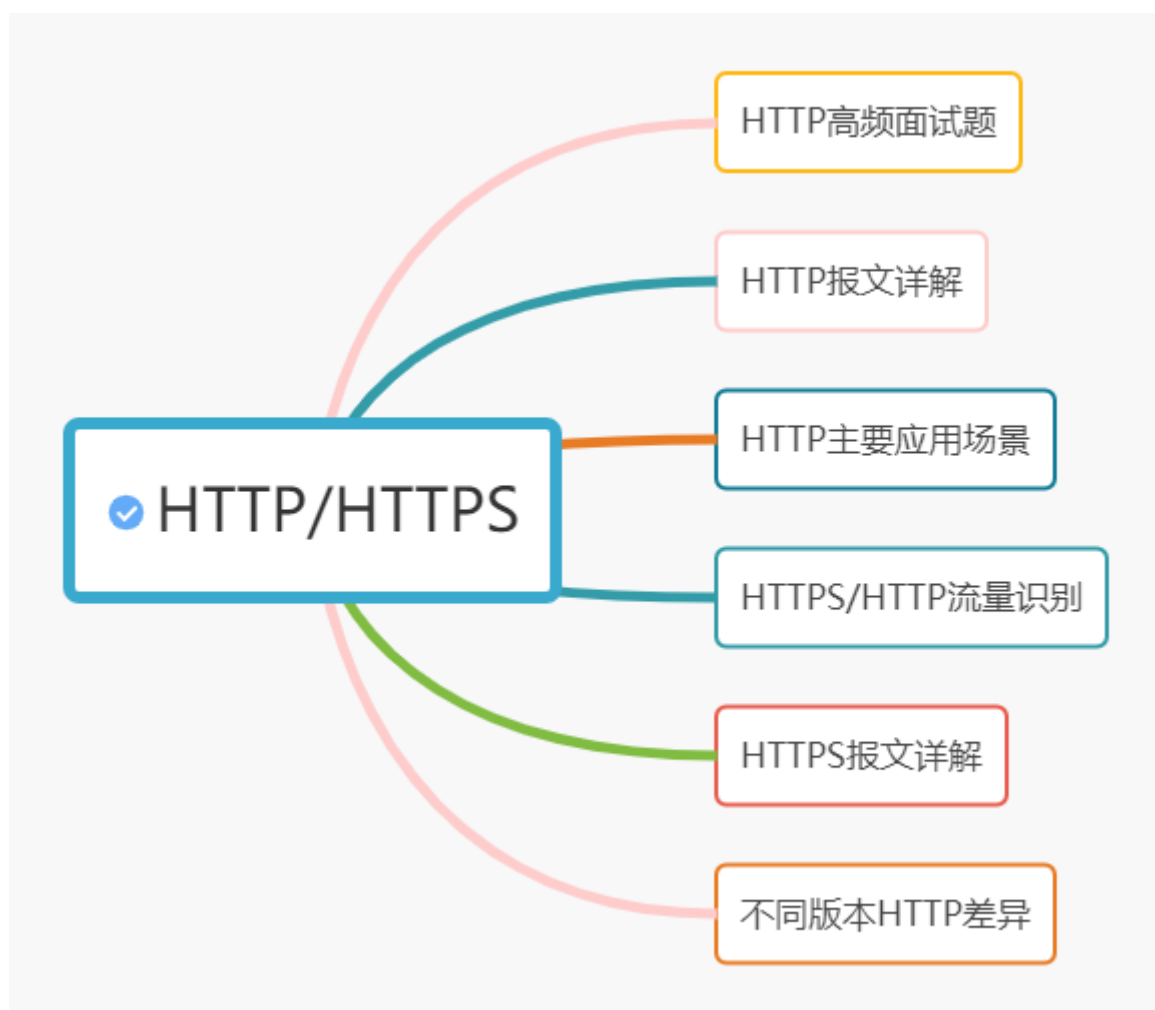


本文将从以下几个方面进行分享。其中包括HTTP发展史，HTTP缓存代理机制，常用的web攻击，HTTP和HTTPS的流量识别，网络协议学习的工具推荐以及高频HTTP与HTTPS的高频面试题解等，开工。

- 1 HTTP应用在哪儿
- 2 HTTP是什么
- 3 不同版本的HTTP
- 4 HTTP报文详解
- 5 HTTPS
- 6 HTTP特点小结
- 7 HTTP识别(还原)
- 8 HTTPS(密文)识别
- 9 HTTP面试题测试
唠嗑



1989年，蒂姆·伯纳斯 - 李（Tim Berners-Lee）在论文中提出可以在互联网上构建超链接文档，并提出了三点。

URI：统一资源标识符。互联网的唯一ID

HTML: 超文本文档

HTTP: 传输超文本的文本传输协议

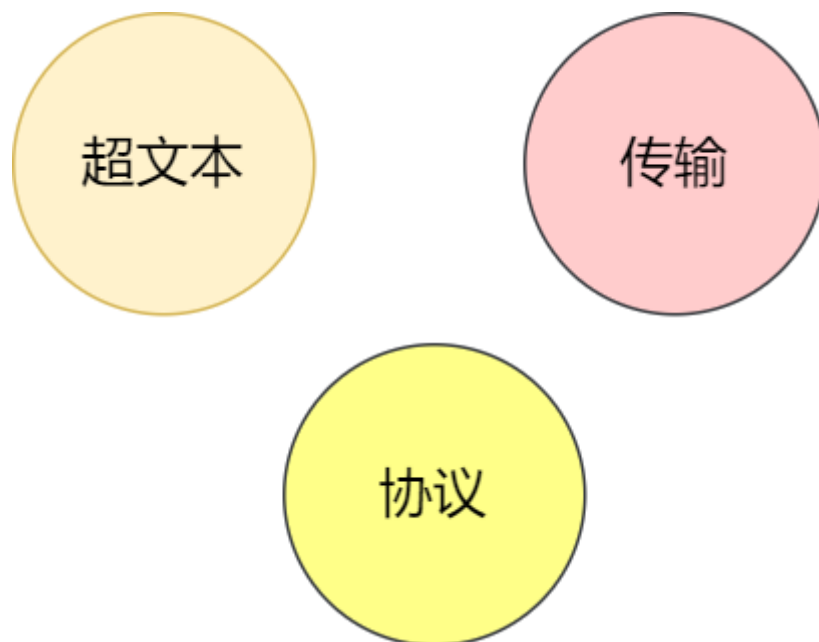
1 HTTP应用在哪儿

学习一门知识，采用五分钟时间看看这个知识是干啥的可能会更加有目的性。HTTP可谓无处不在，这里例举出几个。



2 HTTP是什么

HTTP(hypertext transport protocol)翻译过来为"超文本传输协议"，文本可以理解为简单的字符文字组合，也可以理解为更为复杂的音频或者图像等。那么将这个词语拆分为三个部分。



"超文本"和"文本"相比多了一个字"超"，这样看来比文本丰富，因为它可以将多种文本/图像等进行混合，更重要的是可以从一个文本跳转到另一个文本(文本连接)。

"传输", 传输的过程中需要沟通, 沟通即可能一对一沟通也可能一对多沟通(进行内容协商), 无论怎么样, 参加沟通的人数>1, 想尽一切办法更快更好的完成相应的任务。

"协议", 无规矩不成方圆, 做机密项目之前需要签署保密协议, 找工作要签"三方协议", 三方协议是学校, 公司, 和个人组成的协议, 都是为了让大家受一定的约束, 违反了即有相应的惩罚。



3 不同版本的HTTP

HTTP/0.9

当时网络资源匮乏, 0.9版本相对简单, 采用纯文本格式, 且设置为只读, 所以当时只能使用"Get"的方式从服务器获得HTML文档, 响应以后则关闭。如下图所示

```
GET /Mysite.html
```

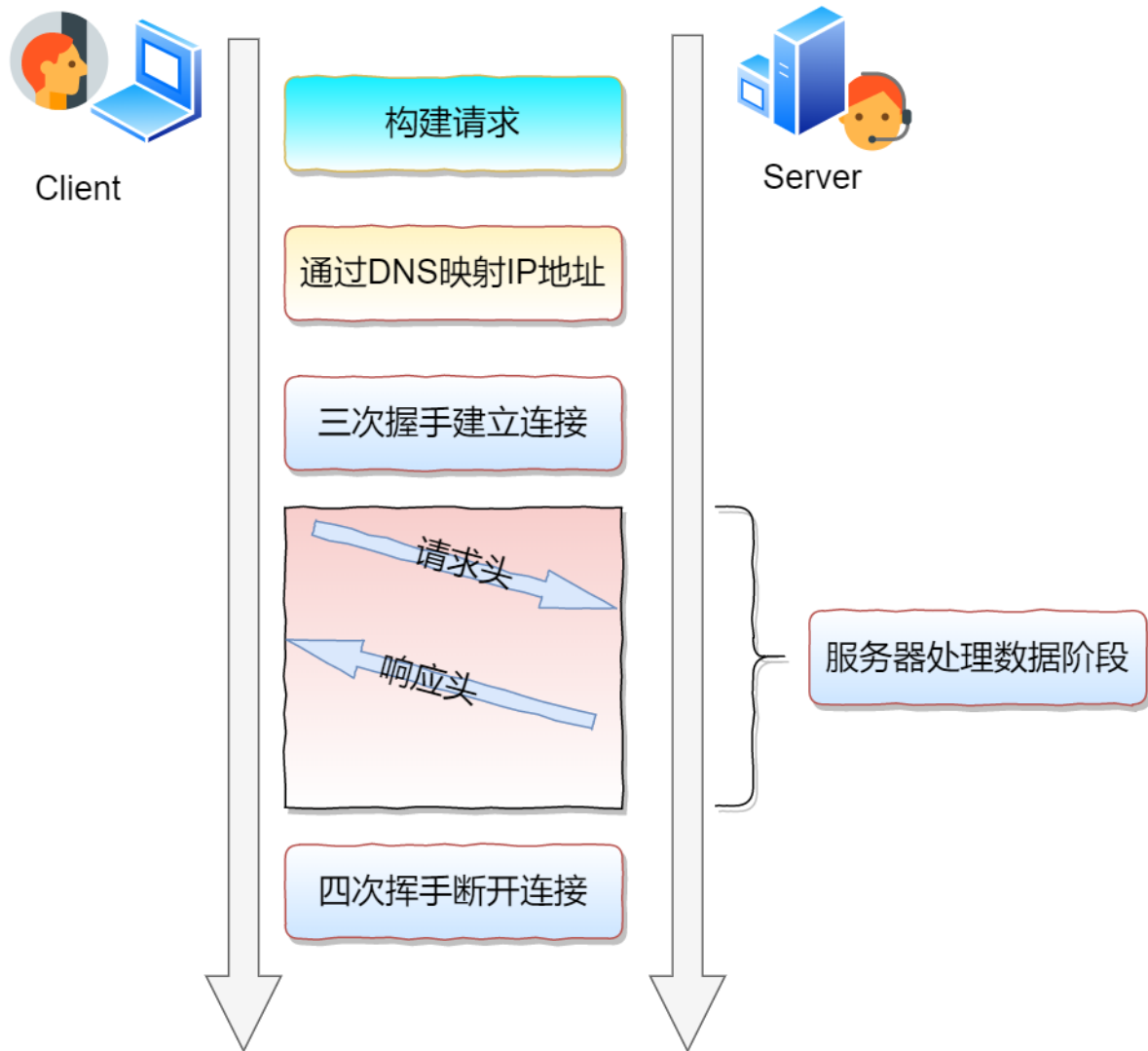
响应中只包含了文档本身。响应内容无响应头, 无错误码, 无状态码, 可以说是"裸奔"。

```
<HTML>
Hello world
</HTML>
HTTP/1.0
```

此时HTTP/0.9请求过程如下

- 应用层的HTTP建立在传输层的TCP之上并运用TCP可靠等特性, 先三次握手建立连接
- 客户端请求建立连接(此时只有GET)
- 服务端响应请求, 数据以 ASCII 字符流返回给客户端
- 传输完成, 断开连接。

HTTP/0.9



HTTP1.0

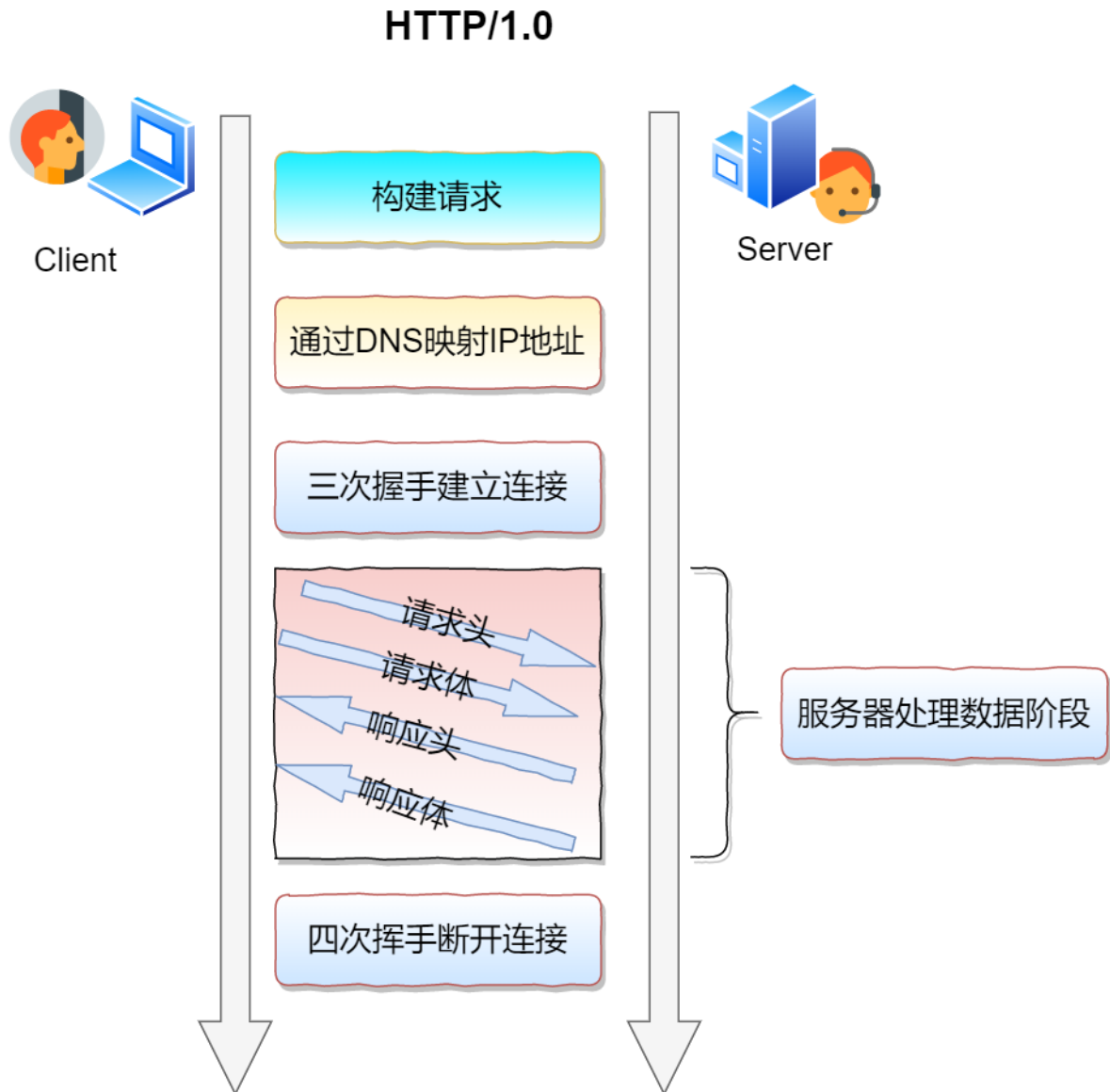
随着时代的进步，仅仅文本的传输无法满足需求，更多情况需要采用图文的方式才能生动的表达出自己的观点。随着1995年开发出Apache，同时其他的多媒体等技术发展迅速，从而进一步的促使HTTP新功能的出现。HTTP1.0在1996年诞生，增加了一下几个方面：

- 之前只有Get方法，现在增加Post(加参数)，Head方法
- 加入协议版本号，同时添加文件处理类型
- 加入HTTP Header，让HTTP处理请求更加灵活
- 增加响应状态码，标记出错的原因
- 提供国际化(不同语言)支持

典型的请求过程

```
GET /image.html HTTP/1.0
User-Agent: Mozilla/5.0 (windows NT 6.1; WOW64)

200 OK
Date: Tue, 17 Nov 2020 09:15:31 GMT
Content-Type: text/html
<HTML>
一个包含图片的页面
  <IMG SRC="/image.gif">
</HTML>
```



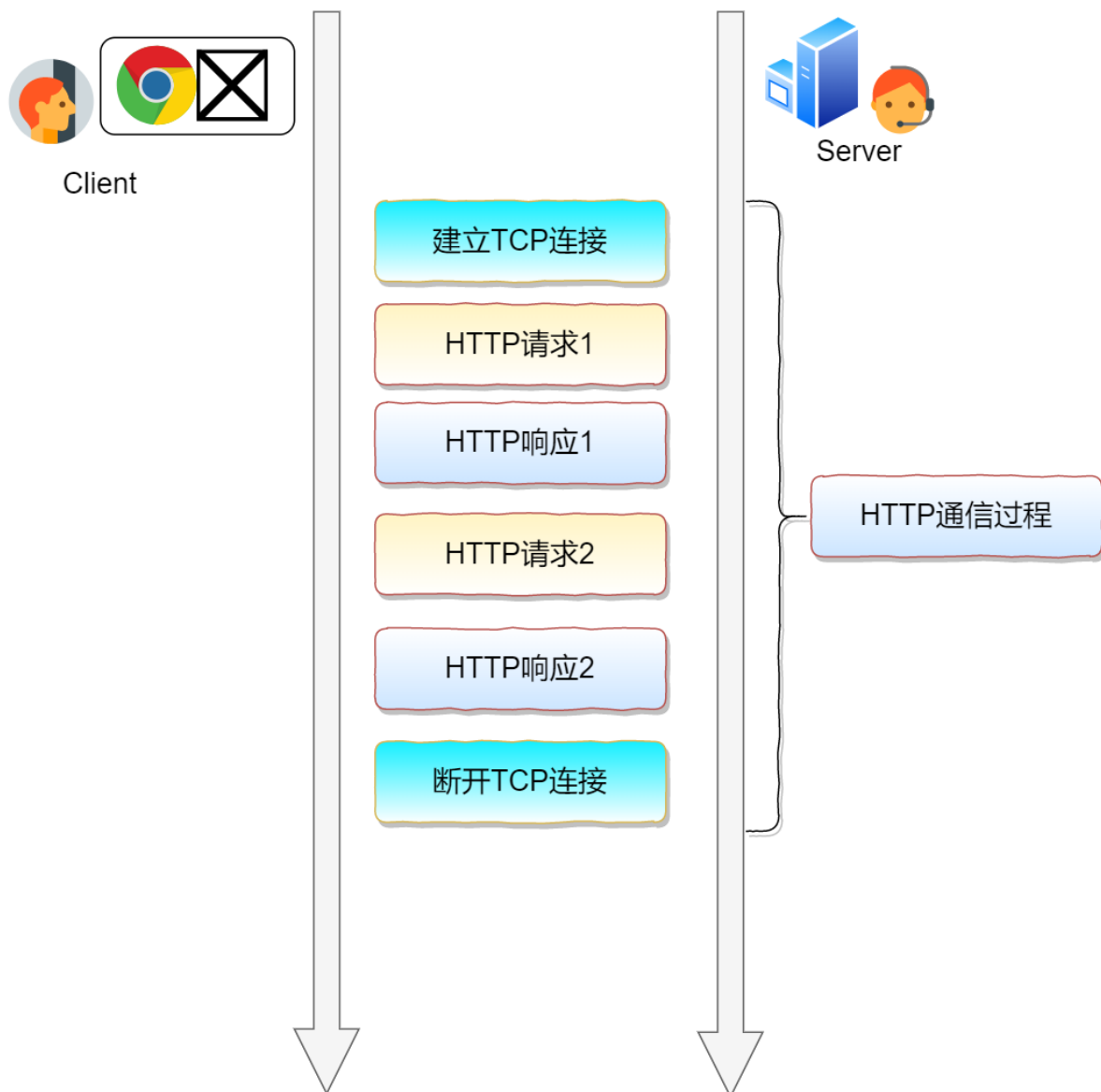
HTTP /1.1

1995年是不平凡的一年，网景公司和微软开启浏览器大战，谁都想当老大。1999年HTTP/1.1发布并成为标准，写入RFC，以为以后不管是网关还是APP等，只要你要使用HTTP，就得遵守这个标准。

- 继续增加了PUT等方法
- 允许持久连接

随着文件越来越大，图片等信息越来越复杂，如果每一次上传下载文件都需要建立连接断开连接的过程将增加大量的开销。为此，提出了持久连接，也就是一次TCP连接可以具有多个HTTP请求。当然持久连接是可选的，如果考虑关闭，只需要使用Connection:close关闭即可。长连接如下图所示

HTTP/1.1长连接



- 强制要求Host头

我们知道，在电商系统中，经常会因为促销活动导致流量飙升，为了缓解流量，其中有种方法即加缓存或者加服务器。如果是单台服务器负载过大，数据库可能分库分表。数据结构算法中分而治之的方法亦是如此。那么HTTP中，同样的道理，如果文件太大，就大文件切分为小文件块发送。

HTTP /2

HTTP/1.1的出现，几年间出来大量牛掰的互联网公司，发展实在是太快，但是HTTP1.1中这几点成为诟病

- 原因1 TCP自带慢启动

顾名思义，"慢启动"从0到1循序渐进。轿车启动不会按下按钮就直接起飞，而是缓慢调节到适合的速度。这不是挺好的？为什么会带来性能问题呢。我们知道一个页面有静态数据，动态页面，很多小文件在加载的过程中就会直接发起请求，这样导致太多的请求都会经历慢启动过程，花费时间太多。

- 原因2 多条TCP连接带宽竞争

带宽固定，多条TCP连接同时发起竞争带宽资源，由于各个TCP连接之间没有通信机制，也无法得知哪些资源优先级更高，从而导致想快速下载的资源反而延迟下载。

- 原因3 头部阻塞

阻塞，在网络编程中，我们采用异步，多路复用(epoll)方式尽量让cpu少等待多干事。在HTTP1.1中，虽然大家共用了一条TCP通道，但是第一个请求没有结束，第二请求就阻塞等待，也就是说不能同时发送接收数据。那么一个网页很多数据文件，如果能够同时发出请求，让部分数据文件能够得到响应并预处理，这样就大大的利用了带宽和cpu的资源。基于这些因素，出现了HTTP2

如何解决头部阻塞呢？

HTTP是一问一答的模式，大家都在这个队列排队导致堵塞，那就多个队列并发进行，也就是"对同一个域名发起多个长连接"。举个例子，在火车站排队买票的时候，如果只有一个窗口可用，大家只能苦等，多开几个窗口就可缓解这个问题。

这个时候用户数 * 并发数(上限6-8)已经不错得效果，但是互联网速度太快，火车站就这么大，窗口也就这么多，怎么办，建新的火车站进行分流(大部分城市都有什么东站 西站)。在这里叫做"域名分片"，使用多个域名，这些域名指向指向同一服务器。

HTTP/3

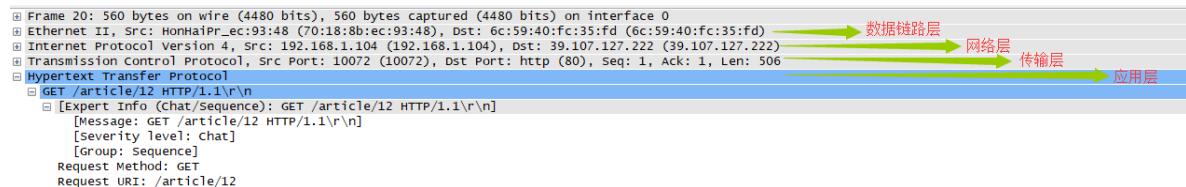
HTTP/2看似很完美了吧，但是Google轮子哥可不服，其他人在研究HTTP/2的时候，它们就在琢磨QUIC。那QUIC有啥牛掰的地方呢

QUIC是Google开发的一个基于UDP且能像TCP一样具有可靠性特点的协议。具备像HTTP/2一样的应用数据二进制分帧传输。其主要解决的问题有两个。

1. 进一步解决线头阻塞问题。通过独立不同流，让各个流之间实现相互独立传输，互不干扰
2. 切换网络时的连接保持。wifi和3G/4G经常需要来回切换。基于TCP的协议，会因为网络的切换导致IP地址的改变。而基于UDP的QUIC协议，及时切换也可以恢复之前与服务器的连接。

4 HTTP报文详解

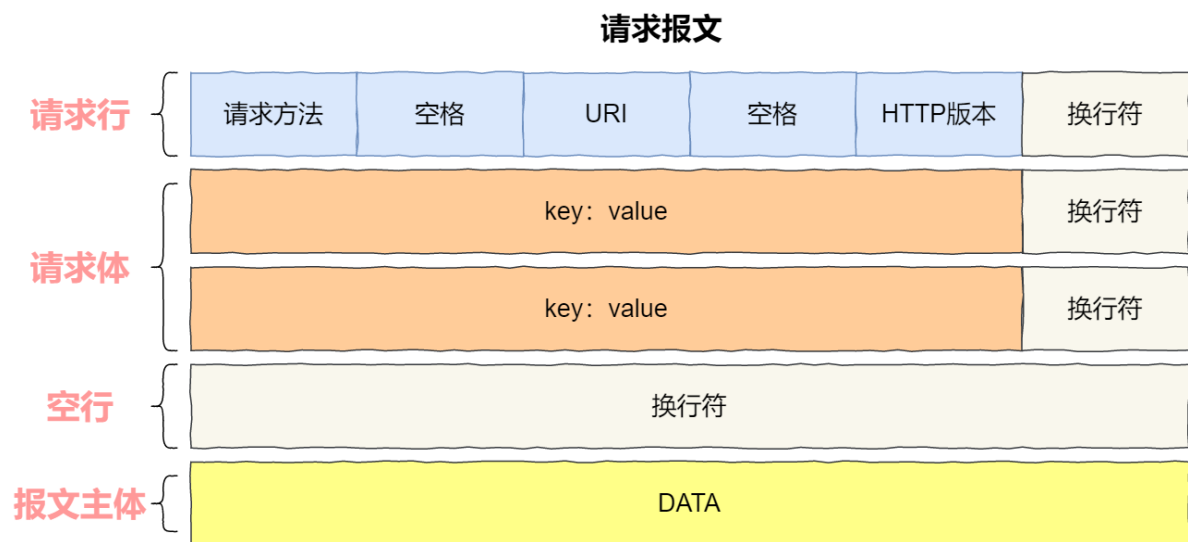
客户端与服务端进行交互的信息为报文。客户端为请求报文，服务端为响应报文。我们先用wireshark抓一个博客看看



```
GET /article/12 HTTP/1.1
Host: www.xxx.cn
Connection: keep-alive
Cache-Control: max-age=0
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/80.0.3987.106 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Cookie: SESSION=so9nlsvenminor5abs65sh9dsa
```

```
HTTP/1.1 200 OK
Server: nginx
Date: Sun, 17 May 2020 17:04:29 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
X-Powered-By: blade-2.0.6-BETA
Content-Encoding: gzip
```

请求报文



请求报文通常由三部分组成：

起始行：描述请求或者响应的基本信息

头部字段集合：key-value形式说明报文

消息正文：实际传输诸如图片等信息。具体如下图试试

1 请求方法：一共有八种方法选择，如下图所示。采用不同的方法获取不同的资源

1	请求方法	方法描述
2	GET	请求服务器某一资源
3	POST	向指定资源提交数据进行处理请求,数据包含在请求体中。
4	HEAD	用于确认URI的有效性 & 资源更新的日期时间, 不返回报文主体, 只返回报文的文首部。
5	PUT	向用来传输文件, 将文件内容放进报文主体中, 保存到URI指定位置上。
6	DELETE	与PUT相反, 请求URI删除指定资源。
7	OPTIONS	查询针对请求URI指定的资源支持的方法。
8	TRACE	用于追踪路径。发送请求时, 首部字段Max-Forwards会指定一个数值, 每经过一个服务器之后, 该数值减1。当该数值为0时, 停止传输, 最后接收到的服务器响应。
9	CONNECT	用于在与代理服务器通信时建立隧道, 实现用隧道协议进行TCP通信。

说一下非常常见的几种请求方法

Get：从服务器中取资源。可以请求图片，视频等

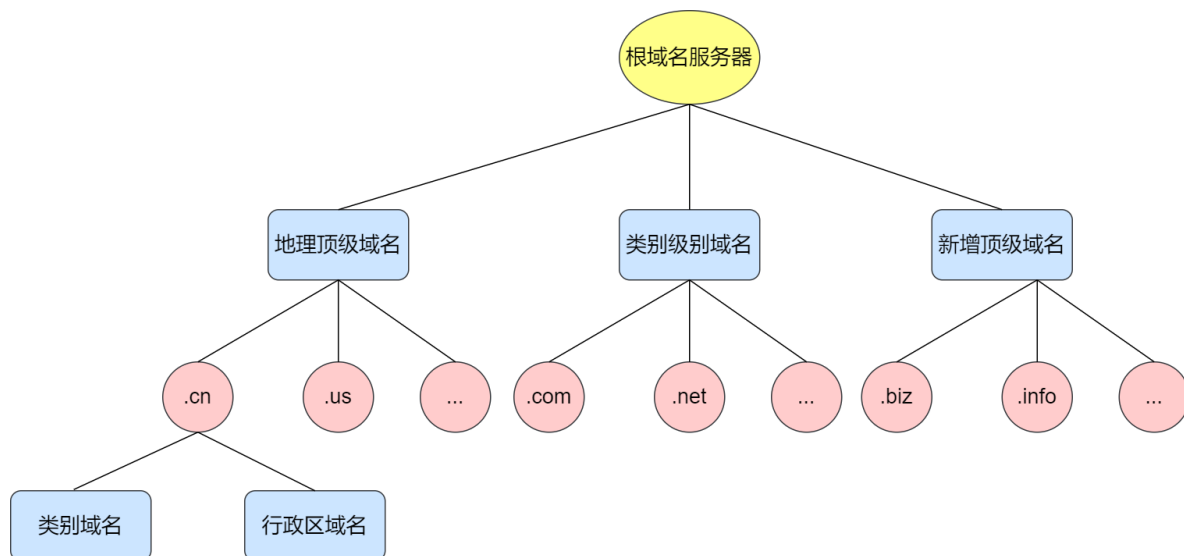
HEAD:和Get类似，但是从服务器请求的资源不会反悔请求的实体数据，只会反悔响应头

POST/PUT：对应于GET，向服务器发送数据

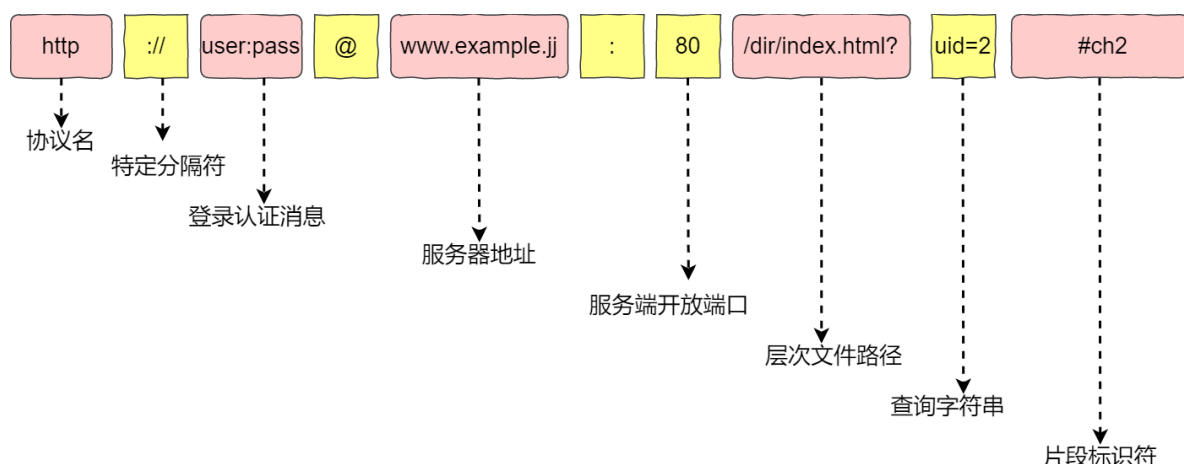
2 URI

统一资源标识符(Uniform Resource Identifier),严格来说不等于网址,它包含URL和URN,可是URL太出名了以致于URL="网址"。无论开发,测试运维配置都离不开URI,所以好好掌握。

网络层的IP主要目的是解决路由和寻址。现在的IP地址按照"."分割,总共2的32次方大约42亿。对于计算机来说比较方便,但是对于人类来说还是不容易记忆,此时出现DNS了,他把IP地址映射为我们平时常见的"redis.org",按照.分割域名,从左到右级别越高,最右边为"顶级域名"。如下图所示



好了,现在TCP提供可靠(数据不丢失)且字节流(数据完整性),而且也有方便我们记忆的域名,但是互联网资源千万种,也不知道访问什么(图片,文字,视频一大堆),这个时候URI(统一资源标识符)出现了,那长啥样?



协议名:HTTP协议,另外还有ftp等协议。告知访问资源时使用什么协议。

紧接着是分隔符:"://"

主机名:标记互联网主机,可以是IP也可以是域名,如果不写端口则使用默认端口,例如HTTP为80,HTTPS为443。

登录认证信息:登录主机时的用户名密码(不建议,直接告诉了别人你的隐私信息)

主机名:此处可以是域名也可以是IP,如果不写端口号则是默认端口。比如HTTP默认端口为80,HTTPS默认端口为443

资源所在位置:资源在主机上的位置,使用"/"分隔多级目录,在这里是"/en/download.html"。注意,必须"/"开头

参数:用"?"开始,表示额外的请求要求。通常使用"key=value"的方式存在,如果多个"key=value"则使用"&"相连。

看几个例子

<http://nginx.org/en/download.html>

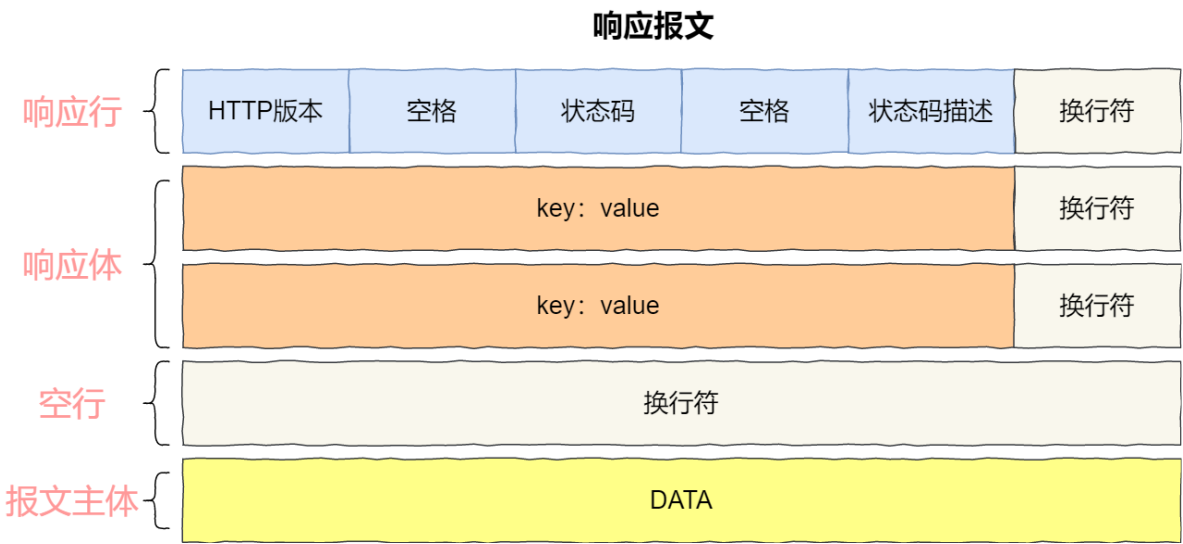
file:///E:/Demo/index/

这里注意是三个"/"，因为前面":"/"作为分隔符，资源路径按照"/"开头。

既然规则这么多，对于接收方而言需要完成的解析也需要遵守规则，全球用户很多使用HTTP，每个国家地区所使用语言不同，HTTP为了能对其进行统一处理，引入了URI编码，方法比较简单，将非ASCII或者特殊字符全部转换为十六进制字节值，同时在前面加入"%". 比如空格被转换为"%20","中国"就编码为"%E4%B8%AD%E5%9B%BD%0A".

3 请求体

响应报文



状态行----服务器响应的状态

<1> 版本号：使用的HTTP什么版本

<2> 状态码：不同数字代表不同的结果，就如我们在编码时，通过返回不同的值代表不同的语义。

状态码一共分为5类。

- 1xx：处于中间状态，还需后续操作
- 2xx：成功收到报文并正确处理

"200 OK"

最常见的成功状态码，表示一切正常，客户端获得期许的处理结果。如果不是Head请求，那么在响应头中通常会有body数据。

"204 No Content"

这个的含义和"200"很相似，不同之处在于它的响应头中没有body数据。

"206 Partial Content"

是 HTTP 分块下载或断点续传的基础，在客户端发送“范围请求”、要求获取资源的部分数据时出现，它与 200 一样，也是服务器成功处理了请求，但 body 里的数据不是资源的全部，而是其中的一部分。状态码 206 通常会伴随着头字段“Content-Range”，表示响应报文里 body 数据的具体范围，供客户端确认，例如“Content-Range: bytes 0-99/5000”，意思是此次获取的是总计 5000 个字节的前 100 个字节。

3xx: 重定向到其他资源位置

"301 Moved Permanently"

"永久重定向"，意思是本地请求的资源以及不存在，使用新的URI再次访问。

"302 Found"

"Moved Temporarily"，"临时重定向"，临时则所请求的资源暂时还在，但是目前需要用另一个URI访问。

301 和 302 通过在字段Location中表明需要跳转的URI。两者最大的不同在于一个是临时改变，一个是永久改变。举个例子，有时候需要将网站全部升级为HTTPS，这种永久性改变就需要配置永久的"301"。有时候晚上更新系统，系统暂时不可用，可以配置"302"临时访问，此时不会做缓存优化，第二天还会访问原来的地址。

"304 Not Modified"

运用于缓存控制。它用于 If-Modified-Since 等条件请求，表示资源未修改，可以理解成"重定向已到缓存的文件"（即"缓存重定向"）。

4xx: 请求报文有误，服务器无法处理

"400 Bad Request"

通用错误码，表示请求报文有错误，但是这个错误过于笼统。不知道是客户端还是哪里的错误，所以在实际应用中，通常会返回含有明确含义的状态码。

"403 Forbidden"

注意了，这一个表示服务器禁止访问资源。原因比如涉及到敏感词汇、法律禁止等。当然，如果能让客户端有一个清晰的认识，可以考虑说明拒绝的原因并返回即可。

"404 Not Found"

这可能是我们都知且都不想看到的状态码之一，它的本意是想要的资源在本地未找到从而无法提供给服务端，但是现在，只要服务器"耍脾气"就会给你返回 404，而我们也无从得知后面到底是真的未找到，还是有什么别的原因，

"405 Method Not Allowed"

获取资源的方法好几种，我们可以对某些方法进行限制。例如不允许 POST 只能 GET；

"406 Not Acceptable"

客户端资源无法满足客户端请求的条件，例如请求中文但只有英文；

"408 Request Timeout"

请求超时，服务器等待了过长的时间；

"409 Conflict":

多个请求发生了冲突，可以理解为多线程并发时的竞态；

413 Request Entity Too Large:

请求报文里的 body 太大；

414 Request-URI Too Long: 请求行里的 URI 太大；

429 Too Many Requests: 客户端发送了太多的请求, 通常是由于服务器的限连策略;

431 Request Header Fields Too Large: 请求头某个字段或总体太大;

5xx: 服务器错误, 服务器对请求出的时候发生内部错误。

“500 Internal Server Error”

和400类似, 属于一个通用的错误码, 但是服务器到底是什么错误我们不得而知。其实这是好事, 尽量少的将服务器资源暴露外网, 尽量保证服务器的安全。

“502 Bad Gateway”

通常是服务器作为网关或者代理时返回的错误码, 表示服务器自身工作正常, 访问后端服务器时发生了错误, 但具体的错误原因也是不知道的。

“503 Service Unavailable”

表示服务器当前很忙, 暂时无法响应服务, 我们上网时有时候遇到的“网络服务正忙, 请稍后重试”的提示信息就是状态码 503。

503 是一个“临时”的状态,

暂时比较忙, 稍后提供服务。在响应报文中的“Retry-After”字段, 指示客户端可以在多久以后再次尝试发送请求。

4 请求体

上面大部分都是涉及到header部分, 还有非常重要的body, everybody

头字段注意事项

<1> 字段名不区分大小写, 例如“Host”也可以写成“host”, 但首字母大写的可读性更好;

<2> 字段名里不允许出现空格, 可以使用连字符“-”, 但不能使用下划线“_”。例如, “test-name”是合法的字段名, 而“test name”“test_name”是不正确的字段名;

<3> 字段名后面必须紧接着“:”, 不能有空格, 而“:”后的字段值前可以有多个空格;

<4> 字段的顺序是没有意义的, 可以任意排列不影响语义;

<5> 字段原则上不能重复, 除非这个字段本身的语义允许, 例如 Set-Cookie。

HTTP的body常常被分为这几类的类别

<1> text:超文本text/html,纯文本text/plain

<2> audio/video:音视频数据

<3> application: 可能是文本, 也可能是二进制, 交给上层应用处理

<4> image: 图像文件。image/png等

但是带宽一定, 数据大了通常考虑使用压缩算法进行压缩, 在HTTP中使用Encoding type表示, 常用的压缩方式有下面几种

<1> gzip:

一种数据格式, 默认且目前仅使用deflate算法压缩data部分

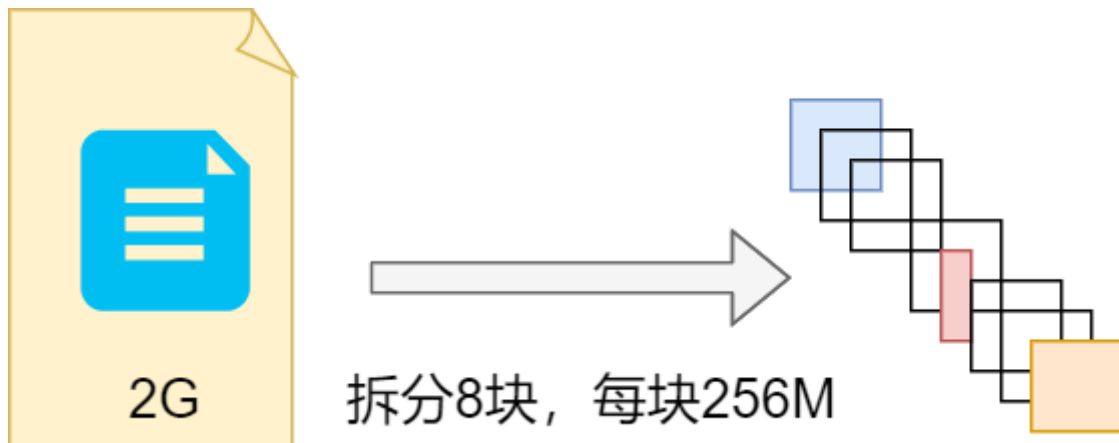
<2> deflate:

deflate是一种压缩算法，是huffman编码的一种加强

<3> br:

br通过变种的LZ77算法、Huffman编码以及二阶文本建模等方式进行数据压缩，其他压缩算法相比，它有着更高的压缩效率

使用相应的压缩方法在带宽一定的情况下确实有不错的效果，但是gzip等主要针对文件压缩效果不错，但是对视频就不行了。这个时候是不是可以使用数据结构中常用的分而治之，大化小再合并的方式呢，



ok，在报文中使用"Transfer-Encoding: chunked"表示，代表body部分数据是分块传输的。另外在body中存在一个content-length字段表示body的长度，两者不能共存，另外很多时候是流式数据，body中没有指明content-length，这个时候一般就是chunked传输了。

现在可以通过采用分块的方式增强带宽的利用率，那他的编码规则如何呢

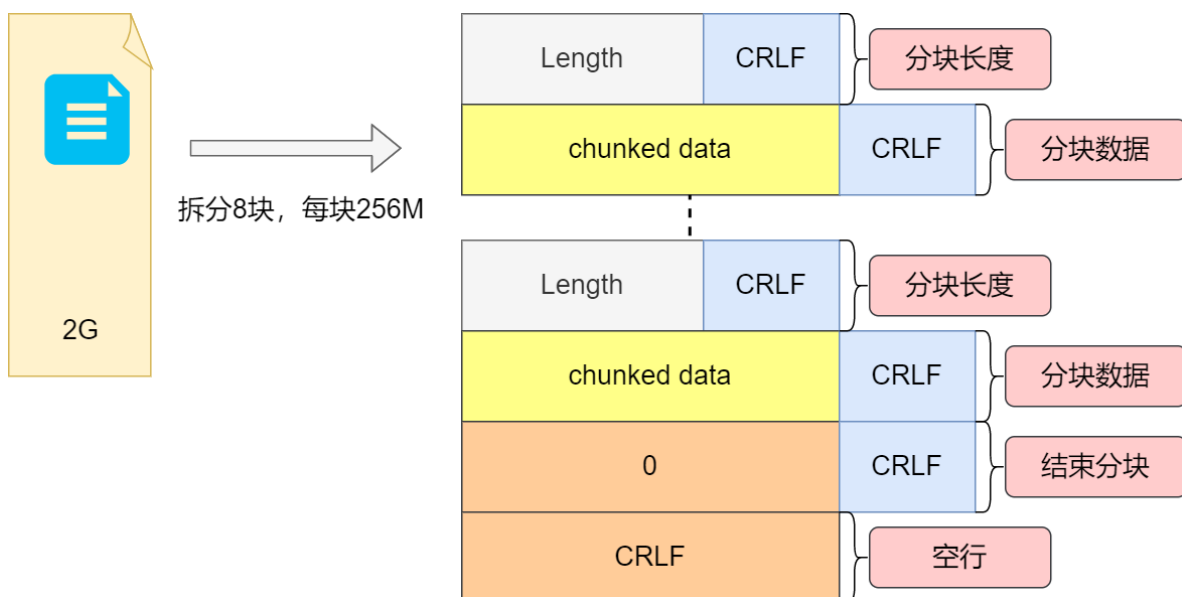
<1> 每一个分块包含长度和数据块

<2> 长度头按照CRLF结束

<3> 数据块在长度块后，且最后CRLF结尾

<4> 使用长度0表示结束，"0\r\n\r\n"

我们还是看图加深印象



分块解决了咱们一部分问题，但是有的时候我们想截断发送怎么办呢。在HTTP中提供了使用字段"Accept-Range: bytes"，明确告知客户端：“我是支持范围请求的”。那么Range范围是怎样的呢，Range从0开始计算，比如Range:0-5则读取前6个字节，服务器收到了这个请求，将如何回应呢

<1> 合法性检查。比如一共只有20字节，但是请求range: 100-200。此时会返回416----"范围请求有误"

<2> 范围正常，则返回216，表示请求数据知识一部分

<3> 服务器端在相应投资端增加Content-Range,格式"bytes x-y/length"。

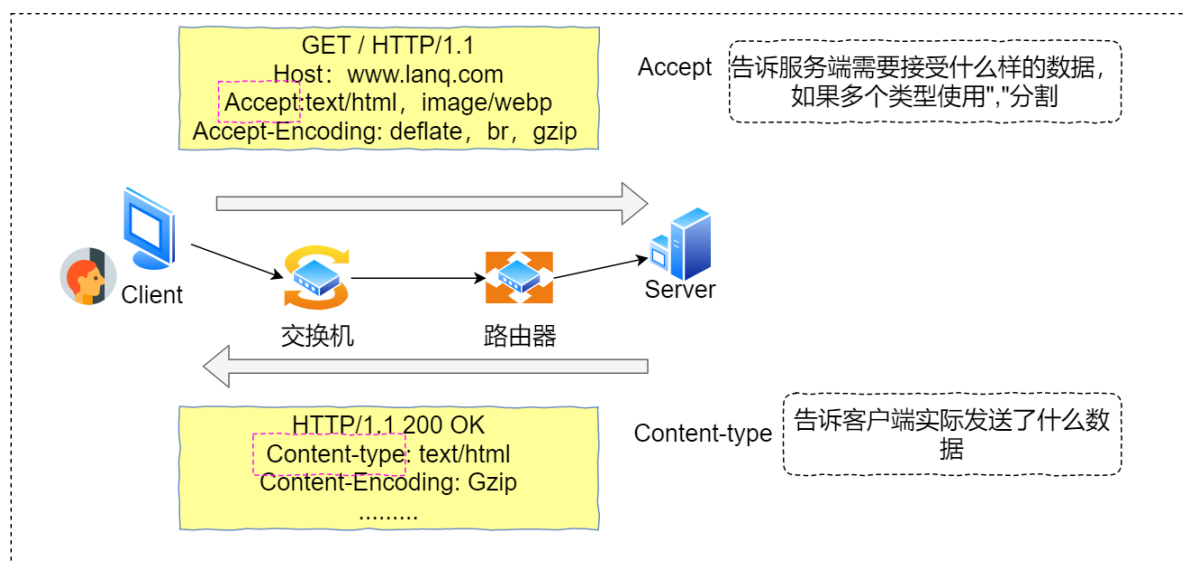
敲黑板：断点续传怎么操作？

<1> 查看服务器是否支持范围请求并记录文件大小

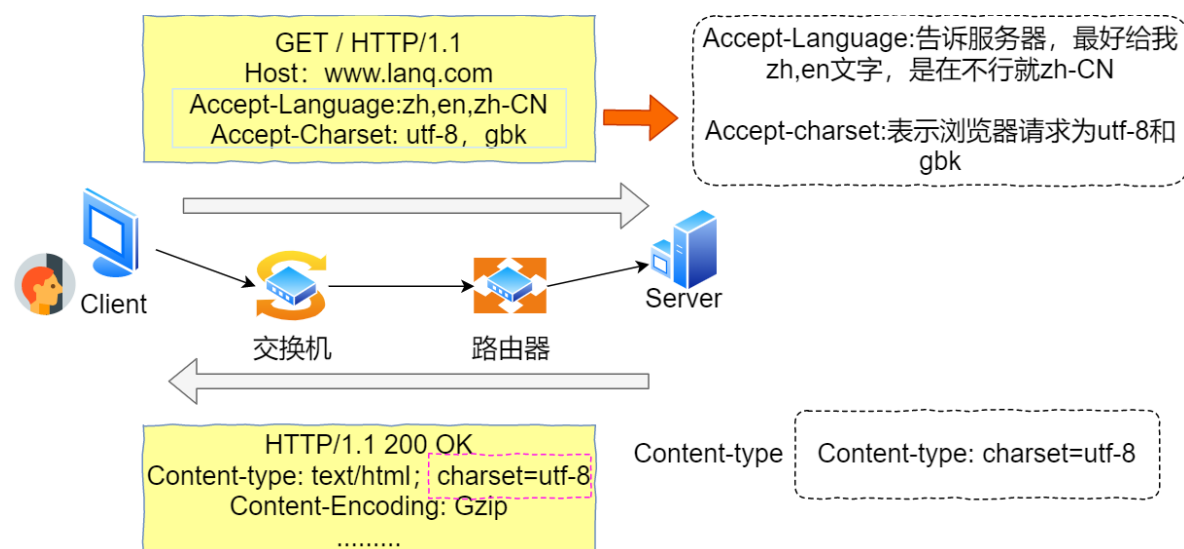
<2> 多个线程分别负责不同的range

<3> 下载同时记录进度，即使因为网络等原因中断也没事，Range请求剩余即可

现在我们通过MIME-TYPE和Encoding-type可以知道body部分的类型，下一步将是对内容进行协商。HTTP中，请求体中使用Accept告诉服务端需要什么类型数据(我能处理哪些类型数据)，响应头中使用Content表明发送了什么类型数据，具体如下图所示

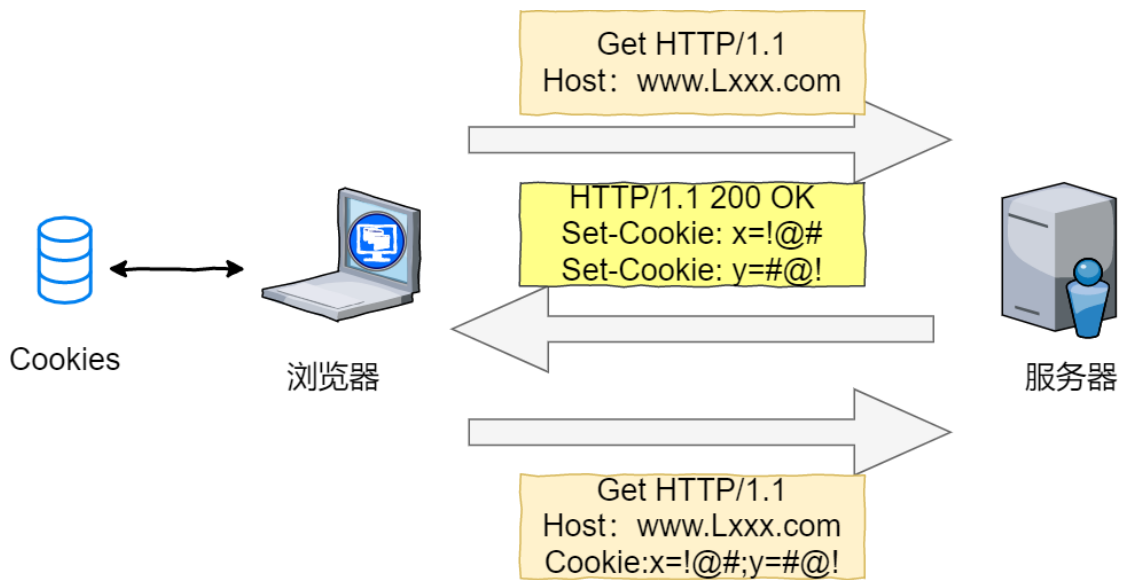


好了，为了各个国家民族顺利友好的沟通和明确的区分。HTTP请求头中使用"type-subtype"，注意此时分隔符是"-"。比如en-GB表示英式英语，zh-CN表示常用的汉语，那对于客户端而言，它通过Accept-Language来标记自己可以理解的自然语言，对应的服务端使用Content-Language表明实体数据使用的语言类型，如下图所示。



Cookie机制

HTTP是无状态、无记忆的，Cookie机制的出现让其有记忆功能，是怎么个实现呢

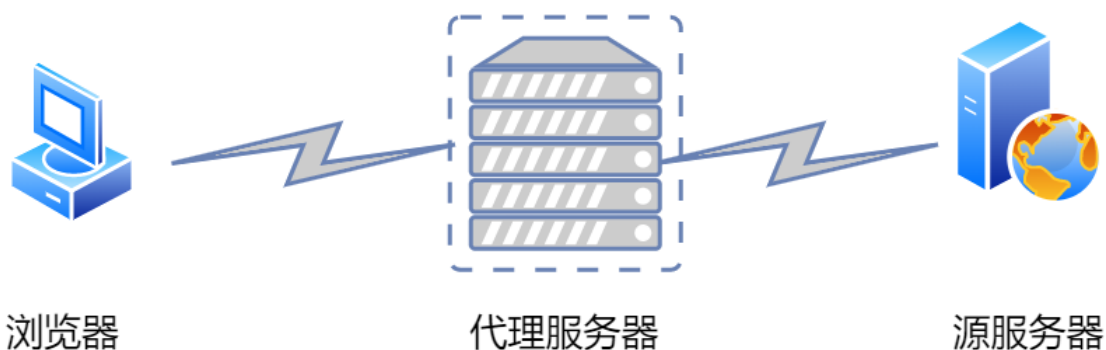


从上图我们可以知道Cookie是由浏览器负责存储，并不是操作系统负责，我们换个浏览器打开同样的网页，服务就认不出来了。

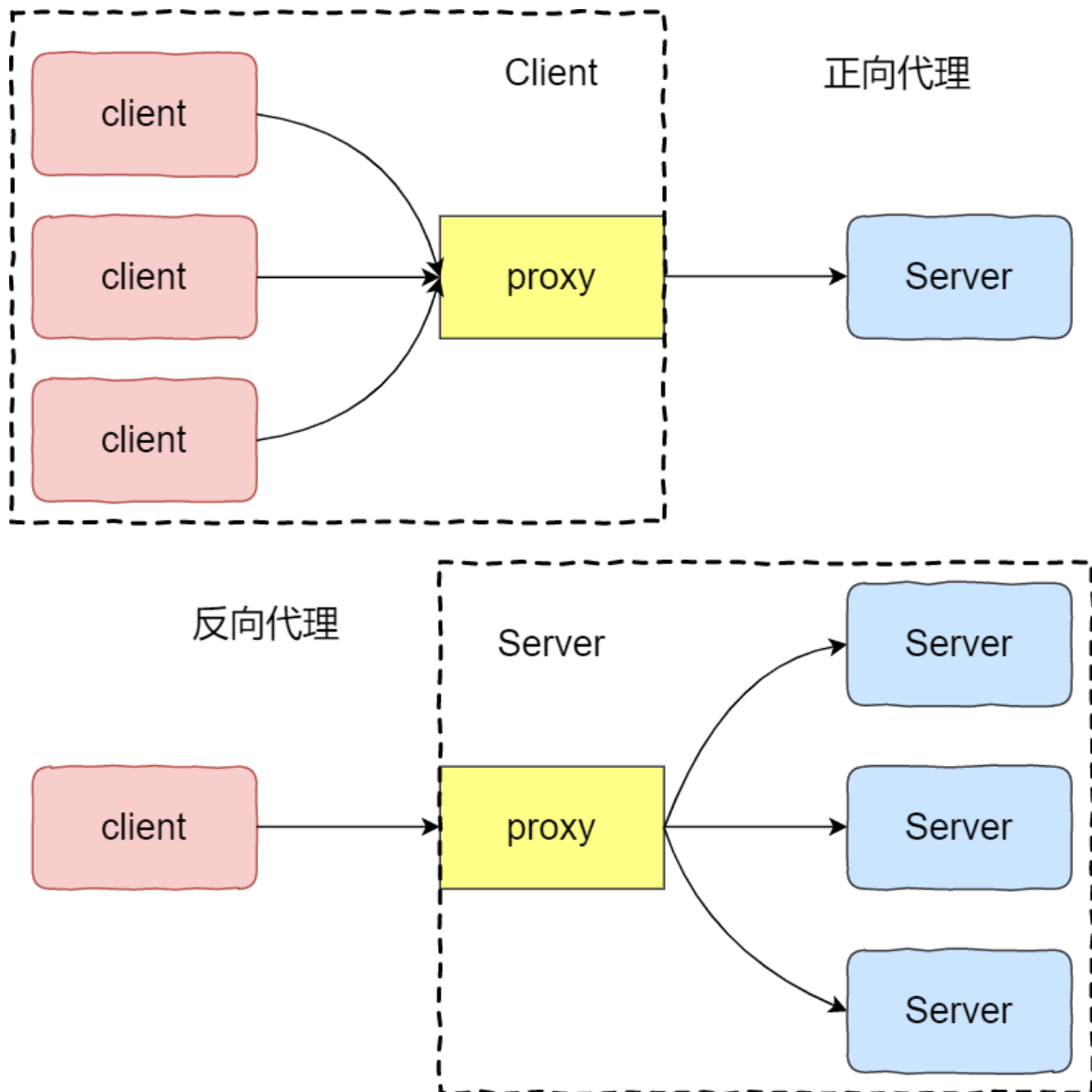
Cookie常见的应用一个是身份识别，一个是广告追踪，比如我们在访问网页视频或者图片的时候，广告商会悄悄给我们Cookie打上标记，方便做关联分析和行为分析，从而给我推荐一些相关内容。

HTTP代理

之前介绍的都是一问一答的情景，但是在大部分的情况下都会存在多台服务器进行通信服务。其中比较常见的就是在请求方与应答方中间增加一个中间代理。



代理作为中间位置，相对请求方为服务端，相当于后端服务端为请求方。代理常见的功能为负载均衡。在负载均衡中需要区分正向代理与反向代理，其中也就会涉及调度算法，比如轮询还是一致性哈希等。

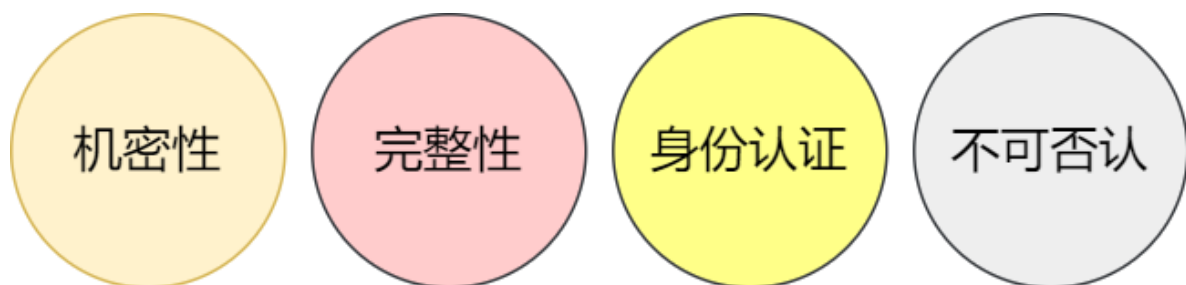


那么问题来了，代理作为隐藏身份，相当于隐藏了真实的客户端与服务端，那在是不是

5 HTTPS

好人占多数，坏人也不少。总有些要搞坏事，因为HTTP是明文，所以需要想办法保护明文，从而出现了https。

安全是什么

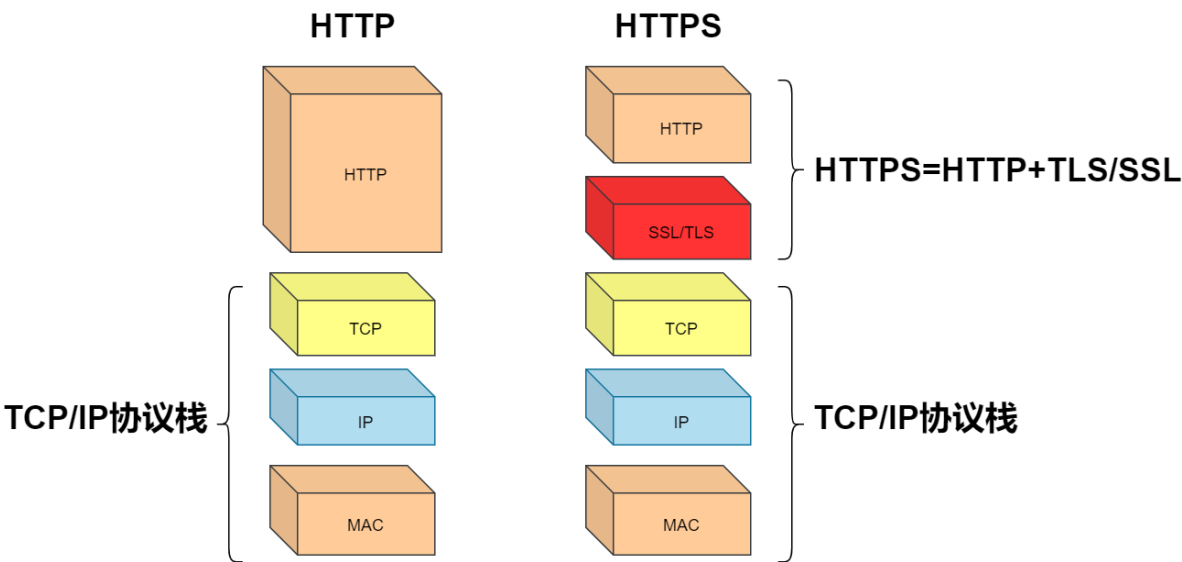


机密性

对信息进行保密，只能可信的人可以访问(让我想起时间管理者)。

完整性

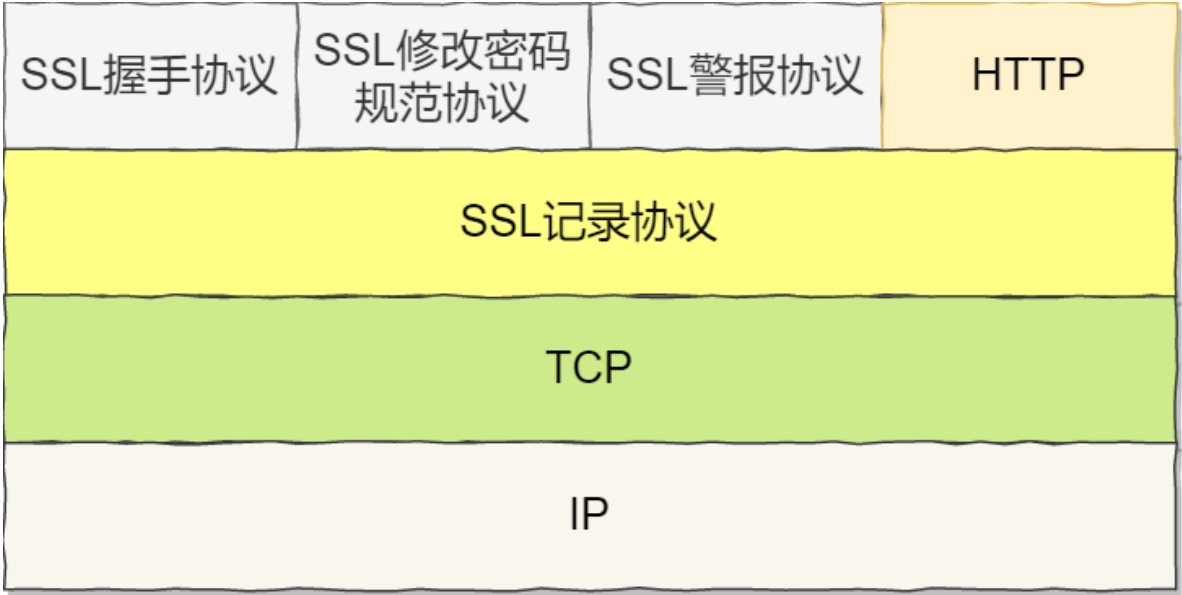
- 数据在传输过程中内容不被"篡改"。虽然机密性对数据进行保密了，但是有上策也有下策(hack)
 - 身份认证
 - 证明自己的身份是本人，保证其消息发给可信的人
 - 不可否认
 - 君子一言驷马难追，说话算数，说过的话做过的事要有所保证
- HTTPS



从上图我们知道HTTPS无非是在传输层和应用层中间加了一层TLS，正是TLS紧跟当代密码学的步伐，尽全力的保障用户的安全。老规矩，我们用wireshark看看长什么样子。

No.	Time	Source	Destination	Protocol	Length	Info
7297	125.218301	192.168.1.104	59.111.179.213	TCP	66	48093 > https [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
7303	125.290874	59.111.179.213	192.168.1.104	TCP	66	https > 48093 [SYN, ACK] Seq=0 Ack=1 Win=2920 Len=0 MSS=1440 SACK_PERM=1 WS=512
7304	125.291082	192.168.1.104	59.111.179.213	TCP	54	48093 > https [ACK] Seq=1 Ack=1 Win=132352 Len=0
7305	125.292222	192.168.1.104	59.111.179.213	SSL	231	Client Hello
7308	125.378134	59.111.179.213	192.168.1.104	TCP	54	https > 48093 [ACK] Seq=1 Ack=178 Win=4096 Len=0
7309	125.378461	59.111.179.213	192.168.1.104	TLSv1.2	1494	Server Hello
7310	125.378464	59.111.179.213	192.168.1.104	TCP	1494	[TCP segment of a reassembled PDU]
7311	125.378705	192.168.1.104	59.111.179.213	TCP	54	48093 > https [ACK] Seq=178 Ack=2881 Win=132352 Len=0
7312	125.379002	59.111.179.213	192.168.1.104	TLSv1.2	386	Certificate
7313	125.383830	192.168.1.104	59.111.179.213	TLSv1.2	147	Client Key Exchange, Change Cipher Spec, Hello Request, Hello Request
7317	125.450613	59.111.179.213	192.168.1.104	TLSv1.2	105	Change Cipher Spec, Hello Request, Hello Request
7318	125.451410	192.168.1.104	59.111.179.213	TLSv1.2	506	Application data
7319	125.543996	59.111.179.213	192.168.1.104	TCP	1494	[TCP segment of a reassembled PDU]
7320	125.543999	59.111.179.213	192.168.1.104	TLSv1.2	993	Application data
7321	125.544182	192.168.1.104	59.111.179.213	TCP	54	48093 > https [ACK] Seq=723 Ack=5643 Win=132352 Len=0
7322	125.544768	192.168.1.104	59.111.179.213	TLSv1.2	85	Encrypted Alert
7323	125.545020	192.168.1.104	59.111.179.213	TCP	54	48093 > https [FIN, ACK] Seq=754 Ack=5643 Win=132352 Len=0

可以看出在交互的过程中多了不少新东西，了解TLS,TLS由SSL握手协议，SSL修改密码规范协议，SSL警报协议，SSL记录协议组成。



SSL握手协议：

相对于三次握手

记录协议

记录为TLS发送接收数据的基本单位。它的自协议需要通过记录协议发出。如果多个纪录数据则可以一个TCP包一次性发出。

警报协议

类似HTTP状态码，通过反馈不同的消息进行不同的策略。

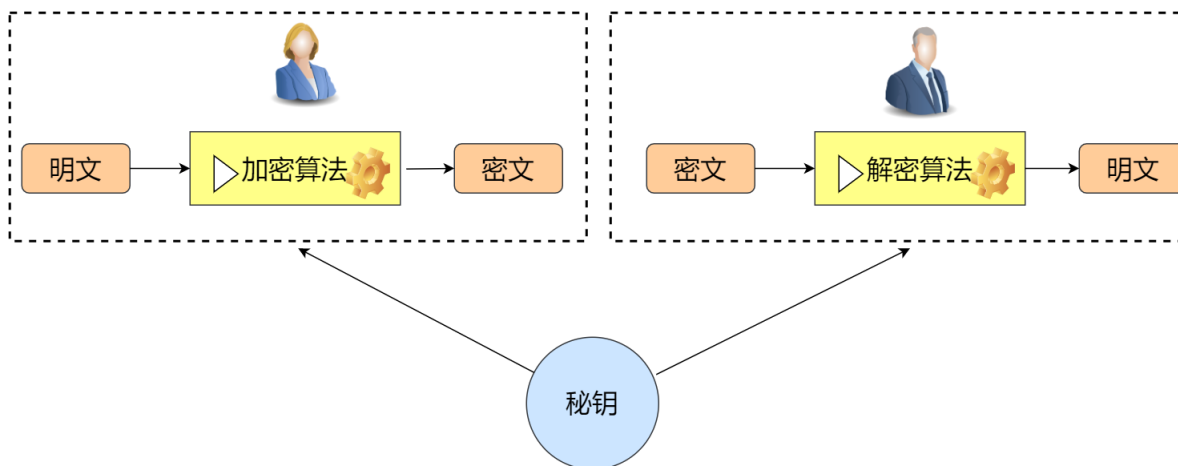
变更密码规范协议

告诉对方，从此刻开始，后续的数据将使用加密算法进行加密再传输。

对称加密与非对称加密

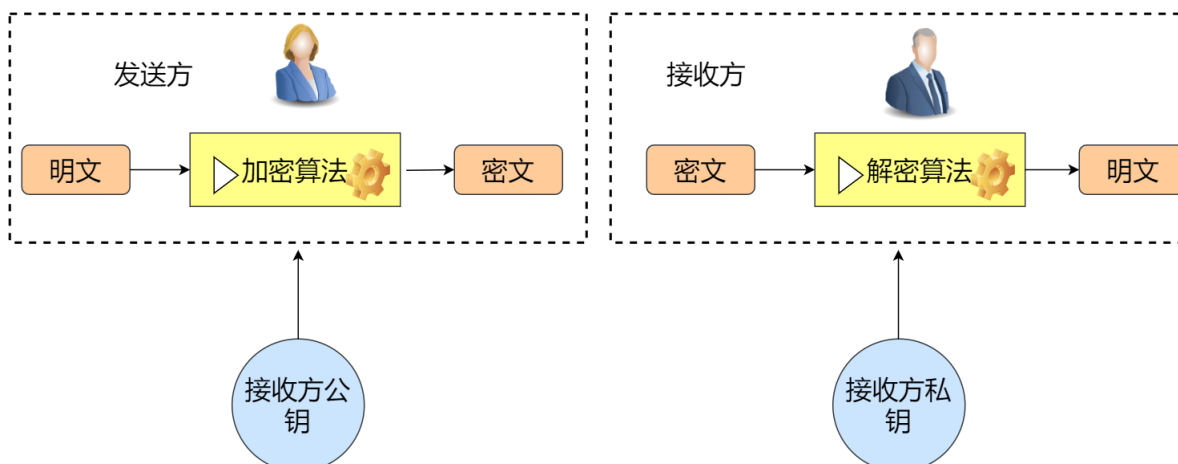
对称加密

对称加密，顾名思义，加密方与解密方使用同一钥匙(密钥)。具体一些就是，发送方通过使用相应的加密算法和密钥，对将要发送的信息进行加密；对于接收方而言，使用解密算法和相同的密钥解锁信息，从而有能力阅读信息。



非对称加密

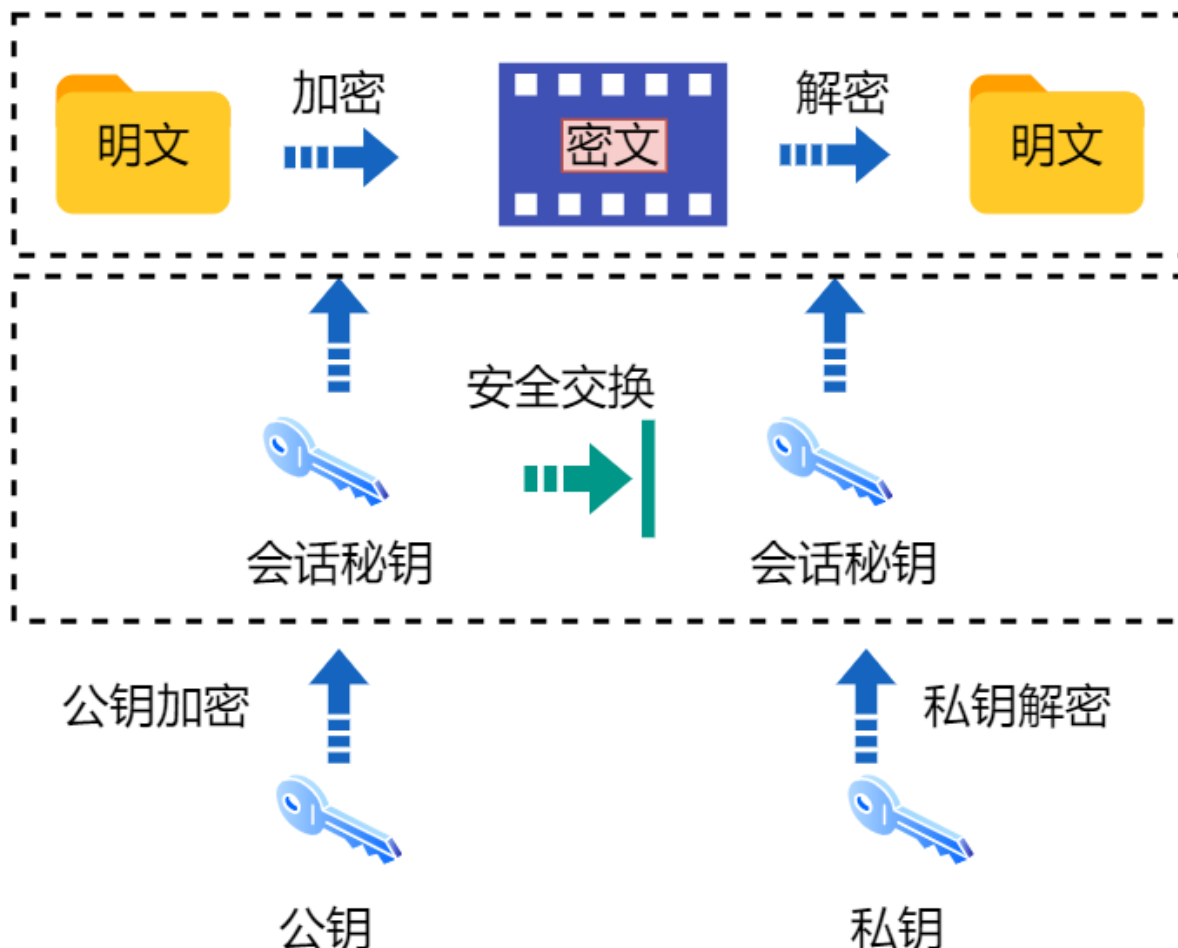
在对称加密中，发送方与接收方使用相同的密钥。那么在非对称加密中则是发送方与接收方使用的不同的密钥。其主要解决的问题是防止在密钥协商的过程中发生泄漏。比如在对称加密中，小蓝将需要发送的消息加密，然后告诉你密码是123balala,ok,对于其他人而言，很容易就能劫持到密码是123balala。那么在非对称的情况下，小蓝告诉所有人密码是123balala,对于中间人而言，拿到也没用，因为没有私钥。所以，非对称密钥其实主要解决了密钥分发的难题。如下图



其实我们经常都在使用非对称加密，比如使用多台服务器搭建大数据平台hadoop，为了方便多台机器设置免密登录，是不是就会涉及到秘钥分发。再比如搭建docker集群也会使用相关非对称加密算法。

混合加密

非对称加密算法，大多数是从数学问题演变而来，运算速度较慢。混合加密所谓取长补短。通信过程中使用RSA等解决密钥交换问题，然后使用随机数产生的在对称算法中的会话密钥，最后使用加密。对方使用私钥解密得到的秘闻取出会话密钥，这样就实现了密钥交换。



通过混淆加密等方式完成了机密性任务，作为Hack只需要伪造发布公钥或者作为中间人窃听密文。但是我们知道安全是四要素，还需要保证数据的完整性，身份认证等。

摘要

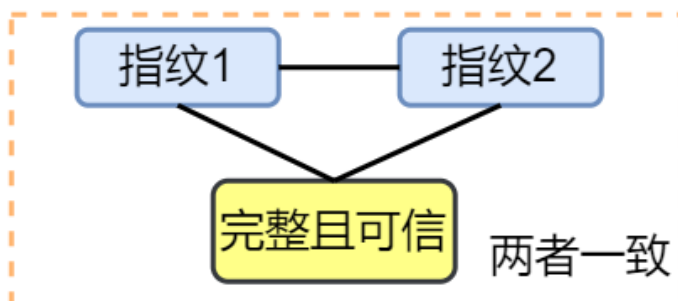
摘要算法可以理解作为一种特殊的"单向"加密算法，无密钥，不可逆。在平时项目中，应该大家都是用过MD5，SHA-1。但是在TLS中使用SHA-2。

假设小A转账5000给小C，小A加上SHA-2摘要。网站计算摘要并对比，如果一致则完整可信。



指纹1

指纹2



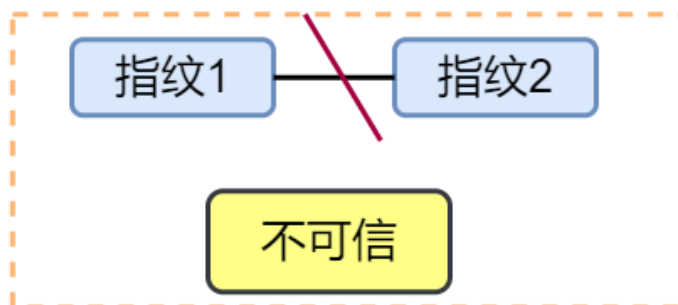
此时小B想修改小A给的money，这个时候网站计算摘要就会发现不一样，不可信



指纹1

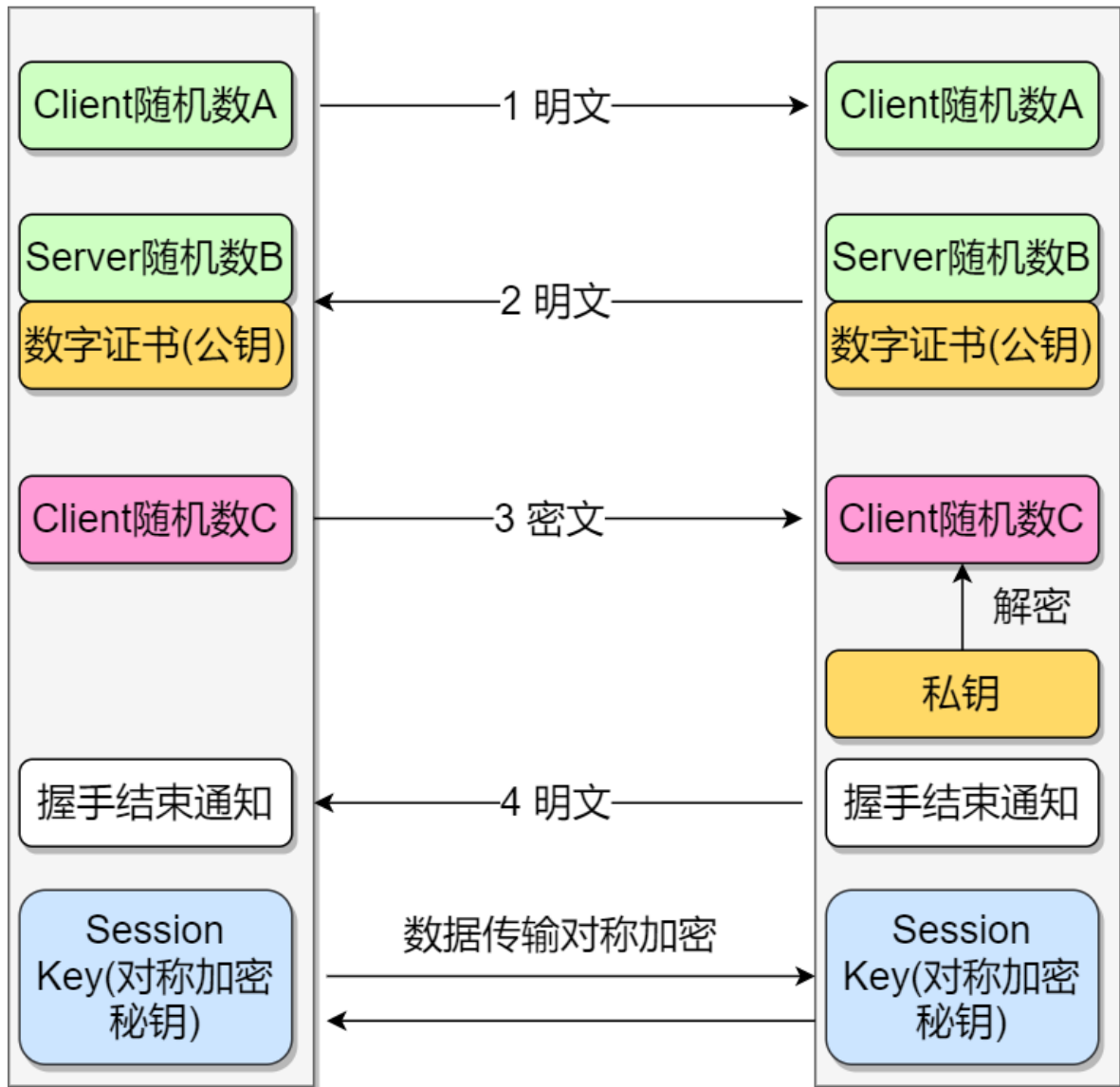
hack

指纹2



HTTPS请求建立连接过程









HTTPS(RSA) 握手阶段











注意:

1. 首先通过非对称加密建立通信过程
2. 在握手阶段, 为什么使用3个随机数, 一方面防止「随机数 C」被猜出, 另一方增加Session key随机性
3. Client发出支持的「对称/非对称加密」算法
4. server返回选用的「对称/非对称加密」算法
5. Client对算法进行确认
6. Server对算法进行确认

根据wireshark结果, 对TLS进一步剖析。TCP三次握手建立连接, 作为礼貌, Client先打招呼"Client Hello"。里面包含了Client的版本号、所支持的密码套件和随机数, 如下图所示

- [-] TLSv1.2 Record Layer: Handshake Protocol: Client Hello
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)
 - Length: 172
 - [-] Handshake Protocol: Client Hello
 - Handshake Type: Client Hello (1)
 - Length: 168
 - Version: TLS 1.2 (0x0303)
 - [-] Random
 - gmt_unix_time: May 19, 2020 09:46:36.0000000000        
 - random_bytes: 506b5f8651b190ab711c011c93fbc650d73e683b09b55219...
 - Session ID Length: 0
 - Cipher Suites Length: 42
 - + Cipher Suites (21 suites)
 - Compression Methods Length: 1
 - + Compression Methods (1 method)
 - Extensions Length: 85
 - [-] Extension: server_name
 - Type: server_name (0x0000)
 - Length: 26
 - + Server Name Indication extension

Server端表示尊重, 回复"Server Hello",同时进行版本校对, 给出随机数(Server Random), 从Client算法列表中选择一个密码套件, 在这里选择的"TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256".

- [-] Random
 - gmt_unix_time: May 19, 2020 09:46:38.0000000000        
 - random_bytes: b7d59be8d83cec8fe519a308ef325595d4758d51444f574e...
 - Session ID Length: 32
 - Session ID: 02de4f0941a55efe0ab31ffa4610972c7eb5c4efccddb40a...
 - Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
 - Compression Method: null (0)
 - Extensions Length: 19

这里的"TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256"什么意思呢

密码套件选择椭圆曲线加RSA、AES、SHA256

双方通过证书验证身份。因为本机服务器选用了ECDHE算法, 为了实现密钥交换算法, 它会发送证书后把椭圆曲线的公钥 (Server Params) 连带"Server Key Exchange"消息发送出去。

- [-] Secure Sockets Layer
 - [-] TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)
 - Length: 300
 - [-] Handshake Protocol: Server Key Exchange
 - Handshake Type: Server Key Exchange (12)
 - Length: 296

意思是, 刚才混合加密套件比较复杂, 给你个算法参数, 好好记住, 别弄丢了。

- [-] TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)

随后服务端回复"hello done"告知打招呼完毕

打完招呼完毕后, 客户端对证书进行核实。然后根据密码套件也生成椭圆曲线的公钥, 用"Client Key Exchange"消息发给服务器

- [-] Secure Sockets Layer
 - [-] TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
 - Content Type: Handshake (22)
 - Version: TLS 1.2 (0x0303)
 - Length: 37

此时客户端和服务端都有了密钥交换的两个参数(Client Params、ServerParams) , 然后通过 ECDHE 算法算出了一个新的值, 叫“Pre-Master”

有了主密钥和会话密钥, 客户端发送“Change Cipher Spec”和“Finished”消息, 最后将所有消息加上摘要发送给服务器验证。

服务器同样发送“Change Cipher Spec”和“Finished”消息, 握手结束, 开始进行HTTP请求与响应

4 初探域名

我们知道域名的出现让我们更容易记忆, 按照"."分割, 越靠近右边级别越高。域名本质是一个名字空间系统, 采用多级域名的方式区分不同的国家, 公司等, 作为一种身份的标识。

根域名服务器 (Root DNS Server) : 管理顶级域名服务器, 返回“com”“net”“cn”等顶级域名服务器的 IP 地址;

顶级域名服务器 (Top-level DNS Server) : 管理各自域名下的权威域名服务器, 比如 com 顶级域名服务器可以返回 apple.com 域名服务器的 IP 地址;

权威域名服务器 (Authoritative DNS Server) : 管理自己域名下主机的 IP 地址, 比如apple.com 权威域名服务器可以返回 www.apple.com 的 IP 地址**

6 HTTP特点小结

写到这里, 说它简单是假的, 简单的东西通常更具有扩展的可能性。根据需求的变更, 越来越复杂。

1: 灵活且易扩展, 他的头部字段很多都是可定制且可扩展

2: 应用广泛。各个领域都有涉及。"跨平台, 跨语言"

3: 无状态。没有记忆功能, 少功能即少占用资源。另外无状态更容易搭建集群, 通过负载均衡将请求转发到任意一台服务器。缺点是无法支持需要连续步骤的"事务"操作。我们知道TCP协议有11种状态, 不同状态代表通信过程中不同的含义。同样操作系统中的进程也有执行, 就绪, 活动阻塞等多种状态。但是HTTP全程都是"懵逼"无状态。比如小华请求服务器获取视频X, 服务器觉得可行就发给小华。小华还想获取视频Y, 这时服务器不会记录之前的状态, 也就不知道这两个请求是否是同一个, 所以小华还得告诉服务器自己的身份。

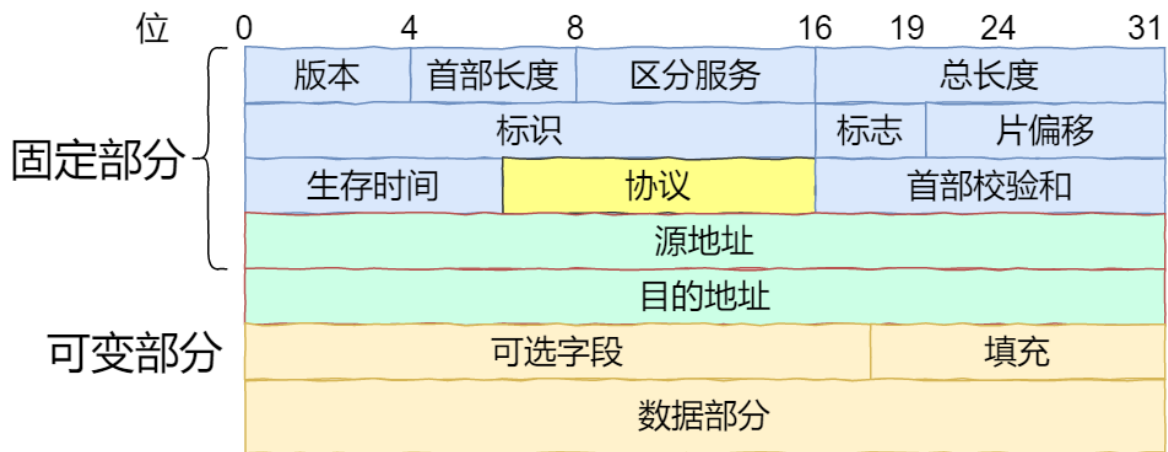
4: 明文。优点是能让开发人员通过wireshark工具更直观的调试。缺点即裸奔互联网, 没隐私可言。

5:可靠传输。HTTP为应用层协议, 基于TCP/IP, 而TCP为“可靠”传输协议, 因此HTTP能在请求应答中"可靠"传输数据。

6: 应用层协议。应用层协议很多, 其中常用的邮件协议SMTP, 上传下载文件ftp, 默认端口22/23, SSH远程登录(XSHELL)。这些应用层协议都太专一, 而HTTP通过各种头部字段, 实体数据的组合, 并综合缓存代理等功能, 不得不说是网络中的冠希哥。

7 HTTP识别(还原)

这里说的识别, 通过代码层面(libpcap封装)实现HTTP的识别, 也能进一步体现TCP/IP协议栈的分层特性。先看回忆一下IP头部格式。



注意头部中的协议字段，如果此字段值为0x0600则为TCP分组。当知道了是TCP分组后，是不是可以通过TCP头部中端口(80)就可以判断为HTTP呢，不能的，很多情况都会使用动态端口的方式进行部署。此时可以通过HTTP中的关键字进行判断。如果为HTTP，再通过头部字段中的"Content-type", charset等确认文本信息，编码方式，最后采用解码算法进行还原。

8 HTTPS(密文)识别

方法一也是比较直接的方法是直接通过抓包工具，插件配置即可。这里想给大家分享另一种思路和在Linux持续捕包的方法。

- 数据集采集

使用python的dpkt库(pip install dpkt即可)，dpkt库方便对每一层协议进行拆解，同时也能进行流的拆分以及特征的提取。下面举一个通过无头浏览的方式自动化采集流量(ps如果需要较大规模的流量采集则可以考虑使用docker集群的方式)

```
def read_pcap_file_to_dict(path):
    """
    读取pcap文件
    :param path: pcap文件全路径
    :return: 分开之后的tcp流，读取的pcap文件
    """
    readfile = open(path, 'rb')
    pcap = dpkt.pcap.Reader(readfile)
    result_pcap = []
    tcp_stream = {}
    for timestamp, buf in pcap:
        result_pcap.append((timestamp, buf))
        try:
            eth = dpkt.ethernet.Ethernet(buf) #处理二层
            if not isinstance(eth.data, dpkt.ip.IP):
                print('Non IP Packet type not supported %s\n' % eth.data.__class__.__name__)
                continue
            ip = eth.data
            if isinstance(ip.data, dpkt.tcp.TCP):
                # Set the TCP data
                tcp = ip.data
                tmp_out = inet_to_str(ip.src) + ' ' + inet_to_str(ip.dst) + ' ' + str(tcp.sport) + ' ' + str(tcp.dport)
                if tmp_out not in tcp_stream.keys():
                    tcp_stream[tmp_out] = [(timestamp, buf)]
                else:
                    tcp_stream[tmp_out].append((timestamp, buf))
            except Exception as e:
                print(e.args)
    return tcp_stream, result_pcap
```

- 根据所提特征生成npz(实际上是numpy提供的数组存储方式)
- 使用开源sklearn库进行模型训练并识别预测，此处假设使用SVM(仅使用默认参数)


```

from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn import tree
import joblib
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
import data
import datetime

# 根据现有的svm模型进行预测
def predict(model_path, test_data, test_label):
    model = joblib.load(model_path)
    predict_res = model.predict(test_data)
    acc_score = accuracy_score(test_label, predict_res)
    prec_score = precision_score(test_label, predict_res, average=None, )
    reca_score = recall_score(test_label, predict_res, average=None)
    f_one_score = f1_score(test_label, predict_res, average=None)
    res = 'accuracy:{}\tprecision:{}\trecall:{}\tf1:{}'.format(acc_score, prec_score, reca_score,
f_one_score)
    print(res)

```

- 识别结果(参数进行适度调整定会有更好的效果)

```

accuracy:0.9703947368421053
precision:[0.94936709 0.98026316 1.          0.95541401]
recall:[0.98684211 0.98026316 0.92763158 0.98684211]
f1:[0.96774194 0.98026316 0.96245734 0.97087379]

```

9 HTTP面试题测试

希望大家看完本文，下面的这些面试是不是可以秒杀了

- Get和Post区别
- HTTP与HTTPS区别
- HTTP通信过程
- 浏览器输入一个地址。到页面展示中间经历了哪些步骤？
- cookies机制和session机制的区别：
- HTTP请求报文与响应报文格式
- 一次完整的HTTP请求所经历的7个步骤
- HTTP优化方案
- 不同版本的HTTP区别
- HTTP优点缺点
- URI和URL的区别
- 如何判断是否为http
- HTTP 1.1引入分块传输编码提供了以下几点好处
- 长连接与短连接的区别，以及应用场景
- 常见web攻击
- 站内跳转和外部重定向有何区别
- HTTP的keep-alive是干什么的？
- 关于Http 2.0 你知道多少？

- 讲讲304缓存的原理
- HTTP与RPC异同
- 从传输协议来说

RPC既可以基于TCP也可以基于HTTP协议，但是HTTP通常都是基于HTTP

- 从性能消耗来说

RPC可以基于thrift实现高效二进制传输。HTTP大部分通过json实现，无论从字节大小还是序列化耗时都比thrift耗时

- 从负载均衡来说

RPC基本上自带负载均衡策略，而HTTP需要配置Nginx实现。

唠嗑

第一篇文章能肝到这么长，也终于体会到各位大佬写文的不容易，不想被「白嫖」，文末点个「在看」吧，让我们一起「看世界」。

Persist

<https://www.chainnews.com/articles/401950499827.htm>

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Basics_of_HTTP/Evolution_of_HTTP