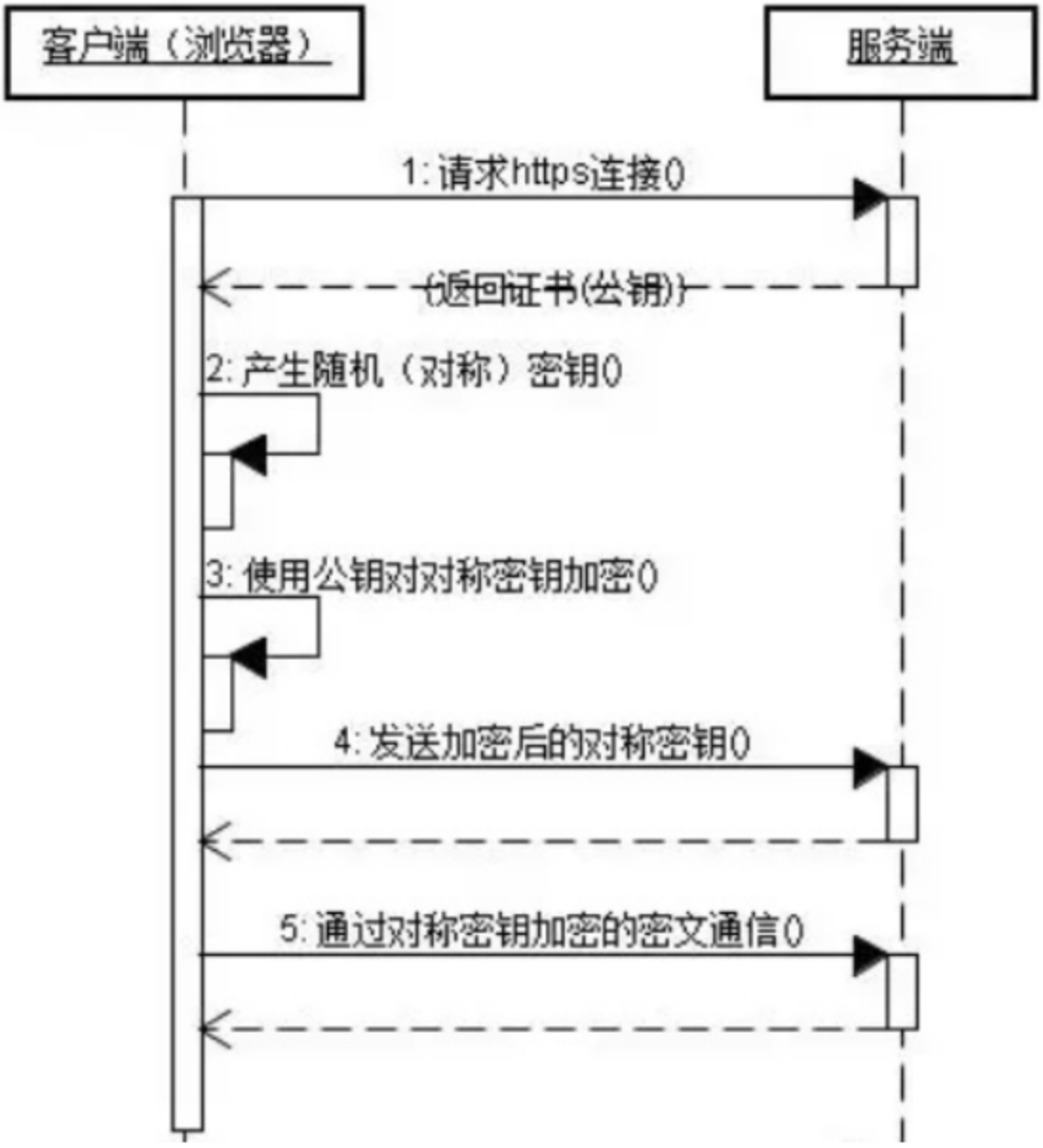
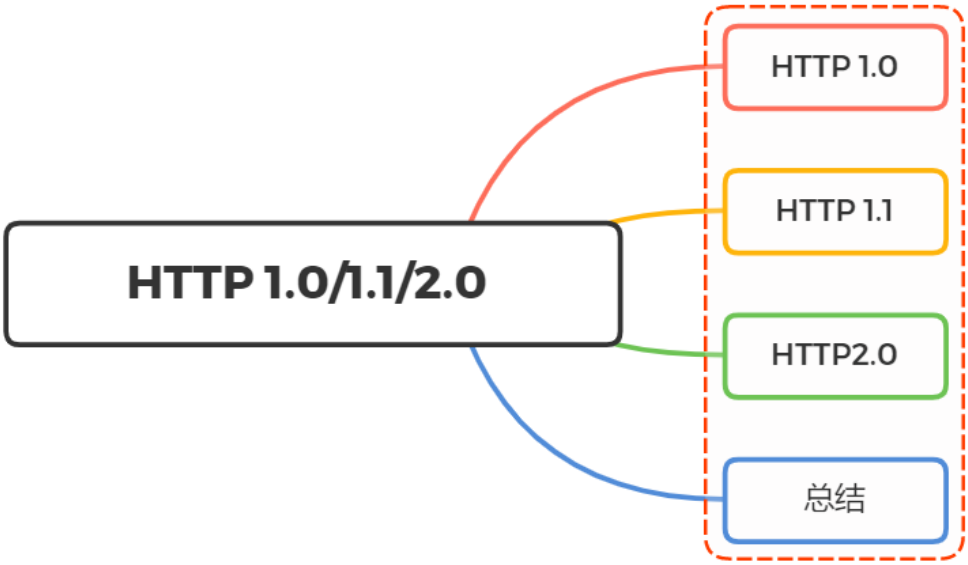


- 首先客户端通过URL访问服务器建立SSL连接
- 服务端收到客户端请求后，会将网站支持的证书信息（证书中包含公钥）传送一份给客户端
- 客户端的服务器开始协商SSL连接的安全等级，也就是信息加密的等级
- 客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站
- 服务器利用自己的私钥解密出会话密钥
- 服务器利用会话密钥加密与客户端之间的通信

8.3. 区别

- HTTPS是HTTP协议的安全版本，HTTP协议的数据传输是明文的，是不安全的，HTTPS使用了SSL/TLS协议进行了加密处理，相对更安全
- HTTP 和 HTTPS 使用连接方式不同，默认端口也不一样，HTTP是80，HTTPS是443
- HTTPS 由于需要设计加密以及多次握手，性能方面不如 HTTP
- HTTPS需要SSL，SSL 证书需要钱，功能越强大的证书费用越高

9. 说说 HTTP1.0/1.1/2.0 的区别?



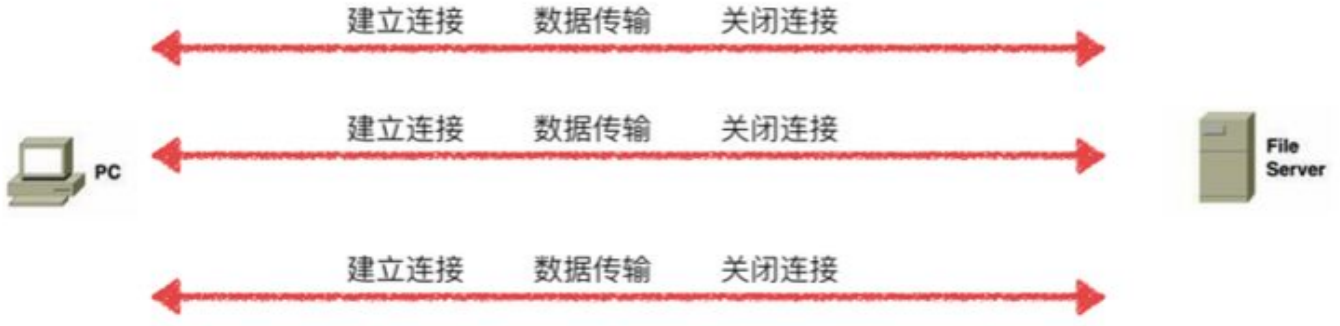
9.1. HTTP1.0

`HTTP` 协议的第二个版本，第一个在通讯中指定版本号的HTTP协议版本

`HTTP 1.0` 浏览器与服务器只保持短暂的连接，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

简单来讲，每次与服务器交互，都需要新开一个连接



例如，解析 `html` 文件，当发现文件中存在资源文件的时候，这时候又创建单独的链接

最终导致，一个 `html` 文件的访问包含了多次的请求和响应，每次请求都需要创建连接、关系连接

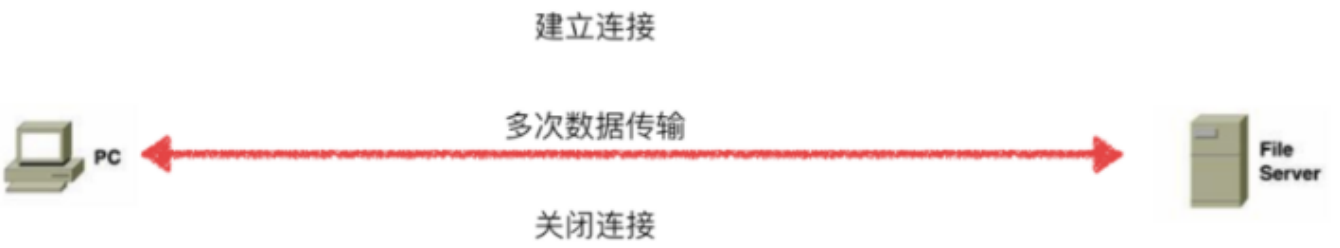
这种形式明显造成了性能上的缺陷

如果需要建立长连接，需要设置一个非标准的Connection字段 `Connection: keep-alive`

9.2. HTTP1.1

在 `HTTP1.1` 中，默认支持长连接（ `Connection: keep-alive` ），即在一个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

建立一次连接，多次请求均由这个连接完成



这样，在加载 `html` 文件的时候，文件中多个请求和响应就可以在一个连接中传输

同时， `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

同时， `HTTP1.1` 在 `HTTP1.0` 的基础上，增加更多的请求头和响应头来完善的功能，如下：

- 引入了更多的缓存控制策略，如If-Unmodified-Since, If-Match, If-None-Match等缓存头来控制缓存策略
- 引入range，允许值请求资源某个部分
- 引入host，实现了在一台WEB服务器上可以在同一个IP地址和端口号上使用不同的主机名来创建多

个虚拟WEB站点

并且还添加了其他的请求方法：`put`、`delete`、`options`...

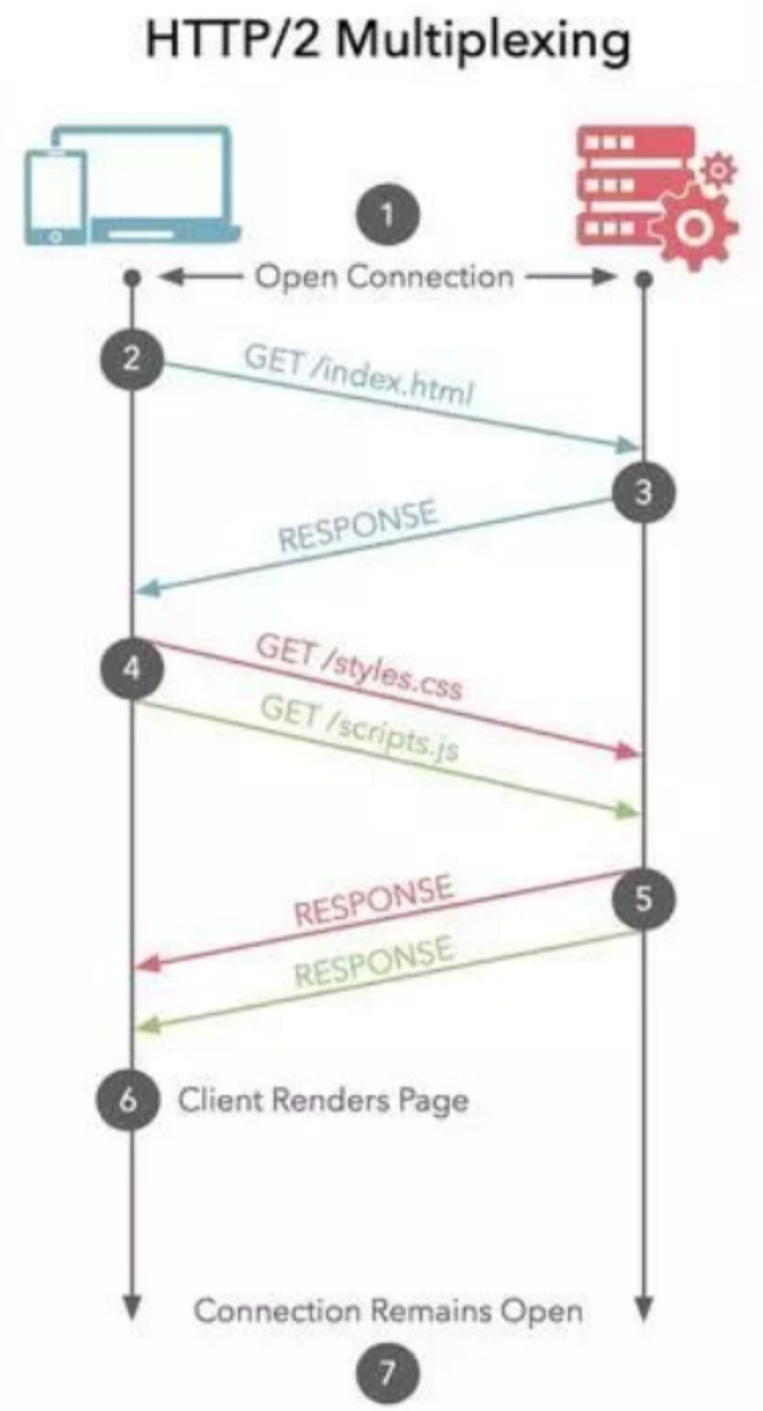
9.3. HTTP2.0

而`HTTP2.0`在相比之前版本，性能上有很大的提升，如添加了一个特性：

- 多路复用
- 二进制分帧
- 首部压缩
- 服务器推送

9.3.1. 多路复用

`HTTP/2` 复用`TCP`连接，在一个连接里，客户端和浏览器都可以**同时**发送多个请求或回应，而且不用按照顺序一一对应，这样就避免了”队头堵塞”



上图中，可以看到第四步中 `css` 、 `js` 资源是同时发送到服务端

9.3.2. 二进制分帧

帧是 `HTTP2` 通信中最小单位信息

`HTTP/2` 采用二进制格式传输数据，而非 `HTTP 1.x` 的文本格式，解析起来更高效

将请求和响应数据分割为更小的帧，并且它们采用二进制编码

`HTTP2` 中，同域名下所有通信都在单个连接上完成，该连接可以承载任意数量的双向数据流

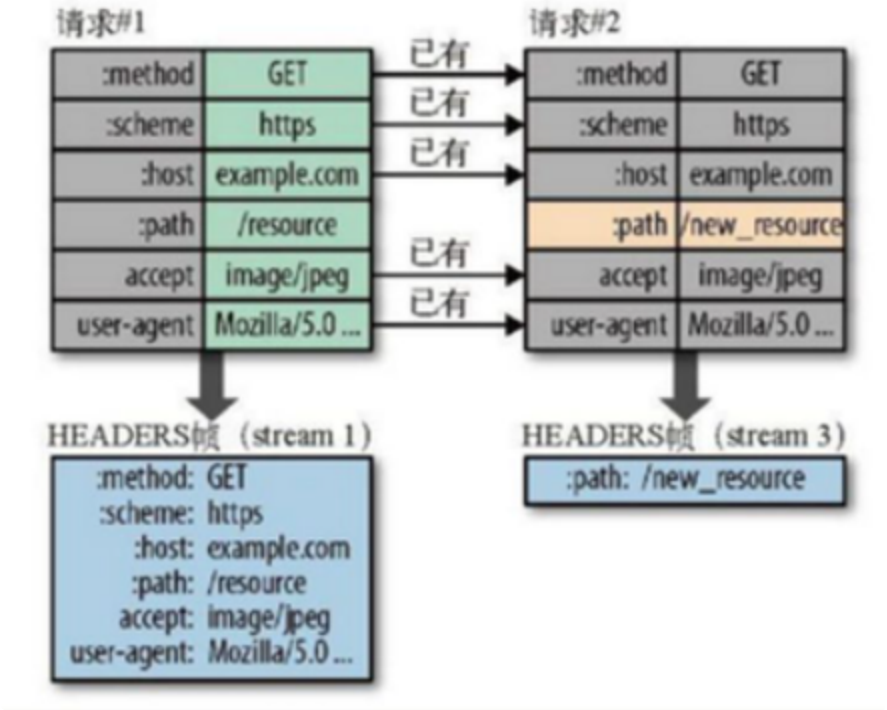
每个数据流都以消息的形式发送，而消息又由一个或多个帧组成。多个帧之间可以乱序发送，根据帧首部的流标识可以重新组装，这也是多路复用同时发送数据的实现条件

9.3.3. 首部压缩

`HTTP/2` 在客户端和服务器端使用“首部表”来跟踪和存储之前发送的键值对，对于相同的数据，不再通过每次请求和响应发送

首部表在 `HTTP/2` 的连接存续期内始终存在，由客户端和服务器共同渐进地更新

例如：下图中的两个请求， 请求一发送了所有的头部字段，第二个请求则只需要发送差异数据，这样可以减少冗余数据，降低开销



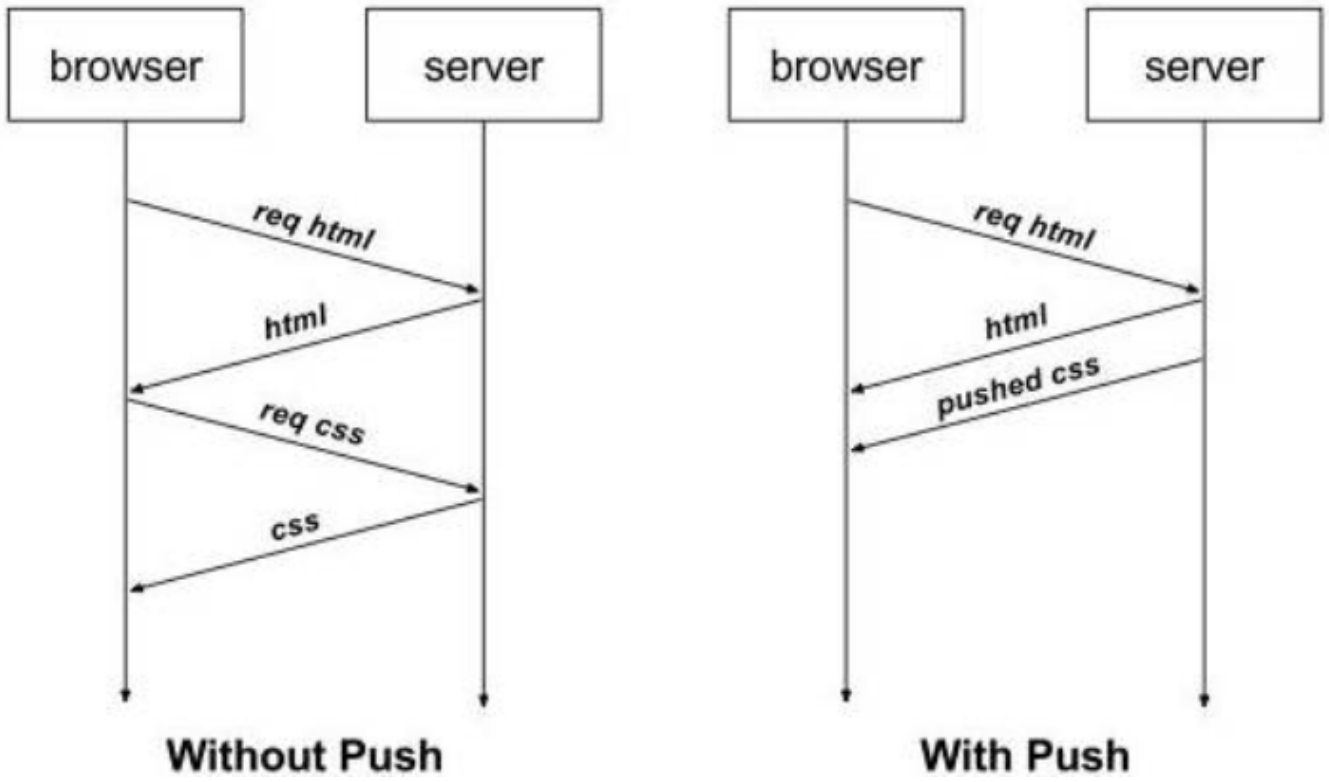
9.3.4. 服务器推送

`HTTP2` 引入服务器推送，允许服务端推送资源给客户端

服务器会顺便把一些客户端需要的资源一起推送到客户端，如在响应一个页面请求中，就可以随同页面的其它资源

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



9.4. 总结

HTTP1.0：

- 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

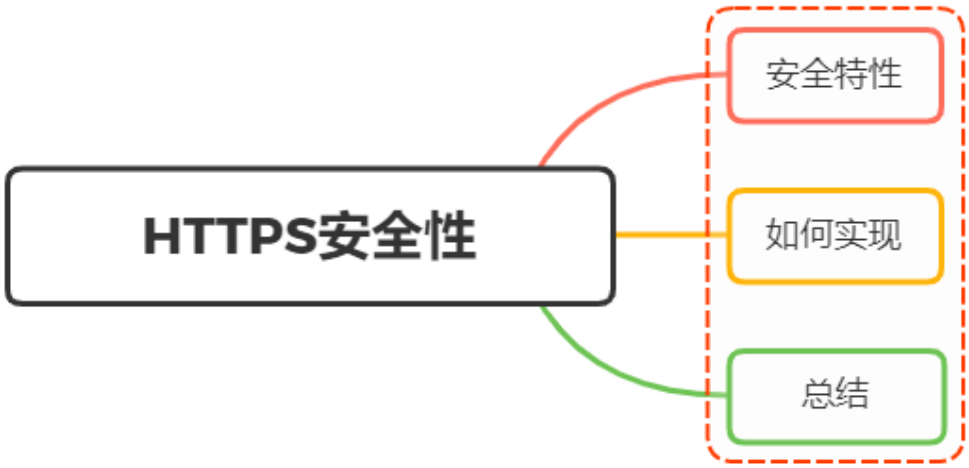
HTTP1.1：

- 引入了持久连接，即TCP连接默认不关闭，可以被多个请求复用
- 在同一个TCP连接里面，客户端可以同时发送多个请求
- 虽然允许复用TCP连接，但是同一个TCP连接里面，所有的数据通信是按次序进行的，服务器只有处理完一个请求，才会接着处理下一个请求。如果前面的处理特别慢，后面就会有许多请求排队等着
- 新增了一些请求方法
- 新增了一些请求头和响应头

HTTP2.0：

- 采用二进制格式而非文本格式
- 完全多路复用，而非有序并阻塞的、只需一个连接即可实现并行
- 使用报头压缩，降低开销
- 服务器推送

10. 为什么说HTTPS比HTTP安全? HTTPS是如何保证安全的?



10.1. 安全特性

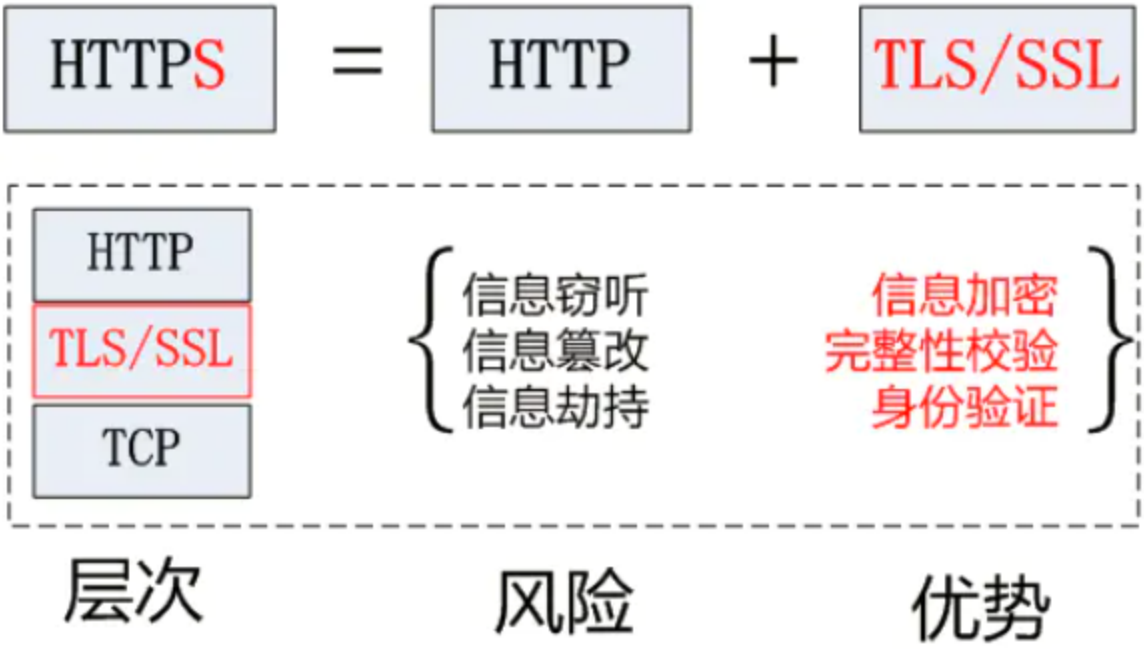
在上篇文章中，我们了解到 `HTTP` 在通信过程中，存在以下问题：

- 通信使用明文（不加密），内容可能被窃听
- 不验证通信方的身份，因此有可能遭遇伪装

而 `HTTPS` 的出现正是解决这些问题， `HTTPS` 是建立在 `SSL` 之上，其安全性由 `SSL` 来保证

在采用 `SSL` 后， `HTTP` 就拥有了 `HTTPS` 的加密、证书和完整性保护这些功能

> SSL(Secure Sockets Layer 安全套接字协议),及其继任者传输层安全（Transport Layer Security，TLS）是为网络通信提供安全及数据完整性的一种安全协议



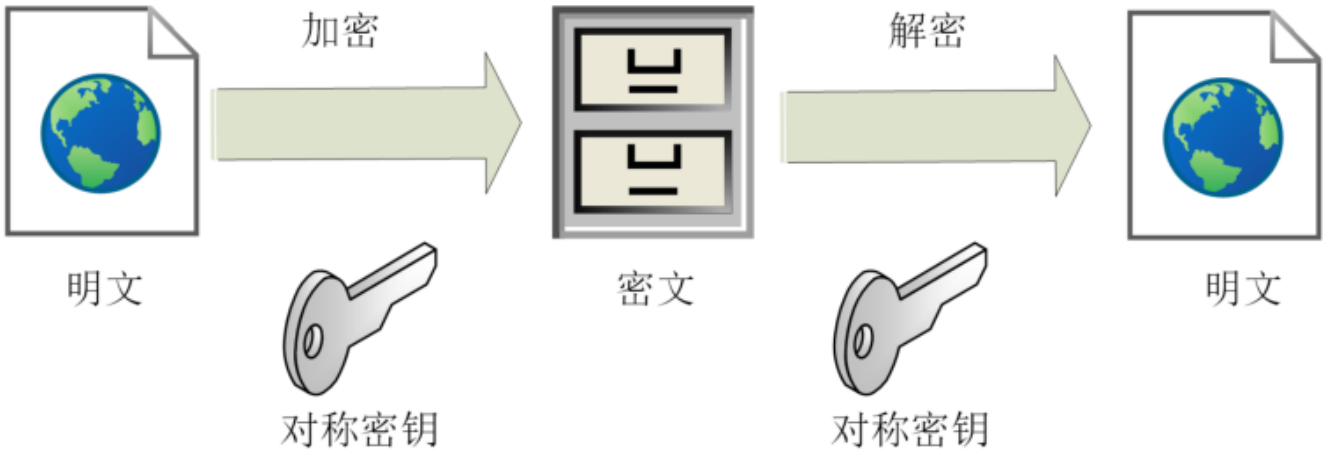
10.2. 如何做

`SSL` 的实现这些功能主要依赖于三种手段：

- 对称加密：采用协商的密钥对数据加密
- 非对称加密：实现身份认证和密钥协商
- 摘要算法：验证信息的完整性
- 数字签名：身份验证

10.2.1. 对称加密

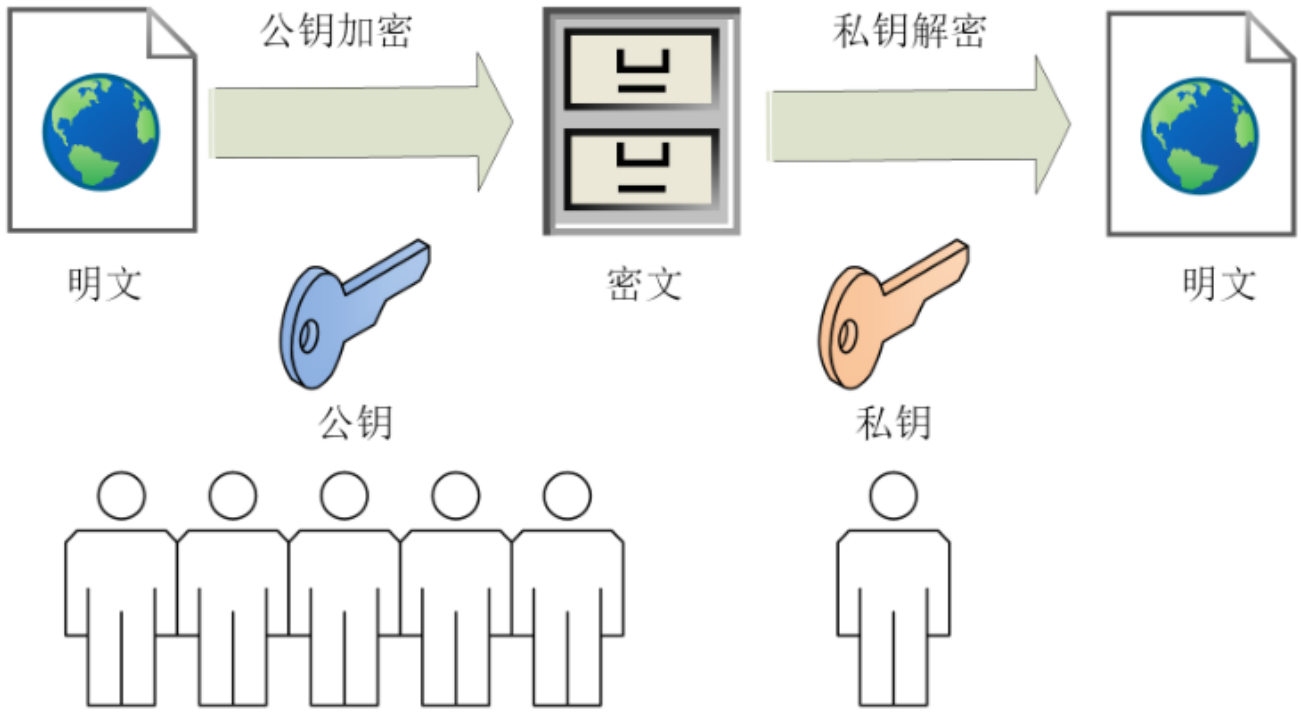
对称加密指的是加密和解密使用的秘钥都是同一个，是对称的。只要保证了密钥的安全，那整个通信过程就可以说具有了机密性



10.2.2. 非对称加密

非对称加密，存在两个秘钥，一个叫公钥，一个叫私钥。两个秘钥是不同的，公钥可以公开给任何人使用，私钥则需要保密

公钥和私钥都可以用来加密解密，但公钥加密后只能用私钥解
密，反过来，私钥加密后也只能用公钥解密



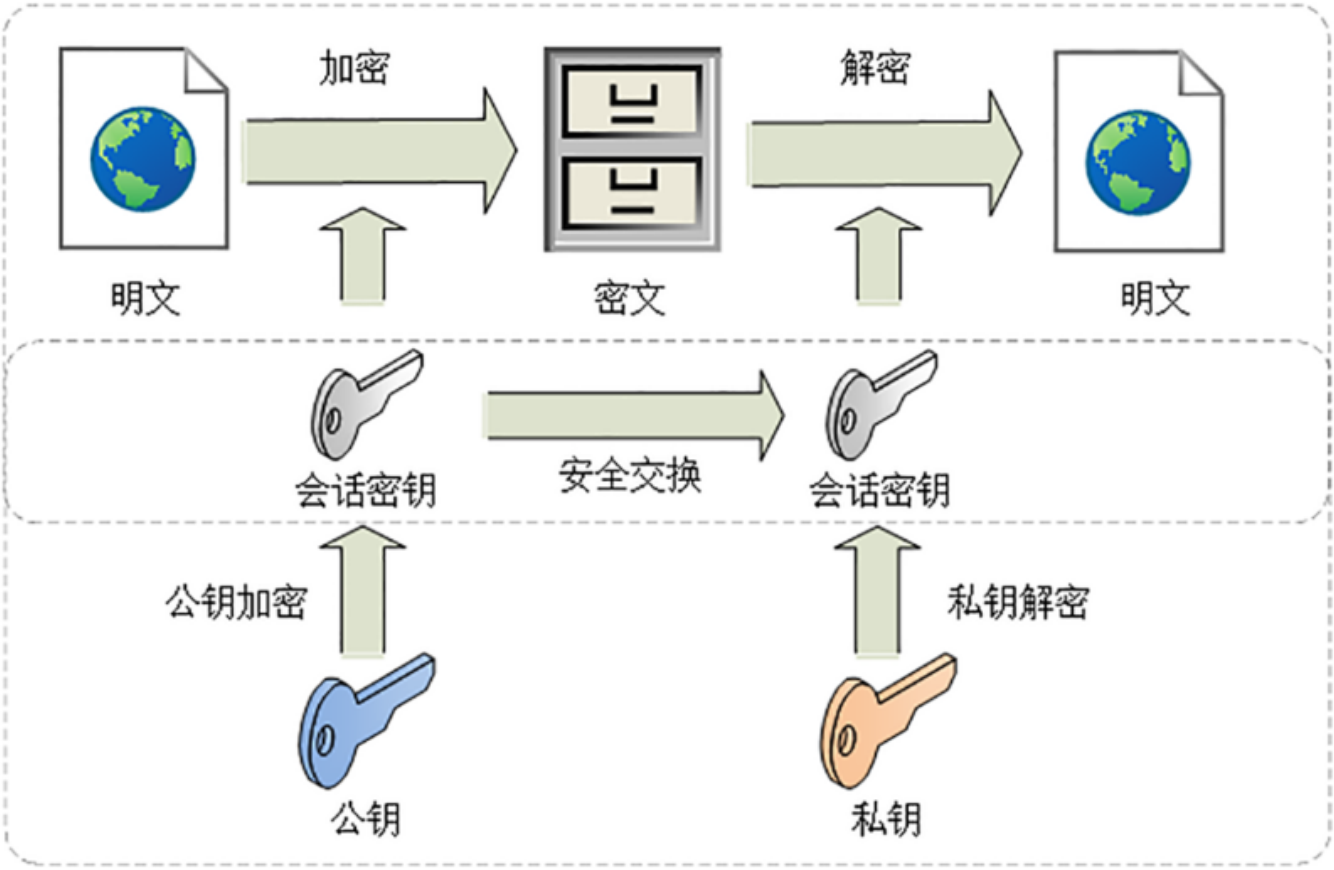
10.2.3. 混合加密

在 HTTPS 通信过程中，采用的是对称加密+非对称加密，也就是混合加密

在对称加密中讲到，如果能够保证了密钥的安全，那整个通信过程就可以说具有了机密性

而 HTTPS 采用非对称加密解决秘钥交换的问题

具体做法是发送密文的一方使用对方的公钥进行加密处理“对称的密钥”，然后对方用自己的私钥解密拿到“对称的密钥”



这样可以确保交换的密钥是安全的前提下，使用对称加密方式进行通信

10.2.3.1. 举个例子

网站秘密保管私钥，在网上任意分发公钥，你想要登录网站只要用公钥加密就行了，密文只能由私钥持有者才能解密。而黑客因为没有私钥，所以就无法破解密文

上述的方法解决了数据加密，在网络传输过程中，数据有可能被篡改，并且黑客可以伪造身份发布公钥，如果你获取到假的公钥，那么混合加密也并无多大用处，你的数据扔被黑客解决

因此，在上述加密的基础上仍需加上完整性、身份验证的特性，来实现真正的安全，实现这一功能则是摘要算法

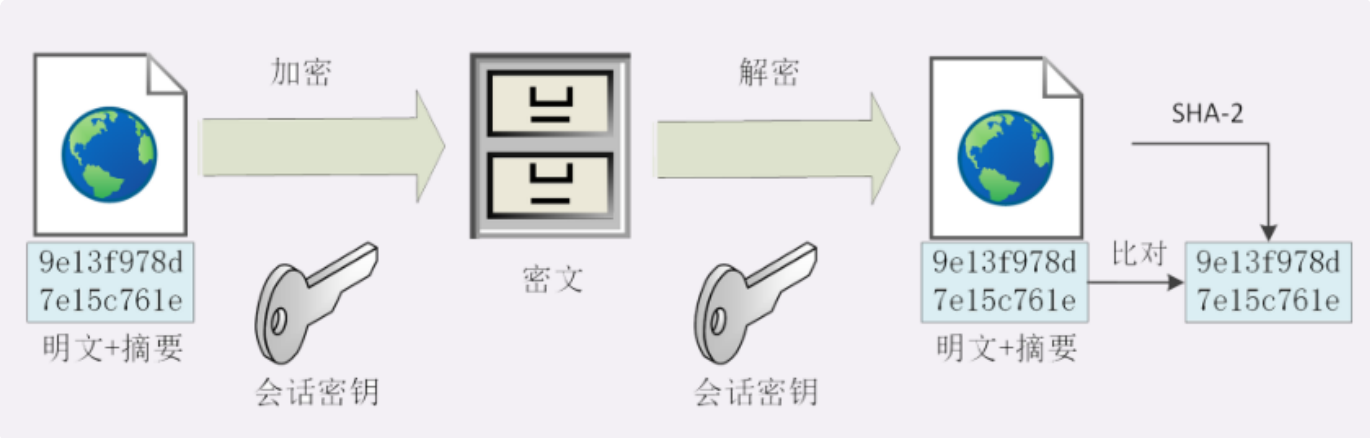
10.2.4. 摘要算法

实现完整性的手段主要是摘要算法，也就是常说的散列函数、哈希函数

可以理解成一种特殊的压缩算法，它能够把任意长度的数据“压缩”成固定长度、而且独一无二的“摘要”字符串，就好像是给这段数据生成了一个数字“指纹”

摘要算法保证了“数字摘要”和原文是完全等价的。所以，我们只要在原文后附上它的摘要，就能够保证数据的完整性

比如，你发了条消息：“转账 1000 元”，然后再加上一个 SHA-2 的摘要。网站收到后也计算一下消息的摘要，把这两份“指纹”做个对比，如果一致，就说明消息是完整可信的，没有被修改

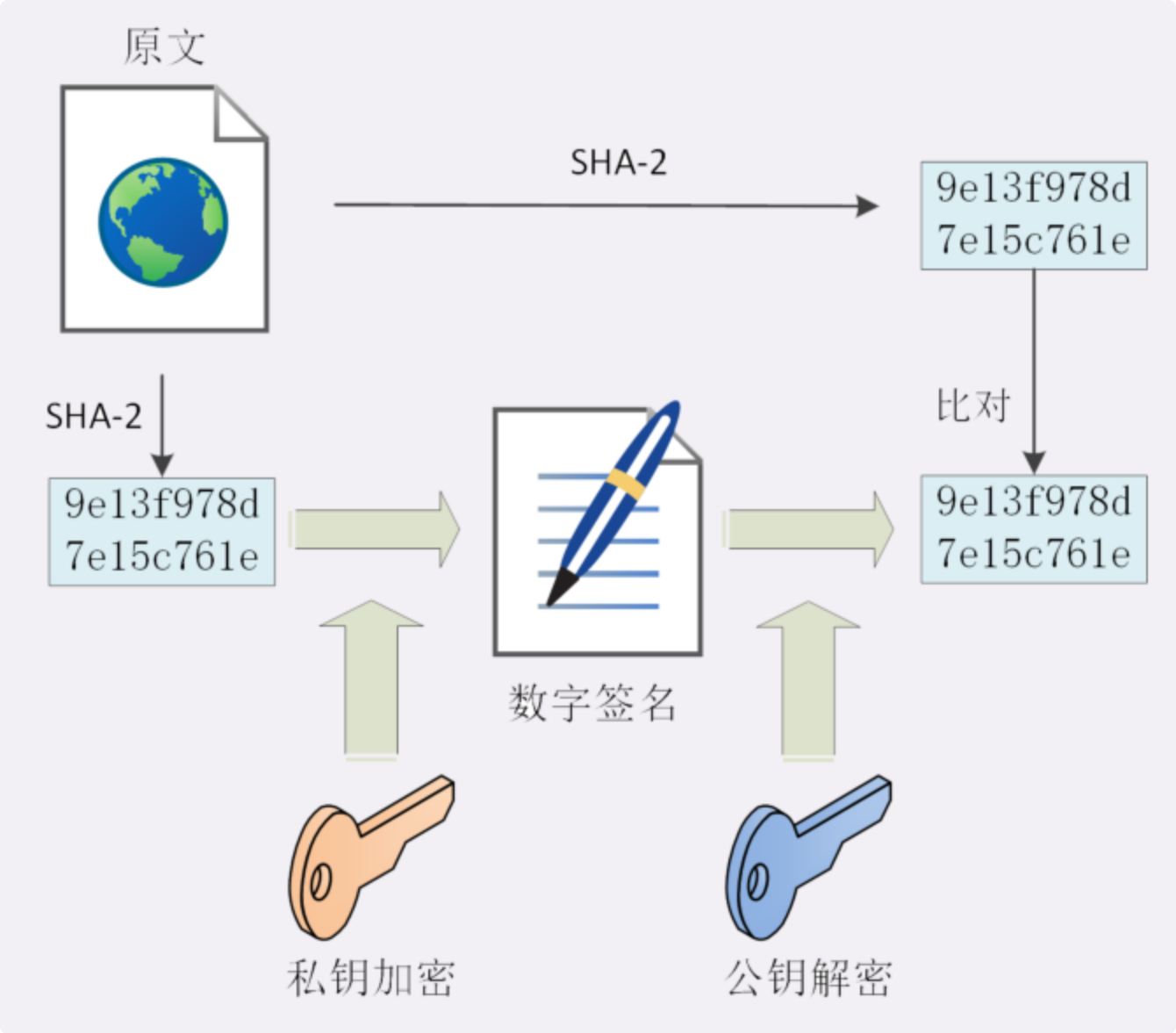


10.2.5. 数字签名

数字签名能确定消息确实是由发送方签名并发出来的，因为别人假冒不了发送方的签名

原理其实很简单，就是用私钥加密，公钥解密

签名和公钥一样完全公开，任何人都可以获取。但这个签名只有用私钥对应的公钥才能解开，拿到摘要后，再比对原文验证完整性，就可以像签署文件一样证明消息确实是你发的



和消息本身一样，因为谁都可以发布公钥，我们还缺少防止黑客伪造公钥的手段，也就是说，怎么判断这个公钥就是你的公钥

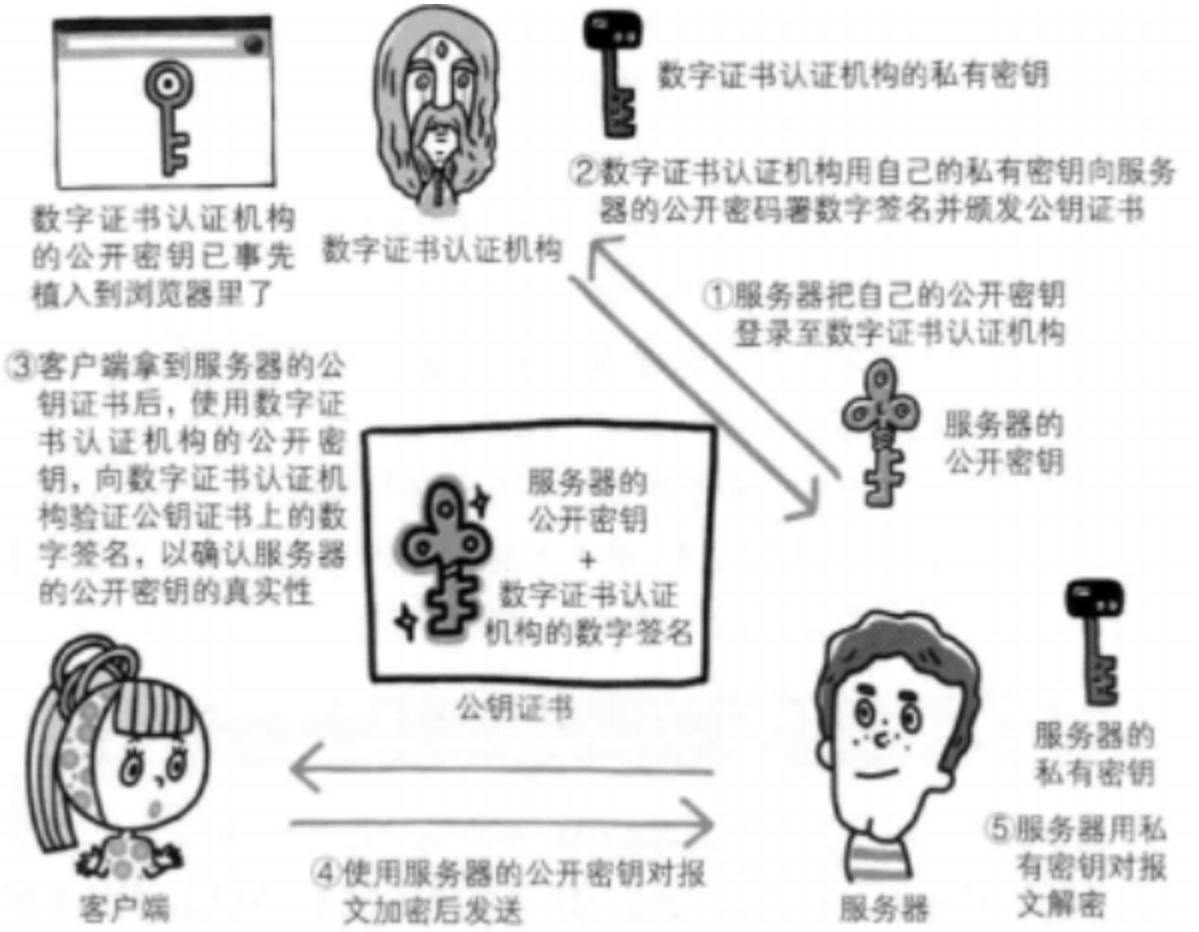
这时候就需要一个第三方，就是证书验证机构

10.2.6. CA验证机构

数字证书认证机构处于客户端与服务器双方都可信赖的第三方机构的立场

CA 对公钥的签名认证要求包括序列号、用途、颁发者、有效时间等等，把这些打成一个包再签名，完整地证明公钥关联的各种信息，形成“数字证书”

流程如下图：



- 服务器的运营人员向数字证书认证机构提出公开密钥的申请
- 数字证书认证机构在判明提出申请者的身份之后，会对已申请的公开密钥做数字签名
- 然后分配这个已签名的公开密钥，并将该公开密钥放入公钥证书后绑定在一起
- 服务器会将这份由数字证书认证机构颁发的数字证书发送给客户端，以进行非对称加密方式通信

接到证书的客户端可使用数字证书认证机构的公开密钥，对那张证书上的数字签名进行验证，一旦验证通过，则证明

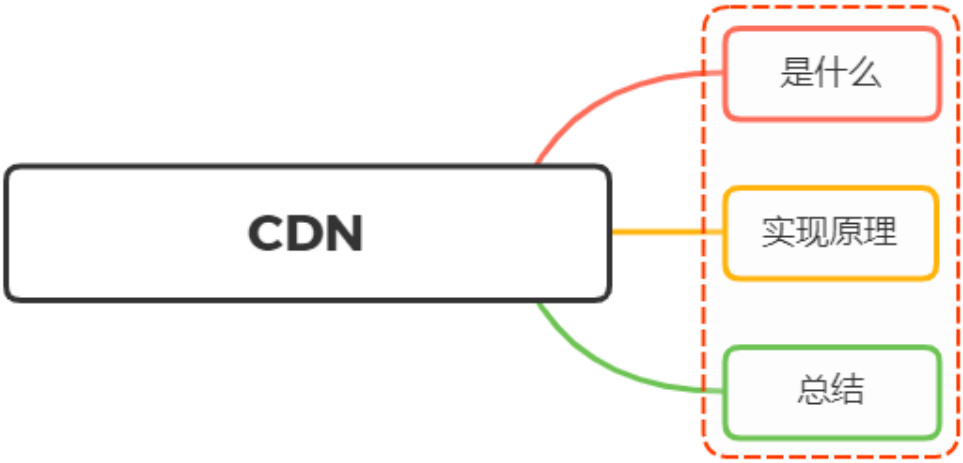
- 认证服务器的公开密钥的是真实有效的数字证书认证机构
- 服务器的公开密钥是值得信赖的

10.3. 总结

可以看到，`HTTPS` 与 `HTTP` 虽然只差一个 `SSL`，但是通信安全得到了大大的保障，通信的四大特性都以解决，解决方式如下：

- 机密性：混合算法
- 完整性：摘要算法
- 身份认证：数字签名
- 不可否定：数字签名

11. 如何理解CDN？说说实现原理？



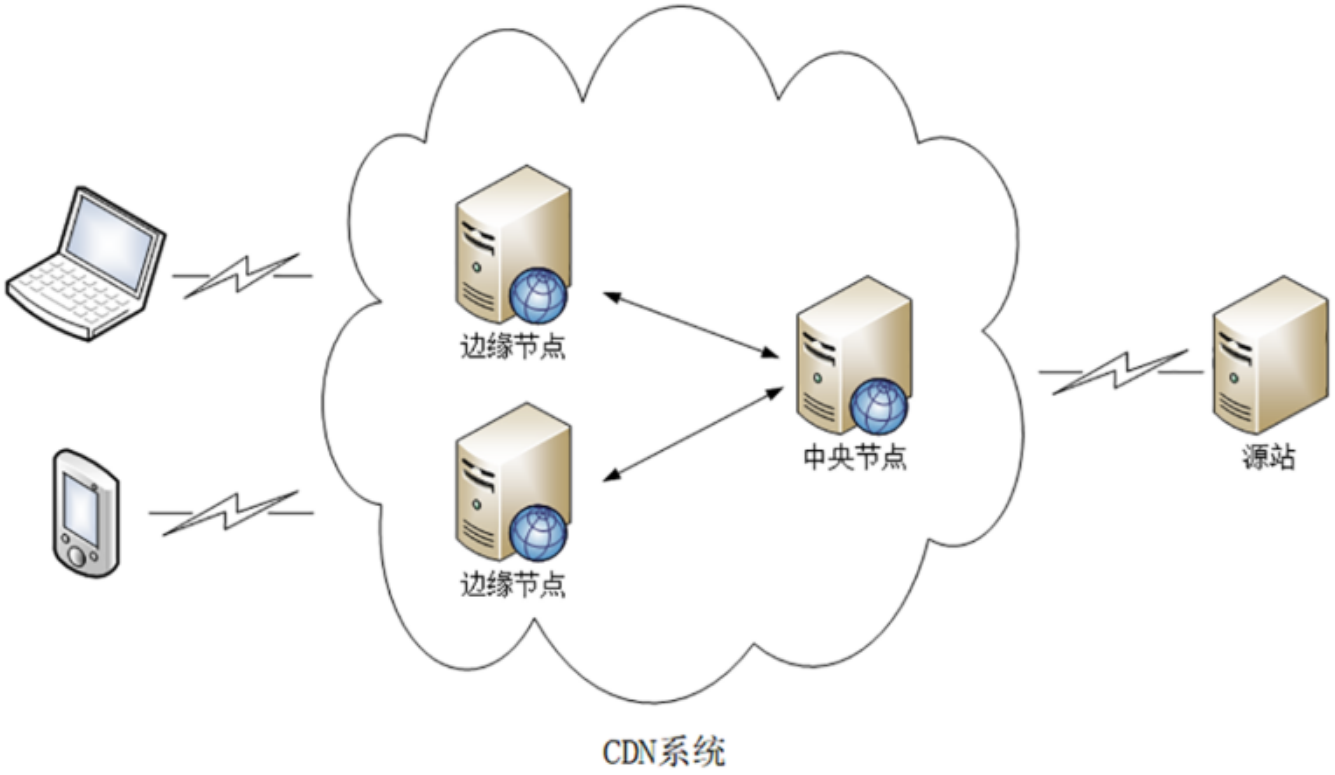
11.1. 是什么

CDN (全称 Content Delivery Network)，即内容分发网络

构建在现有网络基础之上的智能虚拟网络，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，使用户就近获取所需内容，降低网络拥塞，提高用户访问响应速度和命中率。 `CDN` 的关键技术主要有内容存储和分发技术

简单来讲， `CDN` 就是根据用户位置分配最近的资源

于是，用户在上网的时候不用直接访问源站，而是访问离他“最近的”一个 CDN 节点，术语叫**边缘节点**，其实就是缓存了源站内容的代理服务器。如下图：



11.2. 原理分析

在没有应用 `CDN` 时，我们使用域名访问某一个站点时的路径为

> 用户提交域名→浏览器对域名进行解释→ `DNS` 解析得到目的主机的IP地址→根据IP地址访问发出请求→得到请求数据并回复

应用 `CDN` 后， `DNS` 返回的不再是 `IP` 地址，而是一个 `CNAME` (Canonical Name) 别名记录，指向 `CDN` 的全局负载均衡

`CNAME` 实际上在域名解析的过程中承担了中间人（或者说代理）的角色，这是 `CDN` 实现的关键

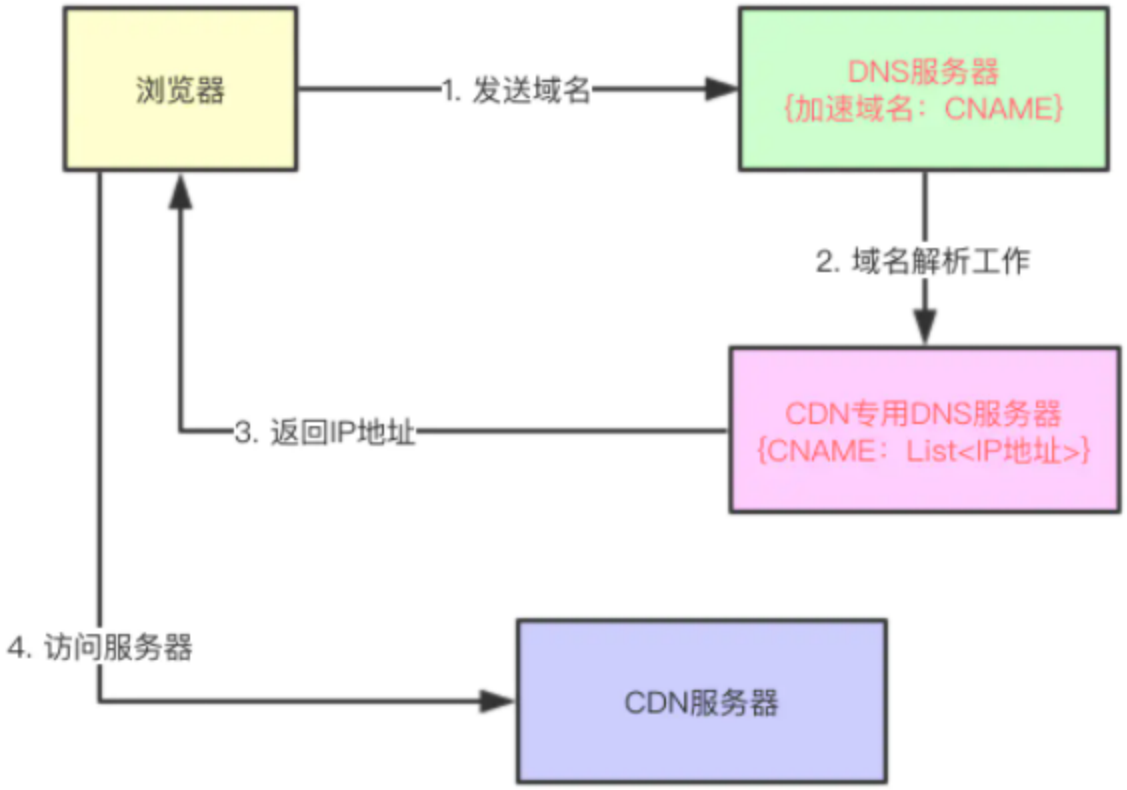
11.2.1.1. 负载均衡系统

由于没有返回 `IP` 地址，于是本地 `DNS` 会向负载均衡系统再发送请求 ，则进入到 `CDN` 的全局负载均衡系统进行智能调度：

- 看用户的 IP 地址，查表得知地理位置，找相对最近的边缘节点
- 看用户所在的运营商网络，找相同网络的边缘节点
- 检查边缘节点的负载情况，找负载较轻的节点
- 其他，比如节点的“健康状况”、服务能力、带宽、响应时间等

结合上面的因素，得到最合适的边缘节点，然后把这个节点返回给用户，用户就能够就近访问 CDN 的缓存代理

整体流程如下图：



11.2.1.2. 缓存代理

缓存系统是 CDN 的另一个关键组成部分，缓存系统会有选择地缓存那些最常用的那些资源

其中有两个衡量 CDN 服务质量的指标：

- 命中率：用户访问的资源恰好在缓存系统里，可以直接返回给用户，命中次数与所有访问次数之比
- 回源率：缓存里没有，必须用代理的方式回源站取，回源次数与所有访问次数之比

缓存系统也可以划分出层次，分成一级缓存节点和二级缓存节点。一级缓存配置高一些，直连源站，二级缓存配置低一些，直连用户

回源的时候二级缓存只找一级缓存，一级缓存没有才回源站，可以有效地减少真正的回源

现在的商业 CDN 命中率都在 90% 以上，相当于把源站的服务能力放大了 10 倍以上

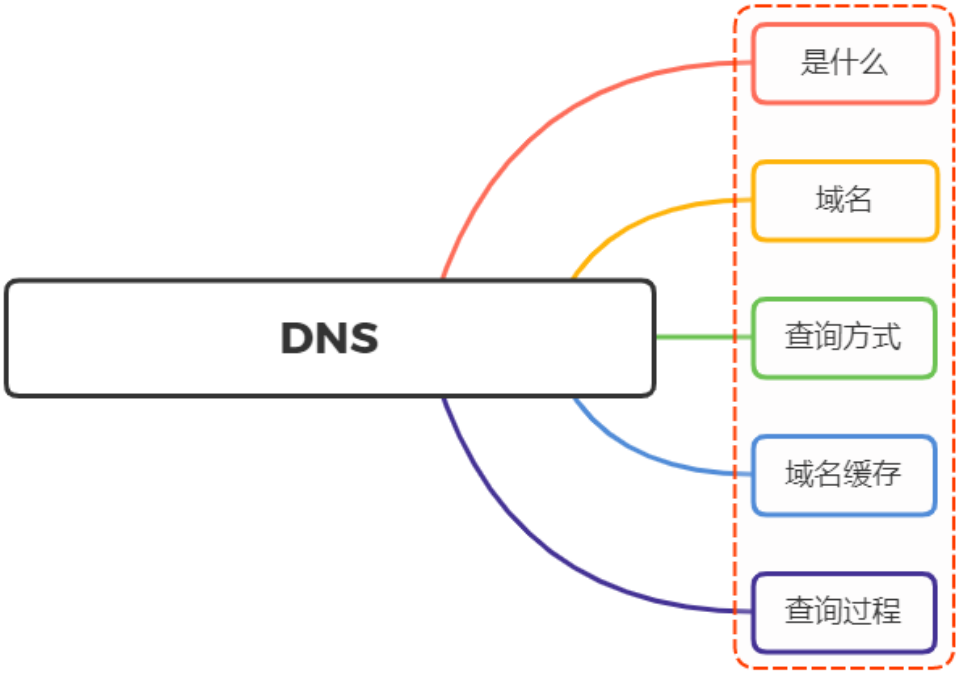
11.3. 总结

CDN 目的是为了改善互联网的服务质量，通俗一点说其实就是提高访问速度

`CDN` 构建了全国、全球级别的专网，让用户就近访问专网里的边缘节点，降低了传输延迟，实现了网站加速

通过 `CDN` 的负载均衡系统，智能调度边缘节点提供服务，相当于 `CDN` 服务的大脑，而缓存系统相当于 `CDN` 的心脏，缓存命中直接返回给用户，否则回源

12. DNS协议 是什么？说说DNS 完整的查询过程？

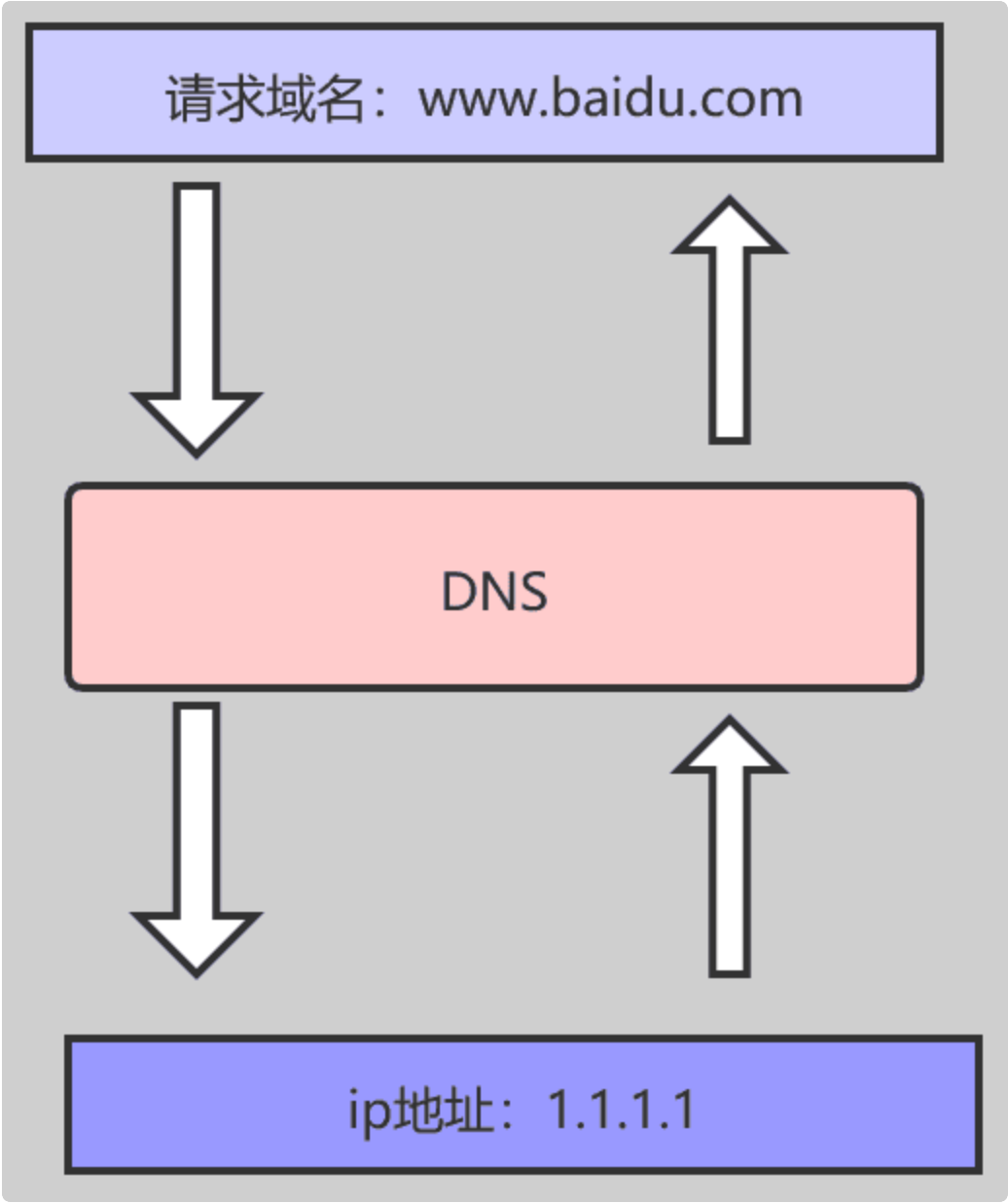


12.1. 是什么

DNS（Domain Names System），域名系统，是互联网一项服务，是进行域名和与之相对应的 IP 地址进行转换的服务器

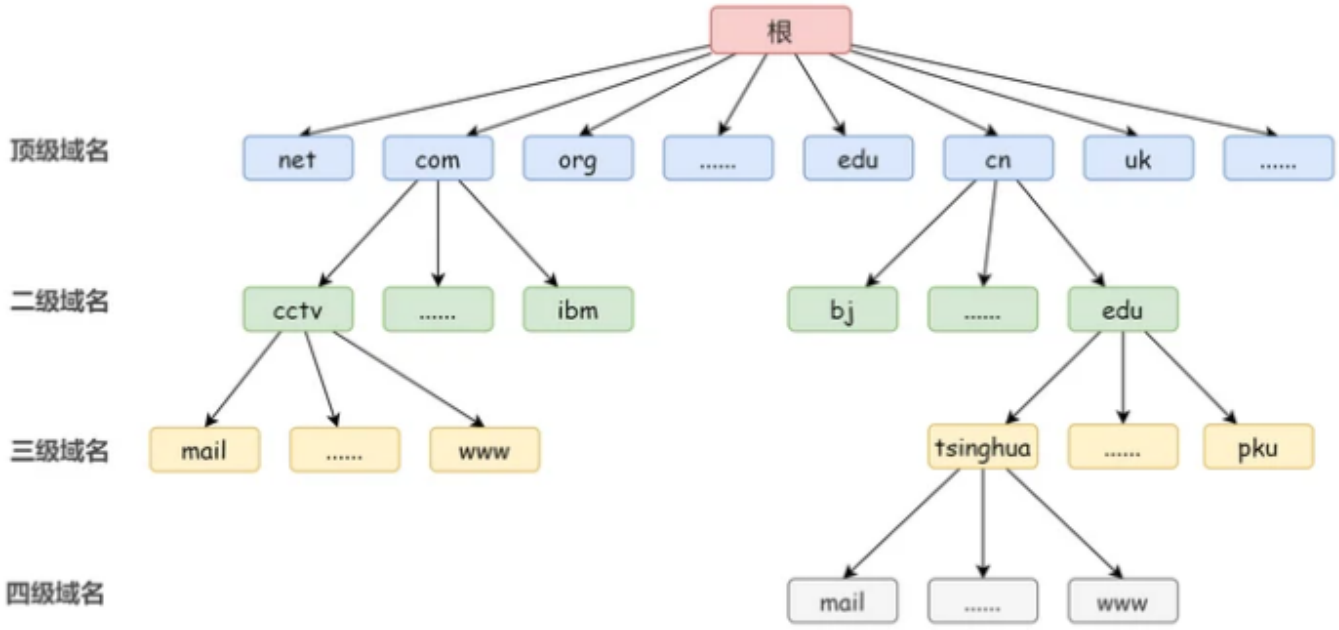
简单来讲，`DNS` 相当于一个翻译官，负责将域名翻译成 `ip` 地址

- IP 地址：一长串能够唯一地标记网络上的计算机的数字
- 域名：是由一串用点分隔的名字组成的 Internet 上某一台计算机或计算机组的名称，用于在数据传输时对计算机的定位标识



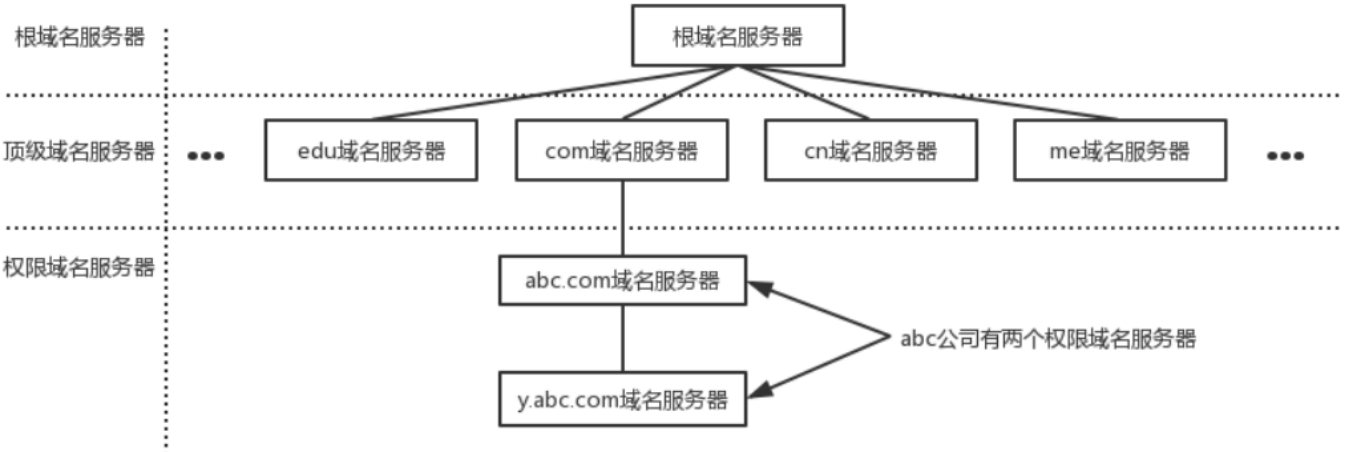
12.2. 二、域名

域名是一个具有层次的结构，从上到下一次为根域名、顶级域名、二级域名、三级域名...



例如 `www.xxx.com`， `www` 为三级域名、 `xxx` 为二级域名、 `com` 为顶级域名，系统为用户做了兼容，域名末尾的根域名 `.` 一般不需要输入

在域名的每一层都会有一个域名服务器，如下图：

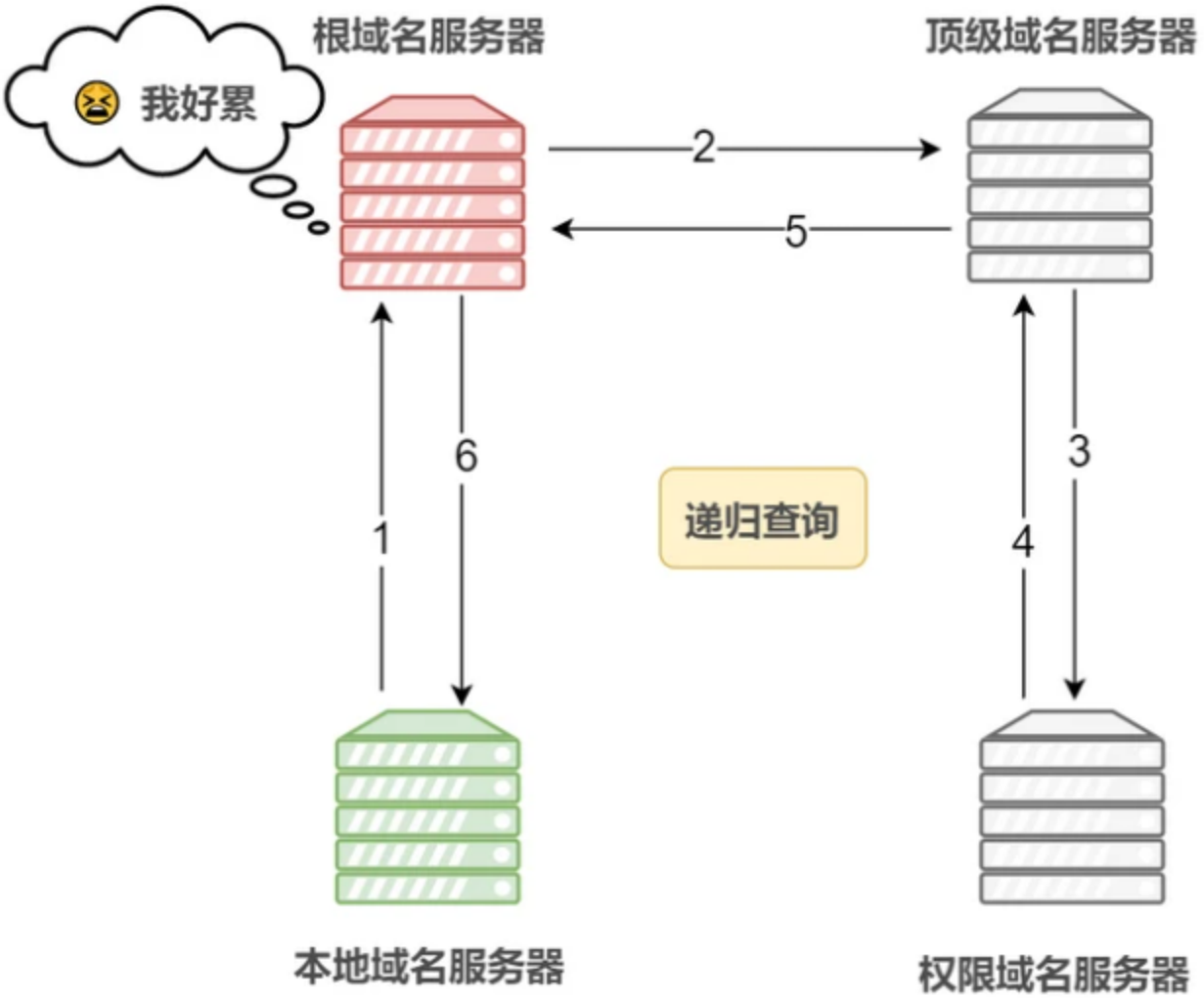


除此之外，还有电脑默认的本地域名服务器

12.3. 查询方式

DNS 查询的方式有两种：

- 递归查询：如果 A 请求 B，那么 B 作为请求的接收者一定要给 A 想要的答案



- 迭代查询：如果接收者 B 没有请求者 A 所需要的准确内容，接收者 B 将告诉请求者 A，如何去获得这个内容，但是自己并不去发出请求