## 计算机网络分层体系结构（五层举例）

**应用层**：为**应用程序提供交互服务**。在互联网中的应用层协议很多，如域名系统DNS、HTTP协议、SMTP协议等。

解决通过应用进程的交互来实现特定网络应用的问题

**传输层**：负责向两台主机进程之间的通信**提供数据传输服务**。传输层的协议主要有传输控制协议TCP和用户数据协议UDP。

解决进程之间基于网络的通信问题

**网络层**：选择合适的**路由和交换结点**，确保**数据及时传送**。主要包括IP协议。

解决分组在多个网络上传输(路由)的问题

**数据链路层**：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。

解决分组在一个网络(或一段链路)上传输的问题

**物理层**：实现相邻节点间**比特流的透明传输**，尽可能屏蔽传输介质和物理设备的差异。

解决使用何种信号来传输比特的问题

## ISO 7层交换（参考）模型

### 物链网传会表应

1. 物理层 通信工程

2. 链路层 内网寻址 ARP ICMP

3. 网络层 外网寻址 IP

4. 传输层 通信稳定性 TCP

5. 会话层 x记录状态

6. 表现层 x统一各个网络结构

7. 应用层 应用细节 HTTP FTP SMTP POP3

### 实际为5层

1. 物理层

2. 链路层

3. 网络层

4. 传输层

5. 应用层

## 域名系统DNS

域名系统 DNS (Domain Name System)是互联网使用的命名系统，用来把便于人们使用的机器名字转换为 IP 地址。域名系统其实就是名字系统。

为什么不叫“名字”而叫“域名”呢？

这是因为在这种互联网的命名系统中使用了许多的“域” (domain)，因此就出现了“域名”这个名词。 “域名系统”很明确地指明这种系统是用在互联网中的。

## DNS解析过程

浏览器搜索自己浏览器的DNS缓存，若没有，则搜索本地操作系统中的DNS缓存和hosts文件。若没有，则操作系统将域名发送至本地域名服务器，本地域名服务器查询自己的DNS缓存，查找成功则返回结果，否则依次向根域名服务器、顶级域名服务器、权限域名服务器发起查询请求，最终返回IP地址给本地域名服务器。本地域名服务器将得到的IP地址返回给操作系统，同时自己也将IP地址缓存起来，操作系统将IP地址返回给浏览器，同时自己也将IP地址缓存起来，浏览器得到域名对应的IP地址。

## HTTP和HTTPS区别

HTTP协议是免费使用的，而 HTTPS 协议需要到CA机构申请证书，需要缴纳费用，HTTP是超文本传输协议，信息是明文传输，HTTPS则是具有安全性的SSL/TLS加密传输协议，信息是密文。

HTTP的连接很简单，是无状态的；HTTPS协议是由SSL/TLS+HTTP协议构建的可进行加密传输、身份认证的网络协议，比HTTP协议安全，和HTTP通信相比HTTPS 通信会由于加减密处理消耗更多的CPU和内存资源。

HTTP和HTTPS使用的端口也不一样，前者是80，后者是443。

## SMTP(简单邮件传输协议(SimpleMailTransferProtocol))

协议定义了在两个或多个通信实体之间交换的报文的格式和顺序，以及报文发送或报文接收或其他事件采取的动作。SMTP是将邮件从源地址向目的地址发送方式的规则，并控制端系统发送的方式。

## TCP和UDP区别和应用

TCP是可靠传输，UDP是不可靠传输。

TCP面向连接，UDP无连接，即发送数据之前不需要建立连接。

TCP传输数据有序，UDP不保证数据的有序性。

TCP面向字节流，把数据看成一连串字节流，UDP是面向报文的。

TCP传输速度相对UDP较慢。

TCP有流量控制和拥塞控制，UDP没有。

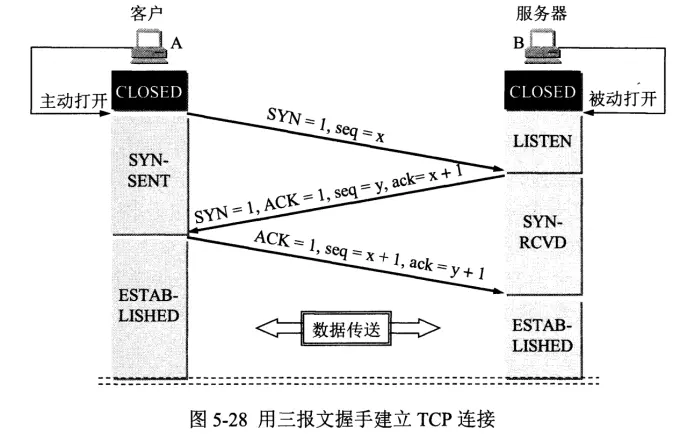
TCP是重量级协议，UDP是轻量级协议。

TCP首部20字节，UDP首部8字节。

TCP连接只能是一对一的（端到端）；UDP支持一对一、一对多、多对一和多对多的通信方式。

## TCP的三次握手和四次挥手

### 三次握手

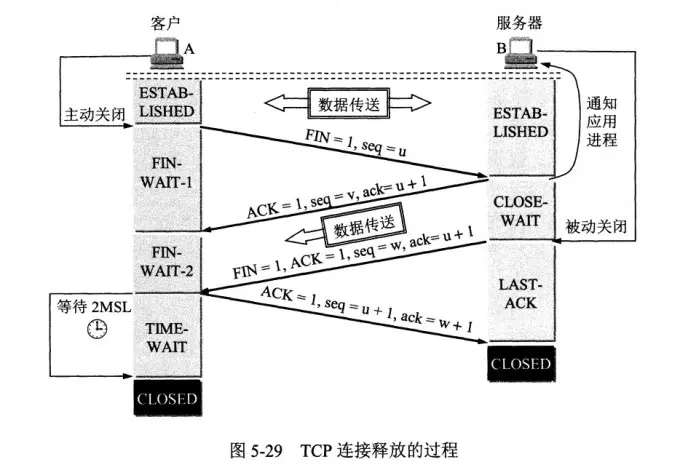


第一次握手：客户端给服务端发一个 SYN 报文，并指明客户端的初始化序列号 ISN，此时客户端处于 SYN\_SEND 状态。

第二次握手：服务器收到客户端的 SYN 报文之后，会以自己的 SYN 报文作为应答，并且也是指定了自己的初始化序列号 ISN。同时会把客户端的 ISN + 1 作为ACK 的值，表示自己已经收到了客户端的 SYN，此时服务器处于 SYN\_REVD 的状态。

第三次握手：客户端收到 SYN 报文之后，会发送一个 ACK 报文，当然，也是一样把服务器的 ISN + 1 作为 ACK 的值，表示已经收到了服务端的 SYN 报文，此时客户端处于 ESTABLISHED 状态。服务器收到 ACK 报文之后，也处于 ESTABLISHED 状态，此时，双方已建立起了连接。

### 四次挥手



第一次挥手： 客户端会发送一个 FIN 报文，报文中会指定一个序列号。此时客户端处于 FIN\_WAIT1 状态。

第二次挥手：服务端收到 FIN 之后，会发送 ACK 报文，且把客户端的序列号值 +1 作为 ACK 报文的序列号值，表明已经收到客户端的报文了，此时服务端处于 CLOSE\_WAIT 状态。

第三次挥手：如果服务端也想断开连接了，和客户端的第一次挥手一样，发给 FIN 报文，且指定一个序列号。此时服务端处于 LAST\_ACK 的状态。

第四次挥手：客户端收到 FIN 之后，一样发送一个 ACK 报文作为应答，且把服务端的序列号值 +1 作为自己 ACK 报文的序列号值，此时客户端处于 TIME\_WAIT 状态。需要过一阵子以确保服务端收到自己的 ACK 报文之后才会进入 CLOSED 状态，服务端收到 ACK 报文之后，就处于关闭连接了，处于 CLOSED 状态。

## 为什么要三次握手?

谢希仁著《计算机网络》中讲“三次握手”的目的是“**为了防止已失效的连接请求报文段 突然又传送到了服务端，因而产生错误**”。

## IP协议

IP协议全称为“网际互连协议（Internet Protocol）”，IP协议是TCP/IP体系中的网络层协议。

1、网络层解决的问题是将数据从一台主机送到另一台主机，因此网络层解决的是主机到主机的问题。

2、一方传输层从上方进程拿到数据后，该数据贯穿网络协议栈进行封装和解包，最终到达对方传输层，此时对方传输层也会将数据向上交给对应的进程，因此传输层解决的是进程到进程的问题。

## 对称加密与非对称加密

### 对称加密：

对称加密是指加密和解密使用同一个密钥的方式，一方通过密钥将信息加密后，把密文传给另一方，另一方通过这个相同的密钥将密文解密，转换成可以理解的明文。

### 非对称加密：

使用一对非对称密钥加密，即公钥和私钥，公钥可以随意发布，任何人都能获得，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。公开密钥与私有密钥是一对，如果用公开密钥对数据进行加密，只有用对应的私有密钥才能解密；如果用私有密钥对数据进行加密，那么只有用对应的公开密钥才能解密。因为加密和解密使用的是两个不同的密钥，所以叫作非对称加密。

## 什么是数字证书/数字签名

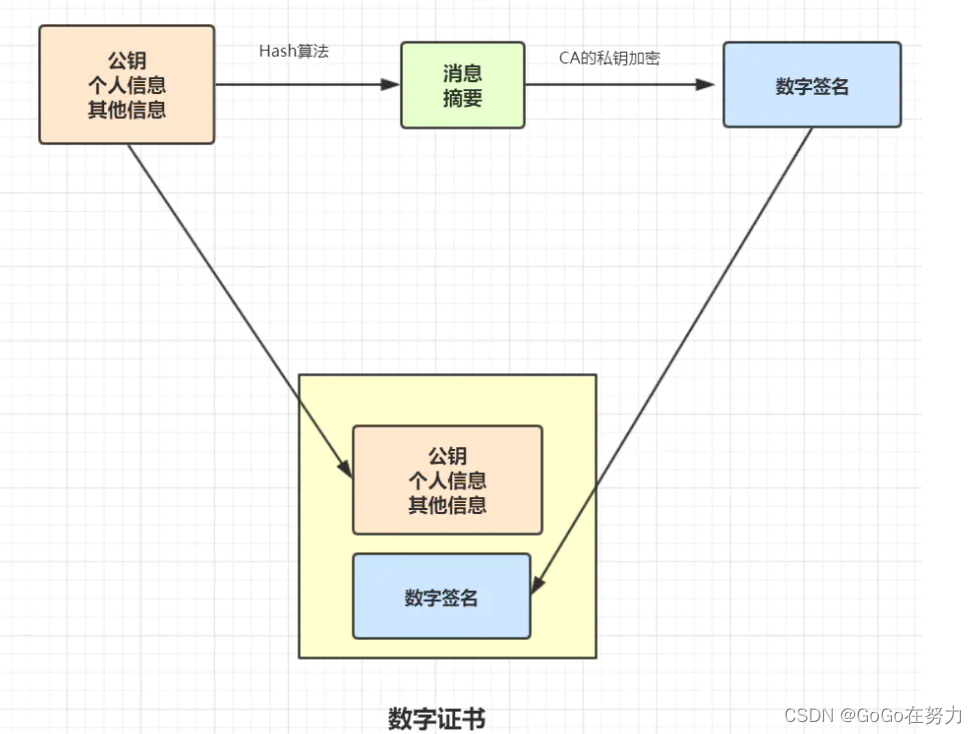
公钥和个人等信息，经过 Hash 摘要算法加密，形成消息摘要；

将消息摘要拿到拥有公信力的认证中心（CA），用它的私钥对消息摘要加密，形成数字签名

公钥和个人信息、数字签名共同构成数字证书

数字证书是指在互联网通讯中标志通讯各方身份信息的一个数字认证，人们可以在网上用它来识别对方的身份。

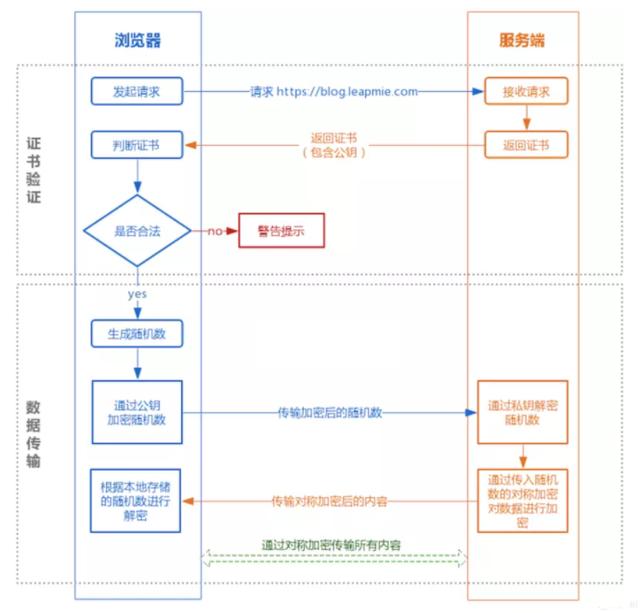
它的出现，是为了避免身份被篡改冒充的。比如 Https 的数字证书，就是为了避免公钥被中间人冒充篡改：



**数字证书**

## HTTPS工作流程

HTTPS协议是安全的，因为HTTPS协议会对传输的数据进行加密，而加密过程是使用了非对称加密实现。但是HTTPS在内容传输使用的是对称加密，在证书验证阶段使用非对称加密。所以HTTPS加密方式是：对称加密 + 非对称加密混合。



① 证书验证阶段：

1）浏览器发起 HTTPS 请求；

2）服务端返回 HTTPS 证书；

3）客户端验证证书是否合法，如果不合法则提示告警。

② 数据传输阶段：

1）当证书验证合法后，在本地生成随机数；

2）通过公钥加密随机数，并把加密后的随机数传输到服务端；

3）服务端通过私钥对随机数进行解密；

1. 服务端通过客户端传入的随机数构造对称加密算法，对返回结果内容进行加密后传输。

## HTTP常见状态码

200 OK：表示从客户端发送给服务器的请求被正常处理并返回成功。

301 Moved Permanently：永久性重定向，表示请求的资源被分配了新的URL，之后应使用更改的URL。

302 Found：临时性重定向，表示请求的资源被分配了新的URL，希望本次访问使用新的URL。

400 Bad Request：表示请求报文存在语法错误。

403 Forbidden：服务器拒绝该次访问（访问权限问题）。

404 Not Found：表示服务器上无法找到请求的资源（也可以在服务器拒绝请求但不想给拒绝原因时使用）。

500 Inter Server Error：表示服务器在执行请求时发生了错误，也有可能是web应用存在的bug或某些临时的错误时。

503 Server Unavailable：表示服务器暂时处于超负载或正在进行停机维护，无法处理请求。

## HTTP长连接和短连接，流水线是什么

长连接和短连接：

HTTP1.0默认使用的是短连接。浏览器和服务器每进行一次HTTP操作，就建立一次连接，任务结束就中断连接。

HTTP/1.1起默认使用长连接。要使用长连接，客户端和服务器的HTTP响应头部的Connection要设置为keep-alive，才能支持长连接。

HTTP长连接，指的是复用TCP连接。多个HTTP请求可以复用同一个TCP连接，这就节省了TCP连接建立和断开的消耗。

流水线：

默认情况下，HTTP请求是按顺序发的，下一个请求只有在当前请求收到响应之后才会被发出，有点像停止等待协议。而在收到下一个请求之前，可能需要很长时间。流水线就是在一个HTTP长连接下连续发出请求，不用等待响应返回，减少延迟。

## ARP协议

ARP（Address Resolution Protocol）是一种计算机网络协议，用于将IP地址解析为物理MAC地址。在数据链路层和网络层之间，IP地址与MAC地址之间存在一种映射关系，ARP就是用来进行这种映射的协议。

当计算机需要与同一局域网中的其他设备进行通信时，它需要知道目标设备的MAC地址。但是，通常情况下，计算机只知道目标设备的IP地址，而不知道其对应的MAC地址。这时，计算机会使用ARP协议来解析目标IP地址的MAC地址。

ARP的工作原理如下：

当计算机需要发送数据到目标IP地址时，首先会检查本地的ARP缓存表。ARP缓存表是一个IP地址与MAC地址的映射表，其中保存了最近通信过的设备的MAC地址。

如果目标IP地址在ARP缓存表中找不到对应的MAC地址，计算机会发送一个ARP请求广播到局域网中的所有设备。ARP请求是一个特殊的数据包，其中包含了计算机的MAC地址和目标IP地址。

局域网中所有设备都会接收到ARP请求广播，但只有目标IP地址与请求中的IP地址匹配的设备会响应ARP请求。

目标设备接收到ARP请求后，它会将自己的MAC地址作为ARP响应返回给请求的计算机。

计算机收到ARP响应后，会将目标IP地址与对应的MAC地址保存到ARP缓存表中，并用这个MAC地址作为目标设备的地址，以便将数据包发送到正确的目标设备。

ARP协议能够动态地解析IP地址和MAC地址的映射关系，从而实现数据包在局域网中的正确传输。ARP是一个非常重要的协议，它在局域网通信中发挥着关键作用。

## XSS攻击

XSS（Cross-Site Scripting）攻击是一种常见的 Web 安全漏洞，其攻击目标是 Web 应用程序中的用户，攻击者通过在 Web 页面中植入恶意脚本，从而实现窃取用户敏感信息、篡改用户数据等目的。

XSS 攻击分为两种类型：**存储型XSS**和**反射型XSS**。存储型 XSS 攻击是将恶意脚本存储到服务器端，用户请求页面时，服务器端将恶意脚本插入到页面中，从而实现攻击的目的。反射型 XSS 攻击是将恶意脚本作为参数发送给服务器端，服务器端将恶意脚本返回给用户，用户浏览器执行恶意脚本，从而实现攻击的目的。

### XSS 攻击的原理

XSS 攻击的原理非常简单，攻击者通过在 Web 应用程序中注入恶意代码，利用浏览器对于 JavaScript 代码的信任，从而实现对用户的攻击。

具体来说，XSS 攻击可以分为以下两种类型：

### 存储型 XSS 攻击

存储型 XSS 攻击是指攻击者将恶意代码存储在 Web 应用程序的数据库中，并在用户访问页面时从数据库中读取并执行该代码的一种攻击方式。

攻击者通常会使用一些常见的漏洞，例如未对用户输入进行过滤、未对用户输入进行转义等，从而成功地将恶意代码注入到 Web 应用程序的数据库中。当用户访问受影响的页面时，恶意代码将在用户的浏览器中执行，从而实现攻击。

## 反射型 XSS 攻击

反射型 XSS 攻击是指攻击者将恶意代码作为 URL 参数发送给 Web 应用程序，并在用户访问页面时从 URL 中读取并执行该代码的一种攻击方式。

攻击者通常会使用一些诱导用户点击链接的手段，例如通过电子邮件、社交媒体等方式发送恶意链接，从而实现攻击。

### XSS 攻击的防范措施

为了防止 XSS 攻击，我们可以采取以下一些措施：

#### 输入验证

首先，我们需要对用户输入进行验证，确保用户输入的数据符合我们的要求。例如，我们可以限制用户输入的长度、格式、类型等，从而避免输入错误或者恶意输入。

#### 过滤和转义

其次，我们需要对用户输入进行过滤和转义，从而确保用户输入的数据不会被误认为是 JavaScript 代码。例如，我们可以使用 PHP 内置的 htmlspecialchars() 函数对用户输入进行转义，从而避免 XSS 攻击。

#### CSP

另外，我们还可以使用 Content Security Policy（CSP）来防止 XSS 攻击。CSP 是一种 Web 安全标准，它允许网站管理员指定哪些来源可以加载特定的资源，从而防止恶意资源的加载。

例如，我们可以在 HTTP 响应头中添加以下 CSP 策略：

Content-Security-Policy: default-src 'self'; script-src 'self' 'unsafe-inline';

上述策略指定了只允许从当前域名加载资源，同时允许使用内联脚本。这样一来，即使攻击者成功注入恶意代码，由于该代码不符合 CSP 策略要求，浏览器也不会执行该代码。

### 总结

XSS 攻击是一种常见的 Web 安全漏洞，它利用了 Web 应用程序对用户输入数据的信任，从而向网站注入恶意代码，使得攻击者能够窃取用户的敏感信息或者实施其他恶意行为。为了防止 XSS 攻击，我们需要对用户输入进行验证、过滤和转义，并且可以使用 Content Security Policy（CSP）来增强 Web 应用程序的安全性。

## IP地址和MAC地址

Mac 地址和 IP 地址是两个不同的概念，它们分别代表了计算机网络中的不同层次和地址。Mac 地址是物理地址，是在计算机硬件中存储的地址，通常是以特定的六进制格式表示。每个设备都有一个唯一的 MAC 地址，它可以用来在计算机之间进行通信，以便在网络上识别设备。IP 地址是逻辑地址，是在计算机软件中使用的地址，用于在网络上识别设备。IP 地址是由四段数字组成的，每段数字代表了一个唯一的网络地址。不同的设备有可能使用同一个 IP 地址，但它们必须属于同一个子网络。

IP地址和MAC地址都是网络中非常重要的地址，它们共同构成了一个设备在网络中的唯一标识符。正确使用这些地址可以提高网络的可靠性、安全性和访问控制。对此大家是怎么看的，欢迎关注我创业者李孟和我一起交流！

### IP地址

IP地址是指Internet协议地址，是IP协议提供的用于唯一标识网络中的设备的地址。IP地址是一个32位的二进制数，由四个字节组成，每个字节的范围是0到255。

#### IP地址可以用于以下几个方面：

1、用户登录：当用户在网络中输入IP地址时，服务器可以验证用户的身份并允许其访问特定的资源。

2、网络寻址：IP地址可以用于寻找网络中的设备，以便将其连接到网络。

3、数据传输：IP地址用于指示数据包从哪个端口号发送出去，以及数据包应该如何路由。

4、安全性：IP地址可以用于确保网络的安全性，防止未经授权的设备访问网络。

1. 路由：IP地址用于路由数据包，确保数据包能够正确地到达目标设备。

IP地址是网络中非常重要的地址，它提供了网络寻址和路由的功能，确保了网络中数据包的正确传输。

### 什么是MAC地址

MAC地址是计算机网络中用于唯一标识一个设备的地址，它是一个16位的二进制数字，由两部分组成：

#### MAC组成

1、特殊控制位（CSR）：CSR是用于标识MAC地址的特殊位。每个MAC地址都有一个唯一的CSR值，这些值是由设备制造商分配的。

1. 数据位（Data）：数据位包含了设备的其他信息，例如生产厂商、产品类型和序列号等。

#### MAC地址可以用于以下几个方面：

1、设备识别：在计算机网络中，每个设备都有一个唯一的MAC地址，这有助于识别和定位设备。

2、访问控制：MAC地址可以用于限制设备的访问。例如，可以将MAC地址与用户账户相关联，以确保只有授权用户才能访问网络资源。

3、安全性：MAC地址也可以用于认证设备，以确保只有经过授权的设备才能访问网络资源。

1. 路由：当一台设备通过路由器时，路由器会将IP地址与MAC地址对应起来，以确保数据包正确地转发到目标设备。

#### MAC地址用途

此外，MAC地址也可以用于限制设备的访问。例如，可以将MAC地址与用户账户相关联，以确保只有授权用户才能访问网络资源。这有助于确保网络的安全性和访问控制。

在网络中，IP地址和MAC地址是相互关联的。当一台设备通过路由器时，路由器会将IP地址与MAC地址对应起来，以确保数据包正确地转发到目标设备。因此，正确使用IP地址和MAC地址可以提高网络的可靠性和安全性。

#### MAC总结

MAC地址是计算机网络中非常重要的地址，它能够帮助我们识别和定位设备，限制设备的访问，提高网络的安全性和可靠性。

## TCP流量控制（滑动窗口机制）

流量控制：控制发送方的发送速率，以便接受方来得及接受。

TCP利用滑动窗口机制实现流量控制：在通信过程中，接收方根据自己的接受缓存大小，即接受窗口rwnd，动态的调整发送方的发送窗口大小（接受方设置确认报文段的窗口字段来将rwnd通知发送方)，发送方的发送窗口取接受窗口rwnd和拥塞窗口cwnd中的最小值。

TCP为每一个连接设有一个持续计时器，只要TCP连接的一方收到对方的零窗口通知，就启动持续计时器；若持续计时器设置的时间到期，就发送一个零窗口探测报文段；接受方收到零窗口探测报文时，给出现在窗口值；若窗口还是0，则发送方重新设置持续计时器。

## TCP拥塞控制

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。在某段时间，若对网络中某一资源的需求超过了该资源所能提供的可用部分，网络的性能就要变坏。这种情况就叫拥塞。

拥塞控制就是为了防止过多的数据注入到网络中，这样就可以使网络中的路由器或链路不致于过载。拥塞控制所要做的都有一个前提，就是网络能够承受现有的网络负荷。拥塞控制是一个全局性的过程，涉及到所有的主机，所有的路由器，以及与降低网络传输性能有关的所有因素。相反，流量控制往往是点对点通信量的控制，是个端到端的问题。流量控制所要做到的就是抑制发送端发送数据的速率，以便使接收端来得及接收。

为了进行拥塞控制，TCP发送方要维持一个拥塞窗口(CWND)的状态变量。拥塞控制窗口的大小取决于网络的拥塞程度，并且动态变化。发送方让自己的发送窗口取为拥塞窗口和接收方的接受窗口中较小的一个。TCP的拥塞控制采用了四种算法，即：慢开始、拥塞避免、快重传和快恢复。在网络层也可以使路由器采用适当的分组丢弃策略（如：主动队列管理AQM），以减少网络拥塞的发生。

慢开始：

慢开始算法的思路是当主机开始发送数据时，如果立即把大量数据字节注入到网络，那么可能会引起网络阻塞，因为现在还不知道网络的符合情况。经验表明，较好的方法是先探测一下，即由小到大逐渐增大发送窗口，也就是由小到大逐渐增大拥塞窗口数值。CWND初始值为1，每经过一个传播轮次，CWND加倍。

拥塞避免： 拥塞避免算法的思路是让拥塞窗口CWND缓慢增大，即每经过一个往返时间RTT就把发送方的CWND加1。

快重传与快恢复： 在TCP/IP中，快速重传和快恢复（FRR）是一种拥塞控制算法，它能快速恢复丢失的数据包。

没有FRR，如果数据包丢失了，TCP将会使用定时器来要求传输暂停。在暂停的这段时间内，没有新的或复制的数据包被发送。有了FRR，如果接收机接收到一个不按顺序的数据段，它会立即给发送机发送一个重复确认。如果发送机接收到三个重复确认，它会假定确认件指出的数据段丢失了，并立即重传这些丢失的数据段。有了FRR，就不会因为重传时要求的暂停被耽误。当有单独的数据包丢失时，快速重传和快恢复（FRR）能最有效地工作。当有多个数据信息包在某一段很短的时间内丢失时，它则不能很有效地工作。

## TCP 如何保证可靠传输

**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。

## DHCP协议

DHCP（动态主机配置协议）是一个局域网的网络协议。

指的是由服务器控制一段IP地址范围，给用户提供了即插即用的联网方式，用户不需要再手动配置IP地址等信息

该协议自动为用户分配TCP/IP参数信息，如：IP地址，子网掩码，网关等信息。

## HTTP请求转发和重定向

### 定义

转发：是指服务器接收到一个请求后，将请求转发给另一个资源进行处理，并将该资源的处理结果返回给客户端。在这个过程中，转发后的资源对客户端是不可见的，客户端只知道自己访问了一个资源，而不知道这个资源是被转发到的。

重定向：是指服务器接收到一个请求后，发现该请求需要访问另一个资源才能得到响应，于是告诉客户端重新发送一个请求，访问另一个资源。在这个过程中，客户端会重新发送一个请求，访问另一个资源，因此客户端会知道自己访问了两个资源。

### 区别

#### 1.可见性

转发对客户端是不可见的，而重定向对客户端是可见的。当服务器通过转发将请求转发给另一个资源进行处理时，客户端只知道访问了一个资源，而不知道这个资源是被转发到的。而当服务器通过重定向告诉客户端重新发送一个请求时，客户端会知道自己访问了两个资源。

#### 2.速度

转发比重定向快。因为转发是在服务器内部进行的，不需要向客户端发送HTTP响应头，也不需要客户端重新发送一个请求，因此速度比重定向快。而重定向需要向客户端发送HTTP响应头，并告诉客户端重新发送一个请求，因此速度较慢。

#### 3.资源路径

转发可以访问相对路径和绝对路径的资源，而重定向只能访问绝对路径的资源。因为转发是在服务器内部进行的，可以访问相对路径和绝对路径的资源。而重定向是在客户端进行的，只能访问绝对路径的资源。

#### 4.请求次数

转发只需要发送一次请求，而重定向需要发送两次请求。因为转发是在服务器内部进行的，只需要发送一次请求即可完成跳转。而重定向需要向客户端发送HTTP响应头，并告诉客户端重新发送一个请求，因此需要发送两次请求。

#### 5.客户端处理

转发是服务器内部的处理机制，客户端不需要进行任何处理。而重定向需要客户端重新发送一个请求，因此需要客户端进行相应的处理。

### 总结

转发和重定向都是网页跳转的方式，它们之间的区别主要在于可见性、速度、资源路径、请求次数和客户端处理等方面。转发对客户端是不可见的，速度比重定向快，可以访问相对路径和绝对路径的资源，只需要发送一次请求，客户端不需要进行任何处理。而重定向对客户端是可见的，速度较慢，只能访问绝对路径的资源，需要发送两次请求，客户端需要进行相应的处理。选择哪种方式取决于具体的需求和情况。

## SQL注入

SQL注入就是将原本的SQL语句的逻辑结构改变，使得SQL语句的执行结果和原本开发者的意图不一样；

比如说我们通过账号密码进行登录，首先需要进行查询判断：

"select \* from t\_user where name = "张三" and password = "123456"

如果我们把or 1 = 1作为拼接在密码末尾传入进来，执行查询的时候sql会变成

"select \* from t\_user where name = "张三" and password = "123456" or 1 = 1

密码永远都是成立的，所以，前面无论密码正确与否都会成功登录

我们使用预编译语句就可以避免这个问题，例如：JDBC中的PrepareStatement或MyBatis中#{}

将SQL预编译，传参数的时候，不会改变SQL语句结构，就可以避免注入。

## URL和URI区别

URL，统一资源定位符，URL 其实就是我们平时上网时输入的网址，它标识一个互联网资源，并指定对其进行操作或获取该资源的方法。例如

http://coderx.cc/blog/28 这个 URL，标识一个特定资源并表示该资源的某种形式是可以通过HTTP协议从相应位置获得。

URI，统一资源标识符，URL是URI的一个子集，两者都定义了资源是什么，而 URL还定义了如何能访问到该资源。URI是一种语义上的抽象概念，可以是绝对的，也可以是相对的，而URL则必须提供足够的信息来定位，是绝对的。简单地说，只要能唯一标识资源的就是 URI，在URI的基础上给出其资源的访问方式的就是 URL。

通俗理解：

URI表示的是一个抽象的地址，URL表示的是一个详细的地址。

抽象的地址：湖北省荆州市（这是一个抽象的地址，相当于URI）

详细的地址：湖北省荆州市荆州区学苑路长江大学（这是一个详细的地址，相当于URL）

为什么URL是URI的子集，长江大学（URL）属于湖北省荆州市（URI）

那么http://coderx.cc 是一个URI（URI只指明了服务器的地址，没有具体到文件是什么类型）

那么http://coderx.cc/blog/28就是一个URL（他具体到了blog文章的位置）

## 粘包与拆包

### 产生原因：

要发送的数据小于TCP发送缓冲区的大小，TCP将多次写入缓冲区的数据一次发送出去，将会发生粘包；

接收数据端的应用层没有及时读取接收缓冲区中的数据，将发生粘包；

要发送的数据大于TCP发送缓冲区剩余空间大小，将会发生拆包；

待发送数据大于MSS（最大报文长度），TCP在传输前将进行拆包。即TCP报文长度-TCP头部长度>MSS。

### 解决策略：

消息定长。发送端将每个数据包封装为固定长度（不够的可以通过补0填充），这样接收端每次接收缓冲区中读取固定长度的数据就自然而然的把每个数据包拆分开来。

设置消息边界。服务端从网络流中按消息边界分离出消息内容。在包尾增加回车换行符进行分割，例如FTP协议。

将消息分为消息头和消息体，消息头中包含表示消息总长度（或者消息体长度）的字段。

更复杂的应用层协议。

什么是阻塞和非阻塞，同步和异步

阻塞和非阻塞：调用者在事件没有发生的时候，一直等待事件发生，不能处理其他任务是阻塞。调用者在事件没有发生的时候，可以去处理别的事务这是非阻塞。

同步和异步：调用者循环查看事件有没有发生，这种情况是同步。调用者不用自己去查看事件有没有发生，而是等待注册在时间上的回调函数自己通知自己，这是异步。

## http1.0，http1.1，http2，http3

### http1.0

HTTP1.0最早在网页中使用是在1996年，那个时候只是使用一些较为简单的网页上和网络请求上,是一种**无状态、无连接**的应用层协议，几年后被HTTP1.1代替并广泛使用

### http1.1

http1.1基于**文本解析**,把所有请求和响应作为纯文本

http1.1加入了**缓存处理（强缓存和协商缓存）**

http1.1拥有长连接，并支持请求**管道化（pipelining）**，

http1.1流控制基于**tcp连接**。当连接建立时，两端通过系统默认机制建立缓冲区。并通过ack报文来通知对方接收窗口大小，因为http1.1 依靠传输层来避免流溢出，每个tcp连接需要一个独立的流控制机制

### 缓存处理（强缓存和协商缓存）

浏览器缓存能优化性能，而浏览器缓存分为强缓存和协商缓存，都是从客户端读取缓存

#### 强缓存

强缓存不发送请求，直接读取资源，可以获得返回200的状态码

利用http头中的Expires和Cache-Control两个字段来控制，都用来表示资源的缓存时间，Expires能设置失效时间，而Cache-Control能做到更多选项更细致，如果同时设置的话，其优先级高于Expires

#### 协商缓存

通过服务器来确定缓存资源是否可用，通过request header判断是否命中请求，命中后返回304状态码，并返回新的request header通知客户端从缓存里取

普通刷新会启用弱缓存，忽略强缓存。只有在地址栏或收藏夹输入网址、通过链接引用资源等情况下，浏览器才会启用强缓存

如果时间过期，则向服务器发送header带有If-None-Match和If-Modified-Since的请求，回到1

### http2

http2相比于http1.1，**性能**大幅度提升

http2通过一个连接来**多路复用**

http2拥有**头部压缩**

http2拥有**新的二进制格式**，使用二进制框架层把所有消息封装成二进制，且仍然保持http语法

http2允许客户端和服务器端实现他们自己的流控制机制，而不是依赖传输层,两端在传输层交换可用的缓冲区大小，来让他们在多路复用流上设置自己的接收窗口

http2让服务器可以将响应主动“**推送**”到客户端缓存中

#### htpp2头部压缩

http2头部压缩又称为HAPCK，设计简单而灵活，是因为HPACK格式有意地简单且不灵活能降低由于实现错误而导致的互操作性或安全问题的风险

http1.1没有头部压缩，随着请求增加，冗余头部字段会不必要地占用带宽，从而显着增加延迟，而头部压缩可消除冗余报头字段，限制已知安全攻击的漏洞，并且在受限环境中使用有限的内存要求

#### http2多路复用

http 性能优化的关键并不在于高带宽，而是低延迟

tcp 连接会随着时间进行自我「调谐」，起初会限制连接的最大速度，如果数据成功传输，会随着时间的推移提高传输的速度,这种调谐则被称为 tcp 慢启动,由于这种原因，让原本就具有突发性和短时性的 http 连接变的十分低效

http/2 通过让所有数据流共用同一个连接，可以更有效地使用 tcp 连接，让高带宽也能真正的服务于 http 的性能提升。而http1.1存在低性能的线头阻塞，一旦有一个请求超时，便会出现阻塞等待的情况

### http3

之前说了http2，那么http3就是为了解决http2相关问题而诞生，它基于一个新的传输层协议QUIC，而http3就是建立一个在QUIC上运行的HTTP新规范，而http3之前的版本都是基于TCP，QUIC就是为了替代TCP，解决TCP的一些缺陷

### tcp

1. **不支持流级复用**，TCP会将所有对象序列化在同一个流中，因此，它不知道TCP段的对象级分区，无法在同一个流中复用数据包
2. **会产生冗余通信**，tco三次连接握手会有冗余的消息交换序列
3. **可能会间歇性地挂起数据传输**，tcp中有个因为序列顺序处理丢失的问题的缺陷称为行头阻塞

### QUIC

同样拥有头部压缩，并优化了对乱序发送的支持，也优化了压缩率

放弃tcp，通过udp建立，提高了连接建立的速度，降低了延迟

tcp本身是无法解决队头拥塞，quic则解决了这个问题

Connection ID使得http3支持连接迁移以及NAT的重绑定

### HTTP1.0对比HTTP1.1

HTTP1.1主要改进了以下几点内容

* keep-alive
* 客户端缓存
* 连接代宽优化
* 请求Host域
* 请求状态码
* 请求方法

#### keep-alive

在HTTP 1.0中，客户端的每次请求都要求建立一次单独的连接，在处理完本次请求后，就自动释放连接。

在HTTP 1.1中则可以在一次连接中处理多个请求，并且多个请求可以重叠进行，不需要等待一个请求结束后再发送下一个请求。

建立起一个 TCP 连接需要经过“三次握手”：

第一次握手：客户端发送syn包(syn=j)到服务器，并进入SYN\_SEND状态，等待服务器确认；

第二次握手：服务器收到syn包，必须确认客户的SYN（ack=j+1），同时自己也发送一个SYN包（syn=k），即SYN+ACK包，此时服务器进入SYN\_RECV状态；

第三次握手：客户端收到服务器的SYN＋ACK包，向服务器发送确认包ACK(ack=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED状态，完成三次握手。

握 手过程中传送的包里不包含数据，三次握手完毕后，客户端与服务器才正式开始传送数据。理想状态下，TCP连接一旦建立，在通信双方中的任何一方主动关闭连 接之前，TCP 连接都将被一直保持下去。断开连接时服务器和客户端均可以主动发起断开TCP连接的请求，断开过程需要经过“四次握手”（过程就不细写 了，就是服务器和客户端交互，最终确定断开）

HTTP 1.1持久连接（PersistentConnection）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TCP连接的消耗和延迟，在HTTP1.1中默认开启Connection： keep-alive，一定程度上弥补了HTTP1.0每次请求都要创建连接的缺点。

#### 客户端缓存

在HTTP/1.0中，使用Expire头域来判断资源的fresh或stale，并使用条件请求（conditional request）来判断资源是否仍有效。例如，cache服务器通过If-Modified-Since头域向服务器验证资源的Last-Modefied头域是否有更新，源服务器可能返回304（Not Modified），则表明该对象仍有效；也可能返回200（OK）替换请求的Cache对象。

此外，HTTP/1.0中还定义了Pragma:no-cache头域，客户端使用该头域说明请求资源不能从cache中获取，而必须回源获取。

HTTP/1.1在1.0的基础上加入了一些cache的新特性，当缓存对象的Age超过Expire时变为stale对象，cache不需要直接抛弃stale对象，而是与源服务器进行重新激活（revalidation）。

HTTP/1.0中，If-Modified-Since头域使用的是绝对时间戳，精确到秒，但使用绝对时间会带来不同机器上的时钟同步问题。而HTTP/1.1中引入了一个ETag头域用于重激活机制，它的值entity tag可以用来唯一的描述一个资源。请求消息中可以使用If-None-Match头域来匹配资源的entitytag是否有变化。

为了使caching机制更加灵活，HTTP/1.1增加了Cache-Control头域（请求消息和响应消息都可使用），它支持一个可扩展的指令子集：例如max-age指令支持相对时间戳；private和no-store指令禁止对象被缓存；no-transform阻止Proxy进行任何改变响应的行为。

Cache使用关键字索引在磁盘中缓存的对象，在HTTP/1.0中使用资源的URL作为关键字。但可能存在不同的资源基于同一个URL的情况，要区别它们还需要客户端提供更多的信息，如Accept-Language和Accept-Charset头域。为了支持这种内容协商机制(content negotiation mechanism)，HTTP/1.1在响应消息中引入了Vary头域，该头域列出了请求消息中需要包含哪些头域用于内容协商。

#### 强缓存、协商缓存

我们在使用浏览器访问一个web页面的时候，浏览器会将该网页中的资源和相应的服务端response header强制缓存起来，在这些头信息中有两个字段来控制强缓存Cache-Control和Expires，如果这两个字段验证通过就使用缓存，如果没有通过就会带着另外两个字段Etag/If-None-Match和Last-Modified/If-Modified-Since去向服务端发送一个验证（协商）请求，如果验证通过就返回304继续使用缓存，如果没有通过就返回200和对应的资源。

#### 连接代宽优化

HTTP1.1支持传送内容的一部分。比方说，当客户端已经有内容的一部分，为了节省带宽，可以只向服务器请求一部分。

HTTP/1.0中，存在一些浪费带宽的现象，例如客户端只是需要某个对象的一部分，而服务器却将整个对象送过来了。例如，客户端只需要显示一个文档的部分内容，又比如下载大文件时需要支持断点续传功能，而不是在发生断连后不得不重新下载完整的包。

HTTP/1.1中在请求消息中引入了range头域，它允许只请求资源的某个部分。

"Content-Type": "application/octet-stream",

"Content-Range": \`bytes ${startPosition}-${endPosition}/${file.size}\`

在响应消息中Content-Range头域声明了返回的这部分对象的起始位置和长度。如果服务器相应地返回了对象所请求范围的内容，则响应码为206（Partial Content），它可以防止Cache将响应误以为是完整的一个对象。我的文件并发上传项目有使用到这个请求方式。

节省带宽资源的一个非常有效的做法就是压缩要传送的数据。Content-Encoding是对消息进行端到端（end-to-end）的编码，它可能是资源在服务器上保存的固有格式（如jpeg图片格式）；在请求消息中加入Accept-Encoding头域，它可以告诉服务器客户端能够解码的编码方式。

而Transfer-Encoding是逐段式（hop-by-hop）的编码，如Chunked编码。在请求消息中加入TE头域用来告诉服务器能够接收的transfer-coding方式，

#### 请求Host域

\*\*Host \*\*请求头指明了请求将要发送到的服务器主机名和端口号。HTTP1.0并没有包含Host，

如果没有包含端口号，会自动使用被请求服务的默认端口（比如HTTPS URL使用443端口，HTTP URL使用80端口）。

如果使用的HTTP1.0我们就不能给一个服务器部署多个服务，HTTP1.1解决了这个问题。

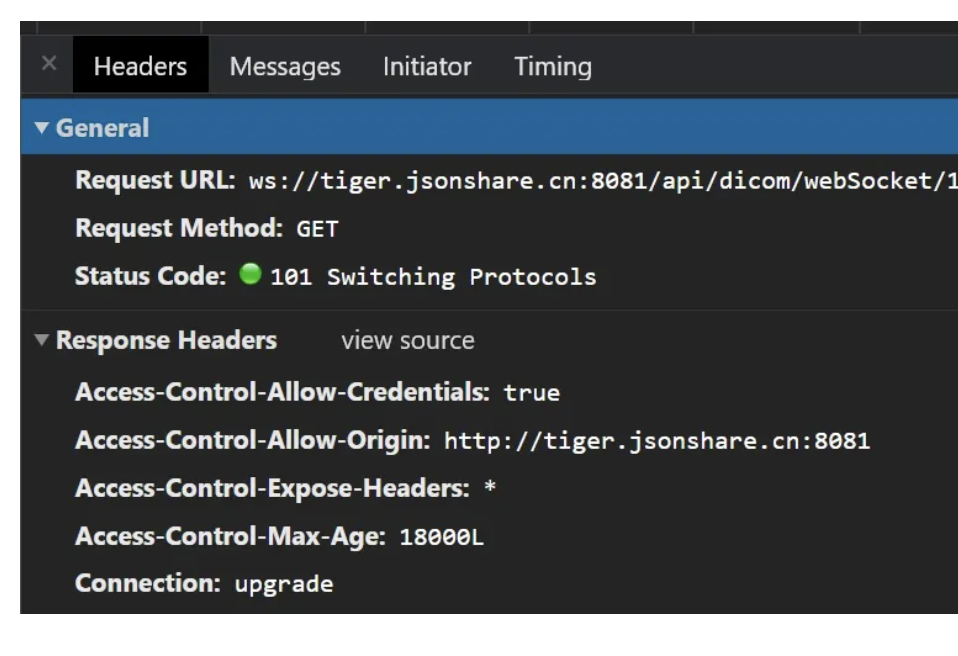
所有HTTP/1.1 请求报文中必须包含一个Host头字段。对于缺少Host头或者含有超过一个Host头的HTTP/1.1 请求，可能会收到400（Bad Request）状态码。

#### 请求状态码

HTTP1.1新增了两个1.0中没有的状态码，

100 (Continue) 状态代码的使用，允许客户端在发request消息body之前先用request header试探一下server，看server要不要接收request body，再决定要不要发request body。

101 (Switching Protocols) 当我们在网页中使用websocket的时候就会出现一个这样的状态码



#### 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。

### HTTP2.0

http2.0是一种安全高效的下一代http传输协议。安全是因为http2.0建立在https协议的基础上，高效是因为它是通过二进制分帧来进行数据传输。

他对于HTTP1.x主要有以下改进

* 二进制分帧（Binary Format）
* 头部压缩（Header Compression）
* 服务端推送（Server Push）
* 多路复用 (Multiplexing) / 连接共享，让多个请求合并在同一 TCP 连接内
* 请求优先级（Request Priorities）

#### 二进制分帧

有别于HTTP/1.1在连接中的明文请求，HTTP/2与SPDY一样，将一个TCP连接分为若干个流（Stream），每个流中可以传输若干消息（Message），每个消息由若干最小的二进制帧（Frame）组成。这也是HTTP/1.1与HTTP/2最大的区别所在。 HTTP/2中，每个用户的操作行为被分配了一个流编号(stream ID)，这意味着用户与服务端之间创建了一个TCP通道

启用http2.0后会给性能带来很大的提升，但同时也会带来新的性能瓶颈。因为现在所有的压力集中在底层一个TCP连接之上，TCP很可能就是下一个性能瓶颈，比如TCP分组的队首阻塞问题，单个TCP packet丢失导致整个连接阻塞，无法逃避，此时所有消息都会受到影响。

具体内容有机会可以单独总结一下，我在另一篇文章总结了HTTP和HTTPS区别。

#### 多路复用

同一域名下多个请求公用同一个连接，不限制请求数量，双向数据流，多个请求之前数据是无序的，浏览器最终会根据每一帧的编号进行组装。

#### header压缩

在HTTP1中每次传输的header都属文本形式传输，当header中的参数比较多时，对于压缩就显得很有必要了，头部会在连接上始终存在，对于多个请求中重复的字段会自动过滤。

#### Server Push

服务器会对于客户端必需的资源在双方协商好的前提下会主动推送。

### HTTP3

HTTP3是一个基于UDP协议的应用层协议，并且集成了HTTP2的加密传输、多路复用的几个特点

QUIC (Quick UDP Internet Connections), 快速 UDP 互联网连接。

QUIC是基于UDP协议的。

#### 线头阻塞(HOL)问题的解决更为彻底：

基于TCP的HTTP/2，尽管从逻辑上来说，不同的流之间相互独立，不会相互影响，但在实际传输方面，数据还是要一帧一帧的发送和接收，一旦某一个流的数据有丢包，则同样会阻塞在它之后传输的流数据传输。而基于UDP的QUIC协议则可以更为彻底地解决这样的问题，让不同的流之间真正的实现相互独立传输，互不干扰。

#### 切换网络时的连接保持

当前移动端的应用环境，用户的网络可能会经常切换，比如从办公室或家里出门，WiFi断开，网络切换为3G或4G。基于TCP的协议，由于切换网络之后，IP会改变，因而之前的连接不可能继续保持。而基于UDP的QUIC协议，则可以内建与TCP中不同的连接标识方法，从而在网络完成切换之后，恢复之前与服务器的连接。

总结：基于UDP和应用层实现了TCP协议。

## 什么是websocket

WebSocket是HTML5下一种新的协议（websocket协议本质上是一个基于tcp的协议）

它实现了浏览器与服务器全双工通信，能更好的节省服务器资源和带宽并达到实时通讯的目的

Websocket是一个持久化的协议

### websocket的原理

websocket约定了一个通信的规范，通过一个握手的机制，客户端和服务器之间能建立一个类似tcp的连接，从而方便它们之间的通信

在websocket出现之前，web交互一般是基于http协议的短连接或者长连接

websocket是一种全新的协议，不属于http无状态协议，协议名为"ws"

### websocket与http的关系

相同点：

都是基于tcp的，都是可靠性传输协议

都是应用层协议

不同点：

WebSocket是双向通信协议，模拟Socket协议，可以双向发送或接受信息

HTTP是单向的

WebSocket是需要浏览器和服务器握手进行建立连接的

而http是浏览器发起向服务器的连接，服务器预先并不知道这个连接

联系：

WebSocket在建立握手时，数据是通过HTTP传输的。但是建立之后，在真正

传输时候是不需要HTTP协议的

总结（总体过程）：

首先，客户端发起http请求，经过3次握手后，建立起TCP连接；http请求里存放WebSocket支持的版本号等信息，如：Upgrade、Connection、WebSocket-Version等；

然后，服务器收到客户端的握手请求后，同样采用HTTP协议回馈数据；

最后，客户端收到连接成功的消息后，开始借助于TCP传输信道进行全双工通信。

### websocket解决的问题

#### 1.http存在的问题

http是一种无状态协议，每当一次会话完成后，服务端都不知道下一次的客户端是谁，需要每次知道对方是谁，才进行相应的响应，因此本身对于实时通讯就是一种极大的障碍

http协议采用一次请求，一次响应，每次请求和响应就携带有大量的header头，对于实时通讯来说，解析请求头也是需要一定的时间，因此，效率也更低下

最重要的是，需要客户端主动发，服务端被动发，也就是一次请求，一次响应，不能实现主动发送

#### 2.long poll(长轮询)

对于以上情况就出现了http解决的第一个方法——长轮询

基于http的特性，简单点说，就是客户端发起长轮询，如果服务端的数据没有发生变更，会 hold

住请求，直到服务端的数据发生变化，或者等待一定时间超时才会返回。返回后，客户端又会立即再次发起下一次长轮询

优点是解决了http不能实时更新的弊端，因为这个时间很短，发起请求即处理请求返回响应，实现了“伪·长连接”

张三取快递的例子，张三今天一定要取到快递，他就一直站在快递点，等待快递一到，立马取走

从例子上来看有个问题：

假如有好多人一起在快递站等快递，那么这个地方是否足够大，（抽象解释：需要有很高的并发，同时有很多请求等待在这里）

总的来看：

推送延迟。服务端数据发生变更后，长轮询结束，立刻返回响应给客户端。

服务端压力。长轮询的间隔期一般很长，例如 30s、60s，并且服务端 hold 住连接不会消耗太多服务端资源。

#### 3.Ajax轮询

基于http的特性，简单点说，就是规定每隔一段时间就由客户端发起一次请求，查询有没有新消息，如果有，就返回，如果没有等待相同的时间间隔再次询问

优点是解决了http不能实时更新的弊端，因为这个时间很短，发起请求即处理请求返回响应，把这个过程放大n倍，本质上还是request =

response

举个形象的例子（假设张三今天有个快递快到了，但是张三忍耐不住，就每隔十分钟给快递员或者快递站打电话，询问快递到了没，每次快递员就说还没到，等到下午张三的快递到了，but，快递员不知道哪个电话是张三的，（可不是只有张三打电话，还有李四，王五），所以只能等张三打电话，才能通知他，你的快递到了）

从例子上来看有两个问题：

假如说，张三打电话的时间间隔为10分钟，当他收到快递前最后一次打电话，快递员说没到，他刚挂掉电话，快递入库了（就是到了），那么等下一次时间到了，张三打电话知道快递到了，那么这样的通讯算不算实时通讯？很显然，不算，中间有十分钟的时间差，还不算给快递员打电话的等待时间（抽象的解释：每次request的请求时间间隔等同于十分钟，请求解析相当于等待）

假如说张三所在的小区每天要收很多快递，每个人都采取主动给快递员打电话的方式，那么快递员需要以多快的速度接到，其他人打电话占线也是问题（抽象解释：请求过多，服务端响应也会变慢）

总的来看，Ajax轮询存在的问题：

推送延迟。

服务端压力。配置一般不会发生变化，频繁的轮询会给服务端造成很大的压力。

推送延迟和服务端压力无法中和。降低轮询的间隔，延迟降低，压力增加；增加轮询的间隔，压力降低，延迟增高

#### 4.websocket的改进

一旦WebSocket连接建立后，后续数据都以帧序列的形式传输。在客户端断开WebSocket连接或Server端中断连接前，不需要客户端和服务端重新发起连接请求。在海量并发及客户端与服务器交互负载流量大的情况下，极大的节省了网络带宽资源的消耗，有明显的性能优势，且客户端发送和接受消息是在同一个持久连接上发起，实现了“真·长链接”，实时性优势明显。