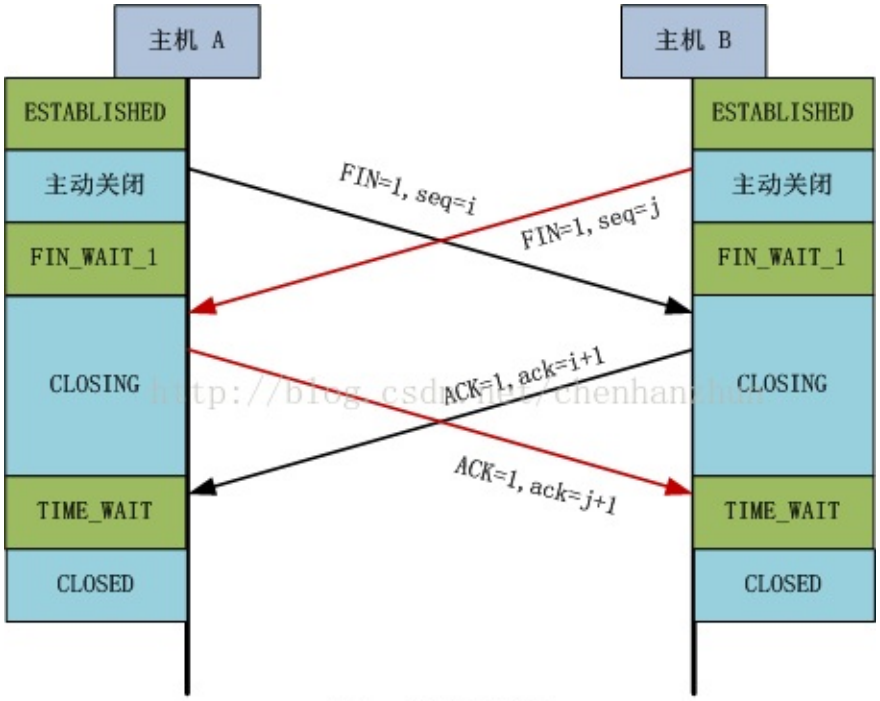

图 5 同时关闭连接

TCP 相关疑问

为什么在 TCP 协议里，建立连接是三次握手，而关闭连接却是四次握手?

因为当处于 LISTEN 状态的服务器端收到来自客户端的 SYN 报文(客户端希望新建一个TCP连接)时，它可以把 ACK (确认应答)和 SYN (同步序号)放在同一个报文里来发送给客户端。但在关闭 TCP 连接时，当收到对方的 FIN 报文时，对方仅仅表示对方已经没有数据发送给你

了，但是你自己可能还有数据需要发送给对方，则等你发送完剩余的数据给对方之后，再发送 FIN 报文给对方来表示你数据已经发送完毕，并请求关闭连接，所以通常情况下，这里的 ACK 报文和 FIN 报文都是分开发送的。

为什么一定要进行三次握手？

当客户端向服务器端发送一个连接请求时，由于某种原因长时间驻留在网络节点中，无法达到服务器端，由于 TCP 的超时重传机制，当客户端在特定的时间内没有收到服务器端的确认应答信息，则会重新向服务器端发送连接请求，且该链接请求得到服务器端的响应并正常建立连接，进而传输数据，当数据传输完毕，并释放了此次 TCP 连接。若此时第一次发送的连接请求报文段延迟了一段时间后，到达了服务器端，本来这是一个早已失效的报文段，但是服务器端收到该链接请求后误以为客户端又发出了一次新的连接请求，于是服务器端向客户端发出确认应答报文段，并同意建立连接。如果没有采用三次握手建立连接，由于服务器端发送了确认应答信息，则表示新的连接已成功建立，但是客户端此时并没有向服务器端发出任何连接请求，因此客户端忽略服务器端的确认应答报文，更不会向服务器端传输数据。而服务器端却认为新的连接已经建立了，并在一直等待客户端发送数据，这样服务器端一直处于等待接收数据，直到超出计数器的设定值，则认为客户端出现异常，并且关闭这个连接。在这个等待的过程中，浪费服务器的资源。如果采用三次握手，客户端就不会向服务端发出确认应答信息，服务器端由于没有收到客户端的确认应答信息，从而判定客户端并没有请求建立连接，从而不建立该连接。

为什么需要在 TIME_WAIT 状态必须等待 2MSL 时间，而不直接给进入 CLOSED 状态？

主要有两个原因：

TIME_WAIT 确保有足够的时间让对端收到了ACK，如果被动关闭的那方没有收到 ACK，就会触发被动端重发 FIN。因为最后一次确认应答 ACK 报文段很有可能丢失，因而使被动关闭方处于在LAST_ACK 状态的，此时被动关闭方会重发这个 FIN+ACK 报文段，在这等待的 2MSL 时间内主动关闭方重新收到这个被动关闭方重发的 FIN+ACK 报文段，因此，主动关闭方会重新发送确认应答信息，从而重新启动 2MSL 计时器，直到通信双方都进入 CLOSED 状态。如果主动关闭方在 TIME_WAIT 状态不等待一段时间就直接释放连接并进入 CLOSED 状态，那么主动关闭方无法收到来自被动关闭方重发的 FIN+ACK 报文段，也就不会再发送一次确认 ACK 报文段，因此被动关闭方就无法正常进入CLOSED 状态。

有足够的时间让这个连接不会跟后面的连接混在一起。防止已失效的请求连接出现在本连接中。在连接处于 2MSL 等待时，任何迟到的报文段将被丢弃，因为处于 2MSL等待的、由该插口（插口是IP和端口对的意思，socket）定义的连接在这段时间内将不能被再用，这样就可以使下一个新的连接中不会出现这种旧的连接之前延迟的报文段。

ISO/OSI模型

ISO/OSI七层模型分别为

- 物理层
- 数据链路层
- 网络层
- 传输层
- 回话层
- 表示层
- 应用层

其中，

网络层负责分组转发和路由选择，根据路由表把分组逐跳地由源站传送到目的站，并能适应网络的负载及拓扑结构变化，动态更新路由表。

传输层传输的PDU称为报文，传输层为源节点和目的节点的用户进程之间提供端到端的可靠的传输协议。

表示层的主要功能是完成数据格式变换、数据加密与解密、数据压缩与恢复功能。

数据链路层负责在单个链路上的节点间传送以帧为PDU的数据。