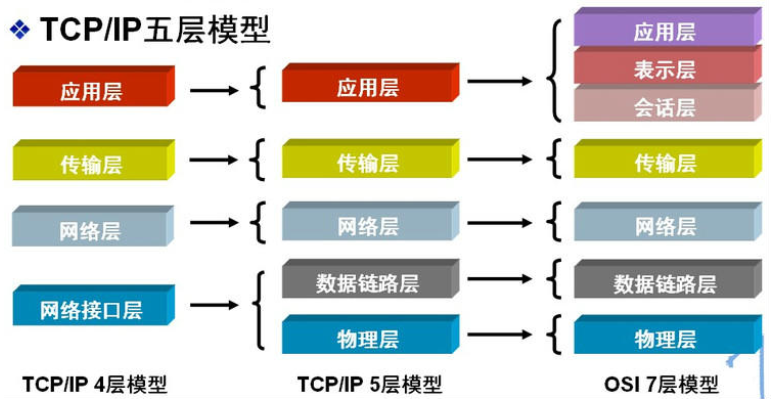
计算机网络面试整理

1. OSI模型层次结构以及各个层的常见协议

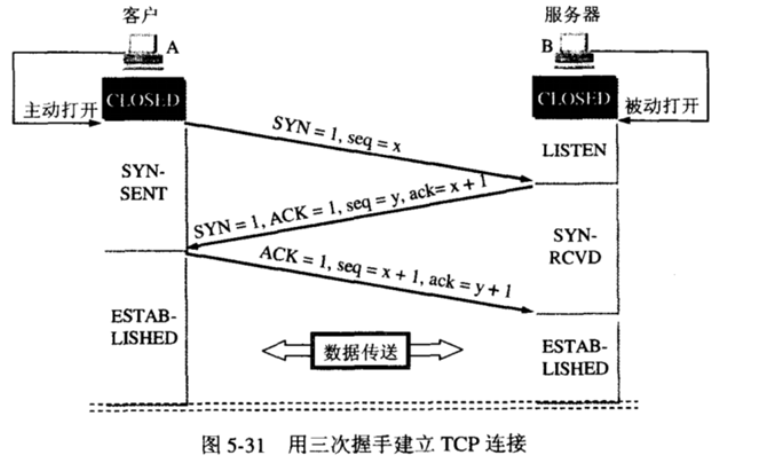
|  |  |
| --- | --- |
| 应用层 | HTTP、DNS、FTP、SMTP、POP3、OSPF、BGP |
| 表示层 | JPEG、MPEG |
| 会话层 | SQL |
| 传输层 | TCP、UDP |
| 网络层 | IP、ICMP |
| 数据链路层 | ARP、MAC、VLAN、HDLC、PPP |
| 物理层 | IEEE802.3 |



2. HTTP协议和HTTPS的区别

|  |  |  |
| --- | --- | --- |
|  | HTTP | HTTPS |
| 端口号 | 80 | 443 |
| 信息传输 | 明文 | 运行于SSL之上，添加了加密和认证 |
| 资源消耗 | 少 | 多 |
| 状态 | 无状态的 | 有状态的 |
| 证书 | 无 | 需要用到CA颁发的证书 |

3. TCP连接建立的三次握手



【问】为什么不能用两次握手进行连接？

第三次握手是为了**防止失效的连接请求到达服务器，让服务器错误打开连接**。

客户端发送的连接请求如果在网络中滞留，那么就会隔很长一段时间才能收到服务器端发回的连接确认。客户端等待一个超时重传时间之后，就会重新请求连接。但是这个滞留的连接请求最后还是会到达服务器，如果不进行三次握手，那么服务器就会打开两个连接。如果有第三次握手，客户端会忽略服务器之后发送的对滞留连接请求的连接确认，不进行第三次握手，因此就不会再次打开连接。

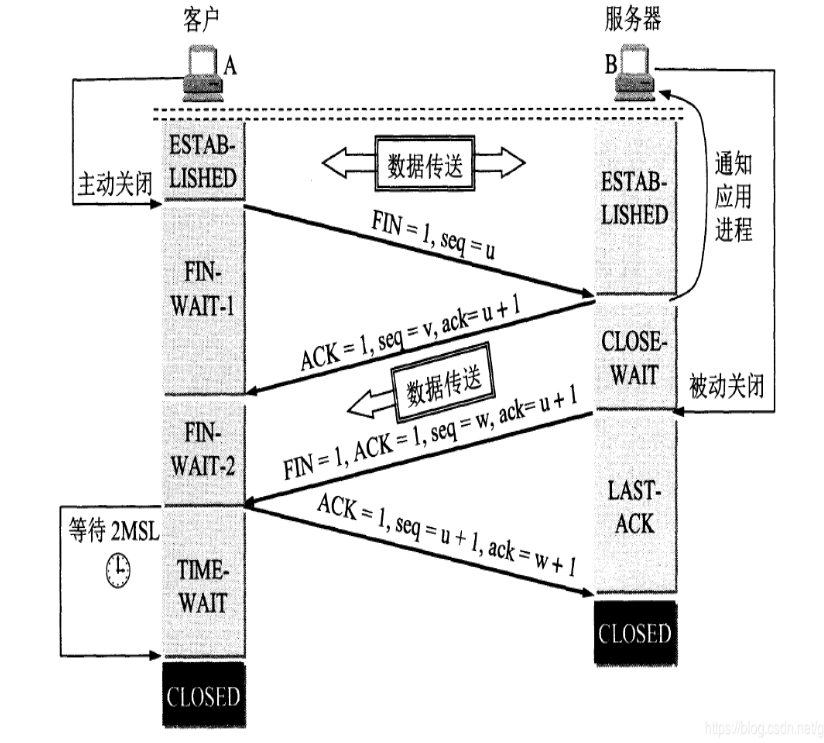
【问】为什么不需要四次握手连接？

四次连接的过程可以概括为：①A发送SYN给B ②B发送ACK给A ③B发送SYN给A ④A发送ACK给B。其中②③两步可以合并，提高传输效率。

【问】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。

4. TCP释放连接的四次挥手



【问】为什么需要四次挥手？

客户端发送了 FIN 连接释放报文之后，服务器收到了这个报文，就进入了 CLOSE-WAIT 状态。这个状态是为了让服务器端发送还未传送完毕的数据，传送完毕之后，服务器会发送 FIN 连接释放报文。

【问】为什么TIME\_WAIT状态需要经过2MSL才能返回到CLOSE状态？

**确保最后一个确认报文能够到达**。如果 B 没收到 A 发送来的确认报文，那么就会重新发送连接释放请求报文（不断重复发送FIN），A 等待一段时间就是为了处理这种情况的发生，如果在等待的过程中收到了FIN，Client会重发ACK并再等待2MSL，直到2MSL内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。同时等待一段时间可以让本连接持续时间内所产生的所有报文都从网络中消失，**使得下一个新的连接不会出现旧的连接请求报文**。

【问】太多TIME\_WAIT和CLOSE\_WAIT状态可能的原因？

过多CLOSE\_WAIT的原因可能是①服务端Socket代码没有写close函数关闭Socket连接，或者出现死循环永远执行不到close ②客户端响应太慢

5. TCP协议如何保证传输的可靠性？（流量、拥塞、顺序、校验）

* **数据包校验**：目的是检测数据在传输过程中的任何变化，若校验出包有错，则丢弃报文段并且不给出响应，这时TCP发送数据端超时后会重发数据；
* **对失序数据包重排序**：既然TCP报文段作为IP数据报来传输，而IP数据报的到达可能会失序，因此TCP报文段的到达也可能会失序。TCP将对失序数据进行重新排序，然后才交给应用层；
* **丢弃重复数据**：对于重复数据，能够丢弃重复数据；
* **应答机制**：当TCP收到发自TCP连接另一端的数据，它将发送一个确认。这个确认不是立即发送，通常将推迟几分之一秒；
* **超时重发**：当TCP发出一个段后，它启动一个定时器，等待目的端确认收到这个报文段。如果不能及时收到一个确认，将重发这个报文段；
* **流量控制**：TCP连接的每一方都有固定大小的缓冲空间。TCP的接收端只允许另一端发送接收端缓冲区所能接纳的数据，这可以防止较快主机致使较慢主机的缓冲区溢出，这就是流量控制。TCP使用的流量控制协议是可变大小的滑动窗口协议。

6. TCP和UDP的区别

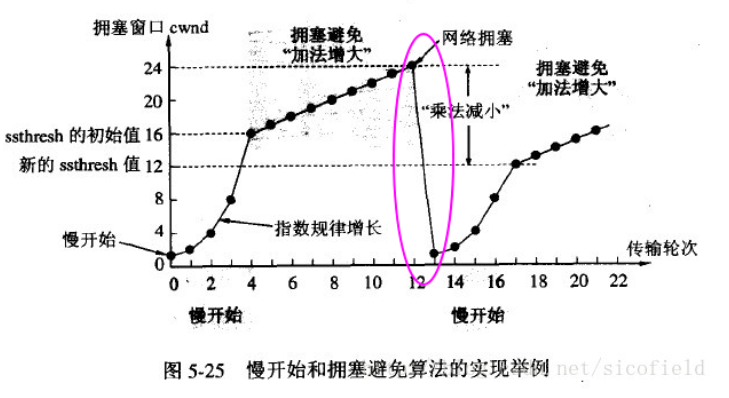
|  |  |
| --- | --- |
| TCP | UDP |
| 面向连接的、可靠的传输层协议 | 无连接的、不可靠的传输层协议 |
| 只支持点对点通信 | 支持多对多的通信模式 |
| 面向字节流的 | 面向报文的 |
| 具有拥塞控制 | 没有拥塞控制 |
| 首部开销大（20字节） | 首部开销小（8字节） |

7. TCP的拥塞处理

拥塞控制就是防止过多的数据注入网络中，这样可以使网络中的路由器或链路不致过载。注意，拥塞控制和流量控制不同，前者是一个全局性的过程，而后者指点对点通信量的控制。拥塞控制的方法主要有以下四种：

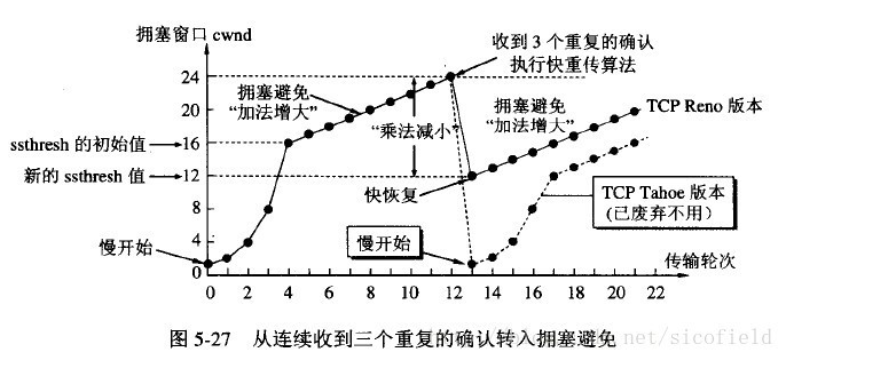
（1）**慢启动**：不要一开始就发送大量的数据，先探测一下网络的拥塞程度，也就是说由小到大逐渐增加拥塞窗口的大小

（2）**拥塞避免**：拥塞避免算法让拥塞窗口缓慢增长，即每经过一个往返时间RTT就把发送方的拥塞窗口cwnd加1，而不是加倍，这样拥塞窗口按线性规律缓慢增长。



（3）**快重传**：快重传要求接收方在**收到一个失序的报文段后就立即发出重复确认**（为的是使发送方及早知道有报文段没有到达对方）而不要等到自己发送数据时捎带确认。快重传算法规定，发送方**只要一连收到三个重复确认就应当立即重传对方尚未收到的报文段**，而不必继续等待设置的重传计时器时间到期。

（4）**快恢复**：快重传配合使用的还有快恢复算法，**当发送方连续收到三个重复确认时，就执行“乘法减小”算法，把ssthresh门限减半**，但是接下去并不执行慢开始算法：因为如果网络出现拥塞的话就不会收到好几个重复的确认，所以发送方现在认为网络可能没有出现拥塞。所以此时不执行慢开始算法，**而是将cwnd设置为ssthresh的大小**，然后执行拥塞避免算法。



8. 浏览器中输入一个网址到得到一个页面的过程

（1）浏览器**查询 DNS，获取域名对应的IP地址**：具体过程包括浏览器搜索自身的**DNS缓存**、搜索操作系统的DNS缓存、读取本地的Host文件和向本地DNS服务器进行查询等。对于向本地DNS服务器进行查询，如果要查询的域名包含在本地配置区域资源中，则返回解析结果给客户机，完成域名解析；如果要查询的域名不由本地DNS服务器区域解析，但该服务器已缓存了此网址映射关系，则调用这个IP地址映射，完成域名解析。如果本地域名服务器并未缓存该网址映射关系，那么将根据其设置**发起递归查询或者迭代查询**；

（2）浏览器获得域名对应的IP地址以后，**浏览器向服务器请求建立链接，发起三次握手**；

（3）TCP/IP链接建立起来后，浏览器**向服务器发送HTTP请求**；

（4）服务器接收到这个请求，并**根据路径参数映射到特定的请求处理器进行处理**，并将处理结果及相应的视图返回给浏览器；

（5）浏览器**解析并渲染视图**，若遇到对JS文件、CSS文件及图片等静态资源的引用，则重复上述步骤并向服务器请求这些资源；

（6）浏览器根据其请求到的资源、数据渲染页面，最终向用户呈现一个完整的页面。

9. Session和Cookie，以及两者之间的区别

Cookie和Session都是客户端与服务器之间保持状态的解决方案，具体来说，Cookie机制采用的是在客户端保持状态的方案，而Session机制采用的是在服务器端保持状态的方案。

（1）Cookie：Cookie实际上是一小段的文本信息。客户端请求服务器，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，而客户端浏览器会把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，以此来辨认用户状态。服务器还可以根据需要修改Cookie的内容。

（2）Session：客户端请求服务器，如果服务器记录该用户状态，就获取Session来保存状态，这时，如果服务器已经为此客户端创建过Session，服务器就按照SessionID把这个Session检索出来使用；如果客户端请求不包含SessionID，则为此客户端创建一个Session并且生成一个与此Session相关联的SessionID，并将这个SessionID在本次响应中返回给客户端保存。**保存这个SessionID的方式可以采用 Cookie机制**，这样在交互过程中浏览器可以自动的按照规则把这个标识发挥给服务器；若浏览器禁用Cookie的话，可以通过**URL重写机制**将SessionID传回服务器。

（3）对比

* **实现机制**：Session的实现常常依赖于Cookie机制，通过Cookie机制回传SessionID；
* **大小限制**：Cookie有大小限制并且浏览器对每个站点也有cookie的个数限制，Session没有大小限制，理论上只与服务器的内存大小有关；
* **安全性**：Cookie存在安全隐患，通过拦截或本地文件找得到Cookie后可以进行攻击，而Session由于保存在服务器端，相对更加安全；
* **服务器资源消耗**：Session是保存在服务器端上会存在一段时间才会消失，如果session过多会增加服务器的压力。

10. 常见的HTTP状态码

* 1×× : 请求处理中，请求已被接受，正在处理
* 2×× : 请求成功，请求被成功处理

200 OK

* 3×× : 重定向，要完成请求必须进行进一步处理

301：永久性转移

302：暂时性转移

304：请求未修改，命中缓存

* 4×× : 客户端错误，请求不合法

400：Bad Request，请求有语法问题

401：未授权

403：拒绝请求

404：客户端所访问的页面不存在

* 5×× : 服务器端错误，服务器不能处理合法请求

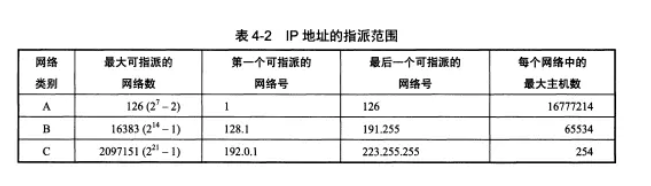
500：服务器内部错误

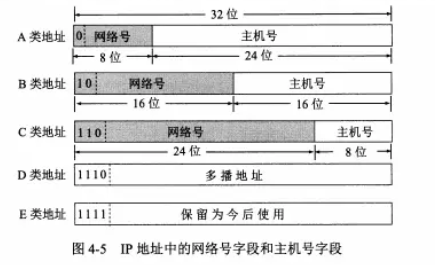
502：Bad Gateway 错误网关

503：服务不可用

11. IP地址的分类：

* 全零（“0．0．0．0”）地址对应于当前主机。
* 全“1”的IP地址（“255．255．255．255”）是当前子网的广播地址。
* 一般来说，A类地址保留给政府机构，B类地址分配给中等规模的公司，C类地址分配给任何需要的人，D类用于组播，E类用于实验，各类可容纳的地址数目不同。





12. NAT地址转换

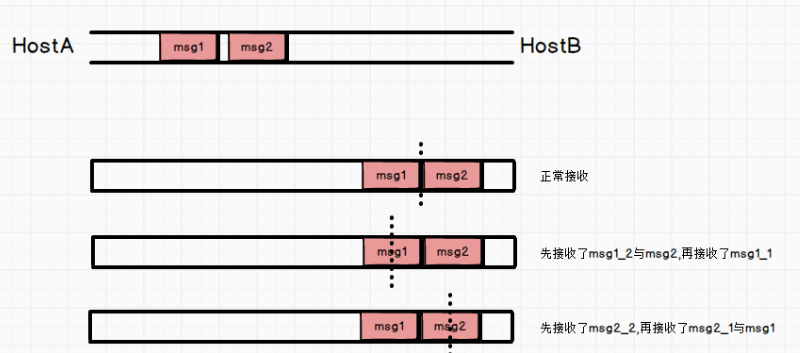
A类：10.0.0.0—10.255.255.255 10.0.0.0/8

B类：172.16.0.0—172.31.255.255 172.16.0.0/12

C类：192.168.0.0—192.168.255.255 192.168.0.0/16

13. TCP粘包问题

TCP是一个基于字节流的传输服务，"流"意味着TCP所传输的数据是没有边界的。这不同于UDP提供基于消息的传输服务，其传输的数据是有边界的。TCP的发送方无法保证对等方每次接收到的是一个完整的数据包。主机A向主机B发送两个数据包，主机B的接收情况可能是：



产生粘包问题的原因有以下几个：

（1）应用层调用write方法，将应用层的缓冲区中的数据拷贝到套接字的发送缓冲区。而**发送缓冲区有一个SO\_SNDBUF的限制**，如果应用层的缓冲区数据大小大于套接字发送缓冲区的大小，则**数据需要进行多次的发送**。

（2）**TCP所传输的报文段有MSS的限制**，如果套接字缓冲区的大小大于MSS，也会导致消息的分割发送。

（3）**链路层最大发送单元MTU**，在IP层会进行数据的分片。

粘包问题的最本质原因在与接收对等方无法分辨消息与消息之间的边界在哪。我们通过使用某种方案给出边界，例如：

（1）发送**定长包**。如果每个消息的大小都是一样的，那么在接收对等方只要累计接收数据，直到数据等于一个定长的数值就将它作为一个消息。

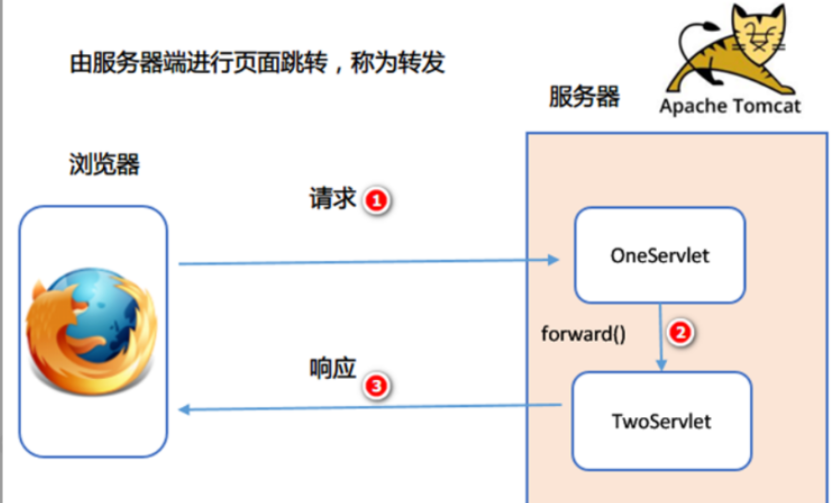
（2）**包尾加上\r\n标记**。FTP协议正是这么做的。但问题在于如果数据正文中也含有\r\n，则会误判为消息的边界。

（3）**包头加上包体长度**。包头是定长的4个字节，说明了包体的长度。接收对等方先接收包体长度，依据包体长度来接收包体。

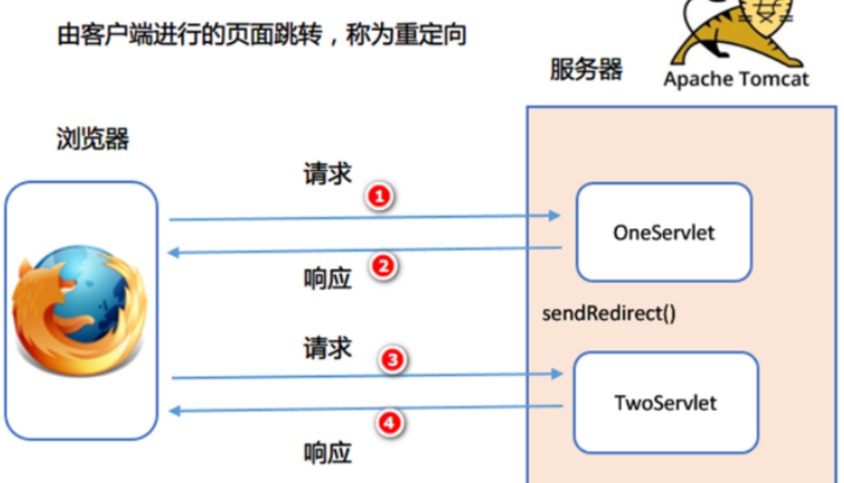
（4）使用更加复杂的应用层协议。

14. 转发和重定向的区别

（1）转发是由服务器端进行的跳转：request.getRequestDispatcher(url).forward(req, res)



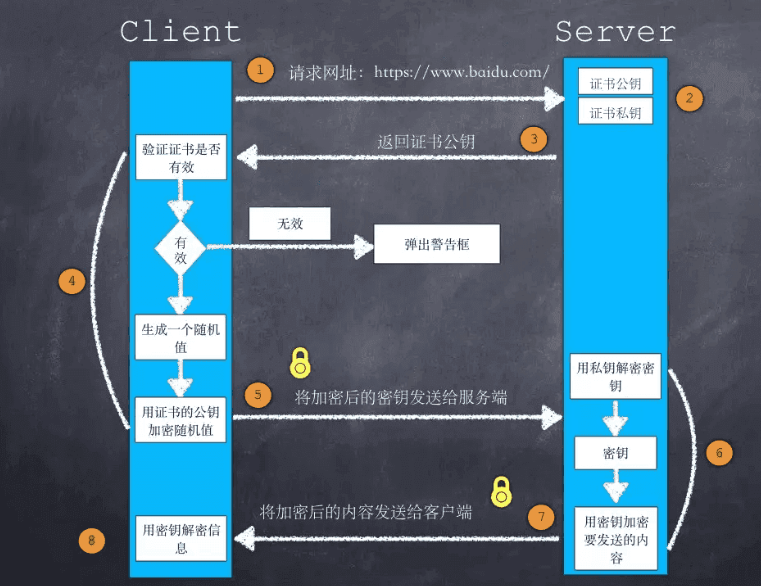
（2）重定向是由浏览器端进行的跳转：response.sendRedirect(url)





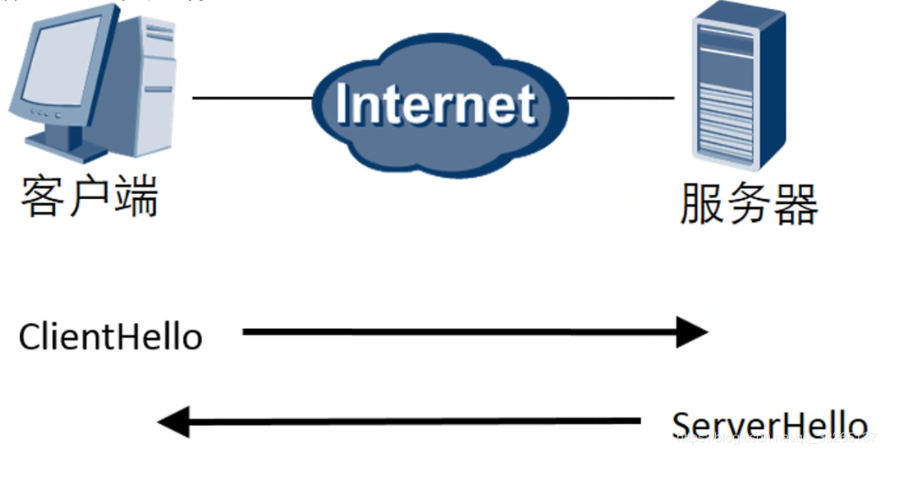
15. HTTPS的安全通信机制

* 服务器把自己的公开密钥登录至数字证书认证机构。
* 数字证书认证机构用自己的私有密钥向服务器的公开密码署数字签名并颁发公钥证书。
* 客户端拿到服务器的公钥证书后，使用数字签名认证机构的公开密钥，向数字证书认证机构验证公钥证书上的数字签名，以确认服务器的公开密钥的真实性。
* 使用服务器的公开密钥对报文加密后发送。
* 服务器用私有密钥对报文解密。

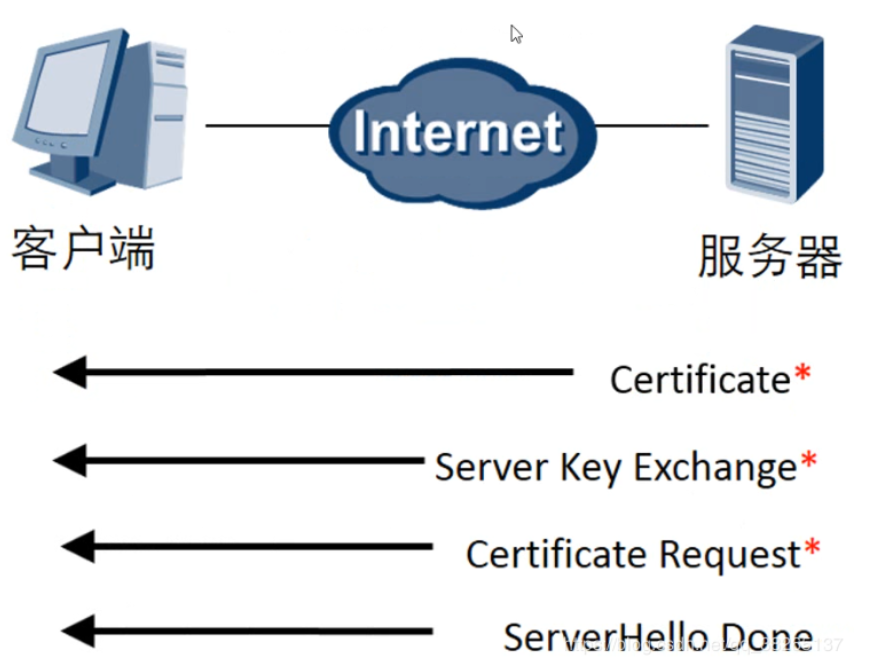


16. SSL协议

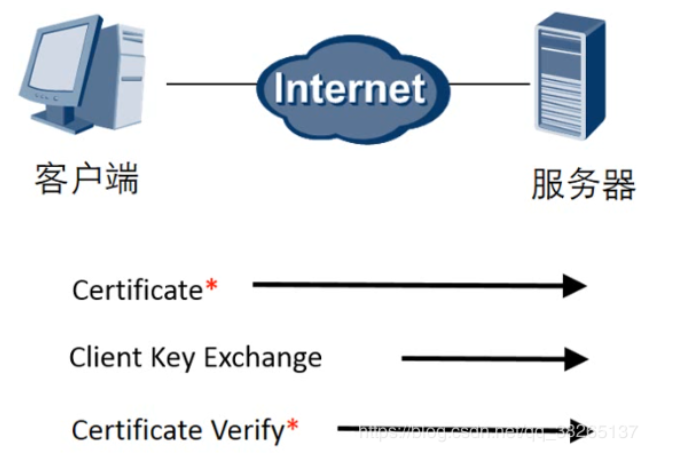
（1）客户端首先发送ClientHello消息到服务端，服务端收到ClientHello消息后，再发送ServerHello消息回应客户端。



（2）服务器向客户端发送消息：服务器将数字证书和到根CA整个链发给客户端，使客户端能用服务器证书中的服务器公钥认证服务器。



（3）客户端收到服务器发送的一系列消息并解析后，将本端相应的消息发送给服务器



（4）完成握手协议，建立SSL连接

