面试 HTTP，99% 的面试官都爱问这些问题

ImportNew 今天

以下文章来源于Java建设者，作者cxuan

（给ImportNew加星标，提高Java技能）

> 作者：cxuan

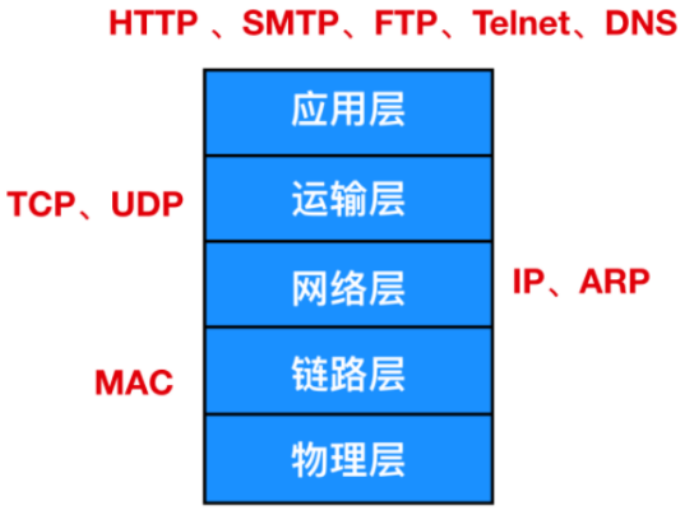
HTTP 和 HTTPS 的区别

HTTP 是一种 超文本传输协议(Hypertext Transfer Protocol)，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

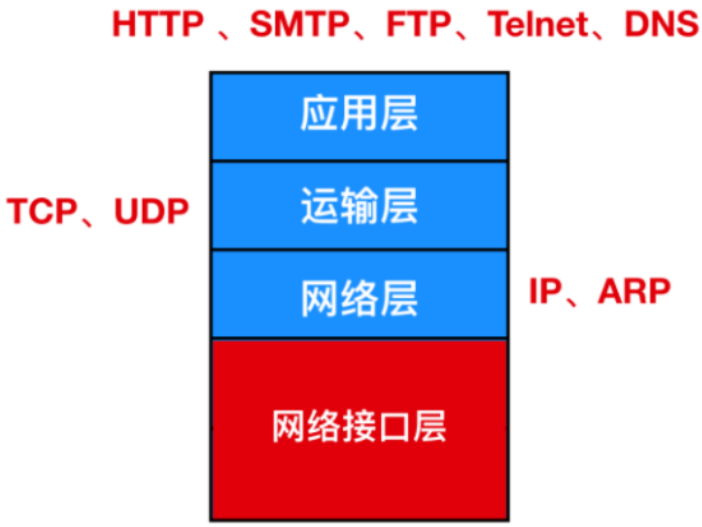
HTTP 主要内容分为三部分，超文本（Hypertext）、传输（Transfer）、协议（Protocol）。

- 超文本就是不单单只是本文，它还可以传输图片、音频、视频，甚至点击文字或图片能够进行超链接的跳转。
- 上面这些概念可以统称为数据，传输就是数据需要经过一系列的物理介质从一个端系统传送到另外一个端系统的过程。通常我们把传输数据包的一方称为请求方，把接到二进制数据包的一方称为应答方。
- 而协议指的就是是网络中(包括互联网)传递、管理信息的一些规范。如同人与人之间相互交流是需要遵循一定的规矩一样，计算机之间的相互通信需要共同遵守一定的规则，这些规则就称为协议，只不过是网络协议。

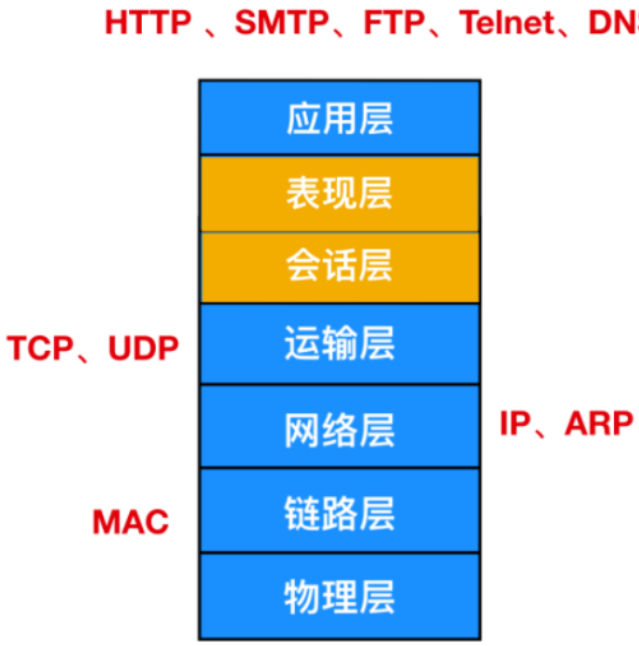
说到 HTTP，不得不提的就是 TCP/IP 网络模型，一般是五层模型。如下图所示



但是也可以分为四层，就是把链路层和物理层都表示为网络接口层

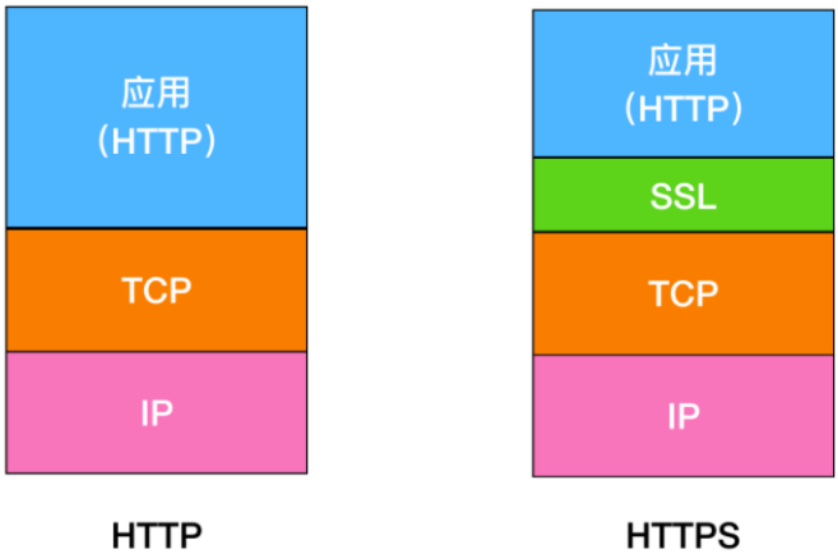


还有一种就是 OSI 七层网络模型，它就是在五层协议之上加了表示层和会话层



而 HTTPS 的全称是 Hypertext Transfer Protocol Secure，从名称我们可以看出 HTTPS 要比 HTTPS 多了 secure 安全性这个概念，实际上， HTTPS 并不是一个新的应用层协议，它其实就是 HTTP + TLS/SSL 协议组合而成，而安全性的保证正是 TLS/SSL 所做的工作。

也就是说，HTTPS 就是身披了一层 SSL 的 HTTP。



那么，HTTP 和 HTTPS 的主要区别是什么呢?

- 最简单的，HTTP 在地址栏上的协议是以 http:// 开头，而 HTTPS 在地址栏上的协议是以 https:// 开头

```
1  http://www.cxuanblog.com/
```

```
https://www.cxuanblog.com/
```

- HTTP 是未经安全加密的协议，它的传输过程容易被攻击者监听、数据容易被窃取、发送方和接收方容易被伪造；而 HTTPS 是安全的协议，它通过 密钥交换算法 - 签名算法 - 对称加密算法 - 摘要算法 能够解决上面这些问题。

- HTTP 的默认端口是 80，而 HTTPS 的默认端口是 443。

HTTP Get 和 Post 区别

HTTP 中包括许多方法，Get 和 Post 是 HTTP 中最常用的两个方法，基本上使用 HTTP 方法中有 99% 都是在使用 Get 方法和 Post 方法，所以有必要我们对这两个方法有更加深刻的认识。

- get 方法一般用于请求，比如你在浏览器地址栏输入 www.cxuanblog.com 其实就是发送了一个 get 请求，它的主要特征是请求服务器返回资源，而 post 方法一般用于

  表单的提交，相当于是把信息提交给服务器，等待服务器作出响应，get 相当于一个是 pull/拉的操作，而 post 相当于是一个 push/推的操作。

- get 方法是不安全的，因为你在发送请求的过程中，你的请求参数会拼在 URL 后面，从而导致容易被攻击者窃取，对你的信息造成破坏和伪造；

```
/test/demo_form.asp?name1=value1&name2=value2
```

而 post 方法是把参数放在请求体 body 中的，这对用户来说不可见。

```
POST /test/demo_form.asp HTTP/1.1
Host: w3schools.com
name1=value1&name2=value2
```

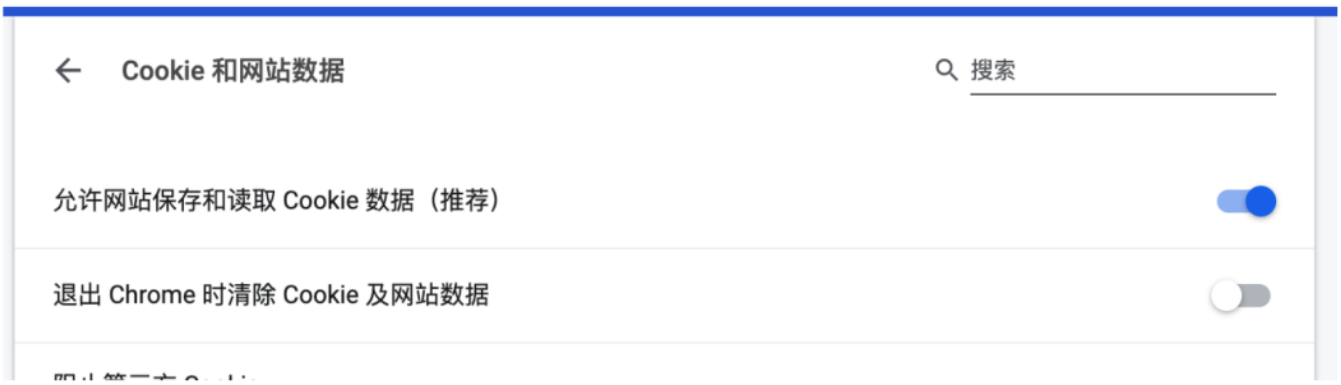
- get 请求的 URL 有长度限制，而 post 请求会把参数和值放在消息体中，对数据长度没有要求。
- get 请求会被浏览器主动 cache，而 post 不会，除非手动设置。
- get 请求在浏览器反复的 回退/前进 操作是无害的，而 post 操作会再次提交表单请求。
- get 请求在发送过程中会产生一个 TCP 数据包；post 在发送过程中会产生两个 TCP 数据包。对于 get 方式的请求，浏览器会把 http header 和 data 一并发送出去，服务器响应 200（返回数据）；而对于 post，浏览器先发送 header，服务器响应 100 continue，浏览器再发送 data，服务器响应 200 ok（返回数据）。

什么是无状态协议，HTTP 是无状态协议吗，怎么解决

无状态协议(Stateless Protocol) 就是指浏览器对于事务的处理没有记忆能力。举个例子来说就是比如客户请求获得网页之后关闭浏览器，然后再次启动浏览器，登录该网站，但是服务器并不知道客户关闭了一次浏览器。

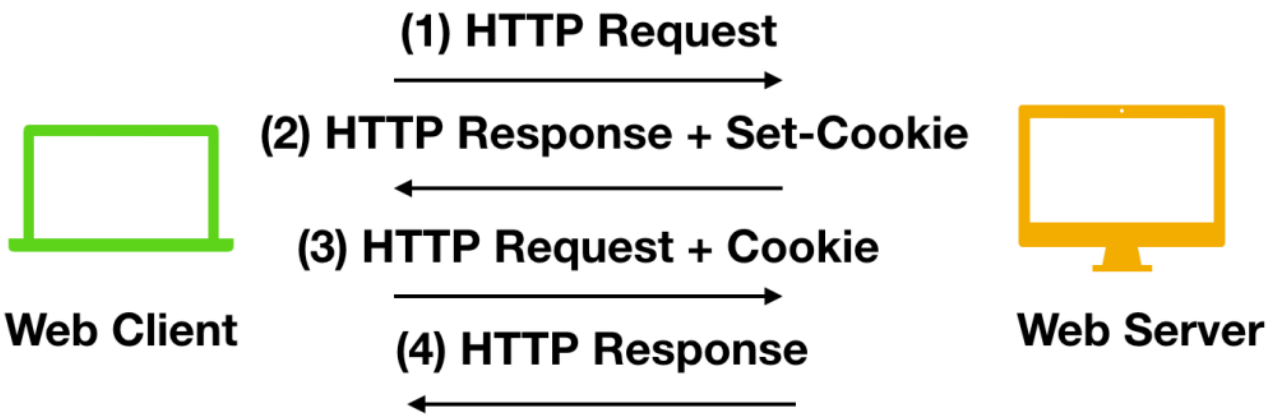
HTTP 就是一种无状态的协议，他对用户的操作没有记忆能力。可能大多数用户不相信，他可能觉得每次输入用户名和密码登陆一个网站后，下次登陆就不再重新输入用户名和密码了。这其实不是 HTTP 做的事情，起作用的是一个叫做 小甜饼(Cookie) 的机制。它能够让浏览器具有记忆能力。

如果你的浏览器允许 cookie 的话，查看方式 chrome://settings/content/cookies

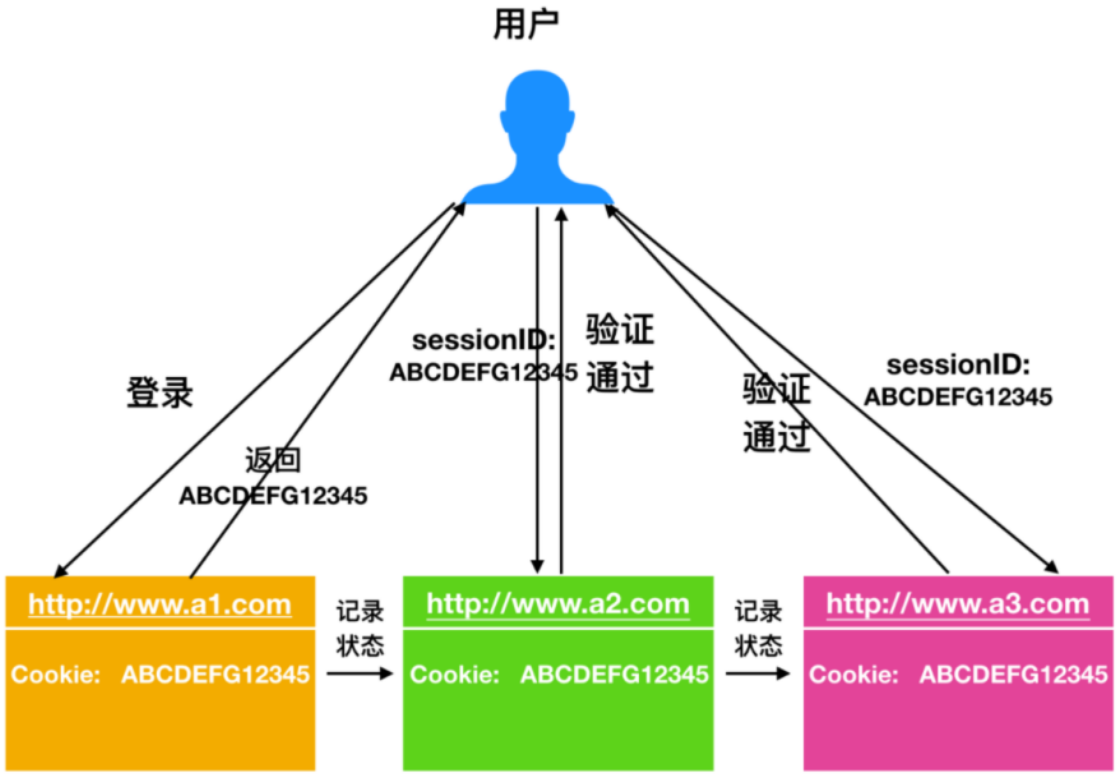


也就说明你的记忆芯片通电了..... 当你向服务端发送请求时，服务端会给你发送一个认证信息，服务器第一次接收到请求时，开辟了一块 Session 空间（创建了Session对象），同时生成一个 sessionId，并通过响应头的 Set-Cookie：JSESSIONID=XXXXXXX 命令，向客户端发送要求设置 Cookie 的响应；客户

端收到响应后，在本机客户端设置了一个 JSESSIONID=XXXXXXX 的 Cookie 信息，该 Cookie 的过期时间为浏览器会话结束；



接下来客户端每次向同一个网站发送请求时，请求头都会带上该 Cookie信息（包含 sessionId ）， 然后，服务器通过读取请求头中的 Cookie 信息，获取名称为 JSESSIONID 的值，得到此次请求的 sessionId。这样，你的浏览器才具有了记忆能力。



还有一种方式是使用 JWT 机制，它也是能够让你的浏览器具有记忆能力的一种机制。与 Cookie 不同，JWT 是保存在客户端的信息，它广泛的应用于单点登录的情况。JWT 具有两个特点

- JWT 的 Cookie 信息存储在客户端，而不是服务端内存中。也就是说，JWT 直接本地进行验证就可以，验证完毕后，这个 Token 就会在 Session 中随请求一起发送到服务器，通过这种方式，可以节省服务器资源，并且 token 可以进行多次验证。

- JWT 支持跨域认证，Cookies 只能用在单个节点的域或者它的子域中有效。如果它们尝试通过第三个节点访问，就会被禁止。使用 JWT 可以解决这个问题，使用 JWT 能够通过多个节点进行用户认证，也就是我们常说的跨域认证。

UDP 和 TCP 的区别

TCP 和 UDP 都位于计算机网络模型中的运输层，它们负责传输应用层产生的数据。下面我们就来聊一聊 TCP 和 UDP 分别的特征和他们的区别

UDP 是什么

UDP 的全称是 User Datagram Protocol，用户数据报协议。它不需要所谓的握手操作，从而加快了通信速度，允许网络上的其他主机在接收方同意通信之前进行数据传输。

数据报是与分组交换网络关联的传输单元。

UDP 的特点主要有

- UDP 能够支持容忍数据包丢失的带宽密集型应用程序
- UDP 具有低延迟的特点
- UDP 能够发送大量的数据包
- UDP 能够允许 DNS 查找，DNS 是建立在 UDP 之上的应用层协议。

TCP 是什么

TCP 的全称是Transmission Control Protocol，传输控制协议。它能够帮助你确定计算机连接到 Internet 以及它们之间的数据传输。通过三次握手来建立 TCP 连接，三次握手就是用来启动和确认 TCP 连接的过程。一旦连接建立后，就可以发送数据了，当数据传输完成后，会通过关闭虚拟电路来断开连接。

TCP 的主要特点有

- TCP 能够确保连接的建立和数据包的发送
- TCP 支持错误重传机制
- TCP 支持拥塞控制，能够在网络拥堵的情况下延迟发送

- TCP 能够提供错误校验和，甄别有害的数据包。

TCP 和 UDP 的不同

下面为你罗列了一些 TCP 和 UDP 的不同点，方便理解，方便记忆。

TCP	UDP
TCP 是面向连接的协议	UDP 是无连接的协议
TCP 在发送数据前先需要建立连接，然后再发送数据	UDP 无需建立连接就可以直接发送大量数据
TCP 会按照特定顺序重新排列数据包	UDP 数据包没有固定顺序，所有数据包都相互独立
TCP 传输的速度比较慢	UDP 的传输会更快
TCP 的头部字节有 20 字节	UDP 的头部字节只需要 8 个字节
TCP 是重量级的，在发送任何用户数据之前，TCP需要三次握手建立连接。	UDP 是轻量级的。没有跟踪连接，消息排序等。
TCP 会进行错误校验，并能够进行错误恢复	UDP 也会错误检查，但会丢弃错误的数据包。
TCP 有发送确认	UDP 没有发送确认
TCP 会使用握手协议，例如 SYN，SYN-ACK，ACK	无握手协议
TCP 是可靠的，因为它可以确保将数据传送到路由器。	在 UDP 中不能保证将数据传送到目标。

TCP 三次握手和四次挥手

TCP 三次握手和四次挥手也是面试题的热门考点，它们分别对应 TCP 的连接和释放过程。下面就来简单认识一下这两个过程

TCP 三次握手

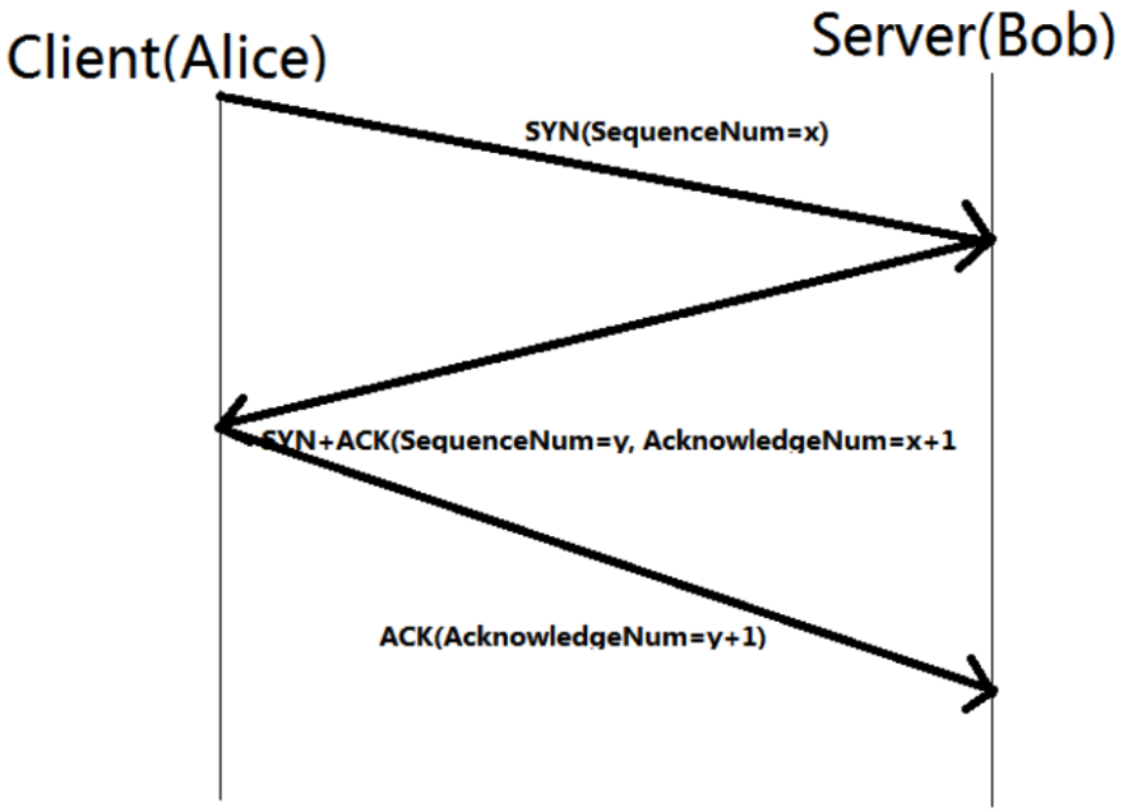
在了解具体的流程前，我们需要先认识几个概念

消息类型	描述
SYN	这个消息是用来初始化和建立连接的。
ACK	帮助对方确认收到的 SYN 消息
SYN-ACK	本地的 SYN 消息和较早的 ACK 数据包
FIN	用来断开连接

- SYN：它的全称是 Synchronize Sequence Numbers，同步序列编号。是 TCP/IP 建立连接时使用的握手信号。在客户机和服务器之间建立 TCP 连接时，首先会发送的一个信号。客户端在接受到 SYN 消息时，就会在自己的段内生成一个随机值 X。

- SYN-ACK：服务器收到 SYN 后，打开客户端连接，发送一个 SYN-ACK 作为答复。确认号设置为比接收到的序列号多一个，即 X + 1，服务器为数据包选择的序列号是另一个随机数 Y。

- ACK： Acknowledge character, 确认字符，表示发来的数据已确认接收无误。最后，客户端将 ACK 发送给服务器。序列号被设置为所接收的确认值即 Y + 1。



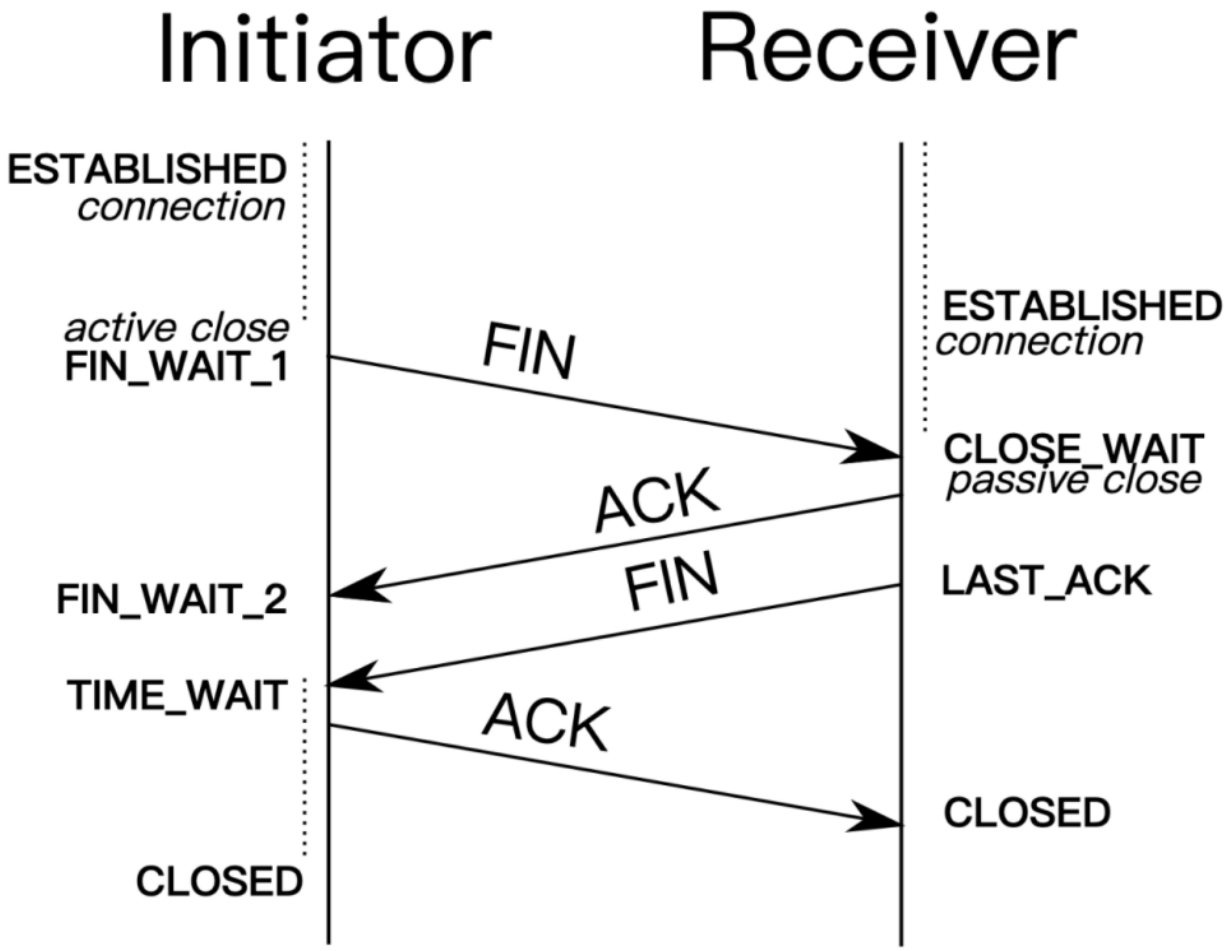
如果用现实生活来举例的话就是

小明 - 客户端 小红 - 服务端

- 小明给小红打电话，接通了后，小明说喂，能听到吗，这就相当于是连接建立。
- 小红给小明回应，能听到，你能听到我说的话吗，这就相当于是请求响应。
- 小明听到小红的回应后，好的，这相当于是连接确认。在这之后小明和小红就可以通话/交换信息了。

TCP 四次挥手

在连接终止阶段使用四次挥手，连接的每一端都会独立的终止。下面我们来描述一下这个过程。



- 首先，客户端应用程序决定要终止连接(这里服务端也可以选择断开连接)。这会使客户端将 FIN 发送到服务器，并进入 FIN_WAIT_1 状态。当客户端处于 FIN_WAIT_1 状态时，它会等待来自服务器的 ACK 响应。
- 然后第二步，当服务器收到 FIN 消息时，服务器会立刻向客户端发送 ACK 确认消息。
- 当客户端收到服务器发送的 ACK 响应后，客户端就进入 FIN_WAIT_2 状态，然后等待来自服务器的 FIN 消息
- 服务器发送 ACK 确认消息后，一段时间（可以进行关闭后）会发送 FIN 消息给客户端，告知客户端可以进行关闭。

- 当客户端收到从服务端发送的 FIN 消息时，客户端就会由 FIN_WAIT_2 状态变为 TIME_WAIT 状态。处于 TIME_WAIT 状态的客户端允许重新发送 ACK 到服务器为了防止信息丢失。客户端在 TIME_WAIT 状态下花费的时间取决于它的实现，在等待一段时间后，连接关闭，客户端上所有的资源（包括端口号和缓冲区数据）都被释放。

还是可以用上面那个通话的例子来进行描述

- 小明对小红说，我所有的东西都说完了，我要挂电话了。
- 小红说，收到，我这边还有一些东西没说。
- 经过若干秒后，小红也说完了，小红说，我说完了，现在可以挂断了
- 小明收到消息后，又等了若干时间后，挂断了电话。

简述 HTTP1.0/1.1/2.0 的区别

HTTP 1.0

HTTP 1.0 是在 1996 年引入的，从那时开始，它的普及率就达到了惊人的效果。

- HTTP 1.0 仅仅提供了最基本的认证，这时候用户名和密码还未经加密，因此很容易收到窥探。
- HTTP 1.0 被设计用来使用短链接，即每次发送数据都会经过 TCP 的三次握手和四次挥手，效率比较低。
- HTTP 1.0 只使用 header 中的 If-Modified-Since 和 Expires 作为缓存失效的标准。
- HTTP 1.0 不支持断点续传，也就是说，每次都会传送全部的页面和数据。
- HTTP 1.0 认为每台计算机只能绑定一个 IP，所以请求消息中的 URL 并没有传递主机名（hostname）。

HTTP 1.1

HTTP 1.1 是 HTTP 1.0 开发三年后出现的，也就是 1999 年，它做出了以下方面的变化

- HTTP 1.1 使用了摘要算法来进行身份验证
- HTTP 1.1 默认使用长连接，长连接就是只需一次建立就可以传输多次数据，传输完成后，只需要一次切断连接即可。长连接的连接时长可以通过请求头中的 keep-alive 来设置

- HTTP 1.1 中新增加了 E-tag，If-Unmodified-Since, If-Match, If-None-Match 等缓存控制标头来控制缓存失效。

- HTTP 1.1 支持断点续传，通过使用请求头中的 Range 来实现。

- HTTP 1.1 使用了虚拟网络，在一台物理服务器上可以存在多个虚拟主机（Multi-homed Web Servers），并且它们共享一个IP地址。

HTTP 2.0

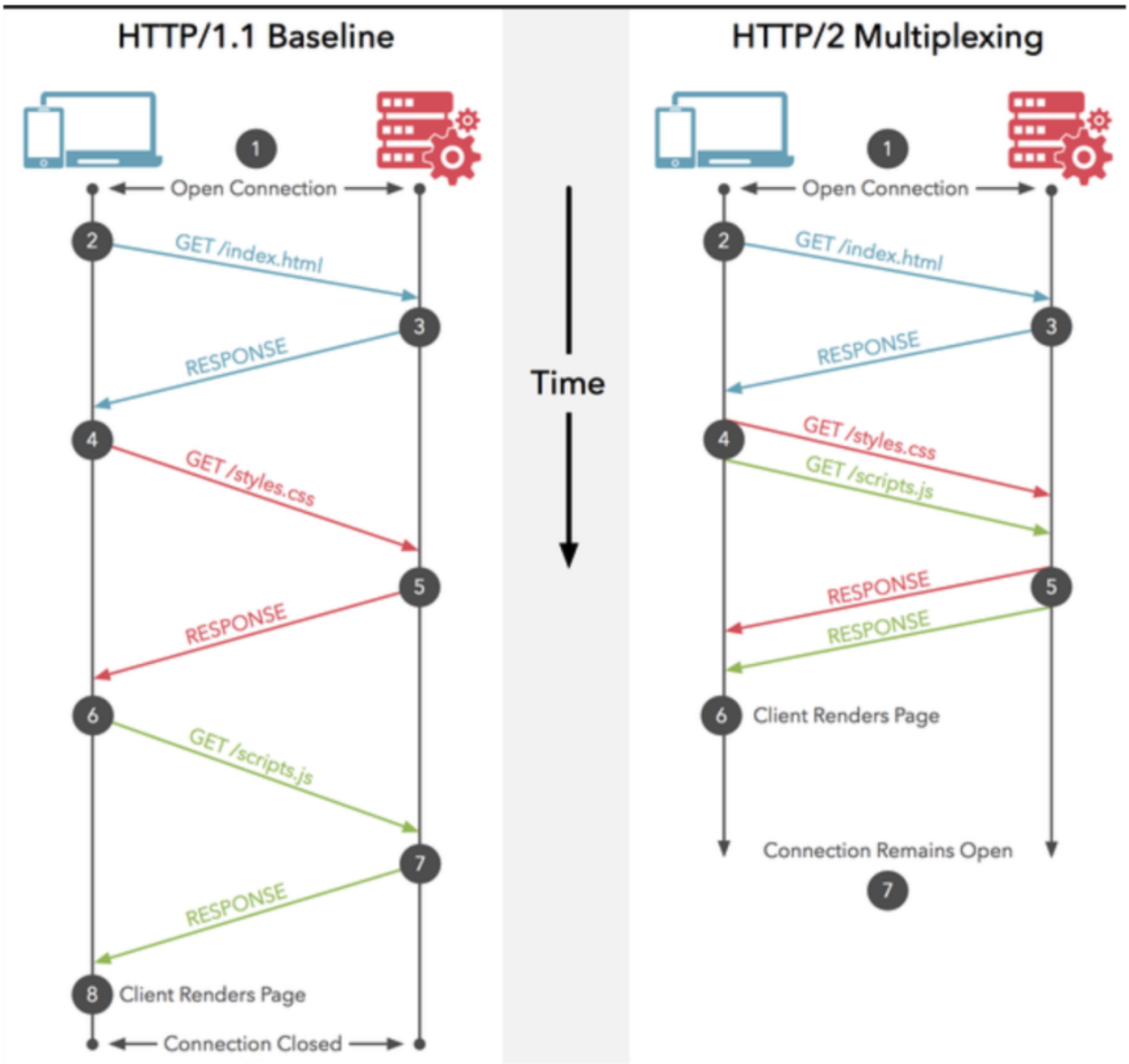
HTTP 2.0 是 2015 年开发出来的标准，它主要做的改变如下

- 头部压缩，由于 HTTP 1.1 经常会出现 User-Agent、Cookie、Accept、Server、Range 等字段可能会占用几百甚至几千字节，而 Body 却经常只有几十字节，所以导致头部偏重。HTTP 2.0 使用 HPACK 算法进行压缩。

- 二进制格式，HTTP 2.0 使用了更加靠近 TCP/IP 的二进制格式，而抛弃了 ASCII 码，提升了解析效率

- 强化安全，由于安全已经成为重中之重，所以 HTTP2.0 一般都跑在 HTTPS 上。

- 多路复用，即每一个请求都是是用作连接共享。一个请求对应一个id，这样一个连接上可以有多个请求。



请你说一下 HTTP 常见的请求头

这个问题比较开放，因为 HTTP 请求头有很多，这里只简单举出几个例子。

HTTP 标头会分为四种，分别是 通用标头、实体标头、请求标头、响应标头。分别介绍一下

通用标头

通用标头主要有三个，分别是 Date、Cache-Control 和 Connection

Date

Date 是一个通用标头，它可以出现在请求标头和响应标头中，它的基本表示如下

```
Date: Wed, 21 Oct 2015 07:28:00 GMT
```

表示的是格林威治标准时间，这个时间要比北京时间慢八个小时

Cache-Control

Cache-Control 是一个通用标头，他可以出现在请求标头和响应标头中，Cache-Control 的种类比较多，虽然说这是一个通用标头，但是有一些特性是请求标头具有的，有一些是响应标头才有的。主要大类有 可缓存性、阈值性、 重新验证并重新加载 和其他特性

Connection

Connection 决定当前事务（一次三次握手和四次挥手）完成后，是否会关闭网络连接。Connection 有两种，一种是持久性连接，即一次事务完成后不关闭网络连接

```
Connection: keep-alive
```

另一种是非持久性连接，即一次事务完成后关闭网络连接

```
Connection: close
```

HTTP1.1 其他通用标头如下

首部字段名	说明
Cache-Control	控制缓存的行为
Connection	逐跳首部、连接的管理
Date	创建报文的日期时间
Pragma	报文指令
Trailer	报文末端的首部一览
Transfer-Encoding	指定报文主体的传输编码方式
Upgrade	升级为其他协议
Via	代理服务器的相关信息
Warning	错误通知

实体标头

实体标头是描述消息正文内容的 HTTP 标头。实体标头用于 HTTP 请求和响应中。头部Content-Length、 Content-Language、 Content-Encoding 是实体头。

- Content-Length 实体报头指示实体主体的大小，以字节为单位，发送到接收方。
- Content-Language 实体报头描述了客户端或者服务端能够接受的语言。
- Content-Encoding 这又是一个比较麻烦的属性，这个实体报头用来压缩媒体类型。Content-Encoding 指示对实体应用了何种编码。

  常见的内容编码有这几种：gzip、compress、deflate、identity ，这个属性可以应用在请求报文和响应报文中

```
Accept-Encoding: gzip, deflate //请求头
Content-Encoding: gzip  //响应头
```

下面是一些实体标头字段

首部字段名	说明
Allow	资源可支持的HTTP方法
Content-Encoding	实体主体适用的编码方式
Content-Language	实体主体的自然语言
Content-Length	实体主体的大小（单位：字节）
Content-Location	替代对应资源的URI
Content-MD5	实体主体的报文摘要
Content-Range	实体主体的位置范围
Content-Type	实体主体的媒体类型
Expires	实体主体过期的日期时间
Last-Modified	资源的最后修改日期时间

请求标头

Host

Host 请求头指明了服务器的域名（对于虚拟主机来说），以及（可选的）服务器监听的 TCP 端口号。如果没有给定端口号，会自动使用被请求服务的默认端口（比如请求一个 HTTP 的 URL 会自动使用 80 作为端口）。

```
Host: developer.mozilla.org
```

上面的 Accpet、 Accept-Language、Accept-Encoding 都是属于内容协商的请求标头。

Referer

HTTP Referer 属性是请求标头的一部分，当浏览器向 web 服务器发送请求的时候，一般会带上 Referer，告诉服务器该网页是从哪个页面链接过来的，服务器因此可以获得一些信息用于处理。

```
Referer: https://developer.mozilla.org/testpage.html
```

If-Modified-Since

If-Modified-Since 通常会与 If-None-Match 搭配使用，If-Modified-Since 用于确认代理或客户端拥有的本地资源的有效性。获取资源的更新日期时间，可通过确认首部字段 Last-Modified 来确定。

大白话说就是如果在 Last-Modified 之后更新了服务器资源，那么服务器会响应 200，如果在 Last-Modified 之后没有更新过资源，则返回 304。

```
If-Modified-Since: Mon, 18 Jul 2016 02:36:04 GMT
```

If-None-Match

If-None-Match HTTP 请求标头使请求成为条件请求。对于 GET 和 HEAD 方法，仅当服务器没有与给定资源匹配的 ETag 时，服务器才会以 200 状态发送回请求的资源。对于其他方法，仅当最终现有资源的 ETag与列出的任何值都不匹配时，才会处理请求。

```
If-None-Match: "c561c68d0ba92bbeb8b0fff2a9199f722e3a621a"
```

Accept

接受请求 HTTP 标头会通告客户端其能够理解的 MIME 类型

Accept-Charset

accept-charset 属性规定服务器处理表单数据所接受的字符集。

常用的字符集有：UTF-8 - Unicode 字符编码 ； ISO-8859-1 - 拉丁字母表的字符编码
Accept-Language

首部字段 Accept-Language 用来告知服务器用户代理能够处理的自然语言集（指中文或英文等），以及自然语言集的相对优先级。可一次指定多种自然语言集。

请求标头我们大概就介绍这几种，后面会有一篇文章详细深挖所有的响应头的，下面是一个响应头的汇总，基于 HTTP 1.1

首部字段名	说明
Accept	用户代理可处理的媒体类型
Accept-Charset	优先的字符集
Accept-Encoding	优先的内容编码
Accept-Language	优先的语言（自然语言）
Authorization	Web认证信息
Expect	期待服务器的特定行为
From	用户的电子邮箱地址
Host	请求资源所在服务器
If-Match	比较实体标记（ETag）
If-Modified-Since	比较资源的更新时间
If-None-Match	比较实体标记（与 If-Match 相反）
If-Range	资源未更新时发送实体 Byte 的范围请求
If-Unmodified-Since	比较资源的更新时间（与If-Modified-Since相反）
Max-Forwards	最大传输逐跳数
Proxy-Authorization	代理服务器要求客户端的认证信息
Range	实体的字节范围请求
Referer	对请求中 URI 的原始获取方
TE	传输编码的优先级
User-Agent	HTTP 客户端程序的信息

响应标头

Access-Control-Allow-Origin

一个返回的 HTTP 标头可能会具有 Access-Control-Allow-Origin ， Access-Control-Allow-Origin 指定一个来源，它告诉浏览器允许该来源进行资源访问。

Keep-Alive

Keep-Alive 表示的是 Connection 非持续连接的存活时间，可以进行指定。

Server

服务器标头包含有关原始服务器用来处理请求的软件的信息。

应该避免使用过于冗长和详细的 Server 值，因为它们可能会泄露内部实施细节，这可能会使攻击者容易地发现并利用已知的安全漏洞。例如下面这种写法

```
Server: Apache/2.4.1 (Unix)
```

Set-Cookie

Set-Cookie 用于服务器向客户端发送 sessionID。

Transfer-Encoding

首部字段 Transfer-Encoding 规定了传输报文主体时采用的编码方式。

HTTP /1.1 的传输编码方式仅对分块传输编码有效。

X-Frame-Options

HTTP 首部字段是可以自行扩展的。所以在 Web 服务器和浏览器的应用上，会出现各种非标准的首部字段。

首部字段 X-Frame-Options 属于 HTTP 响应首部，用于控制网站内容在其他 Web 网站的 Frame 标签内的显示问题。其主要目的是为了防止点击劫持（clickjacking）攻击。

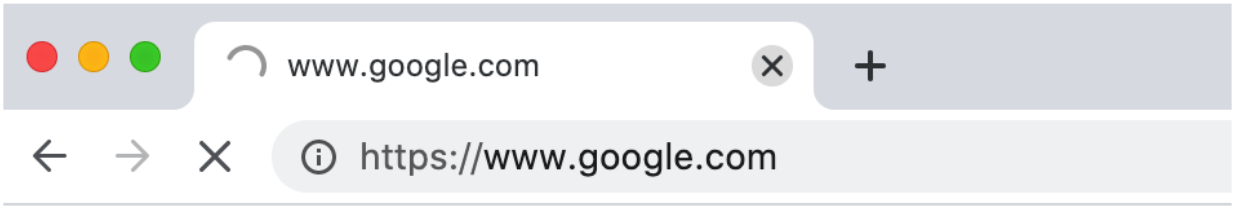
下面是一个响应头的汇总，基于 HTTP 1.1

首部字段名	说明
Accept-Ranges	是否接受字节范围请求
Age	推算资源创建经过时间
ETag	资源的匹配信息
Location	令客户端重定向至指定URI
Proxy-Authenticate	代理服务器对客户端的认证信息
Retry-After	对再次发起请求的时机要求
Server	HTTP服务器的安装信息
Vary	代理服务器缓存的管理信息
WWW-Authenticate	服务器对客户端的认证信息

地址栏输入 URL 发生了什么

这道题也是一道经常会考的面试题。那么下面我们就来探讨一下从你输入 URL 后到响应，都经历了哪些过程。

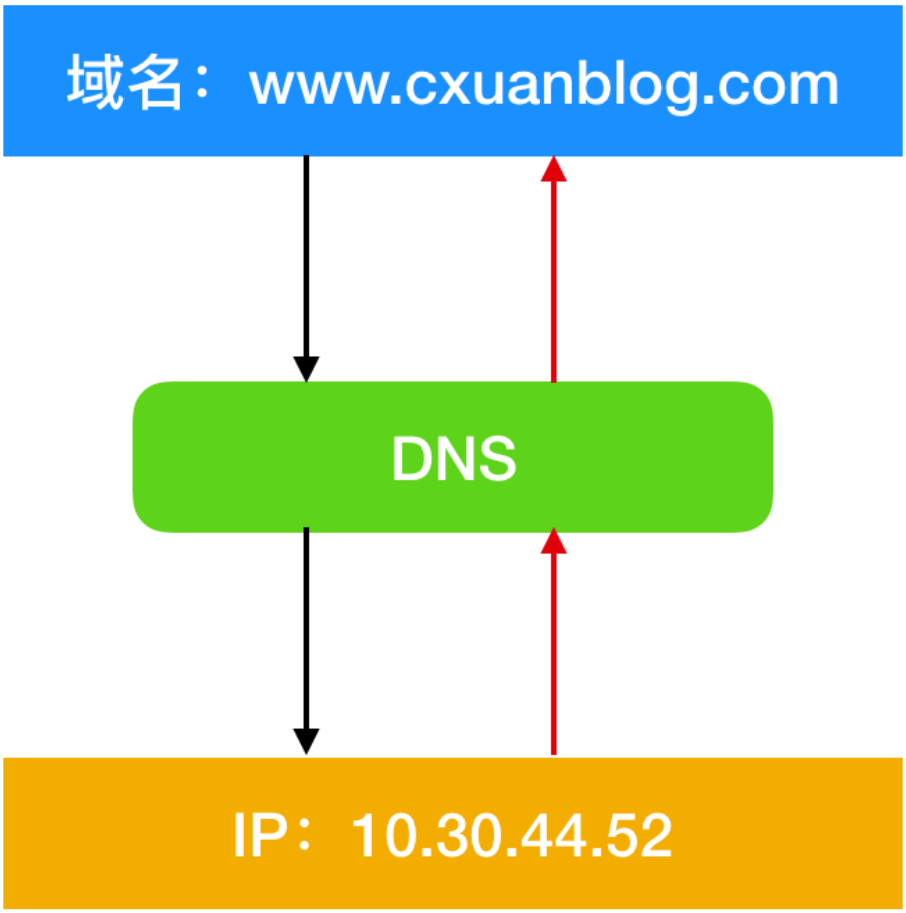
- 首先，你需要在浏览器中的 URL 地址上，输入你想访问的地址，如下



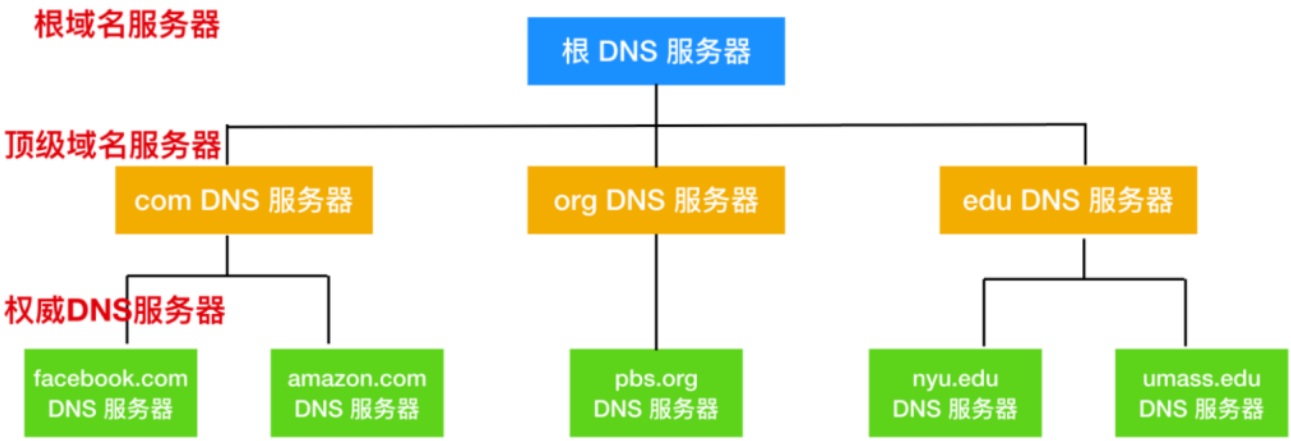
你应该访问不到的，对不对~

- 然后，浏览器会根据你输入的 URL 地址，去查找域名是否被本地 DNS 缓存，不同浏览器对 DNS 的设置不同，如果浏览器缓存了你想访问的 URL 地址，那就直接返回 ip。如果没有缓存你的 URL 地址，浏览器就会发起系统调用来查询本机 hosts 文件是否有配置 ip 地址，如果找到，直接返回。如果找不到，就向网络中发起一个 DNS 查询。

- 首先来看一下 DNS 是啥，互联网中识别主机的方式有两种，通过主机名和 IP 地址。我们人喜欢用名字的方式进行记忆，但是通信链路中的路由却喜欢定长、有层次结构的 IP 地址。所以就需要一种能够把主机名到 IP 地址的转换服务，这种服务就是由 DNS 提供的。DNS 的全称是 Domain Name System 域名系统。DNS 是一种由分层的 DNS 服务器实现的分布式数据库。DNS 运行在 UDP 上，使用 53 端口。



DNS 是一种分层数据库，它的主要层次结构如下



一般域名服务器的层次结构主要是以上三种，除此之外，还有另一类重要的 DNS 服务器，它是 本地 DNS 服务器(local DNS server)。严格来说，本地 DNS 服务器并不属于上述层次结构，但是本地 DNS 服务器又是至关重要的。每个 ISP(Internet Service Provider) 比如居民区的 ISP 或者一个机构的 ISP 都有一台本地 DNS 服务器。当主机和 ISP 进行连接时，该 ISP 会提供一台主机的 IP 地址，该主机会具有一台或多台其本地 DNS 服务器的 IP地址。通过访问网络连接，用户能够容易的确定 DNS 服务器的 IP地址。当主机发出 DNS 请求后，该请求被发往本地 DNS 服务器，它起着代理的作用，并将该请求转发到 DNS 服务器层次系统中。

首先，查询请求会先找到本地 DNS 服务器来查询是否包含 IP 地址，如果本地 DNS 无法查询到目标 IP 地址，就会向根域名服务器发起一个 DNS 查询。

注意：DNS 涉及两种查询方式：一种是递归查询(Recursive query)，一种是迭代查询(Iteration query)。《计算机网络：自顶向下方法》竟然没有给出递归查询和迭代查询的区别，找了一下网上的资料大概明白了下。

如果根域名服务器无法告知本地 DNS 服务器下一步需要访问哪个顶级域名服务器，就会使用递归查询；

如果根域名服务器能够告知 DNS 服务器下一步需要访问的顶级域名服务器，就会使用迭代查询。

在由根域名服务器 -> 顶级域名服务器 -> 权威 DNS 服务器后，由权威服务器告诉本地服务器目标 IP 地址，再有本地 DNS 服务器告诉用户需要访问的 IP 地址。

- 第三步，浏览器需要和目标服务器建立 TCP 连接，需要经过三次握手的过程，具体的握手过程请参考上面的回答。

- 在建立连接后，浏览器会向目标服务器发起 HTTP-GET 请求，包括其中的 URL，HTTP 1.1 后默认使用长连接，只需要一次握手即可多次传输数据。

- 如果目标服务器只是一个简单的页面，就会直接返回。但是对于某些大型网站的站点，往往不会直接返回主机名所在的页面，而会直接重定向。返回的状态码就不是 200，而是 301,302 以 3 开头的重定向码，浏览器在获取了重定向响应后，在响应报文中 Location 项找到重定向地址，浏览器重新第一步访问即可。

- 然后浏览器重新发送请求，携带新的 URL，返回状态码 200 OK，表示服务器可以响应请求，返回报文。

HTTPS 的工作原理

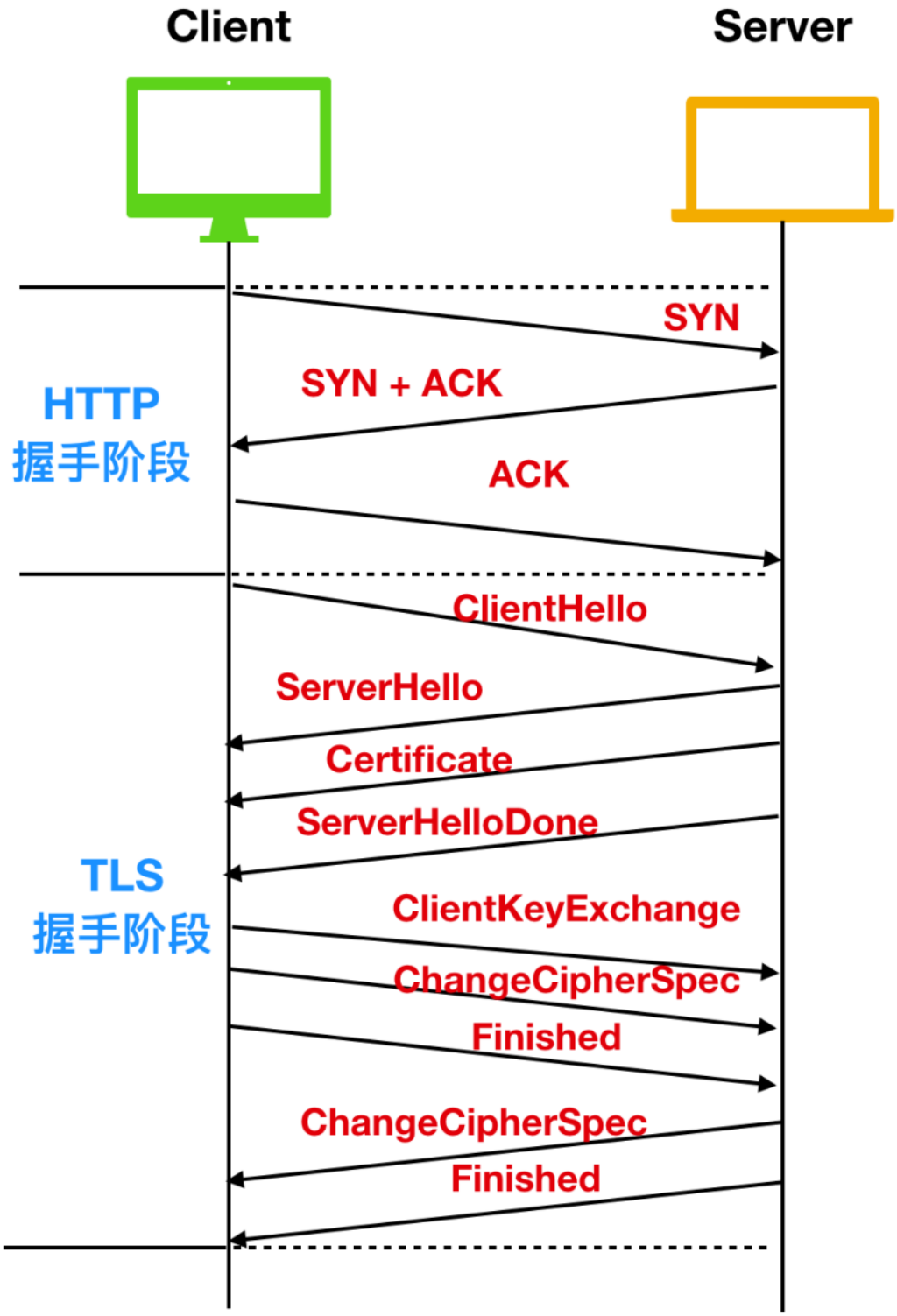
我们上面描述了一下 HTTP 的工作原理，下面来讲述一下 HTTPS 的工作原理。因为我们知道 HTTPS 不是一种新出现的协议，而是

所以，我们探讨 HTTPS 的握手过程，其实就是 SSL/TLS 的握手过程。

TLS 旨在为 Internet 提供通信安全的加密协议。TLS 握手是启动和使用 TLS 加密的通信会话的过程。在 TLS 握手期间，Internet 中的通信双方会彼此交换信息，验证密码套件，交换会话密钥。

每当用户通过 HTTPS 导航到具体的网站并发送请求时，就会进行 TLS 握手。除此之外，每当其他任何通信使用HTTPS（包括 API 调用和在 HTTPS 上查询 DNS）时，也会发生 TLS 握手。

TLS 具体的握手过程会根据所使用的密钥交换算法的类型和双方支持的密码套件而不同。我们以RSA 非对称加密来讨论这个过程。整个 TLS 通信流程图如下



- 在进行通信前，首先会进行 HTTP 的三次握手，握手完成后，再进行 TLS 的握手过程
- ClientHello：客户端通过向服务器发送 hello 消息来发起握手过程。这个消息中会夹带着客户端支持的 TLS 版本号(TLS1.0 、TLS1.2、TLS1.3) 、客户端支持的密码套件、以及一串 客户端随机数。
- ServerHello：在客户端发送 hello 消息后，服务器会发送一条消息，这条消息包含了服务器的 SSL 证书、服务器选择的密码套件和服务器生成的随机数。
- 认证(Authentication)：客户端的证书颁发机构会认证 SSL 证书，然后发送 Certificate 报文，报文中包含公开密钥证书。最后服务器发送 ServerHelloDone 作为 hello 请求的响应。第一部分握手阶段结束。

- 加密阶段：在第一个阶段握手完成后，客户端会发送 ClientKeyExchange 作为响应，这个响应中包含了一种称为 The premaster secret 的密钥字符串，这个字符串就是使用上面公开密钥证书进行加密的字符串。随后客户端会发送 ChangeCipherSpec，告诉服务端使用私钥解密这个 premaster secret 的字符串，然后客户端发送 Finished 告诉服务端自己发送完成了。

  Session key 其实就是用公钥证书加密的公钥。

- 实现了安全的非对称加密：然后，服务器再发送 ChangeCipherSpec 和 Finished 告诉客户端解密完成，至此实现了 RSA 的非对称加密。

推荐阅读 点击标题可跳转

一文详细解读 Dubbo 中的 http 协议

HttpClient 连接池设置引发的一次雪崩

简单的 HTTP 调用，为什么时延这么大？