2020 面试题总结——网络篇

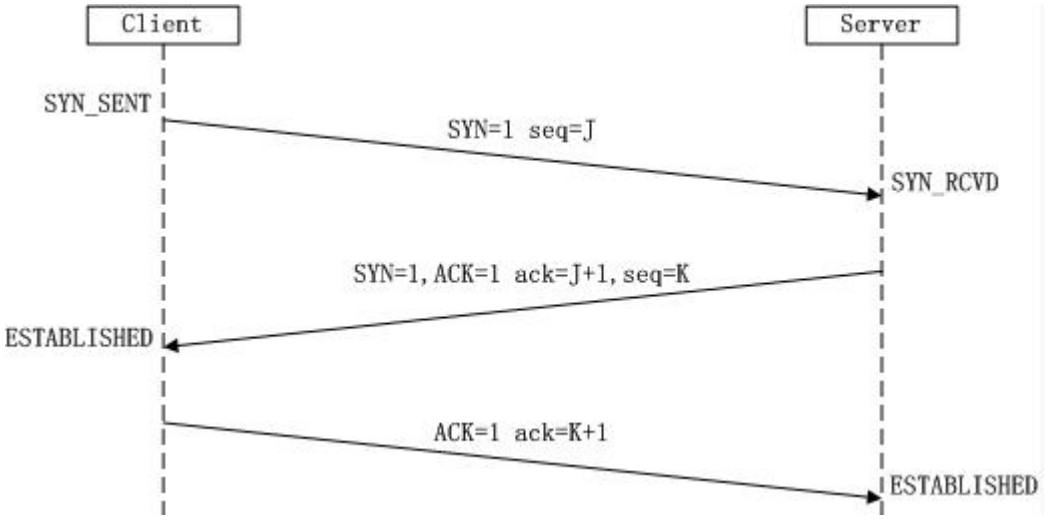
1、http1.0 和 http1.1 的区别。

主要是如下的 8 点：

- 可拓展性。
- 缓存。
- **带宽优化，带来了分块传输。**
- **长连接，HTTP1.1 支持长连接（默认开启 Connect: keep-alive）和请求的流水线处理，在一个 TCP 连接上可以传送多个 HTTP 请求和响应，减少了建立和关闭连接的消耗和延迟。**
- 消息传递。
- Host 头域。
- 错误提示。
- 内容协商。

2、TCP 三次握手和四次挥手的流程，为什么断开连接要四次，如果握手只有两次，会出现什么。

三次握手：



主要流程为：

- 第一次握手(SYN=1, seq=x)，发送完毕后，客户端进入 SYN_SEND 状态。
- 第二次握手(SYN=1, ACK=1, seq=y, ACKnum=x+1)， 发送完毕后，服务器端进入 SYN_RCVD 状态。

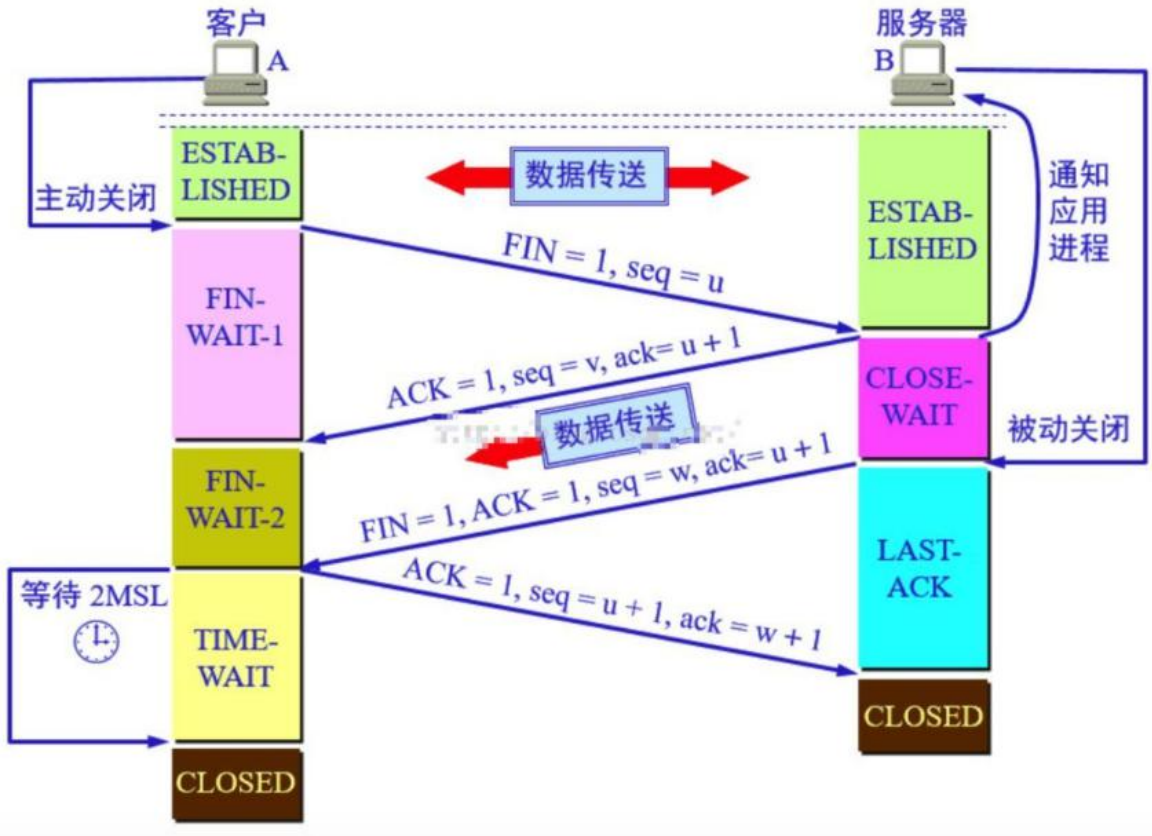
- 第三次握手(ACK=1，ACKnum=y+1)，发送完毕后，客户端进入 ESTABLISHED 状态，当服务器端接收到这个包时，也进入 ESTABLISHED 状态，TCP 握手，即可以开始数据传输。

为什么 TCP 连接需要三次握手，两次不可以么，为什么？

为了防止**已失效的连接请求**报文突然又传送到了服务端，因而产生错误。

- 客户端发出的连接请求报文并未丢失，而是在某个网络节点长时间滞留了，以致延误到链接释放以后的某个时间才到达 Server 。
- 若不采用"三次握手"，那么只要 Server 发出确认数据包，新的连接就建立了。由于 Client 此时并未发出建立连接的请求，所以其不会理睬 Server 的确认，也不与 Server 通信；而这时 Server 一直在等待 Client 的请求，这样 Server 就白白浪费了一定的资源。
- 若采用"三次握手"，在这种情况下，由于 Server 端没有收到来自客户端的确认，则就会知道 Client 并没有要求建立请求，就不会建立连接。

四次挥手：



主要流程为：

- 第一次挥手(FIN=1，seq=a)，发送完毕后，客户端进入 FIN_WAIT_1 状态。

- 第二次挥手(ACK=1，ACKnum=a+1)，发送完毕后，服务器端进入 CLOSE_WAIT 状态，客户端接收到这个确认包之后，进入 FIN_WAIT_2 状态。
- 第三次挥手(FIN=1，seq=b)，发送完毕后，服务器端进入 LAST_ACK 状态，等待来自客户端的最后一个 ACK。
- 第四次挥手(ACK=1，ACKnum=b+1)，客户端接收到来自服务器端的关闭请求，发送一个确认包，并进入 TIME_WAIT 状态，等待了某个固定时间（两个最大段生命周期，2MSL，2 Maximum Segment Lifetime）之后，没有收到服务器端的 ACK ，认为服务器端已经正常关闭连接，于是自己也关闭连接，进入 CLOSED 状态。服务器端接收到这个确认包之后，关闭连接，进入 CLOSED 状态。

为什么需要四次挥手：因为 TCP 连接是全双工的网络协议，允许同时通信的双方同时进行数据的收发，同样也允许收发两个方向的连接被独立关闭，以避免 client 数据发送完毕，向 server 发送 FIN 关闭连接，而 server 还有发送到 client 的数据没有发送完毕的情况。所以关闭 TCP 连接需要进行四次握手，每次关闭一个方向上的连接需要 FIN 和 ACK 两次握手。

握手过程如果只有两次，可能会出现已失效的连接请求报文突然又传送到了服务端，因而产生错误。

在三次握手过程中，为了保证服务端能收接受到客户端的信息并能做出正确的应答而进行前两次(第一次和第二次)握手，为了保证客户端能够接收到服务端的信息并能做出正确的应答而进行后两次(第二次和第三次)握手。

3、TIME_WAIT 和 CLOSE_WAIT 的区别。

TIME_WAIT 表示主动关闭，CLOSE_WAIT 表示被动关闭。

TCP 协议规定，对于已经建立的连接，网络双方要进行四次挥手才能断开连接，如果缺少了其中某个步骤，将会使连接处于假死状态，连接本身占用的资源不会被释放。网络服务器程序要同时管理大量连接，所以很有必要保证无用连接完全断开，否则大量僵死的连接会浪费许多服务器资源。在众多 TCP 状态中，最值得注意的状态有两个：CLOSE_WAIT 和 TIME_WAIT。

- TIME_WAIT 是主动关闭链接时形成的，等待 2MSL 时间，约 4 分钟。一方面是为了把原来的连接里面的重复数据包都已经在网络中消逝。避免老的连接的数据影响新建立的连接（新老连接的 IP 和端口号相同，新的被称为老的连接到化身）。 另一方面， 假如客户端回复的 ACK 丢失，服务端会重发 FIN，客户端此时还能接收到 FIN，还能再回复一个 ACK（此时 time_wait 会重新计时）（MSL 是指一个包的最大存活时间，一般是两分钟。）
- 另一种对于 TIME_WAIT 的解释：如果没有 TIME_WEIT 这个等待，释放的端口可能会重连刚断开的服务器端口，这样依然存活在网络里的老的

TCP 报文可能与新 TCP 连接报文冲突，造成数据冲突，为避免此种情况，需要耐心等待网络老的 TCP 连接的活跃报文全部死翘翘，2MSL 时间可以满足这个需求（尽管非常保守）！

- CLOSE_WAIT 是被动关闭连接是形成的。根据 TCP 状态机，服务器端收到客户端发送的 FIN，则按照 TCP 实现发送 ACK，因此进入 CLOSE_WAIT 状态。但如果服务器端不执行 close()，就不能由 CLOSE_WAIT 迁移到 LAST_ACK，则系统中会存在很多 CLOSE_WAIT 状态的连接。此时，可能是系统忙于处理读、写操作，而未将已收到 FIN 的连接，进行 close。此时，recv/read 已收到 FIN 的连接 socket，会返回 0。

参考链接：TIME_WAIT 和 CLOSE_WAIT 状态区别

4、说说你知道的几种 HTTP 响应码，比如 200，302 和 404。

HTTP 响应码主要分为五种：

- 1XX：请求处理中，请求已被接收，正在处理。
- 2XX：请求成功，请求被成功处理。比如 200，OK，表示客户端请求成功。
- 3XX：重定向，要完成请求必须进行进一步处理。比如 301，Moved Permanently，永久重定向，使用域名跳转；302，Found，临时重定向，未登录的用户访问用户中心重定向到登陆界面。
- 4XX：客户端错误，请求不合符。比如 400，Bad Request，客户端请求有语法错误，不能被服务器所理解；401，Unauthrized，请求未经授权，这个状态代码必须和 WWW-Authenticate 报头域一起使用；403，Forbidden，服务器收到请求，但是拒绝提供服务；404，Not Found，请求资源不存在，输入了错误的 URL。
- 5XX：服务器端错误，服务器不能处理合法请求。比如 500，Internal Servel Error，服务器发生不可预期的错误；503，Server Unavailable，服务器当前不能处理客户端的请求，一段时间后可能恢复正常。

5、当你用浏览器打开一个链接（如：http://www.baidu.com ）的时候，计算机做了哪些工作步骤。

计算机的工作主要是将域名解析成 ip 地址。

主机解析域名的顺序依次是：

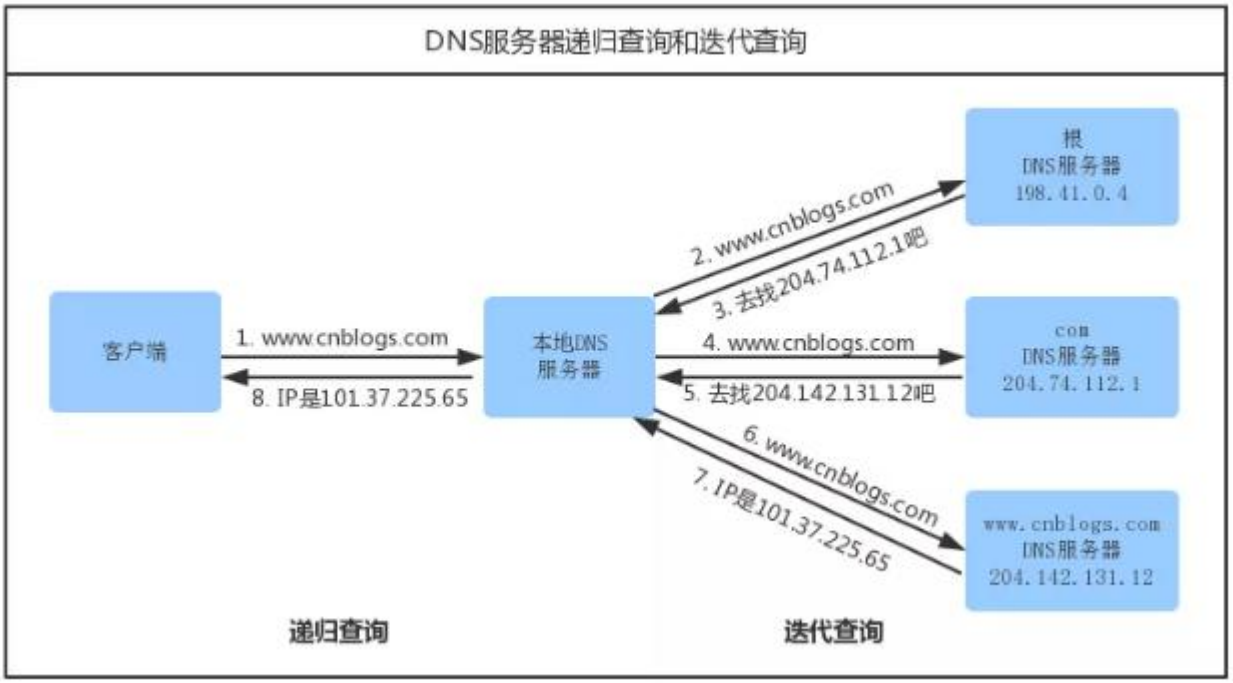
- 浏览器缓存。
- 找本机的 hosts 文件。
- 路由缓存。

- 找 DNS 服务器（本地域名、顶级域名、根域名），主要分为递归查询和迭代查询。

需要注意的是：

- 主机向本地域名服务器的查询一般都是采用递归查询。所谓递归查询就是：如果主机所询问的本地域名服务器不知道被查询的域名的 IP 地址，那么本地域名服务器就以 DNS 客户的身份，向其它根域名服务器继续发出查询请求报文(即替主机继续查询)，而不是让主机自己进行下一步查询。因此，递归查询返回的查询结果或者是所要查询的 IP 地址，或者是报错，表示无法查询到所需的 IP 地址。
- 本地域名服务器向根域名服务器的查询的迭代查询。迭代查询的特点：当根域名服务器收到本地域名服务器发出的迭代查询请求报文时，要么给出所要查询的 IP 地址，要么告诉本地服务器：“你下一步应当向哪一个域名服务器进行查询”。然后让本地服务器进行后续的查询。根域名服务器通常是把自己知道的顶级域名服务器的 IP 地址告诉本地域名服务器，让本地域名服务器再向顶级域名服务器查询。顶级域名服务器在收到本地域名服务器的查询请求后，要么给出所要查询的 IP 地址，要么告诉本地服务器下一步应当向哪一个权限域名服务器进行查询。

递归和迭代查询示意图如下：



参考文章：DNS 原理总结及其解析过程详解（递归查询+迭代查询）

6、TCP/IP 如何保证可靠性，说说 TCP 头的结构。

TCP 提供一种面向连接的、可靠的字节流服务。其中，面向连接意味着两个使用 TCP 的应用（通常是一个客户和一个服务器）在彼此交换数据之前必须先建立一个 TCP 连接。

对于可靠性，TCP 通过以下方式进行保证：

- **数据包校验：**目的是检验数据在传输过程中的变化，若检验包有错，则丢弃报文段并且不给出响应，这时 TCP 发送数据超时后会重发数据。
- **序号机制（序号、确认号）：**确保了数据是按序、完整到达。
- **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。TCP 传输时将每个字节的数据都进行了编号，这就是序列号。TCP 传输的过程中，每次接收方收到数据后，都会对传输方进行确认应答。也就是发送 ACK 报文。这个 ACK 报文当中带有对应的确认序列号，告诉发送方，接收到了哪些数据，下一次的数据从哪里发。
- **丢弃重复数据：**对于重复数据，能够丢弃重复数据。这是在超时重传情况下可能发生，判断依据就是序列号。
- **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，他启动一个定时器，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 根据接收端对数据的处理能力，决定发送端的发送速度，这个机制就是流量控制。在 TCP 协议的报头信息当中，有一个 16 位字段的窗口大小。在介绍这个窗口大小时我们知道，窗口大小的内容实际上是接收端接收数据缓冲区的剩余大小。这个数字越大，证明接收端接收缓冲区的剩余空间越大，网络的吞吐量越大。接收端会在确认应答发送 ACK 报文时，将自己的即时窗口大小填入，并跟随 ACK 报文一起发送过去。而发送方根据 ACK 报文里的窗口大小的值的改变进而改变自己的发送速度。如果接收到窗口大小的值为 0，那么发送方将停止发送数据。并定期的向接收端发送窗口探测数据段，让接收端把窗口大小告诉发送端。TCP 使用的流量控制协议是可变大小的滑动窗口协议。
- **拥塞控制：** TCP 传输过程中，发送端开始发送数据的时候，如果刚开始就发送大量的数据，那么就可能造成一些问题，网络可能在开始的时候就很拥堵，如果给网络再扔出大量数据，那么这个拥堵就会加剧。拥堵的加剧就会产生大量的丢包，以及大量的超时重传，严重影响传输。所以 TCP 引入了慢启动的机制，在刚开始发送数据时，先发送少量的数据探路，探清当前的网络状态如何，再决定多大的速度进行传输。这个时候就引入了一个叫做拥塞窗口的概念。拥塞窗口是发送端根据网络拥塞情况确定的窗口值。在刚刚开始发送报文的时候，先把拥塞窗口设置 1，每经过一个传输轮次（把拥塞窗口所允许发送的报文段都连续发送出去，并收到接受方的确认应答），拥塞窗口就加倍，这个增长速度是指数级

别的，为了控制拥塞窗口的增长，不能使拥塞窗口单纯的加倍，设置一个拥塞窗口的阈值，当拥塞窗口大小超过阈值时，不能再按照指数来增长，而是线性的增长。慢开始的“慢”并不是指拥塞窗口的增长速率慢，而是指在 TCP 开始发送报文段时先设置拥塞窗口=1，使得发送方在开始时只发送一个报文段（目的是试探一下网络的拥塞情况），然后再逐渐增大拥塞窗口。在发送数据之前，首先将拥塞窗口与接收端反馈的窗口大小比对，取最小的值作为实际发送的窗口。一旦造成网络拥塞，发生超时重传时，慢启动的阈值会为原来的一半（这里的原来指的是发生网络拥塞时拥塞窗口的大小），同时拥塞窗口重置为 1。

参考文章：网络基础：TCP 协议-如何保证传输可靠性

7、如何避免浏览器缓存。

主要有以下几种方法：

- Cache-Control/Pragma 这个 HTTP Head 字段用于指定所有缓存机制在整个请求/响应链中必须服从的指令，如果知道该页面是否为缓存，不仅可以控制浏览器，还可以控制和 HTTP 协议相关的缓存或代理服务器。
- Expires 通常的使用格式是 Expires:Sat,25Feb201212:22:17GMT，后面跟着一个日期和时间，超过这个时间值后，缓存的内容将失效，也就是浏览器在发出请求之前检查这个页面的这个字段，看该页面是否已经过期了，过期了就重新向服务器发起请求。
- Last-Modified/EtagLast-Modified 字段一般用于表示一个服务器上的资源的最后修改时间，资源可以是静态（静态内容自动加上 Last-Modified 字段）或者动态的内容（如 Servlet 提供了一个 getLastModified 方法用于检查某个动态内容是否已经更新），通过这个最后修改时间可以判断当前请求的资源是否是最新的。
- 在每次请求后面加上一个随机数，这样保证每次发送的都是不一样的请求。比如 url = "http://localhost:8080/test/login/index"?r=Math.random();

参考连接：面试官：你了解过浏览器缓存机制吗？

8、如何理解 HTTP 协议的无状态性。

无状态，是指协议对于事务处理没有记忆功能。HTTP 是一个无状态协议，这意味着每个请求都是独立的，Keep-Alive 没能改变这个结果。无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很快。

无状态，更容易做服务的扩容，支撑更大的访问量。

9、简述 Http 请求中 get 和 post 的区别及数据包格式。

GET：对服务器资源的简单请求，把参数包含在 URL 中。

POST：用于发送包含用户提交数据的请求，通过 request body 传递阐述。

另外，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）。

10、HTTP 中有哪些 method。

GET：对服务器资源的简单请求。

HEAD：类似于 GET，但服务器在响应中只返回首部，不返回实体的主体部分。

PUT：向指定资源位置上传其最新内容，是幂等的。

POST：用于发送包含用户提交数据的请求，是不幂等的，当我们多次发出同样的 POST 请求后，其结果是创建出了若干个资源。

TRACE：发送一个请求副本，以跟踪其处理进程。

OPTIONS：返回所有可用方法，检查服务器支持哪些方法。DELETE：请求服务器删除请求 URL 指定的资源。

另外，对幂等理解，幂等是数学的一个用于，对于单个输入或者无输入的运算方法，如果每次都是同样的结果，则称其是幂等的。方法 GET，HEAD，PUT，DELETE 都有这种性质。POST 方法不是幂等的。

11、简述 HTTP 请求的报文格式。

HTTP 的请求报文如图，由四部分构成：

- 请求行，用来说明请求类型、要访问的资源以及所使用的 HTTP 版本。例如 GET /books/java.html HTTP/1.1
- 请求头部，紧接着请求行（即第一行）之后的部分，用来说明服务器要使用的附加信息，第二行起为请求头部。
- 空行，请求头部后面的空行是必须的。
- 请求数据，也叫主体，可以添加任意的其他数据。

HTTP 的响应报文如图，也是由四部分构成：

- 状态行，由 HTTP 协议版本号、状态码、状态消息三部分构成。
- 消息报文，用来说明客户端要使用的一些附加消息。
- 空行，消息报文后面的空行是必须的。
- 响应报文，服务器返回给客户端的文本信息。

12、HTTP 的长连接是什么意思。

HTTP1.0 规定浏览器与服务器只保持短暂的连接，浏览器的每次请求都需要与服务器建立一个 TCP 连接，服务器完成请求处理后立即断开 TCP 连接，服务器不跟踪每个客户也不记录过去的请求，此外，由于大多数网页的流量都比较小，一次 TCP 连接很少能通过 slow-start 区，不利于提高带宽利用率。

HTTP 1.1 支持**长连接（PersistentConnection）和请求的流水线（Pipelining）**处理，在一个 TCP 连接上可以传送多个 HTTP 请求和响应，减少了建立和关闭连接的消耗和延迟。例如：一个包含有许多图像的网页文件的多个请求和应答可以在一个连接中传输，但每个单独的网页文件的请求和应答仍然需要使用各自的连接。

HTTP 1.1 还允许客户端不用等待上一次请求结果返回，就可以发出下一次请求，但服务器端必须按照接收到客户端请求的先后顺序依次回送响应结果，以保证客户端能够区分出每次请求的响应内容，这样也显著地减少了整个下载过程所需要的时间（请求的流水线）。

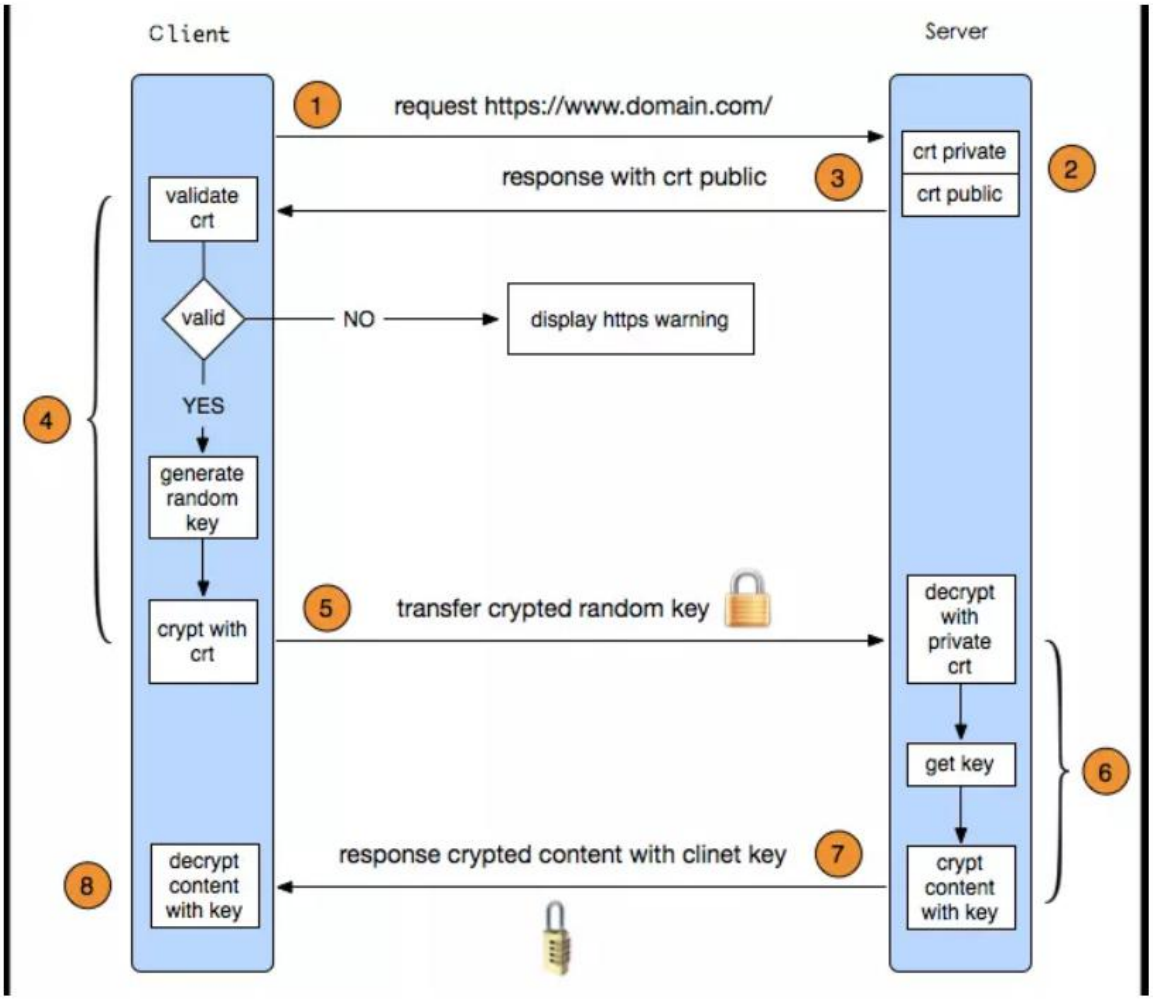
在 HTTP/1.0 中，要建立长连接，可以在请求消息中包含 Connection: Keep-Alive 头域，如果服务器愿意维持这条连接，在响应消息中也会包含一个 Connection: Keep-Alive 的头域。同时，可以加入一些指令描述该长连接的属性，如 max，timeout 等。

13、HTTPS 的加密方式是什么，讲讲整个加密解密流程。

HTTP 直接通过明文在浏览器和服务器之间传递消息，容易被监听抓取到通信内容。

HTTPS 采用对称加密和非对称加密结合的方式来进行通信，HTTPS 不是应用层的新协议，而是 HTTP 通信接口用 SSL/TLS 来加强加密和认证机制。

整个加密流程如下：



需要注意的是，第一次服务器向客户端传输证书的具体过程为：

- 把公钥以及服务器的个人信息通过 Hash 算法生成信息摘要；

- 为了防止信息摘要被人调换，服务器还会用 CA 提供的私钥对信息摘要进行加密来形成数字签名；
- 最后还会把原来没 Hash 算法之前的个人信息以及公钥 和 数字签名合并在一起，形成数字证书。

当客户端拿到这份数字证书之后，就会用 CA 提供的公钥来对数字证书里面的数字签名进行解密来得到信息摘要，然后对数字证书里服务器的公钥以及个人信息进行 Hash 得到另外一份信息摘要。最后把两份信息摘要进行对比，如果一样，则证明这个人是服务器，否则就不是。

整个的流程如下：

1. 客户端向服务器发起 HTTPS 请求，连接到服务器的 443 端口。
2. 服务器端有一个密钥对，即公钥和私钥，是用来进行非对称加密使用的，服务器端保存着私钥，不能将其泄露，公钥可以发送给任何人。
3. 服务器将自己的公钥发送给客户端。
4. 客户端收到服务器端的公钥之后，会对公钥进行检查，验证其合法性，如果发现发现公钥有问题，那么 HTTPS 传输就无法继续。严格的说，这里应该是验证服务器发送的数字证书的合法性，关于客户端如何验证数字证书的合法性，下文会进行说明。如果公钥合格，那么客户端会生成一个随机值，这个随机值就是用于进行对称加密的密钥，我们将该密钥称之为 client key，即客户端密钥，这样在概念上和服务器端的密钥容易进行区分。然后用服务器的公钥对客户端密钥进行非对称加密，这样客户端密钥就变成密文了，至此，HTTPS 中的第一次 HTTP 请求结束。
5. 客户端会发起 HTTPS 中的第二个 HTTP 请求，将加密之后的客户端密钥发送给服务器。
6. 服务器接收到客户端发来的密文之后，会用自己的私钥对其进行非对称解密，解密之后的明文就是客户端密钥，然后用客户端密钥对数据进行对称加密，这样数据就变成了密文。
7. 然后服务器将加密后的密文发送给客户端。
8. 客户端收到服务器发送来的密文，用客户端密钥对其进行对称解密，得到服务器发送的数据。这样 HTTPS 中的第二个 HTTP 请求结束，整个 HTTPS 传输完成。

一共是两次非对称加密+一次对称加密。

14、HTTPS 握手。

HTTPS 有四次握手，具体如上图所示。

15、什么是分块传输。

通常情况下，HTTP 的响应消息体 message body 是作为整包发送到客户端的，用头『Content-Length』 来表示消息体的长度， 这个长度对客户端非常重要，因为对于持久连接 TCP 并不会在请求完立马结束，而是可以发送多次请求/响应，客户端需要知道哪个位置才是响应消息的结束，以及后续响应的开始，因此 Content-Length 显得尤为重要，服务端必须精确地告诉客户端 message body 的长度是多少， 如果 Content-Length 比实际返回的长度短，那么就会造成内容截断，如果比实体内容长，客户端就一直处于 pendding 状态，直到所有的 message body 都返回了请求才结束。

分块传输编码：它把数据分解为一系列的数据块，并以多个块发送给客户端，服务器发送数据时不再需要预先告诉客户端发送内容的总大小，只需在响应头里面添加 Transfer-Encoding: chunked，以此来告诉浏览器我使用的是分块传输编码，这样就不需要 Content-Length 了。

分块编码传输优点：

- HTTP 分块传输编码允许服务器为动态生成的内容维持 HTTP 持久链接。通常，持久链接需要服务器在开始发送消息体前发送 Content-Length 消息头字段，但是对于动态生成的内容来说，在内容创建完之前是不可知的。
- 分块传输编码允许服务器在最后发送消息头字段。对于那些头字段值在内容被生成之前无法知道的情形非常重要，例如消息的内容要使用散列进行签名，散列的结果通过 HTTP 消息头字段进行传输。没有分块传输编码时，服务器必须缓冲内容直到完成后计算头字段的值并在发送内容前发送这些头字段的值。
- HTTP 服务器有时使用压缩（gzip）以缩短传输花费的时间。分块传输编码可以用来分隔压缩对象的多个部分。在这种情况下，块不是分别压缩的，而是整个负载进行压缩，压缩的输出使用本文描述的方案进行分块传输。在压缩的情形中，分块编码有利于一边进行压缩一边发送数据，而不是先完成压缩过程以得知压缩后数据的大小。

参考连接：HTTP 中的分块传输编码是怎么回事？

16、Session 和 cookie 的区别。

Session 和 cookie 都是实现对话管理的方案。主要区别在于：

- Session 在服务端，Cookie 存储在客户端。
- Session 的运行依赖 Session ID，而 Session ID 是存在 Cookie 中的，也就是说，如果浏览器禁用了 Cookie，同时 Session 也会失效，但是，可以通过其他方式实现，比如在 url 参数中传递 Session ID。
- Tomcat 中的 Session 是存在服务器内存中，不过也可以通过特殊的方式做持久化处理（memcache，redis），方便 Session 共享。另外，PHP 中的 Session 是存在文件中的。

- cookie 不是很安全，别人可以分析存放在本地的 cookie 并进行 cookie 欺骗，考虑到安全应当使用 session。

参考连接：Session 是怎么实现的？存储在哪里？

17、用户在浏览器输入一个 URL 并回车，这个过程涉及到哪些网络协议，请具体描述。

浏览器输入一个 URL 并回车：

1. 首先进行域名解析，浏览器搜索自己的 DNS 缓存，缓存中维护一张域名与 IP 地址的对应表。若没有，则搜索操作系统的 DNS 缓存；若没有，则将域名发送至本地域名服务器（递归查询方式），本地域名服务器查询自己的 DNS 缓存，查找成功则返回结果，否则，本地的 DNS 服务器向根域名服务器发出查询请求，根域名服务器告知该域名的一级域名服务器，然后本地服务器给该一级域名服务器发送查询请求，然后依次类推直到查询到该域名的 IP 地址。DNS 服务器是基于 UDP 的，因此会用到 UDP 协议。
2. 得到 IP 地址以后，浏览器就要与服务器建立一个 HTTP 连接，因此要用到 HTTP 协议。HTTP 生成一个 GET 请求报文。
3. 接下来到了传输层，选择传输协议，TCP 或者 UDP，TCP 是可靠的传输控制协议，对 HTTP 请求进行封装，加入了端口号等信息。
4. 然后到了网络层，通过 IP 协议将 IP 地址封装为 IP 数据报；然后此时会用到 ARP 协议，主机发送信息时将包含目标 IP 地址的 ARP 请求广播到网络上的所有主机，并接收返回消息，以此确定目标的物理地址，找到目的 MAC 地址。
5. 接下来到了数据链路层，把网络层交下来的 IP 数据报添加首部和尾部，封装为 MAC 帧，现在根据目的 mac 开始建立 TCP 连接，三次握手，接收端在收到物理层上交的比特流后，根据首尾的标记，识别帧的开始和结束，将中间的数据部分上交给网络层，然后层层向上传递到应用层。
6. 服务器响应请求并请求客户端要的资源，传回给客户端。
7. 断开 TCP 连接，浏览器对页面进行渲染呈现给客户端。

18、一致性哈希问题。

在解决分布式系统中负载均衡的问题时候可以使用 Hash 算法让固定的一部分请求落到同一台服务器上，这样每台服务器固定处理一部分请求（并维护这些请求的信息），起到负载均衡的作用。但是普通的余数 hash（hash(比如用户 id)%服务器机器数）算法伸缩性很差，当新增或者下线服务器机器时候，用户 id 与服务器的映射关系会大量失效。一致性 hash 则利用 hash 环对其进行了改进。

一致性 hash 主要是建立起一个在[0,2^32-1]分布的哈希环。根据资源的 Key 的 Hash 值（也是分布为[0,,2^32-1]）H1，在环上顺时针的找到离 H1 最近（第一个大于或等于 H1）的一个节点，就建立了资源和节点的映射关系。

另外，当一致性哈希算法在服务节点太少时，容易因为节点分布不均匀而造成数据倾斜问题。为了解决这个问题，一致性哈希算法引入了虚拟节点机制，即对每一个服务节点计算多个哈希，每个计算结果位置都放置一个此服务节点，称为虚拟节点。具体做法可以在服务器 ip 或主机名的后面增加编号来实现。

具体信息建议直接去看文章：深入浅出一致性 Hash 原理和一致性哈希(hash)算法

19、ETag 的含义和作用。

ETag 是 Entity Tag 的缩写，中文译过来就是实体标签的意思。在 HTTP1.1 协议中的其实就是请求 Head 中的一个属性。比如

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2019 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close
```

ETag 是 HTTP1.1 中才加入的一个属性，类似于资源指纹，用来帮助服务器控制 Web 端的缓存验证。它的原理是这样的，当浏览器请求服务器的某项资源（A）时，服务器根据 A 算出一个哈希值（3f80f-1b6-3e1cb03b）并通过 ETag 返回给浏览器，浏览器把"3f80f-1b6-3e1cb03b"和 A 同时缓存在本地，当下次再次向服务器请求 A 时，会通过类似 If-None-Match: "3f80f-1b6-3e1cb03b"的请求头把 Etag 信息发送给服务器，服务器再次计算 A 的哈希值并和浏览器请求的值作比较，如果发现 A 发生了变化就把 A 返回给浏览器（200），如果发现 A 没有变化，就给浏览器返回一个 304 未修改。这样通过控制浏览器端的缓存，可以节省服务器的带宽，因为服务器不需要每次都把全量数据返回给客户端。

注意的是，HTTP 并没有指定如何生成 ETag，哈希是比较理想的选择。但在一些服务器情况下，并不会用哈希来计算 ETag，因为这会严重浪费服务器资源，所以很多时候通过资源的版本或者修改时间来生成 ETag。如果通过资源修改时间来生成 ETag，那么效果和 HTTP 协议里面的另外一个控制属性(Last-Modified)就雷同了，使用 Last-Modified 的问题在于它的精度在秒(s)的级别，比较适合不太敏感的静态资源。

20、转发和重定向的区别。

转发是指 RequestDispatcher.forward，而重定向是指 HttpServletResponse.sendRedirect。主要区别如下：

- forword 方法只能将请求转发给同一个 WEB 应用中的组件，而 sendRedirect 方法不仅可以重定向到当前应用程序的其他资源，还可以重定向到同一个站点上的其他应用程序中的资源。
- forword 方法的请求转发过程结束后，浏览器地址栏保持初始的 URL 地址不变；而 sendRedirect 方法重定向的访问过程结束后，浏览器地址栏中显示的 URL 会发现改变，由初始的 URL 地址变成重定向的目标 URL。
- forword 方法服务器端内部将请求转发给另外一个资源，浏览器只知道发出了请求并得到了响应结果，并不知道在服务器程序内部发生了转发行为；而 sendRedirect 方法对浏览器的请求直接作出响应，响应的结果就是告诉浏览器去重新发出对另外一个 URL 的访问请求。
- forword 方法的调用者与被调用者之间共享相同的 request 对象和 response 对象，它们属于同一个访问请求和响应过程；而 sendRedirect 方法调用者与被调用者使用各自的 request 对象和 response 对象，它们属于两个独立的访问请求和响应过程。

注意：

- 转发比重定向快，因为重定向需要经过客户端，而转发没有。
- 使用重定向不太方便的地方是，使用它无法将值轻松地传递给目标页面。而采用转发，则可以简单地将属性添加到 Model,使得目标视图可以轻松访问。

重定向的使用场景：当提交产品表单的时候，执行保存的方法将会被调用，并执行相应的动作；这在一个真实的应用程序中，很有可能将表单中的所有产品信息加入到数据库中。但是如果在提交表单后，重新加载页面，执行保存的方法就很有可能再次被调用。同样的产品信息就将可能再次被添加，为了避免这种情况，提交表单后，你可以将用户重定向到一个不同的页面，这样的话，这个网页任意重新加载都没有副作用。

21、HTTP2.0 带来的变化。

HTTP2.0 和 HTTP1.x 相比的新特性为：

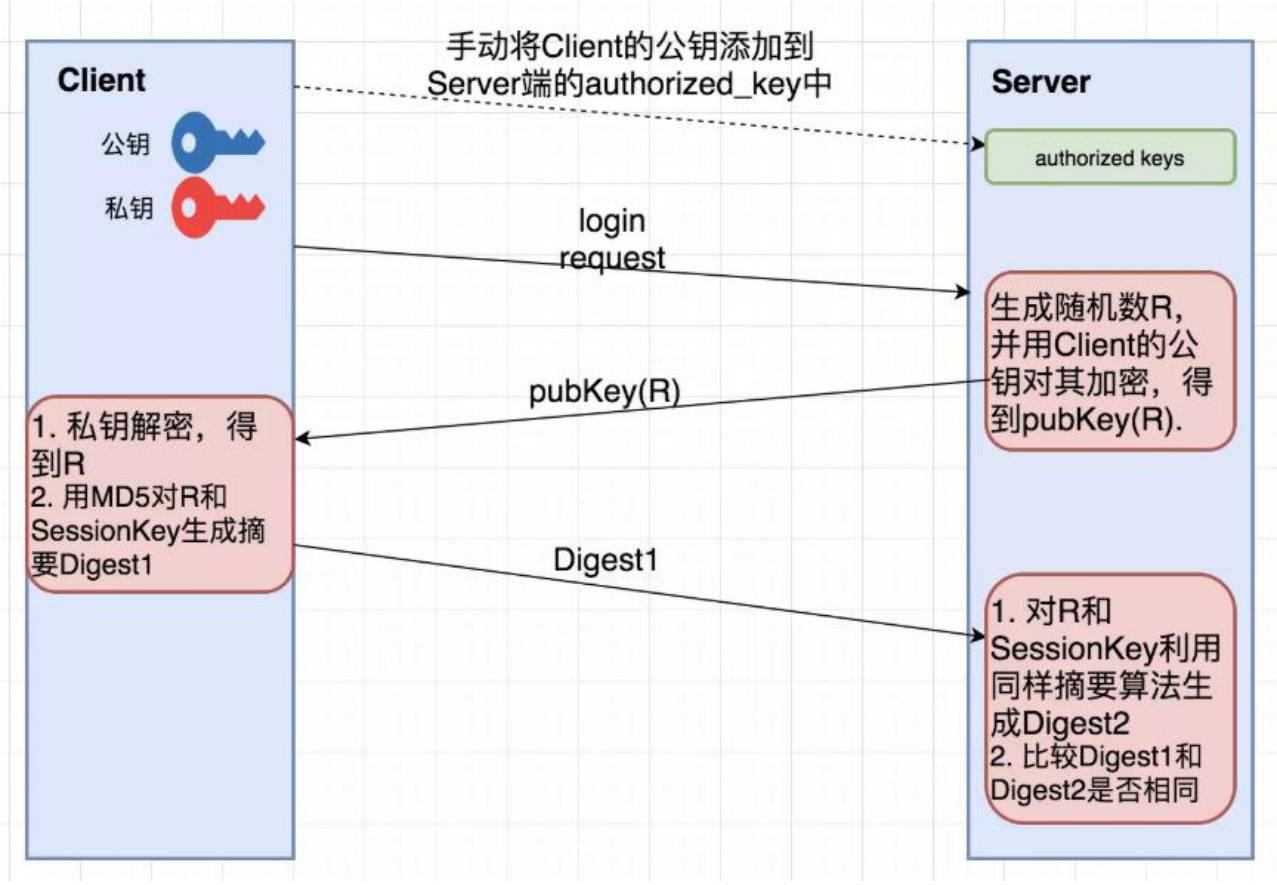
- **新的二进制格式**，HTTP1.x 的解析是基于文本。基于文本协议的格式解析存在天然缺陷，文本的表现形式有多样性，要做到健壮性考虑的场景必然很多，二进制则不同，只认 0 和 1 的组合。基于这种考虑 HTTP2.0 的协议解析决定采用二进制格式，实现方便且健壮。
- **多路复用**，即连接共享，做到同一个连接并发处理多个请求，而且并发请求的数量比 HTTP1.1 大了好几个数量级。

- **多个请求可以同时在一个连接上并行执行**，某个请求任务耗时严重，不会影响到其他连接的正常执行。这块和 HTTP1.1 的长连接有区别，长连接是串行化执行多个请求。
- **首部压缩**，HTTP1.1 不支持 header 数据的压缩，HTTP2.0 使用 HPACK 算法对 header 的数据进行压缩，这样数据体积小了，在网络上传输就会更快。
- **服务器推送**，当我们对支持 HTTP2.0 的 web server 请求数据的时候，服务器会顺便把一些客户端需要的资源（比如 css、js 文件）一起推送到客户端，免得客户端再次创建连接发送请求到服务器端获取。**这种方式非常合适加载静态资源**。还没有收到浏览器的请求，服务器就把各种资源推送给浏览器了。

22、SSH 相关。

SSH 是一种协议标准，其目的是实现远程登录以及其他安全网络服务。SSH 和 telnet、ftp 等协议主要的区别在于安全性。

SSH 中用了非对称加密方式。但与 HTTPS 可以通过 CA 认证不同，SSH 的公钥和密钥都是自己生成的，没法公证，只能通过 client 端自己对公钥进行确认。一般是在第一次登陆的过程时实现。具体过程如下。



23、网络模型

OSI七层网络模型	TCP/IP四层概念模型	对应网络协议
应用层（Application）	应用层	HTTP、TFTP, FTP, NFS, WAIS、SMTP
表示层（Presentation）		Telnet, Rlogin, SNMP, Gopher
会话层（Session）		SMTP, DNS
传输层（Transport）	传输层	TCP, UDP
网络层（Network）	网络层	IP, ICMP, ARP, RARP, AKP, UUCP
数据链路层（Data Link）	数据链路层	FDDI, Ethernet, Arpanet, PDN, SLIP, PPP
物理层（Physical）		IEEE 802.1A, IEEE 802.2到IEEE 802.11

自上而下分别是：

- 应用层（数据）：确定进程之间通信的性质以满足用户需要以及提供网络与用户应用。
- 表示层（数据）：主要解决用户信息的语法表示问题，如加密解密。在表示层进行代码/编码转换。
- 会话层（数据）：提供包括访问验证和会话管理在内的建立和维护应用之间通信的机制，如服务器验证用户登录便是由会话层完成的。在会话层封装会话控制参数。
- 传输层（段）：实现网络不同主机上用户进程之间的数据通信，可靠与不可靠的传输，传输层的错误检测，流量控制等。在传输层封装传输控制。
- 网络层（包）：提供逻辑地址（IP）、选路，数据从源端到目的端的传输。在网络层加上逻辑寻址地址。
- 数据链路层（帧）：将上层数据封装成帧，用 MAC 地址访问媒介，错误检测与修正。在数据链路层封装基于 MAC 的信息。
- 物理层（比特流）：设备之间比特流的传输，物理接口，电气特性等。在物理层连接到线缆系统进行实际传递。

24、网络层的 ARP 协议工作原理。

网络层的 ARP 协议完成了 IP 地址与物理地址的映射。

首先，每台主机都会在自己的 ARP 缓冲区中建立一个 ARP 列表，以表示 IP 地址和 MAC 地址的对应关系。

当源主机需要将一个数据包要发送到目的主机时，会首先检查自己 ARP 列表中是否存在该 IP 地址对应的 MAC 地址：

- 如果有，就直接将数据包发送到这个 MAC 地址。
- 如果没有，就向本地网段发起一个 ARP 请求的广播包，查询此目的主机对应的 MAC 地址。

25、对于 TCP 连接，客户端不断进行请求链接会怎么样？

服务器端准备为每个请求创建一个链接，并向其发送确认报文，然后等待客户端进行确认后创建。如果此时客户端一直不确认，会造成 SYN 攻击，即：SYN 攻击，英文为 SYN Flood ，是一种典型的 DoS/DDoS 攻击。

- 客户端向服务端发送请求连接数据包。
- 服务端向客户端发送确认数据包。
- 客户端不向服务端发送确认数据包，服务器一直等待来自客户端的确认。

这时候服务器上有大量的半连接状态，特别是源 IP 地址是随机的，基本可以断定是一次 SYN 攻击。

对于 SYN 攻击，只能预防，没有彻底根治的办法，除非不使用 TCP 。方法主要如下：

- 限制同时打开 SYN 半链接的数目。
- 缩短 SYN 半链接的超时时间。
- 关闭不必要的服务。
- 增加半链接数据。
- 过滤网关防护。

26、TCP 和 UDP 的区别。

TCP 和 UDP 都属于传输层协议，它们之间的区别在于：

- TCP 是面向连接的；UDP 是无连接的。
- TCP 是可靠的；UDP 是不可靠的。
- TCP 只支持点对点通信；UDP 支持一对一、一对多、多对一、多对多的通信模式。
- TCP 是面向字节流的；UDP 是面向报文的。
- TCP 有拥塞控制机制；UDP 没有拥塞控制，适合媒体通信。
- TCP 首部开销(20 个字节)，比 UDP 的首部开销(8 个字节)要大。

27、TCP 中 RST 标志位的作用。

TCP 报文中一共有 6 个标志位，分别为：URG/ACK/PSH/RST/SYN/FIN。

- SYN：TCP 三次握手时，如果 A 是发起端，则 A 就对服务器发送一个 SYN 报文，表示想要建立连接。
- ACK：收到数据或者请求后发送响应时发送 ACK 报文。
- RST：关闭异常连接。
- FIN：TCP 四次挥手时，表示关闭连接。
- PSH：发送端需要发送一段数据，这个数据需要接收端一收到就进行向上交付。而接收端在收到 PSH 标志位有效的数据时，迅速将数据交付给应用层，所以 PSH 又叫做急迫比特。
- URG：紧急指针，意思为 URG 位有效的数据包，是一个紧急需要处理的数据包，需要接收端在接收到之后迅速处理。

RST 标志位应用场景如下。

TCP 正常关闭连接的时候使用 FIN，但是如果是关闭异常连接，则使用 RST，发送 RST 包。与 FIN 包存在两点不同：

- RST 不必等缓冲区的包都发出去，直接就丢弃缓冲区的包去发送 RST 包，而 FIN 需要先处理完缓冲区的包才行。
- 接收端收到 RST 包之后，不需要发送 ACK 包进行确认，而接收端接收到 FIN 包的时候需要 ACK 包应答。

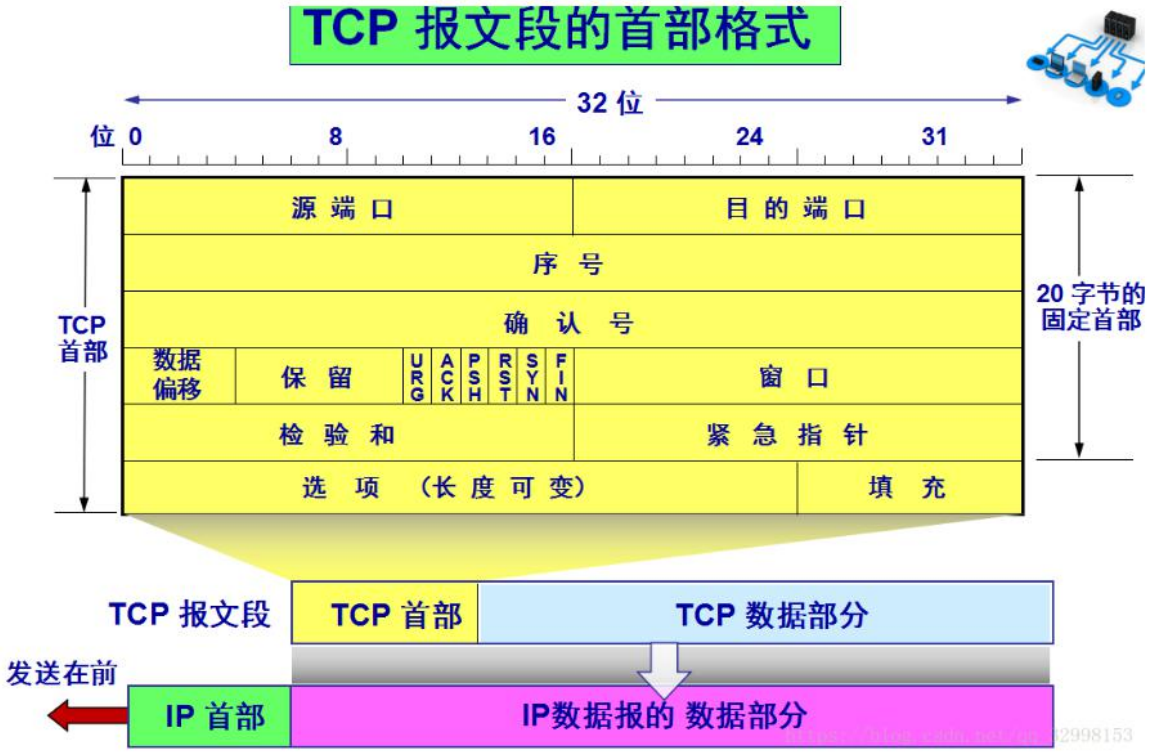
28、TCP 和 UDP 报文段的首部格式。

TCP 报文分为首部和数据两部分，TCP 的全部功能体现在首部的各字段作用上。

首部固定部分各字段的意义。

- **源端口和目的端口：**各占两个字节。
- **序号：**占四个字节。
- **确认号：**占四个字节。
- **数据偏移：**占四个字节，指出 TCP 报文段的数据起始处距离 TCP 报文段的起始处有多远。
- **保留：**占六个字节，保留为今后使用，但目前应置为 0。

还有六个控制位，就在 27 题里讲述了。TCP 示意图如下所示。



UDP 报文首部相对简单。主要分为四部分：

- 源端口和目的端口。
- 数据包长度。
- 校验值。