

- 计算机网络体系结构
 - 网络协议是什么？
 - 为什么要对网络协议分层？
- TCP/IP 协议族
 - 应用层
 - 运输层
 - 网络层
 - 数据链路层
 - 物理层
 - TCP/IP 协议族
- TCP的三次握手四次挥手
 - TCP报文的头部结构
 - 三次握手
 - 四次挥手
- 常见面试题
 - 为什么TCP连接的时候是3次？2次可以吗？
 - 为什么TCP连接的时候是3次，关闭的时候却是4次？
 - 为什么客户端发出第四次挥手的确认报文后要等2MSL的时间才能释放TCP连接？
 - 如果已经建立了连接，但是客户端突然出现故障了怎么办？

- 什么是HTTP，HTTP 与 HTTPS 的区别
- 常用HTTP状态码
- GET和POST区别
- 什么是对称加密与非对称加密
- 什么是HTTP2
- Session、Cookie和Token的主要区别
- Servlet是线程安全的吗
- Servlet接口中有哪些方法及Servlet生命周期探秘
- 如果客户端禁止 cookie 能实现 session 还能用吗？

计算机网络体系结构

在计算机网络的基本概念中，分层次的体系结构是最基本的。计算机网络体系结构的抽象概念较多，在学习时要多思考。这些概念对后面的学习很有帮助。

网络协议是什么？

在计算机网络要做到有条不紊地交换数据，就必须遵守一些事先约定好的规则，比如交换数据的格式、是否需要发送一个应答信息。这些规则被称为网络协议。

为什么要对网络协议分层？

- 简化问题难度和复杂度。由于各层之间独立，我们可以分割大问题为小问题。
- 灵活性好。当其中一层的技术变化时，只要层间接口关系保持不变，其他层不受影响。
- 易于实现和维护。
- 促进标准化工作。分开后，每层功能可以相对简单地被描述。

网络协议分层的缺点：功能可能出现在多个层里，产生了额外开销。

为了使不同体系结构的计算机网络都能互联，国际标准化组织 ISO 于1977年提出了一个试图使各种计算机在世界范围内互联成网的标准框架，即著名的开放系

统互联基本参考模型 OSI/RM，简称为OSI。

OSI 的七层协议体系结构的概念清楚，理论也较完整，但它既复杂又不实用，TCP/IP 体系结构则不同，但它现在却得到了非常广泛的应用。TCP/IP 是一个四层体系结构，它包含应用层，运输层，网际层和网络接口层（用网际层这个名字是强调这一层是为了解决不同网络的互连问题），不过从实质上讲，TCP/IP 只有最上面的三层，因为最下面的网络接口层并没有什么具体内容，因此在学习计算机网络的原理时往往采用折中的办法，即综合 OSI 和 TCP/IP 的优点，采用一种只有五层协议的体系结构，这样既简洁又能将概念阐述清楚，有时为了方便，也可把最底下两层称为网络接口层。

四层协议，五层协议和七层协议的关系如下：

- TCP/IP是一个四层的体系结构，主要包括：应用层、运输层、网际层和网络接口层。
- 五层协议的体系结构主要包括：应用层、运输层、网络层，数据链路层和物理层。
- OSI七层协议模型主要包括是：应用层（Application）、表示层（Presentation）、会话层（Session）、运输层（Transport）、网络层（Network）、数据链路层（Data Link）、物理层（Physical）。

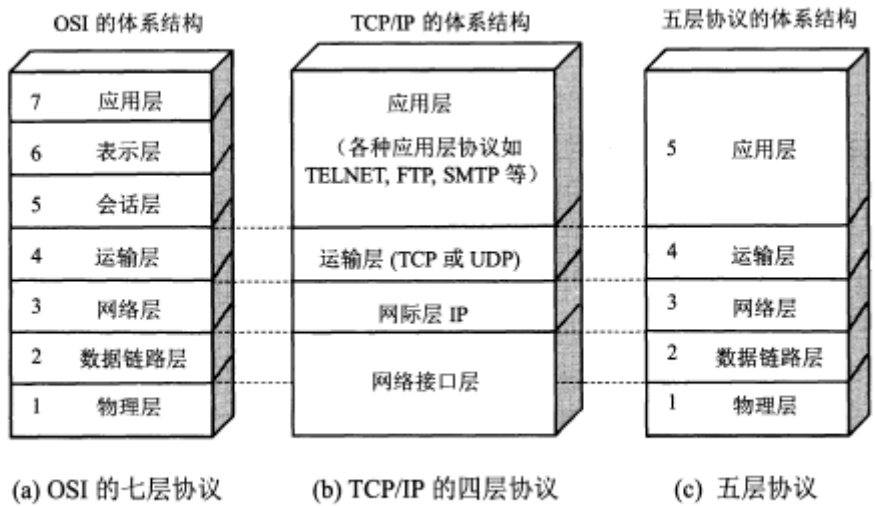


图 1-18 计算机网络体系结构

在这里插入图片描述

注：五层协议的体系结构只是为了介绍网络原理而设计的，实际应用还是 TCP/IP 四层体系结构。

TCP/IP 协议族

应用层

应用层(application-layer) 的任务是通过应用进程间的交互来完成特定网络应用。应用层协议定义的是应用进程（进程：主机中正在运行的程序）间的通信和交互的规则。

对于不同的网络应用需要不同的应用层协议。在互联网中应用层协议很多，如域名系统 DNS，支持万维网应用的 HTTP 协议，支持电子邮件的 SMTP 协议等等。

运输层

运输层(transport layer)的主要任务就是负责向两台主机进程之间的通信提供通用的**数据传输服务**。应用进程利用该服务传送应用层报文。

运输层主要使用一下两种协议

- 1. 传输控制协议-TCP：提供面向连接的，可靠的数据传输服务。
- 2. 用户数据协议-UDP：提供无连接的，尽最大努力的数据传输服务（不保证数据传输的可靠性）。

UDP	TCP	
是否连接	无连接	面向连接
是否可靠	不可靠传输，不使用流量控制和拥塞控制	可靠传输，使用流量控制和拥塞控制
连接对象个数	支持一对一，一对多，多对一和多对多交互通信	只能是一对一通信
传输方式	面向报文	面向字节流
首部开销	首部开销小，仅8字节	首部最小20字节，最大60字节
场景	适用于实时应用（IP电话、视频会议、直播等）	适用于要求可靠传输的应用，例如文件传输

每一个应用层（TCP/IP参考模型的最高层）协议一般都会使用到两个传输层协议之一：

运行在TCP协议上的协议：

- HTTP（Hypertext Transfer Protocol，超文本传输协议），主要用于普通浏览。
- HTTPS（HTTP over SSL，安全超文本传输协议），HTTP协议的安全版本。
- FTP（File Transfer Protocol，文件传输协议），用于文件传输。

- POP3 (Post Office Protocol, version 3, 邮局协议) , 收邮件用。
- SMTP (Simple Mail Transfer Protocol, 简单邮件传输协议) , 用来发送电子邮件。
- TELNET (Teletype over the Network, 网络电传) , 通过一个终端 (terminal) 登陆到网络。
- SSH (Secure Shell, 用于替代安全性差的TELNET) , 用于加密安全登陆用。

运行在UDP协议上的协议:

- BOOTP (Boot Protocol, 启动协议) , 应用于无盘设备。
- NTP (Network Time Protocol, 网络时间协议) , 用于网络同步。
- DHCP (Dynamic Host Configuration Protocol, 动态主机配置协议) , 动态配置IP地址。

运行在TCP和UDP协议上:

- DNS (Domain Name Service, 域名服务) , 用于完成地址查找, 邮件转发等工作。

网络层

网络层的任务就是选择合适的网间路由和交换结点, 确保计算机通信的数据及时传送。在发送数据时, 网络层把运输层产生的报文段或用户数据报封装成分组和包进行传送。在 TCP/IP 体系结构中, 由于网络层使用 IP 协议, 因此分组也叫 IP 数据报, 简称数据报。

互联网是由大量的异构 (heterogeneous) 网络通过路由器 (router) 相互连接起来的。互联网使用的网络层协议是无连接的网际协议 (Intert Prococol) 和许多路由选择协议, 因此互联网的网络层也叫做网际层或 IP 层。

数据链路层

数据链路层(data link layer)通常简称为链路层。两台主机之间的数据传输, 总是在一段一段的链路上传送的, 这就需要使用专门的链路层的协议。

在两个相邻节点之间传送数据时, 数据链路层将网络层交下来的 IP 数据报组装成帧, 在两个相邻节点间的链路上传送帧。每一帧包括数据和必要的控制信息 (如同步信息, 地址信息, 差错控制等) 。

在接收数据时, 控制信息使接收端能够知道一个帧从哪个比特开始和到哪个比特结束。

一般的web应用的通信传输流是这样的:



img

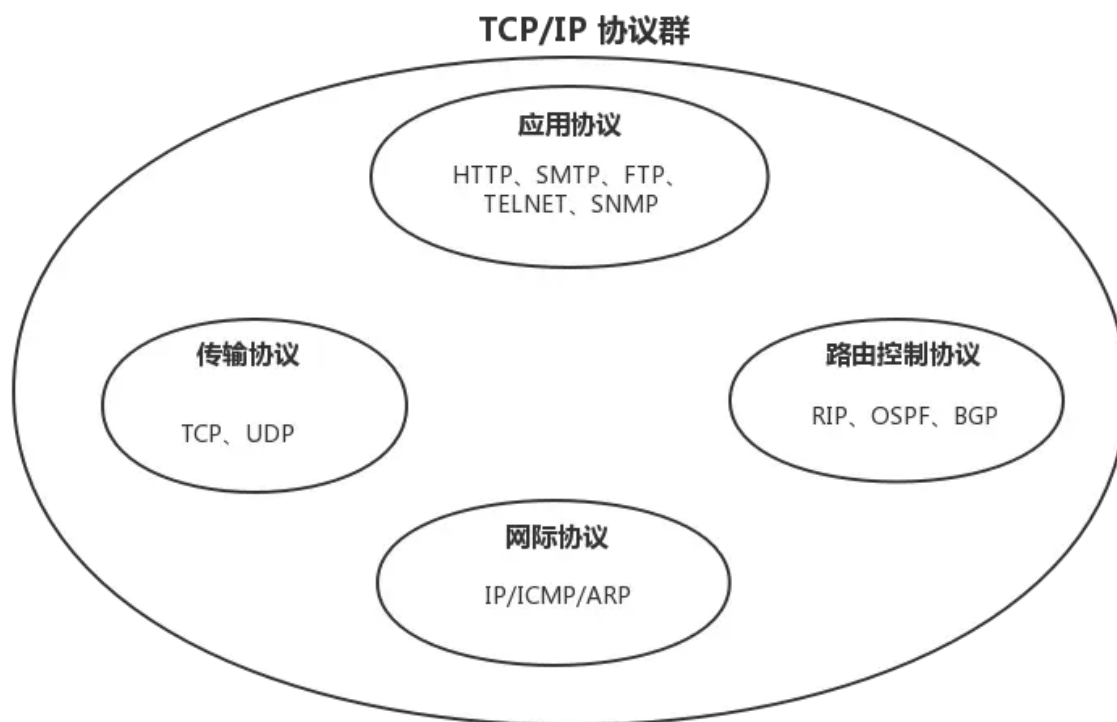
发送端在层与层之间传输数据时，每经过一层时会被打上一个该层所属的首部信息。反之，接收端在层与层之间传输数据时，每经过一层时会把对应的首部信息去除。

物理层

在物理层上所传送的数据单位是比特。物理层(physical layer)的作用是实现相邻计算机节点之间比特流的透明传送，尽可能屏蔽掉具体传输介质和物理设备的差异。使其上面的数据链路层不必考虑网络的具体传输介质是什么。“透明传送比特流”表示经实际电路传送后的比特流没有发生变化，对传送的比特流来说，这个电路好像是看不见的。

TCP/IP 协议族

在互联网使用的各种协议中最重要和最著名的就是 TCP/IP 两个协议。现在人们经常提到的 TCP/IP 并不一定是单指 TCP 和 IP 这两个具体的协议，而往往是表示互联网所使用的整个 TCP/IP 协议族。



img

互联网协议套件（英语：Internet Protocol Suite，缩写**IPS**）是一个网络通讯模型，以及一整个网络传输协议家族，为网际网络的基础通讯架构。它常被通称为TCP/IP协议族（英语：**TCP/IP Protocol Suite**，或**TCP/IP Protocols**），简称**TCP/IP**。因为该协定家族的两个核心协定：**TCP**（传输控制协议）和**IP**（网际协议），为该家族中最早通过的标准。

划重点：

TCP（传输控制协议）和**IP**（网际协议）是最先定义的两个核心协议，所以才统称为**TCP/IP协议族**

TCP的三次握手四次挥手

TCP是一种面向连接的、可靠的、基于字节流的传输层通信协议，在发送数据前，通信双方必须在彼此间建立一条连接。所谓的“连接”，其实是客户端和服务端保存的一份关于对方的信息，如ip地址、端口号等。

TCP可以看成是一种字节流，它会处理IP层或以下的层的丢包、重复以及错误问题。在连接的建立过程中，双方需要交换一些连接的参数。这些参数可以放在TCP头部。

一个TCP连接由一个4元组构成，分别是两个IP地址和两个端口号。一个TCP连接通常分为三个阶段：连接、数据传输、退出（关闭）。**通过三次握手建立一个**

链接，通过四次挥手来关闭一个连接。

当一个连接被建立或被终止时，交换的报文段只包含TCP头部，而没有数据。

TCP报文的头部结构

在了解TCP连接之前先来了解一下TCP报文的头部结构。

TCP Header																																					
Offsets	Octet	0								1								2								3											
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31				
0	0	Source port (16位源端口号)																Destination port (16位目标端口号)																			
4	32	Sequence number (32位序列号)																																			
8	64	Acknowledgment number (if ACK set) (32位确认号)																																			
12	96	Data offset (4位头部长度)				Reserved (6位保留)						U	A	P	R	S	F	Window Size (16位窗口大小)																			
																R	C																	S	S	Y	I
																G	K																	H	T	N	N
16	128	Checksum (16位校验和)																Urgent pointer (if URG set) (16紧急指针)																			
20	160	Options (if data offset > 5. Padded at the end with "0" bytes if necessary.)																																			
...	...	选项，长度可变，最多40个字节[kind(1字节)，length(1字节)，info(n字节)]																																			

TCPHeader.png

上图中有几个字段需要重点介绍下：

(1) 序号：seq序号，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。

(2) 确认序号：ack序号，占32位，只有ACK标志位为1时，确认序号字段才有效，ack=seq+1。

(3) 标志位：共6个，即URG、ACK、PSH、RST、SYN、FIN等，具体含义如下：

- ACK：确认序号有效。
- FIN：释放一个连接。
- PSH：接收方应该尽快将这个报文交给应用层。
- RST：重置连接。
- SYN：发起一个新连接。
- URG：紧急指针（urgent pointer）有效。

需要注意的是：

- 不要将确认序号ack与标志位中的ACK搞混了。
- 确认方ack=发起方seq+1，两端配对。

三次握手

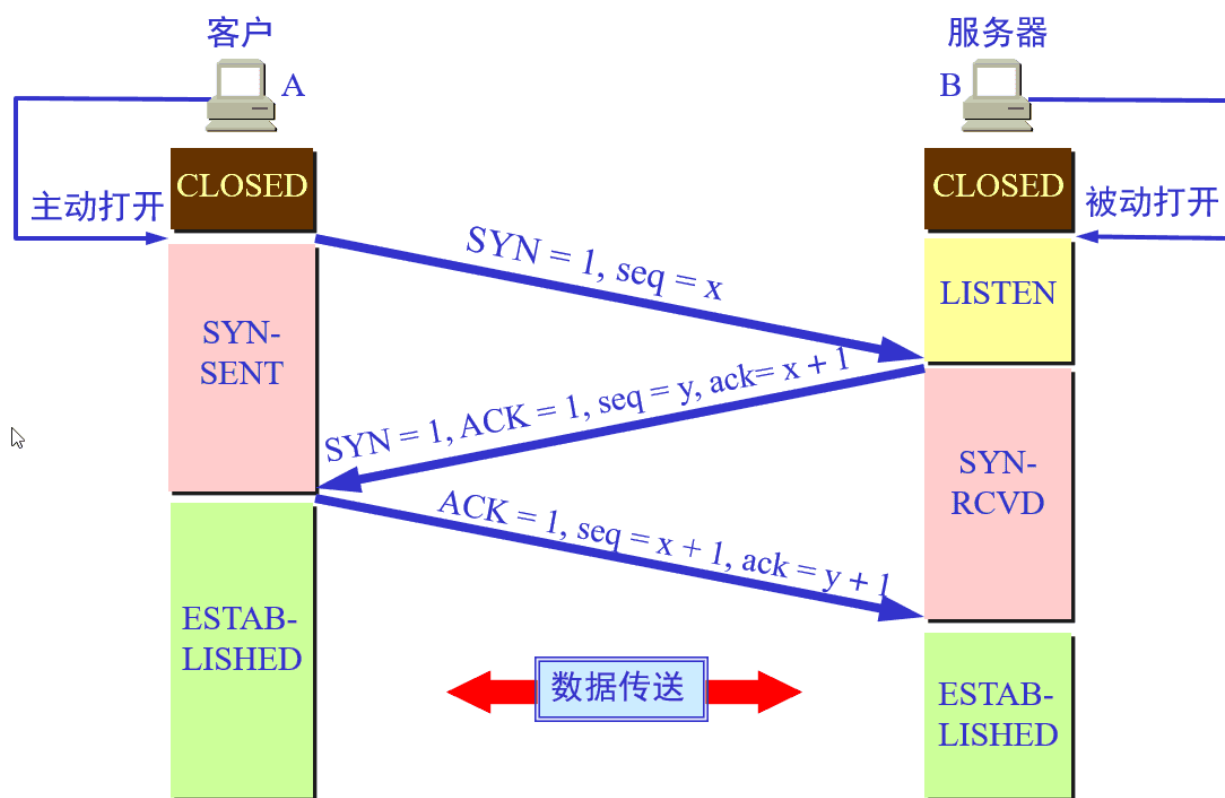
三次握手的本质是确认通信双方收发数据的能力

首先，我让信使运输一份信件给对方，对方收到了，那么他就知道了我的发件能力和他的收件能力是可以的。

于是他给我回信，**我若收到了，我便知我的发件能力和他的收件能力是可以的，并且他的发件能力和我的收件能力是可以。**

然而此时他还不知道他的发件能力和我的收件能力到底可不可以，于是我最后回馈一次，**他若收到了，他便清楚了他的发件能力和我的收件能力是可以的。**

这，就是三次握手，这样说，你理解了吗？



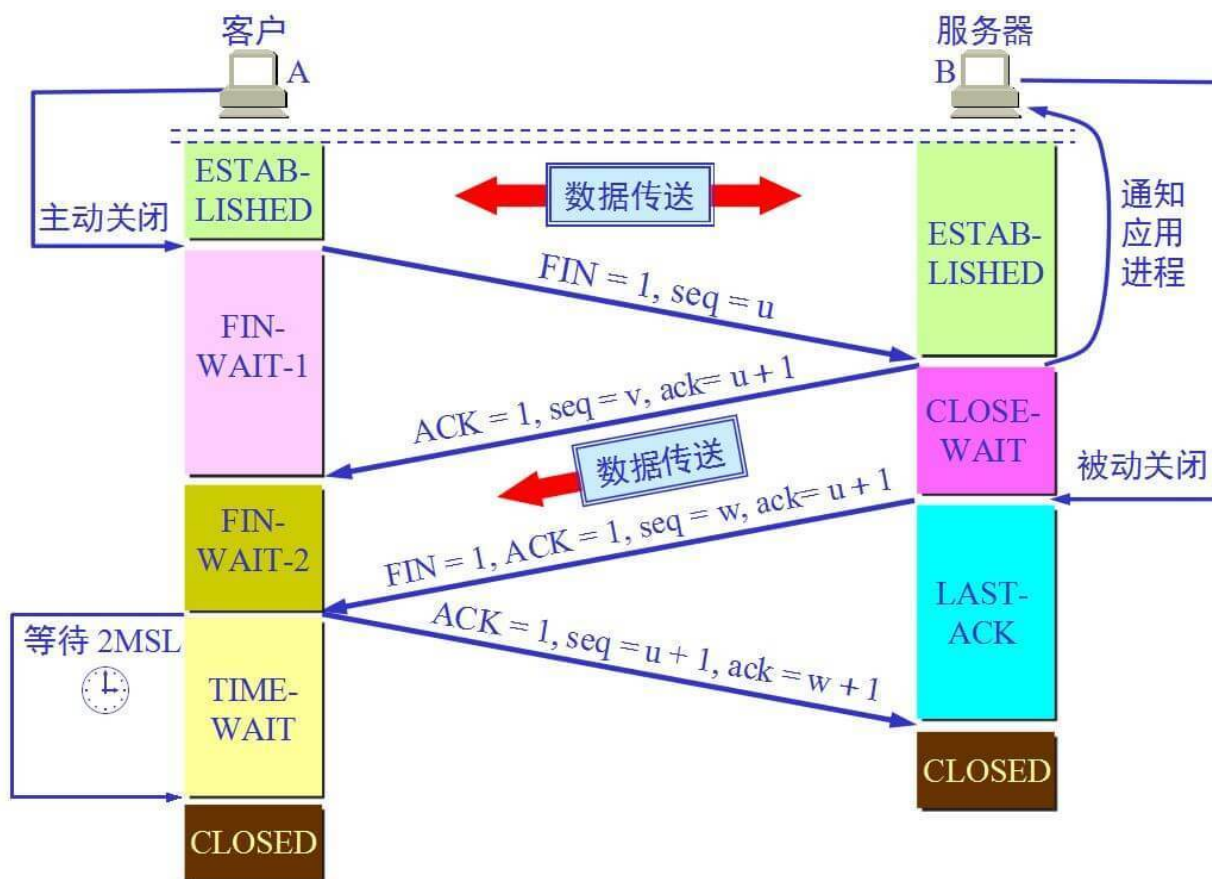
三次握手.png

- **第一次握手**：客户端要向服务端发起连接请求，首先客户端随机生成一个起始序列号ISN(比如是100)，那客户端向服务端发送的报文段包含SYN标志位(也就是SYN=1)，序列号seq=100。
- **第二次握手**：服务端收到客户端发过来的报文后，发现SYN=1，知道这是一个连接请求，于是将客户端的起始序列号100存起来，并且随机生成一个服务端的起始序列号(比如是300)。然后给客户端回复一段报文，回复报文包含SYN和ACK标志(也就是SYN=1,ACK=1)、序列号seq=300、确认号ack=101(客户端发过来的序列号+1)。
- **第三次握手**：客户端收到服务端的回复后发现ACK=1并且ack=101,于是知道服务端已经收到了序列号为100的那段报文；同时发现SYN=1，知道了服务端同意了这次连接，于是就将服务端的序列号300给存下来。然后客户端再回复一段报文给服务端，报文包含ACK标志位(ACK=1)、ack=301(服务端序列号+1)、seq=101(第一次握手时发送报文是占据一个序列号的，所以这次seq就从101开始，需要注意的是不携带数据的ACK报文是不占据序列号的，所以后面第一次正式发送数据时seq还是

101)。当服务端收到报文后发现ACK=1并且ack=301，就知道客户端收到序列号为300的报文了，就这样客户端和服务端通过TCP建立了连接。

四次挥手

四次挥手的目的是关闭一个连接



四次挥手.jpeg

比如客户端初始化的序列号ISA=100，服务端初始化的序列号ISA=300。TCP连接成功后客户端总共发送了1000个字节的数据，服务端在客户端发FIN报文前总共回复了2000个字节的数据。

- **第一次挥手：**当客户端的数据都传输完成后，客户端向服务端发出连接释放报文（当然数据没发完时也可以发送连接释放报文并停止发送数据），释放连接报文包含FIN标志位($FIN=1$)、序列号 $seq=1101(100+1+1000)$ ，其中的1是建立连接时占的一个序列号)。需要注意的是客户端发出FIN报文段后只是不能发数据了，但是还可以正常收数据；另外FIN报文段即使不携带数据也要占据一个序列号。
- **第二次挥手：**服务端收到客户端发的FIN报文后给客户端回复确认报文，确认报文包含ACK标志位($ACK=1$)、确认号 $ack=1102$ (客户端FIN报文序列号1101+1)、序列号 $seq=2300(300+2000)$ 。此时服务端处于关闭等待状态，而不是立马给客户端发FIN报文，这个状态还要持续一段时间，因为服务端可能还有数据没发完。

- **第三次挥手**：服务端将最后数据(比如50个字节)发送完毕后就向客户端发出连接释放报文，报文包含FIN和ACK标志位(FIN=1,ACK=1)、确认号和第二次挥手一样ack=1102、序列号seq=2350(2300+50)。
- **第四次挥手**：客户端收到服务端发的FIN报文后，向服务端发出确认报文，确认报文包含ACK标志位(ACK=1)、确认号ack=2351、序列号seq=1102。注意客户端发出确认报文后不是立马释放TCP连接，而是要经过2MSL(最长报文段寿命的2倍时长)后才释放TCP连接。而服务端一旦收到客户端发出的确认报文就会立马释放TCP连接，所以服务端结束TCP连接的时间要比客户端早一些。

常见面试题

为什么TCP连接的时候是3次？2次不可以吗？

因为需要考虑连接时丢包的问题，如果只握手2次，第二次握手时如果服务端发给客户端的确认报文段丢失，此时服务端已经准备好了收发数(可以理解服务端已经连接成功)据，而客户端一直没收到服务端的确认报文，所以客户端就不知道服务端是否已经准备好了(可以理解为客户端未连接成功)，这种情况下客户端不会给服务端发数据，也会忽略服务端发过来的数据。

如果是三次握手，即便发生丢包也不会有问题，比如如果第三次握手客户端发的确认ack报文丢失，服务端在一段时间内没有收到确认ack报文的话就会重新进行第二次握手，也就是服务端会重发SYN报文段，客户端收到重发的报文段后会再次给服务端发送确认ack报文。

为什么TCP连接的时候是3次，关闭的时候却是4次？

因为只有在客户端和服务端都没有数据要发送的时候才能断开TCP。而客户端发出FIN报文时只能保证客户端没有数据发了，服务端还有没有数据发客户端是不知道的。而服务端收到客户端的FIN报文后只能先回复客户端一个确认报文来告诉客户端我服务端已经收到你的FIN报文了，但我服务端还有一些数据没发完，等这些数据发完了服务端才能给客户端发FIN报文(所以不能一次性将确认报文和FIN报文发给客户端，就是这里多出来了一次)。

为什么客户端发出第四次挥手的确认报文后要等2MSL的时间才能释放TCP连接？

这里同样是要考虑丢包的问题，如果第四次挥手的报文丢失，服务端没收到确认ack报文就会重发第三次挥手的报文，这样报文一去一回最长时间就是2MSL，所以需要等这么长时间来确认服务端确实已经收到了。

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设有一个保活计时器，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。

什么是HTTP，HTTP 与 HTTPS 的区别

HTTP 是一个在计算机世界里专门在两点之间传输文字、图片、音频、视频等超文本数据的约定和规范

区别	HTTP	HTTPS
协议	运行在 TCP 之上，明文传输，客户端与服务器端都无法验证对方的身份	身披 SSL(Secure Socket Layer)外壳的 HTTP，运行于 SSL 上，SSL 运行于 TCP 之上，是添加了加密和认证机制的 HTTP。
端口	80	443
资源消耗	较少	由于加解密处理，会消耗更多的 CPU 和内存资源
开销	无需证书	需要证书，而证书一般需要向认证机构购买
加密机制	无	共享密钥加密和公开密钥加密并用的混合加密机制
安全性	弱	由于加密机制，安全性强

常用HTTP状态码

HTTP状态码表示客户端HTTP请求的返回结果、标识服务器处理是否正常、表明请求出现的错误等。

状态码的类别：

类别	原因短语
1XX	Informational（信息性状态码） 接受的请求正在处理

2XX	Success（成功状态码） 请求正常处理完毕
3XX	Redirection（重定向状态码） 需要进行附加操作以完成请求
4XX	Client Error（客户端错误状态码） 服务器无法处理请求
5XX	Server Error（服务器错误状态码） 服务器处理请求出错

常用HTTP状态码：

2XX	成功（这系列表明请求被正常处理了）
200	OK，表示从客户端发来的请求在服务器端被正确处理
204	No content，表示请求成功，但响应报文不含实体的主体部分
206	Partial Content，进行范围请求成功
3XX	重定向（表明浏览器要执行特殊处理）
301	moved permanently，永久性重定向，表示资源已被分配了新的 URL
302	found，临时性重定向，表示资源临时被分配了新的 URL
303	see other，表示资源存在着另一个 URL，应使用 GET 方法获取资源（对于 301/302/303响应，几乎所有浏览器都会删除报文主体并自动用GET重新请求）
304	not modified，表示服务器允许访问资源，但请求未满足条件的情况（与重定向无关）
307	temporary redirect，临时重定向，和302含义类似，但是期望客户端保持请求方法不变向新的地址发出请求
4XX	客户端错误
400	bad request，请求报文存在语法错误
401	unauthorized，表示发送的请求需要有通过 HTTP 认证的认证信息
403	forbidden，表示对请求资源的访问被服务器拒绝，可在实体主体部分返回原因描述
404	not found，表示在服务器上没有找到请求的资源
5XX	服务器错误
500	internal sever error，表示服务器端在执行请求时发生了错误
501	Not Implemented，表示服务器不支持当前请求所需要的某个功能
503	service unavailable，表明服务器暂时处于超负载或正在停机维护，无法处理请求

GET和POST区别

说道GET和POST，就不得不提HTTP协议，因为浏览器和服务器的交互是通过HTTP协议执行的，而GET和POST也是HTTP协议中的两种方法。

HTTP全称为Hyper Text Transfer Protocol，中文翻译为超文本传输协议，目的是保证浏览器与服务器之间的通信。HTTP的工作方式是客户端与服务端之间的请求-应答协议。

HTTP协议中定义了浏览器和服务端进行交互的不同方法，基本方法有4种，分别是GET，POST，PUT，DELETE。这四种方法可以理解为，对服务器资源的查，改，增，删。

- GET：从服务器上获取数据，也就是所谓的查，仅仅是获取服务器资源，不进行修改。
- POST：向服务器提交数据，这就涉及到了数据的更新，也就是更改服务器的数据。
- PUT：英文含义是放置，也就是向服务器新添加数据，就是所谓的增。
- DELETE：从字面意思也能看出，这种方式就是删除服务器数据的过程。

GET和POST区别

1. Get是不安全的，因为在传输过程，数据被放在请求的URL中；Post的所有操作对用户来说都是不可见的。但是这种做法也不绝对，大部分人的做法也是按照上面的说法来的，但是也可以在get请求加上 request body，给 post请求带上 URL 参数。
2. Get请求提交的url中的数据最多只能是2048字节，这个限制是浏览器或者服务器给添加的，http协议并没有对url长度进行限制，目的是为了保证服务器和浏览器能够正常运行，防止有人恶意发送请求。Post请求则没有大小限制。
3. Get限制Form表单的数据集的值必须为ASCII字符；而Post支持整个ISO10646字符集。
4. Get执行效率却比Post方法好。Get是form提交的默认方法。
5. GET产生一个TCP数据包；POST产生两个TCP数据包。

对于GET方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据）；

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

什么是对称加密与非对称加密

对称密钥加密是指加密和解密使用同一个密钥的方式，**这种方式存在的最大问题就是密钥发送问题，即如何安全地将密钥发给对方；**

而非对称加密是指使用一对非对称密钥，即公钥和私钥，公钥可以随意发布，但私钥只有自己知道。发送密文的一方使用对方的公钥进行加密处理，对方接收到加密信息后，使用自己的私钥进行解密。

由于非对称加密的方式不需要发送用来解密的私钥，所以可以保证安全性；但是和对称加密比起来，非常的慢

什么是HTTP2

HTTP2 可以提高了网页的性能。

在 HTTP1 中浏览器限制了同一个域名下的请求数量（Chrome 下一般是六个），当在请求很多资源的时候，由于队头阻塞当浏览器达到最大请求数量时，剩余的资源需等待当前的六个请求完成后才能发起请求。

HTTP2 中引入了多路复用的技术，这个技术可以只通过一个 TCP 连接就可以传输所有的请求数据。多路复用可以绕过浏览器限制同一个域名下的请求数量的问题，进而提高了网页的性能。

Session、Cookie和Token的主要区别

HTTP协议本身是无状态的。什么是无状态呢，即服务器无法判断用户身份。

什么是cookie

cookie是由Web服务器保存在用户浏览器上的小文件（key-value格式），包含用户相关的信息。客户端向服务器发起请求，如果服务器需要记录该用户状态，就使用response向客户端浏览器颁发一个Cookie。客户端浏览器会把Cookie保存起来。当浏览器再请求该网站时，浏览器把请求的网址连同该Cookie一同提交给服务器。服务器检查该Cookie，以此来辨认用户身份。

什么是session

session是依赖Cookie实现的。session是服务器端对象

session 是浏览器和服务端会话过程中，服务器分配的一块储存空间。服务器默认认为浏览器在cookie中设置 sessionid，浏览器在向服务器请求过程中传输 cookie 包含 sessionid，服务器根据 sessionid 获取出会话中存储的信息，然后确定会话的身份信息。

cookie与session区别

- 存储位置与安全性：cookie数据存放在客户端上，安全性较差，session数据放在服务器上，安全性相对更高；
- 存储空间：单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie，session无此限制
- 占用服务器资源：session一定时间内保存在服务器上，当访问增多，占用服务器性能，考虑到服务器性能方面，应当使用cookie。

什么是Token

Token的引入：Token是在客户端频繁向服务端请求数据，服务端频繁的去数据库查询用户名和密码并进行对比，判断用户名和密码正确与否，并作出相应提示，在这样的背景下，Token便应运而生。

Token的定义：Token是服务端生成的一串字符串，以作客户端进行请求的一个令牌，当第一次登录后，服务器生成一个Token便将此Token返回给客户端，以后客户端只需带上这个Token前来请求数据即可，无需再次带上用户名和密码。

使用Token的目的：Token的目的是为了减轻服务器的压力，减少频繁的查询数据库，使服务器更加健壮。

Token 是在服务端产生的。如果前端使用用户名/密码向服务端请求认证，服务端认证成功，那么在服务端会返回 Token 给前端。前端可以在每次请求的时候带上 Token 证明自己的合法地位

session与token区别

- session机制存在服务器压力增大，CSRF跨站伪造请求攻击，扩展性不强等问题；
- session存储在服务器端，token存储在客户端
- token提供认证和授权功能，作为身份认证，token安全性比session好；
- session这种会话存储方式只适用于客户端代码和服务端代码运行在同一台服务器上，token适用于项目级的前后端分离（前后端代码运行在不同的服务器下）

Servlet是线程安全的吗

Servlet不是线程安全的，多线程并发的读写会导致数据不同步的问题。

解决的办法是尽量不要定义name属性，而是要把name变量分别定义在doGet()和doPost()方法内。虽然使用synchronized(name){}语句块可以解决问题，但是会造成线程的等待，不是很科学的办法。

注意：多线程的并发的读写Servlet类属性会导致数据不同步。但是如果只是并发地读取属性而不写入，则不存在数据不同步的问题。因此Servlet里的只读属性最好定义为final类型的。

Servlet接口中有哪些方法及Servlet生命周期探秘

在Java Web程序中，**Servlet**主要负责接收用户请求**HttpServletRequest**，在**doGet()**，**doPost()**中做相应的处理，并将回应**HttpServletResponse**反馈给用户。Servlet可以设置初始化参数，供Servlet内部使用。

Servlet接口定义了5个方法，其中**前三个方法与Servlet生命周期相关**：

- **void init(ServletConfig config) throws ServletException**
- **void service(ServletRequest req, ServletResponse resp) throws ServletException, java.io.IOException**
- **void destroy()**
- java.lang.String getServletInfo()
- ServletConfig getServletConfig()

生命周期：

Web容器加载Servlet并将其实例化后，Servlet生命周期开始，容器运行其**init()**方法进行Servlet的初始化；

请求到达时调用Servlet的**service()方法**，service()方法会根据需要调用与请求对应的**doGet或doPost**等方法；

当服务器关闭或项目被卸载时服务器会将Servlet实例销毁，此时会调用Servlet的**destroy()方法**。

init方法和destory方法只会执行一次，service方法客户端每次请求Servlet都会执行。Servlet中有时会用到一些需要初始化与销毁的资源，因此可以把初始化资源的代码放入init方法中，销毁资源的代码放入destroy方法中，这样就不需要每次处理客户端的请求都要初始化与销毁资源。

如果客户端禁止 cookie 能实现 session 还能用吗？

Cookie 与 Session，一般认为是两个独立的东西，Session采用的是在服务器端保持状态的方案，而Cookie采用的是在客户端保持状态的方案。

但为什么禁用Cookie就不能得到Session呢？因为Session是用Session ID来确定当前对话所对应的服务器Session，而Session ID是通过Cookie来传递的，禁用Cookie相当于失去了Session ID，也就得不到Session了。

假定用户关闭Cookie的情况下使用Session，其实现途径有以下几种：

1. 手动通过URL传值、隐藏表单传递Session ID。
2. 用文件、数据库等形式保存Session ID，在跨页过程中手动调用。