# 网络协议相关

**http，http2，https**

**HTTP**

HTTP 超文本传输协议是位于 TCP/IP 体系结构中的应用层协议，它是万维网数据通信的基础。

当我们访问一个网站时，需要通过统一资源定位符（uniform resource locator，URL）来定位服务器并获取资源。

HTTP 的默认端口是 80，通常可以省略。

HTTP 协议是个无状态协议，不会保存状态

### **HTTP 连接建立过程**

我们来看一下在浏览器输入 URL 后获取 HTML 页面的过程。

1. 先通过域名系统（Domain Name System，DNS）查询将域名转换为 IP 地址。即将 test.com 转换为 221.239.100.30 这一过程。
2. 通过三次握手（稍后会讲）建立 TCP 连接。
3. 发起 HTTP 请求。
4. 目标服务器接收到 HTTP 请求并处理。
5. 目标服务器往浏览器发回 HTTP 响应。
6. 浏览器解析并渲染页面。

**影响一个 HTTP 网络请求的因素主要有两个：带宽和延迟。**

· **带宽**：如果说我们还停留在拨号上网的阶段，带宽可能会成为一个比较严重影响请求的问题，但是现在网络基础建设已经使得带宽得到极大的提升，我们不再会担心由带宽而影响网速，那么就只剩下延迟了。

· **延迟**：

· **浏览器阻塞（HOL blocking）**：浏览器会因为一些原因阻塞请求。浏览器对于同一个域名，同时只能有 4 个连接（这个根据浏览器内核不同可能会有所差异），超过浏览器最大连接数限制，后续请求就会被阻塞。

* **DNS 查询（DNS Lookup）**：浏览器需要知道目标服务器的 IP 才能建立连接。将域名解析为 IP 的这个系统就是 DNS。这个通常可以利用DNS缓存结果来达到减少这个时间的目的。
* **建立连接（Initial connection）**：HTTP 是基于 TCP 协议的，浏览器最快也要在第三次握手时才能捎带 HTTP 请求报文，达到真正的建立连接，但是这些连接无法复用会导致每次请求都经历三次握手和慢启动。三次握手在高延迟的场景下影响较明显，慢启动则对文件类大请求影响较大。

## **HTTP/2**

HTTP/2 是 HTTP/1.x 的扩展，而非替代。所以 HTTP 的语义不变，提供的功能不变，HTTP 方法、状态码、URL 和首部字段等这些核心概念也不变。

之所以要递增一个大版本到 2.0，主要是因为它改变了客户端与服务器之间交换数据的方式。HTTP 2.0 增加了新的二进制分帧数据层，而这一层并不兼容之前的 HTTP 1.x 服务器及客户端——是谓 2.0。

### http2 新特性

● **新的二进制格式（Binary Format）**，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（MultiPlexing）**，即连接共享，即每一个request都是是用作连接共享机制的

**HTTPS**

HTTPS 是最流行的 HTTP 安全形式,使用 HTTPS 时，所有的 HTTP 请求和响应数据在发送之前，都要进行加密。加密可以使用 SSL 或 TLS

要想了解 HTTPS 为何安全，还得继续了解一下这些概念：**加密算法**、**摘要算法**、**数字签名**和**数字证书**。

## HTTPS与HTTP的一些区别

1. HTTPS协议需要到CA申请证书，一般免费证书很少，需要交费。
2. HTTP协议运行在TCP之上，所有传输的内容都是明文，HTTPS运行在SSL/TLS之上，SSL/TLS运行在TCP之上，所有传输的内容都经过加密的。
3. HTTP和HTTPS使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。
4. HTTPS可以有效的防止运营商劫持，解决了防劫持的一个大问题。

**HTTP缓存方式**

强制缓存 服务端设置响应头Cache-Control:max-age=xxx，并且设置Expires响应头过期时间，客户端自行判断是否读取缓存

协商缓存 通过状态码304告诉客户端该走缓存

修改时间：通过文件的最后修改时间判断该不该读取缓存，服务端设置响应头Last-Modified,客户端把上次服务端响应头中的Last-modified值通过if-modified-since 传递给服务端 ， 服务端通过比较当前文件的修改时间和上次修改时间(上次传给客户端的值),如果相等那么说明文件修改时间没变也就是没变化

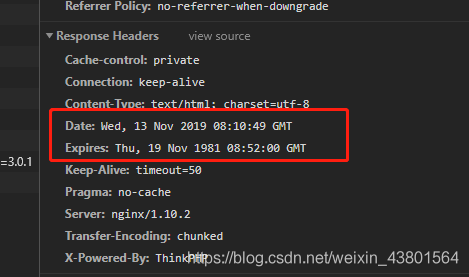
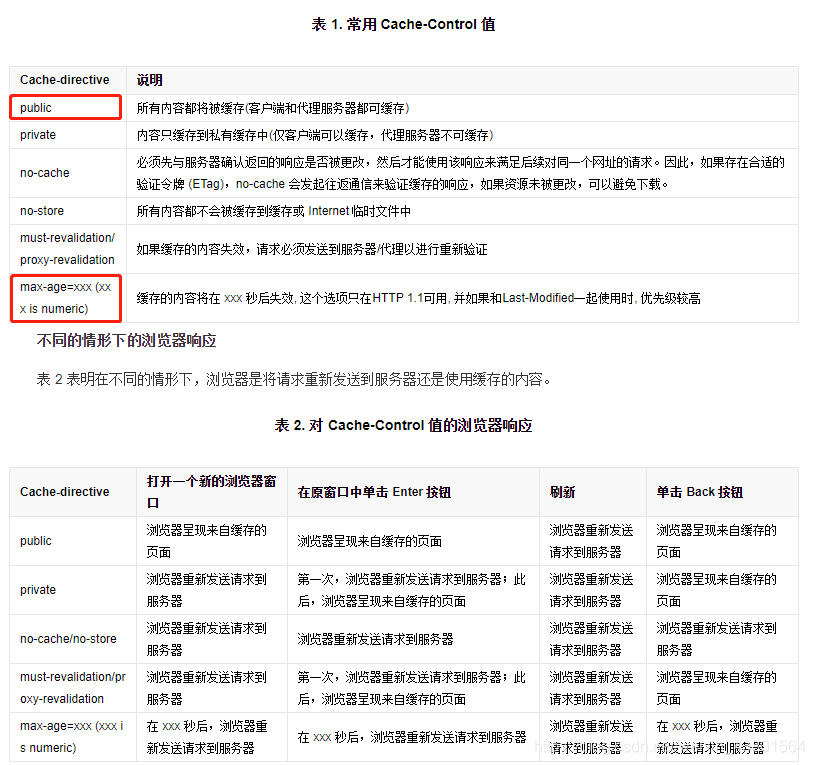
文件内容：通过文件的内容来判断该不该读取缓存，服务端通过把文件内容读取出来，通过md5进行base64加密得出hash值，把这个值设置响应头Etag，客户端下一次请求通过if-none-match带过来，服务端再比对当前文件内容加密得出的hash值和上次是否一样，如果一样说明文件内容没有发生改变，这种方式是最准确的方式，但是也是最耗性能

## **浏览器缓存分类**

****1.强缓存:****  
指浏览器直接从浏览器缓存中读取资源,不请求服务器,状态码显示200,并且size显示from disk cache或from memory cache;

****2.协商缓存****  
指浏览器向服务器发送请求,并且根据request headers中携带的数据判断是否需要从缓存中读取,如果判断资源未发生变动,则返回304,并且从通过新的response header让浏览器从缓存中读取资源;

### **强缓存**

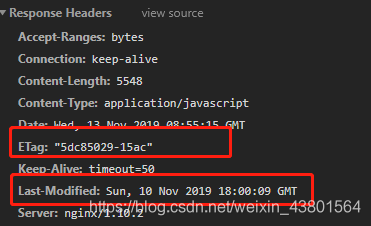
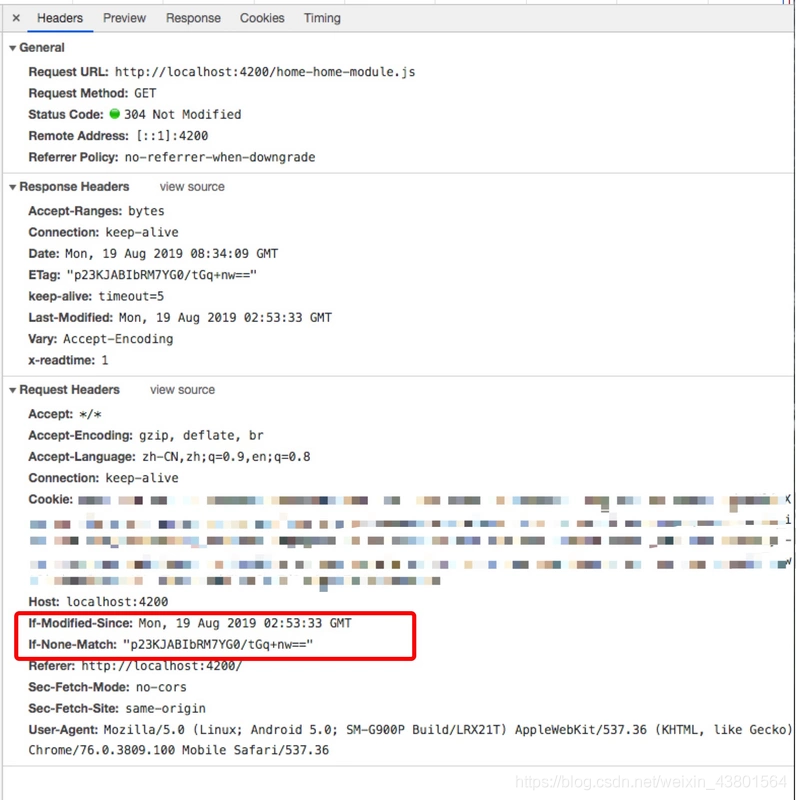
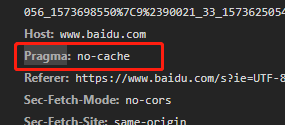
资源强缓存,主要是跟max-age有关,也就初次请求资源中携带的过期时间,主要跟expires和cache-control有关  
****expires****  
指资源到期时间,是个具体的时间,如果当前时间在这个expires时间内,则命中强缓存(上面Date表示资源请求的服务器时间),如果本地时间跟服务器时间差别较大时,可能会有点影响.  
  
****cache-control****  
贴一张cache-control的值,当值为public或者max-age=\*\*\*时可能会命中强缓存.不过现在资源一般都使用no-cache  


**cache-control的优先级高于expires,expires是http1.0的产物,而cache-control是http1.1的产物,在不支持http1.1的情况下可能就需要expires来保持兼容.**

### **协商缓存**

简单来讲,就是首次请求资源时,资源返回的response header中会带有ETag或者Last-Modified,之后重新请求资源的时候回带上这部分信息,跟服务器上对应的值进行对应,判断资源是否发生改变

ETag 对应 If-None-Match  
Last-Modified 对应 If-Modified-Since

  
  
还有一个Pragma  
  
功能同cache-control:no-cache,为了兼容一些支持http1.0的请求  
****ETag和Last-Modified的区别****

* 1.ETag更精准,Last-Modified的时间精度是秒,如果一个文件在一秒内改变多次,则Last-Modified不会发生变化,同时如果设置了负载均衡,则不同的服务器上的Last-Modified也可能不同,而ETag每次文件改变都会生成一个全新的值;
* 2.Last-Modified性能更好,Last-Modified只是个时间,而ETag需要执行一定算法生成;
* 3.ETag优先级高于Last-Modified

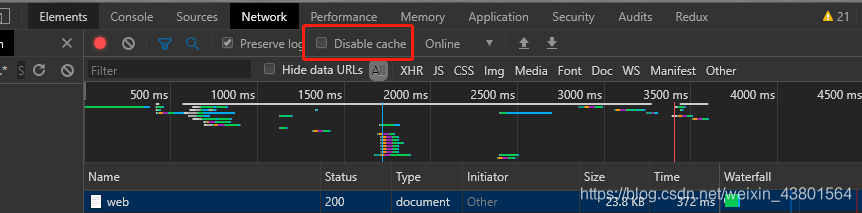
## **浏览器资源请求的流程**

* 1.**首次请求资源,请求服务器,服务器返回带有request header的资源,状态码200;**
* 2.**再次请求资源,判断资源过期时间,  
  资源未过期:命中强缓存,状态码200;**  
  **资源过期:向服务器发送请求;**
* **3.服务器根据request header中的资源标识(etag,last-modified),与服务器上的资源进行对应:  
  如果资源改变,带上新的资源标识返回,状态码200  
  如果资源未变,命中协商缓存,从浏览器缓存读取资源,状态码304**

### **用户对浏览器缓存机制的控制**

* 1.地址栏访问,按下回车,执行浏览器缓存机制;
* 2.F5刷新,跳过强缓存,向服务器发送请求,判断是否需要执行协商缓存;
* 3.Ctrl+F5,跳过强缓存和协商缓存,直接从服务器拉取资源.

### **注意**

****1.开启浏览器缓存需要打开这个开关****  


**HTTP的请求方式**

HTTP的几种请求方法及其用途

**GET 方法**

发送一个请求来取得服务器上的某一资源

**POST 方法**

向 URL 指定的资源提交数据或附加新的数据

**PUT 方法**

跟 POST 方法很像，也是想服务器提交数据。但是，它们之间有不同。 PUT 指定了资源

在服务器上的位置，而 POST 没有

**HEAD 方法**

只请求页面的首部

**DELETE 方法**

删除服务器上的某资源

**OPTIONS 方法**

它用于获取当前 URL 所支持的方法。如果请求成功，会有一个 Allow 的头包含类

似 “GET,POST” 这样的信息

**TRACE 方法**

TRACE 方法被用于激发一个远程的，应用层的请求消息回路

**CONNECT 方法**

把请求连接转换到透明的 TCP/IP 通道

**Post 和 Get 的区别**

Get 请求能缓存， Post 不能

Post 相对 Get 安全一点点，因为 Get 请求都包含在 URL 里，且会被浏览器保存历 史纪录， Post 不会，但是在抓包的情况下都是一样的。

Post 可以通过 request body 来传输比 Get 更多的数据， Get 没有这个技术

URL 有长度限制，会影响 Get 请求，但是这个长度限制是浏览器规定的，不是 RFC 规 定的

Post 支持更多的编码类型且不对数据类型限制

**HTTP常见状态码**

### **1XX：**信息状态码（**通知）**

1XX系列响应代码仅在与HTTP服务器沟通时使用。

**100("Continue")**  
重要程度：中等，但（写操作时）很少用。

这是对HTTP LBYL（look-before-you-leap）请求的一个可能的响应。该响应代码表明：客户端应重新发送初始请求，并在请求中附上第一次请求时未提供的（可能很大或者包含敏感信息的）表示。客户端这次发送的请求不会被拒绝。对LBYL请求的另一个可能的响应是417("Expectation Failed")。

请求报头：要做一个LBYL请求，客户端必须把Expect请求报头设为字符串"100-continue"。除此以外，客户端还需要设置其他一些报头，服务器将根据这些报头决定是响应100还是417。

100 Continue 继续，

一般在发送 post 请求时，已发送了 http header 之后服务端将 返回此信息，表示确认，之后发送具体参数信息

**101("Switching Protocols")**重要程度：非常低。

**当客户端通过在请求里使用Upgrade报头，以通知服务器它想改用除HTTP协议之外的其他协议时，客户端将获得此响应代码**。101响应代码表示“行，我现在改用另一个协议了”。通常HTTP客户端会在收到服务器发来的101响应后关闭与服务器的TCP连接。**101响应代码意味着，该客户端不再是一个HTTP客户端，而将成为另一种客户端。**

**2XX 成功**

**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**

205 Reset Content ，表示请求成功，但响应报文不含实体的主体部分，但是与 204 响

应不同在于要求请求方重置内容

**206 Partial Content ，表示客户端进行了范围请求**，**并且服务器成功执行了这部分的GET请求，响应报文中包含由Content-Range指定范围的实体内容。**

**3XX 重定向**

**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 方法丁香获取资源

**304 not modified ，表示服务器允许访问资源，但因发生请求未满足条件的情况**

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**

**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**

**HTTP request报文结构是怎样的**

1. 首行是Request-Line包括：请求方法，请求URI，协议版本，CRLF
2. 首行之后是若干行请求头，包括general-header，request-header或者entity-header，每 个一行以CRLF结束
3. 请求头和消息实体之间有一个CRLF分隔
4. 根据实际请求需要可能包含一个消息实体 一个请求报文例子如下：

**GET /Protocols/rfc2616/rfc2616-sec5.html HTTP/1.1**

**Host: www.w3.org**

**Connection: keep-alive**

**Cache-Control: max-age=0**

**Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8**

**User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Ge**

**Referer: https://www.google.com.hk/**

**Accept-Encoding: gzip,deflate,sdch**

**Accept-Language: zh-CN,zh;q=0.8,en;q=0.6**

**Cookie: authorstyle=yes**

**If-None-Match: "2cc8-3e3073913b100"**

**If-Modified-Since: Wed, 01 Sep 2004 13:24:52 GMT**

**name=qiu&age=25**



****常见HTTP请求报文头属性****

****Accept****  
请求报文可通过一个“Accept”报文头属性告诉服务端 客户端接受什么类型的响应。   
如下报文头相当于告诉服务端，俺客户端能够接受的响应类型仅为纯文本数据啊，你丫别发其它什么图片啊，视频啊过来，那样我会歇菜的~~~

Accept:text/plain

Accept属性的值可以为一个或多个MIME类型的值（描述消息内容类型的因特网标准， 消息能包含文本、图像、音频、视频以及其他应用程序专用的数据）

****cookie****

客户端的Cookie就是通过这个报文头属性传给服务端的哦！如下所示：

Cookie: $Version=1; Skin=**new**;jsessionid=5F4771183629C9834F8382E23

服务端是怎么知道客户端的多个请求是隶属于一个Session呢？注意到后台的那个jsessionid = 5F4771183629C9834F8382E23木有？原来就是通过HTTP请求报文头的Cookie属性的jsessionid的值关联起来的！（当然也可以通过重写URL的方式将会话ID附带在每个URL的后面哦）。

****Referer****  
表示这个请求是从哪个URL过来的，假如你通过google搜索出一个商家的广告页面，你对这个广告页面感兴趣，鼠标一点发送一个请求报文到商家的网站，这个请求报文的Referer报文头属性值就是http://www.google.com。

****Cache-Control****

对缓存进行控制，如一个请求希望响应返回的内容在客户端要被缓存一年，或不希望被缓存就可以通过这个报文头达到目的。

**HTTP response报文结构是怎样的**

首行是状态行包括：HTTP版本，状态码，状态描述，后面跟一个CRLF

首行之后是若干行响应头，包括：通用头部，响应头部，实体头部 响应头部和响应实体之间用一个CRLF空行分隔

最后是一个可能的消息实体 响应报文例子如下：

**HTTP/1.1 200 OK**

**Date: Tue, 08 Jul 2014 05:28:43 GMT**

**Server: Apache/2**

**Last-Modified: Wed, 01 Sep 2004 13:24:52 GMT**

**ETag: "40d7-3e3073913b100"**

**Accept-Ranges: bytes**

**Content-Length: 16599**

**Cache-Control: max-age=21600**

**Expires: Tue, 08 Jul 2014 11:28:43 GMT**

**P3P: policyref="http://www.w3.org/2001/05/P3P/p3p.xml"**

**Content-Type: text/html; charset=iso-8859-1**

**{"name": "qiu", "age": 25}**

****常见的HTTP响应报文头属性****

****Cache-Control****   
响应输出到客户端后，服务端通过该报文头属告诉客户端如何控制响应内容的缓存。

常见的取值有private、public、no-cache、max-age，no-store，默认为private。  
private:             客户端可以缓存  
public:              客户端和代理服务器都可缓存（前端的同学，可以认为public和private是一样的）  
max-age=xxx:   缓存的内容将在 xxx 秒后失效  
no-cache:          需要使用对比缓存来验证缓存数据  
no-store:           所有内容都不会缓存

默认为private，缓存时间为31536000秒（365天）也就是说，在365天内再次请求这条数据，都会直接获取缓存数据库中的数据，直接使用。

****ETag****   
一个代表响应服务端资源（如页面）版本的报文头属性，如果某个服务端资源发生变化了，这个ETag就会相应发生变化。它是Cache-Control的有益补充，可以让客户端“更智能”地处理什么时候要从服务端取资源，什么时候可以直接从缓存中返回响应。

****Location****   
我们在JSP中让页面Redirect到一个某个A页面中，其实是让客户端再发一个请求到A页面，这个需要Redirect到的A页面的URL，其实就是通过响应报文头的Location属性告知客户端的，如下的报文头属性，将使客户端redirect到iteye的首页中：

Location: http://www.iteye.com

****Set-Cookie****   
服务端可以设置客户端的Cookie，其原理就是通过这个响应报文头属性实现的：

Set-Cookie: UserID=JohnDoe; Max-Age=3600; Version=1

Content-Type（内容类型），一般是指网页中存在的 Content-Type，用于定义网络文件的类型和网页的编码，决定浏览器将以什么形式、什么编码读取这个文件，这就是经常看到一些 PHP 网页点击的结果却是下载一个文件或一张图片的原因。

Content-Type 标头告诉客户端实际返回的内容的内容类型。

语法格式：

Content-Type: text/html; charset=utf-8Content-Type: multipart/form-data; boundary=something

cookie机制：

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。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。

Cookie的maxAge决定着Cookie的有效期，单位为秒（Second）。Cookie中通过getMaxAge()方法与setMaxAge(int maxAge)方法来读写maxAge属性。

如果maxAge属性为正数，则表示该Cookie会在maxAge秒之后自动失效。

如果maxAge为负数，则表示该Cookie仅在本浏览器窗口以及本窗口打开的子窗口内有效，关闭窗口后该Cookie即失效。

如果maxAge为0，则表示删除该Cookie。

Cookie并不提供修改、删除操作。如果要修改某个Cookie，只需要新建一个同名的Cookie，添加到response中覆盖原来的Cookie。

如果要删除某个Cookie，只需要新建一个同名的Cookie，并将maxAge设置为0，并添加到response中覆盖原来的Cookie。

Cookie cookie = new Cookie("username","helloweenvsfei");   // 新建Cookie

cookie.setMaxAge(0);                          // 设置生命周期为0，不能为负数

response.addCookie(cookie);                    // 必须执行这一句 输出到客户端

**OSI参考模型----网络互联模型**

在通信主机上完成的功能：应用层，表示层，会话层，传输层

在网络设备上实现的功能：网络层，数据链路层，物理层

**OSI第七层：应用层功能及介绍**

OSI七层模型的顶层是第七层：应用层，Application Layer

OSI应用层功能：应用层提供各种各样的应用层协议，这些协议嵌入在各种我们使用的应用程序中，为用户与网络之间提供一个打交道的接口。

举几个OSI应用层与我们打交道的常见例子：  
****我们要看网页，打开IE浏览器，输入一个网址，就进入了相应的网站，那么这个IE浏览器就是我浏览网页的应用工具，也是工作在应用层的。IE浏览器是基于HTTP协议开发的，HTTP是一个应用层的协议。  
我们要使用FTP上传下载一个文件，会首先打开FTP客户端，然后去连接FTP服务器，这个客户端，就是应用层的工具。FTP客户端是基于FTP协议开发的，FTP协议也是一个应用层的协议。  
我要发一封E-mail，我会在电脑装一个Foxmail、Outlook等邮件客户端软件，然后编辑邮件，发送给相应的人。Outlook、Foxmail是基于SMTP和POP3协议开发的，SMTP、POP3也是应用层的协议。****

应用层的PDU为：APDU（应用层协议数据单元）

**TCP与UDP差异**

网络协议是每个前端工程师都必须要掌握的知识，TCP/IP 中有两个具有代表性的传输层协议，分别是 TCP 和 UDP，本文将介绍下这两者以及它们之间的区别。

### **一、TCP/IP网络模型**

TCP/IP模型是互联网的基础，它是一系列网络协议的总称。这些协议可以划分为四层，分别为链路层、网络层、传输层和应用层。

* 链路层：负责封装和解封装IP报文，发送和接受ARP/RARP报文等。
* 网络层：负责路由以及把分组报文发送给目标网络或主机。
* 传输层：负责对报文进行分组和重组，并以TCP或UDP协议格式封装报文。
* 应用层：负责向用户提供应用程序，比如HTTP、FTP、Telnet、DNS、SMTP等。



### **二、UDP**

UDP协议全称是用户数据报协议，在网络中它与TCP协议一样用于处理数据包，是一种无连接的协议。在OSI模型中，在第四层——传输层，处于IP协议的上一层。UDP有不提供数据包分组、组装和不能对数据包进行排序的缺点，也就是说，当报文发送之后，是无法得知其是否安全完整到达的。

它有以下几个特点：

#### **1. 面向无连接**

#### **2. 有单播，多播，广播的功能**

#### **3. UDP是面向报文的**

#### **4. 不可靠性**

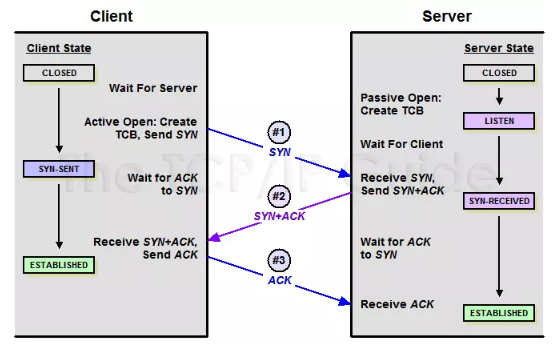
#### **5. 头部开销小，传输数据报文时是很高效的。**

### **三、TCP**

**TCP协议全称是传输控制协议是一种面向连接的、可靠的、基于字节流的传输层通信协议，**由 IETF 的RFC 793定义。TCP 是面向连接的、可靠的流协议。流就是指不间断的数据结构，你可以把它想象成排水管中的水流。

#### **TCP连接过程**

如下图所示，可以看到建立一个TCP连接的过程为（三次握手的过程）:



**第一次握手**

客户端向服务端发送连接请求报文段。该报文段中包含自身的数据通讯初始序号。请求发送后，客户端便进入 SYN-SENT 状态。

**第二次握手**

服务端收到连接请求报文段后，如果同意连接，则会发送一个应答，该应答中也会包含自身的数据通讯初始序号，发送完成后便进入 SYN-RECEIVED 状态。

**第三次握手**

当客户端收到连接同意的应答后，还要向服务端发送一个确认报文。客户端发完这个报文段后便进入 ESTABLISHED 状态，服务端收到这个应答后也进入 ESTABLISHED 状态，此时连接建立成功。

这里可能大家会有个疑惑：为什么 TCP 建立连接需要三次握手，而不是两次？这是因为这是为了防止出现失效的连接请求报文段被服务端接收的情况，从而产生错误。

#### **TCP断开链接**

**第一次握手**

若客户端 A 认为数据发送完成，则它需要向服务端 B 发送连接释放请求。

**第二次握手**

B 收到连接释放请求后，会告诉应用层要释放 TCP 链接。然后会发送 ACK 包，并进入 CLOSE\_WAIT 状态，此时表明 A 到 B 的连接已经释放，不再接收 A 发的数据了。但是因为 TCP 连接是双向的，所以 B 仍旧可以发送数据给 A。

**第三次握手**

B 如果此时还有没发完的数据会继续发送，完毕后会向 A 发送连接释放请求，然后 B 便进入 LAST-ACK 状态。

**第四次握手**

A 收到释放请求后，向 B 发送确认应答，此时 A 进入 TIME-WAIT 状态。该状态会持续 2MSL（最大段生存期，指报文段在网络中生存的时间，超时会被抛弃） 时间，若该时间段内没有 B 的重发请求的话，就进入 CLOSED 状态。当 B 收到确认应答后，也便进入 CLOSED 状态。