HTTP

目录

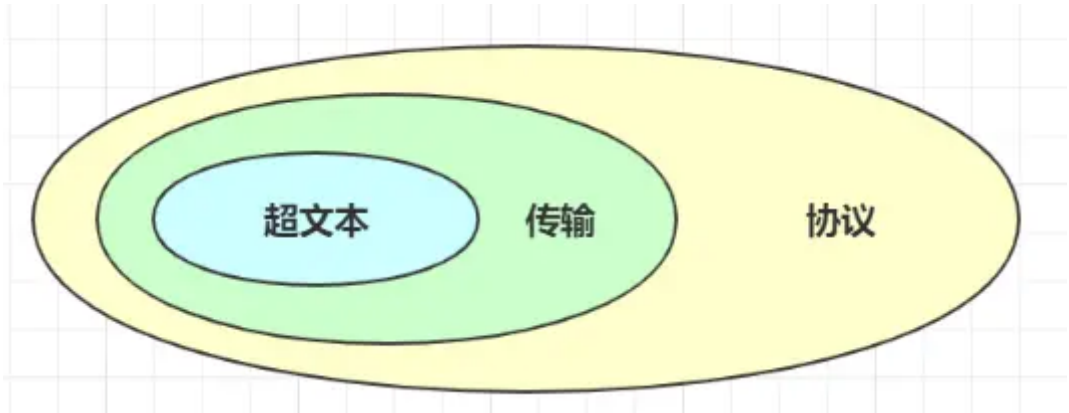
HTTP 基本概念

HTTP 是什么?

HTTP 是超文本传输协议，也就是HyperText Transfer Protocol。

HTTP 的名字「超文本协议传输」，它可以拆成三个部分：

- 超文本
- 传输
- 协议



1. *「协议」*

生活中的协议，本质上与计算机中的协议是相同的，协议的特点:

「协」字，代表的意思是必须有两个以上的参与者。例如三方协议里的参与者有三个：你、公司、学校三个；租房协议里的参与者有两个：你和房东。

「议」字，代表的意思是对参与者的一种行为约定和规范。例如三方协议里规定试用期期限、毁约金等；租房协议里规定租期期限、每月租金金额、违约如何处理等。

针对 HTTP 协议，我们可以这么理解。

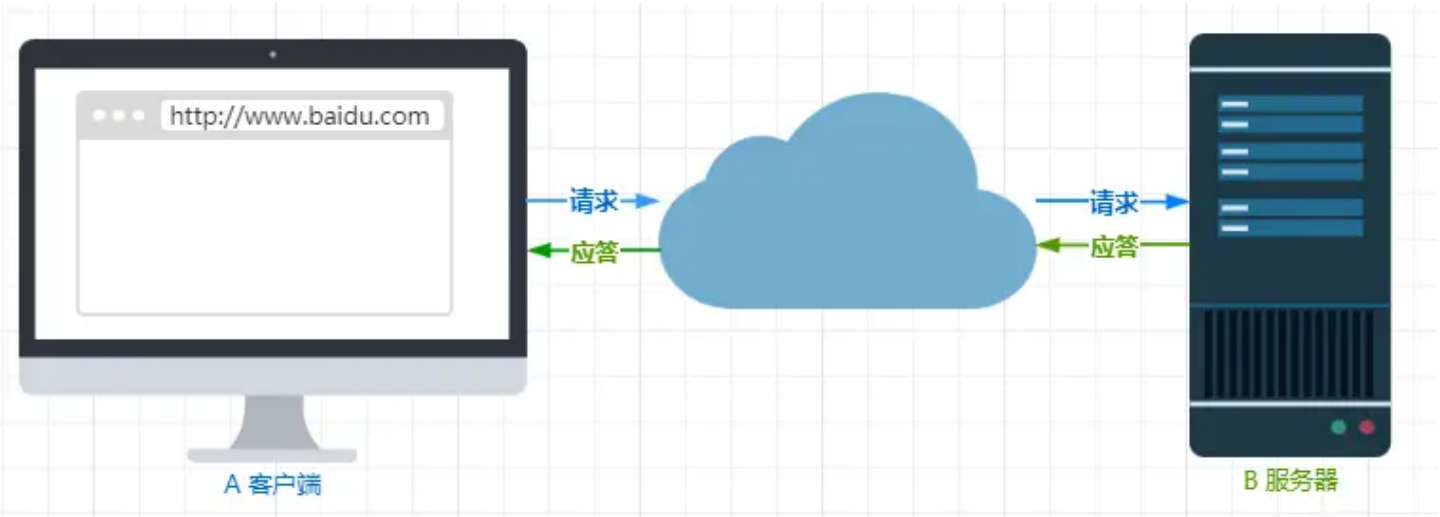
HTTP 是一个用在计算机世界里的协议。它使用计算机能够理解的语言确立了一种计算机之间交流通信的规范（两个以上的参与者），以及相关的各种控制和错误处理方式（行为约定和规范）。

2. *「传输」*

所谓的「传输」，很好理解，就是把一堆东西从 A 点搬到 B 点，或者从 B 点 搬到 A 点。

HTTP 协议是一个双向协议

我们在上网冲浪时，浏览器是请求方 A，百度网站就是应答方 B。双方约定用 HTTP 协议来通信，于是浏览器把请求数据发送给网站，网站再把一些数据返回给浏览器，最后由浏览器渲染在屏幕，就可以看到图片、视频了。



数据虽然是在 A 和 B 之间传输，但允许中间有中转或接力。

就好像第一排的同学想传递纸条给最后一排的同学，那么传递的过程中就需要经过好多个同学（中间人），这样的传输方式就从「A < --- > B」，变成了「A <-> N <-> M <-> B」。

而在 HTTP 里，需要中间人遵从 HTTP 协议，只要不打扰基本的数据传输，就可以添加任意额外的东西。

针对传输，我们可以进一步理解了 HTTP。

HTTP 是一个在计算机世界里专门用来在两点之间传输数据的约定和规范。

3. *「超文本」*

HTTP 传输的内容是「超文本」。

「文本」，在互联网早期的时候只是简单的字符文字，但现在「文本」的涵义已经可以扩展为图片、视频、压缩包等，在 HTTP 眼里这些都算作「文本」。

「超文本」，它就是超越了普通文本的文本，它是文字、图片、视频等的混合体，最关键有超链接，能从一个超文本跳转到另外一个超文本。

HTML 就是最常见的超文本了，它本身只是纯文字文件，但内部用很多标签定义了图片、视频等的链接，再经过浏览器的解释，呈现给我们的就是一个文字、有画面的网页了。

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 是用于从互联网服务器传输超文本到本地浏览器的协议"
>
> 这种说法是不正确的。因为也可以是「服务器<--->服务器」，所以采用两点之间的描述会更准确。

HTTP 常见的状态码有哪些?

`1xx` 类状态码属于提示信息，是协议处理中的一种中间状态，实际用到的比较少。

`2xx` 类状态码表示服务器成功处理了客户端的请求，也是我们最愿意看到的状态。

- 「200 OK」是最常见的成功状态码，表示一切正常。如果是非 `HEAD` 请求，服务器返回的响应头都会有 body 数据。
- 「204 No Content」也是常见的成功状态码，与 200 OK 基本相同，但响应头没有 body 数据。

- 「206 Partial Content」是应用于 HTTP 分块下载或断点续传，表示响应返回的 body 数据并不是资源的全部，而是其中的一部分，也是服务器处理成功的状态。

`3xx` 类状态码表示客户端请求的资源发生了变动，需要客户端用新的 URL 重新发送请求获取资源，也就是重定向。

- 「301 Moved Permanently」表示永久重定向，说明请求的资源已经不存在了，需改用新的 URL 再次访问。
- 「302 Found」表示临时重定向，说明请求的资源还在，但暂时需要用另一个 URL 来访问。

301 和 302 都会在响应头里使用字段 `Location`，指明后续要跳转的 URL，浏览器会自动重定向新的 URL。

- 「304 Not Modified」不具有跳转的含义，表示资源未修改，重定向已存在的缓冲文件，也称缓存重定向，也就是告诉客户端可以继续使用缓存资源，用于缓存控制。

`4xx` 类状态码表示客户端发送的报文有误，服务器无法处理，也就是错误码的含义。

- 「400 Bad Request」表示客户端请求的报文有错误，但只是个笼统的错误。
- 「403 Forbidden」表示服务器禁止访问资源，并不是客户端的请求出错。
- 「404 Not Found」表示请求的资源在服务器上不存在或未找到，所以无法提供给客户端。

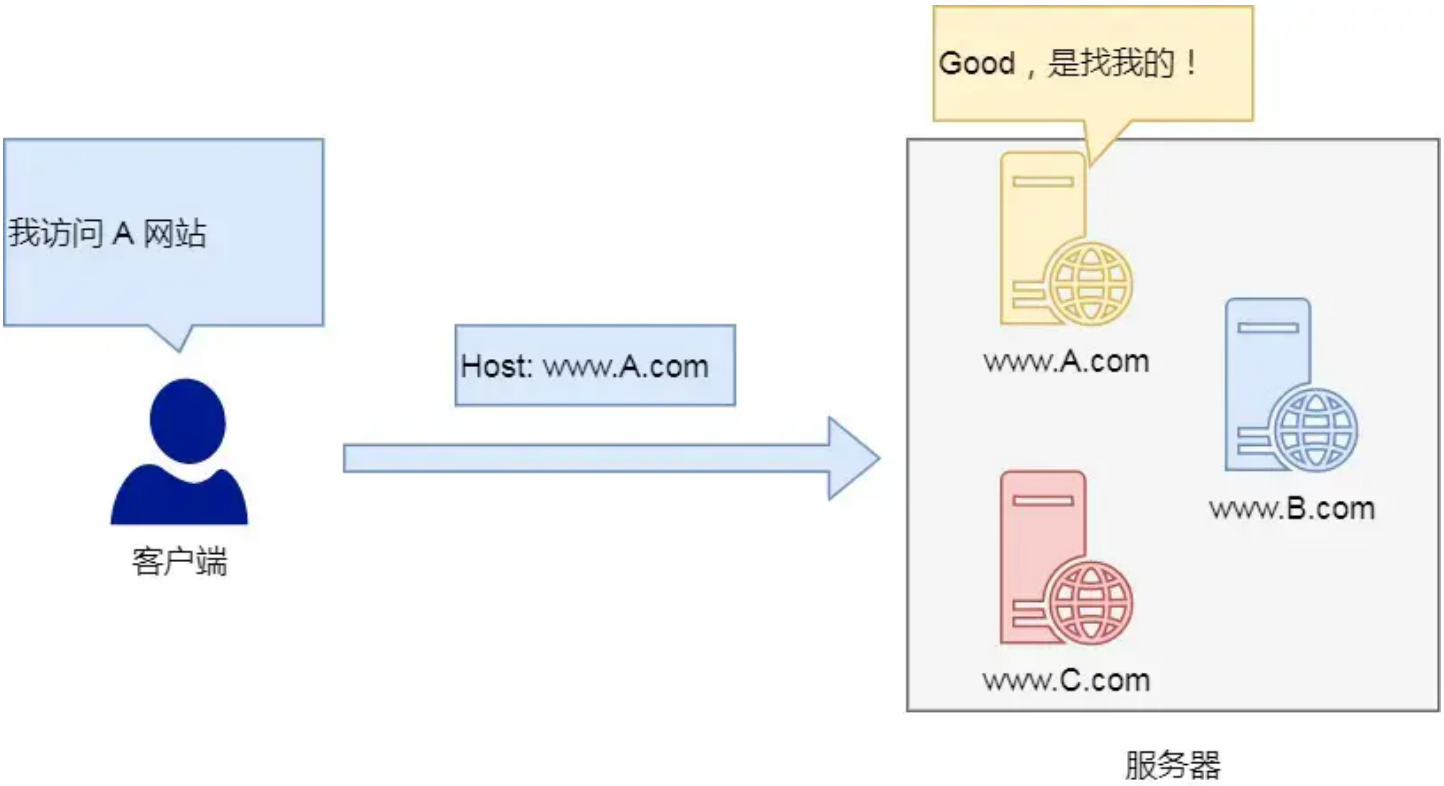
`5xx` 类状态码表示客户端请求报文正确，但是服务器处理时内部发生了错误，属于服务器端的错误码。

- 「500 Internal Server Error」与 400 类型，是个笼统通用的错误码，服务器发生了什么错误，我们并不知道。
- 「501 Not Implemented」表示客户端请求的功能还不支持，类似“即将开业，敬请期待”的意思。
- 「502 Bad Gateway」通常是服务器作为网关或代理时返回的错误码，表示服务器自身工作正常，访问后端服务器发生了错误。
- 「503 Service Unavailable」表示服务器当前很忙，暂时无法响应客户端，类似“网络服务正忙，请稍后重试”的意思。

HTTP 常见字段有哪些?

Host 字段

客户端发送请求时，用来指定服务器的域名。



有了 `Host` 字段，就可以将请求发往「同一台」服务器上的不同网站。

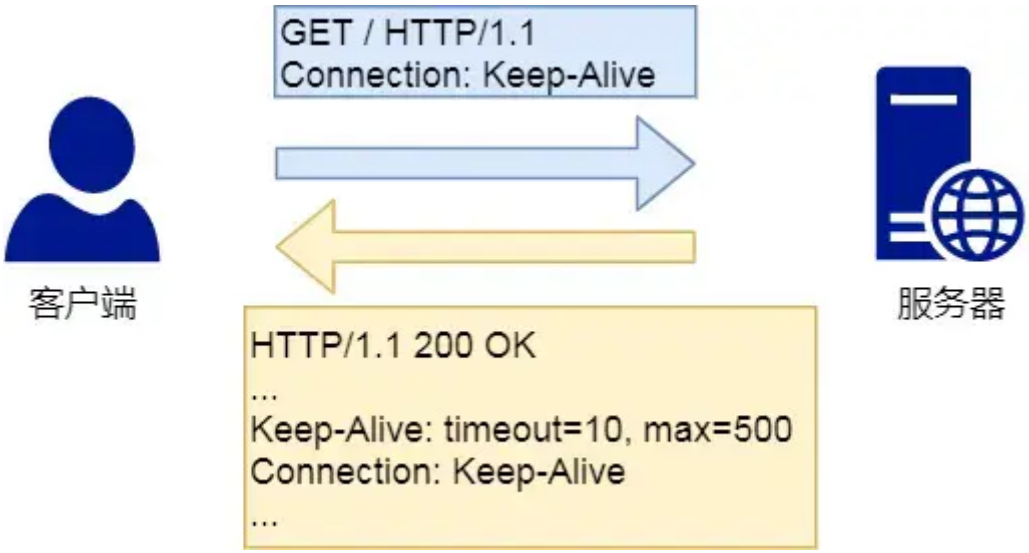
Content-Length 字段

服务器在返回数据时，会有 `Content-Length` 字段，表明本次回应的数据长度。

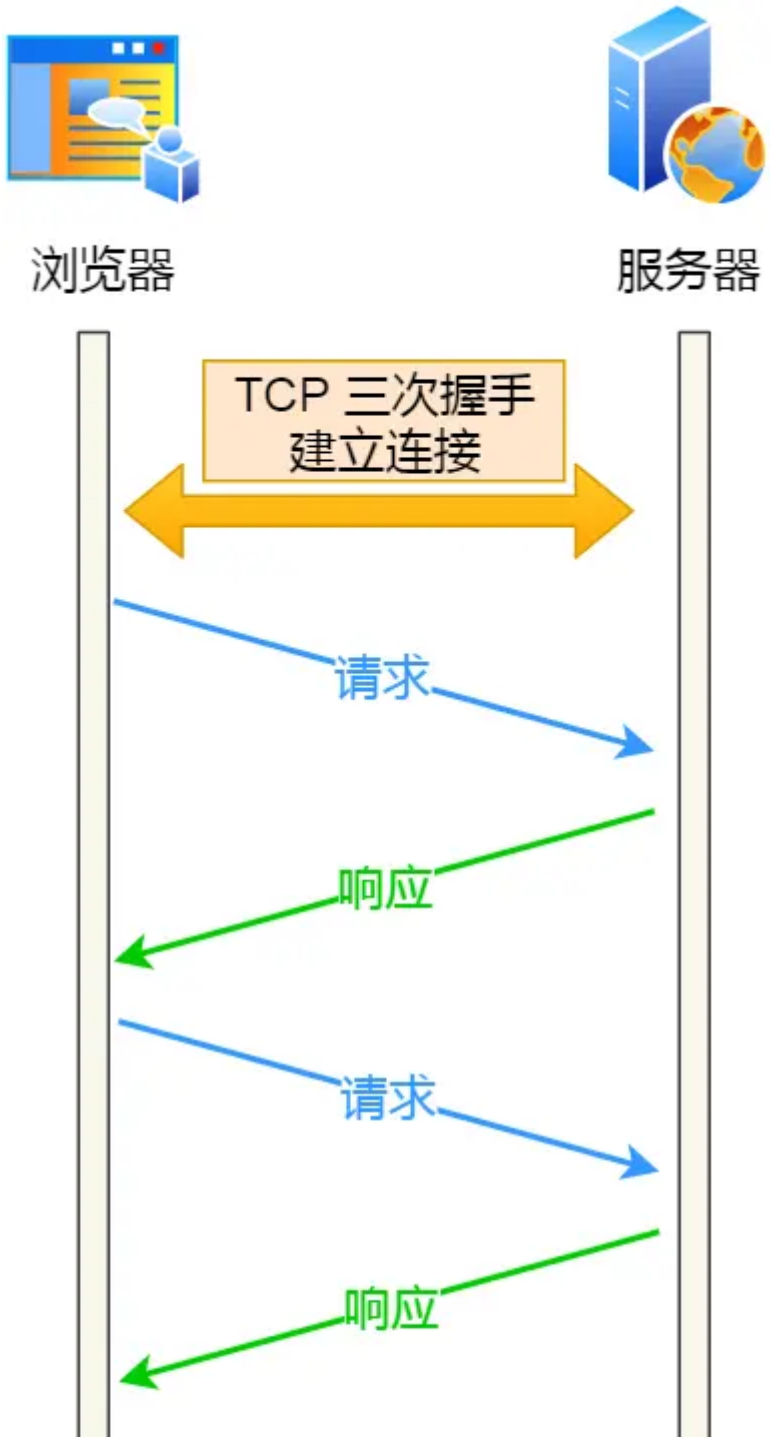
如上面则是告诉浏览器，本次服务器回应的数据长度是 1000 个字节，后面的字节就属于下一个回应了。

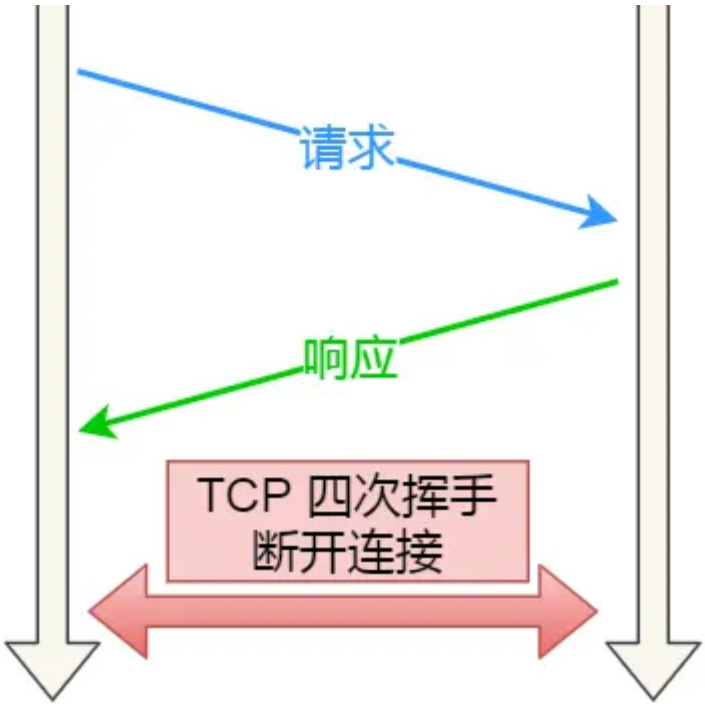
Connection 字段

`Connection` 字段最常用于客户端要求服务器使用「HTTP 长连接」机制，以便其他请求复用。



HTTP 长连接的特点是，只要任意一端没有明确提出断开连接，则保持 TCP 连接状态。



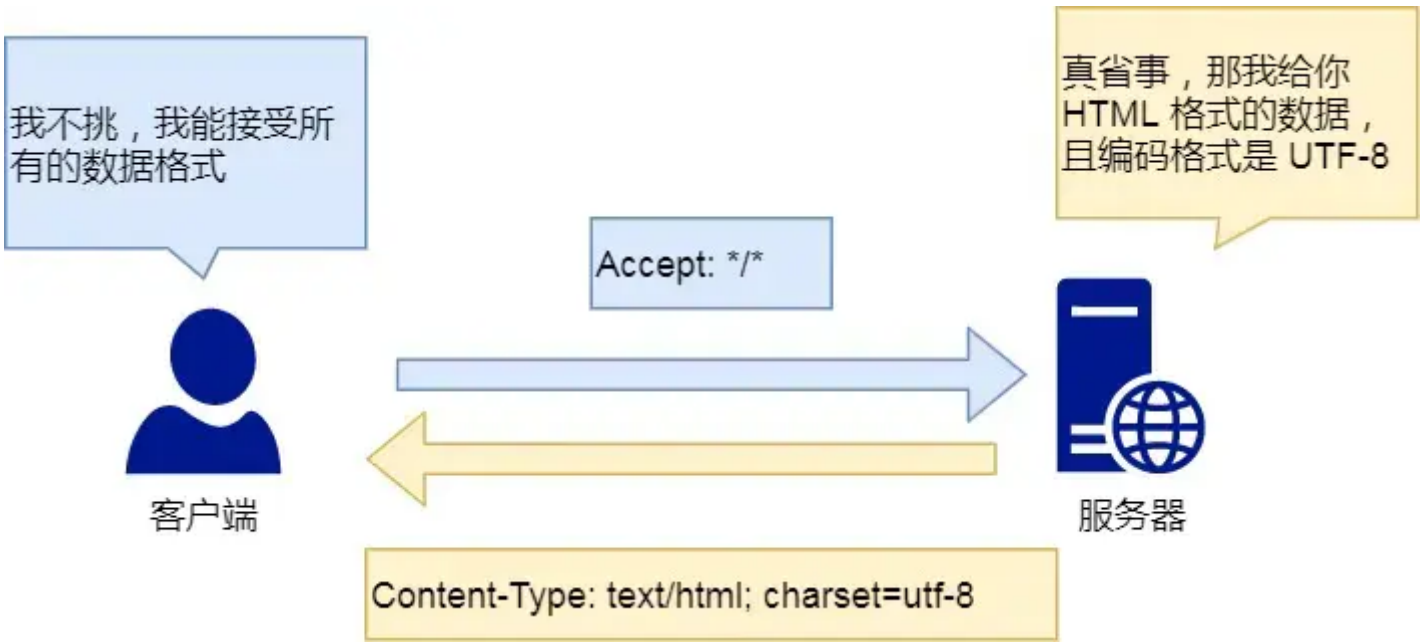


HTTP/1.1 版本的默认连接都是长连接，但为了兼容老版本的 HTTP，需要指定 `Connection` 首部字段的值为 `Keep-Alive`。

开启了 HTTP Keep-Alive 机制后， 连接就不会中断，而是保持连接。当客户端发送另一个请求时，它会使用同一个连接，一直持续到客户端或服务器端提出断开连接。

Content-Type 字段

`Content-Type` 字段用于服务器回应时，告诉客户端，本次数据是什么格式。



上面的类型表明，发送的是网页，而且编码是UTF-8。

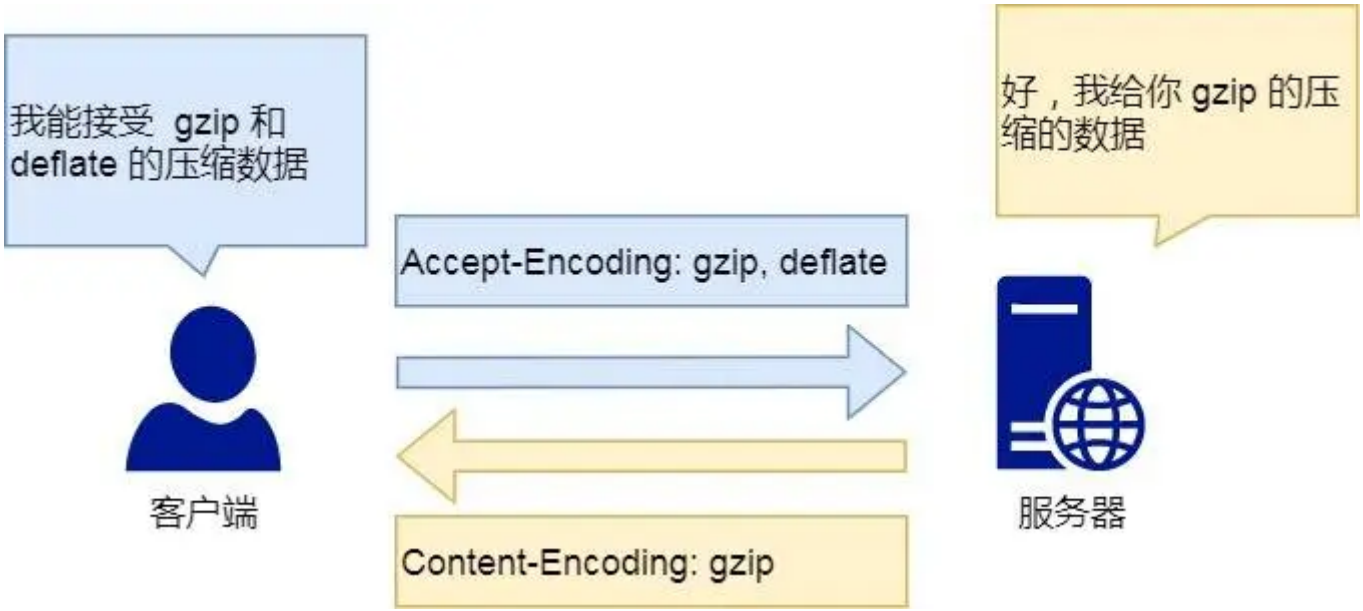
客户端请求的时候，可以使用 `Accept` 字段声明自己可以接受哪些数据格式。

```
代码块
1   Accept: /
```

上面代码中，客户端声明自己可以接受任何格式的数据。

Content-Encoding 字段

`Content-Encoding` 字段说明数据的压缩方法。表示服务器返回的数据使用了什么压缩格式



上面表示服务器返回的数据采用了 gzip 方式压缩，告知客户端需要用此方式解压。

客户端在请求时，用 `Accept-Encoding` 字段说明自己可以接受哪些压缩方法。

```
代码块
1   Accept-Encoding: gzip, deflate
```

GET 与 POST

GET 和 POST 有什么区别?

根据 RFC 规范，GET 的语义是从服务器获取指定的资源，这个资源可以是静态的文本、页面、图片视频等。GET 请求的参数位置一般是写在 URL 中，URL 规定只能支持 ASCII，所以 GET 请求的参数只允许 ASCII 字符 ，而且浏览器会对 URL 的长度有限制（HTTP协议本身对 URL长度并没有做任何规定）。

根据 RFC 规范，POST 的语义是根据请求负荷（报文body）对指定的资源做出处理，具体的处理方式视资源类型而不同。POST 请求携带数据的位置一般是写在报文 body 中，body 中的数据可以是任意

格式的数据，只要客户端与服务端协商好即可，而且浏览器不会对 body 大小做限制。

GET 和 POST 方法都是安全和幂等的吗？

先说明下安全和幂等的概念：

- 在 HTTP 协议里，所谓的「安全」是指请求方法不会「破坏」服务器上的资源。
- 所谓的「幂等」，意思是多次执行相同的操作，结果都是「相同」的。

如果从 RFC 规范定义的语义来看：

- GET 方法就是安全且幂等的，因为它是「只读」操作，无论操作多少次，服务器上的数据都是安全的，且每次的结果都是相同的。所以，可以对 GET 请求的数据做缓存，这个缓存可以做到浏览器本身上（彻底避免浏览器发请求），也可以做到代理上（如nginx），而且在浏览器中 GET 请求可以保存为书签。
- POST 因为是「新增或提交数据」的操作，会修改服务器上的资源，所以是不安全的，且多次提交数据就会创建多个资源，所以不是幂等的。所以，浏览器一般不会缓存 POST 请求，也不能把 POST 请求保存为书签。

做个简要的小结。

GET 的语义是请求获取指定的资源。GET 方法是安全、幂等、可被缓存的。

POST 的语义是根据请求负荷（报文主体）对指定的资源做出处理，具体的处理方式视资源类型而不同。POST 不安全，不幂等，（大部分实现）不可缓存。

注意， 上面是从 RFC 规范定义的语义来分析的。

但是实际过程中，开发者不一定会按照 RFC 规范定义的语义来实现 GET 和 POST 方法。比如：

- 可以用 GET 方法实现新增或删除数据的请求，这样实现的 GET 方法自然就不是安全和幂等。
- 可以用 POST 方法实现查询数据的请求，这样实现的 POST 方法自然就是安全和幂等。

要避免传输过程中数据被窃取，就要使用 HTTPS 协议，这样所有 HTTP 的数据都会被加密传输。

GET 请求可以带 body 吗？

RFC 规范并没有规定 GET 请求不能带 body 的。理论上，任何请求都可以带 body 的。只是因为 RFC 规范定义的 GET 请求是获取资源，所以根据这个语义不需要用到 body。

另外，URL 中的查询参数也不是 GET 所独有的，POST 请求的 URL 中也可以有参数的。

HTTP缓存技术

HTTP 缓存有哪些实现方式？

对于一些具有重复性的 HTTP 请求，比如每次请求得到的数据都一样的，我们可以把这对「请求-响应」的数据都缓存在本地，那么下次就直接读取本地的数据，不必在通过网络获取服务器的响应了，这样的话HTTP/1.1 的性能肯定肉眼可见的提升。

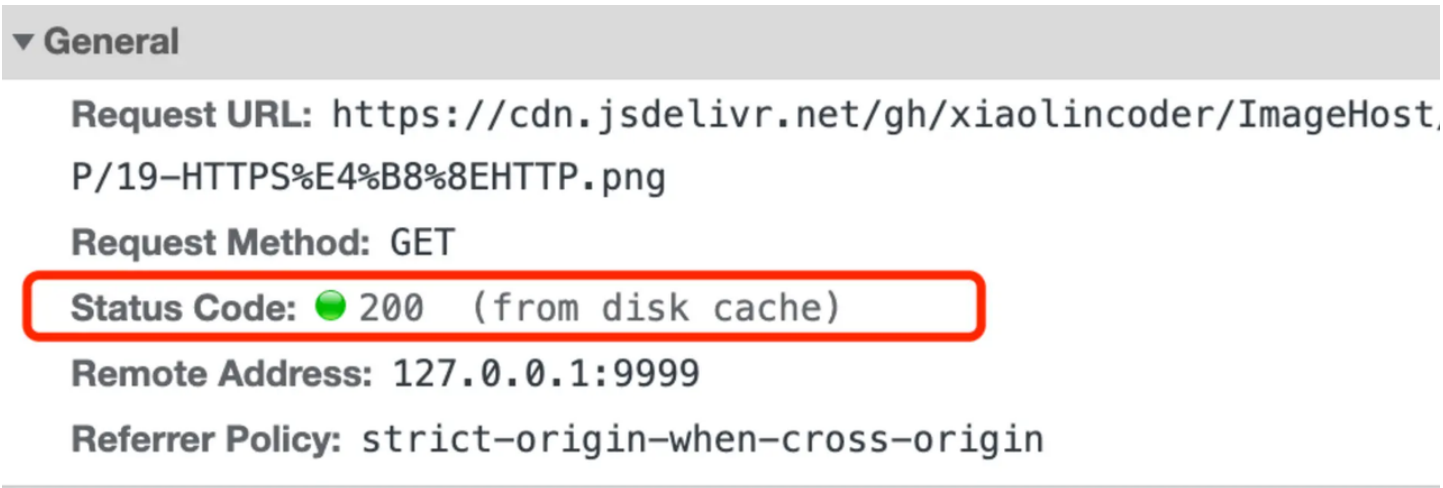
所以，避免发送 HTTP 请求的方法就是通过缓存技术，HTTP 设计者早在之前就考虑到了这点，因此HTTP协议的头部有不少是针对缓存的字段。

HTTP 缓存有两种实现方式，分别是强制缓存和协商缓存。

什么是强制缓存?

强缓存指的是只要浏览器判断缓存没有过期，则直接使用浏览器的本地缓存，决定是否使用缓存的主动性在于浏览器这边。

如下图中，返回的是 200 状态码，但在 size 项中标识的是 from disk cache，就是使用了强制缓存。



强缓存是利用下面这两个 HTTP 响应头部(Response Header)字段实现的，它们都用来表示资源在客户端缓存的有效期:

`Cache-Control`， 是一个相对时间；

`Expires`，是一个绝对时间；

如果 HTTP 响应头部同时有 Cache-Control和 Expires 字段的话，Cache-Control的优先级高于 Expires。

Cache-control选项更多一些，设置更加精细，所以建议使用 Cache-Control 来实现强缓存。具体的实现

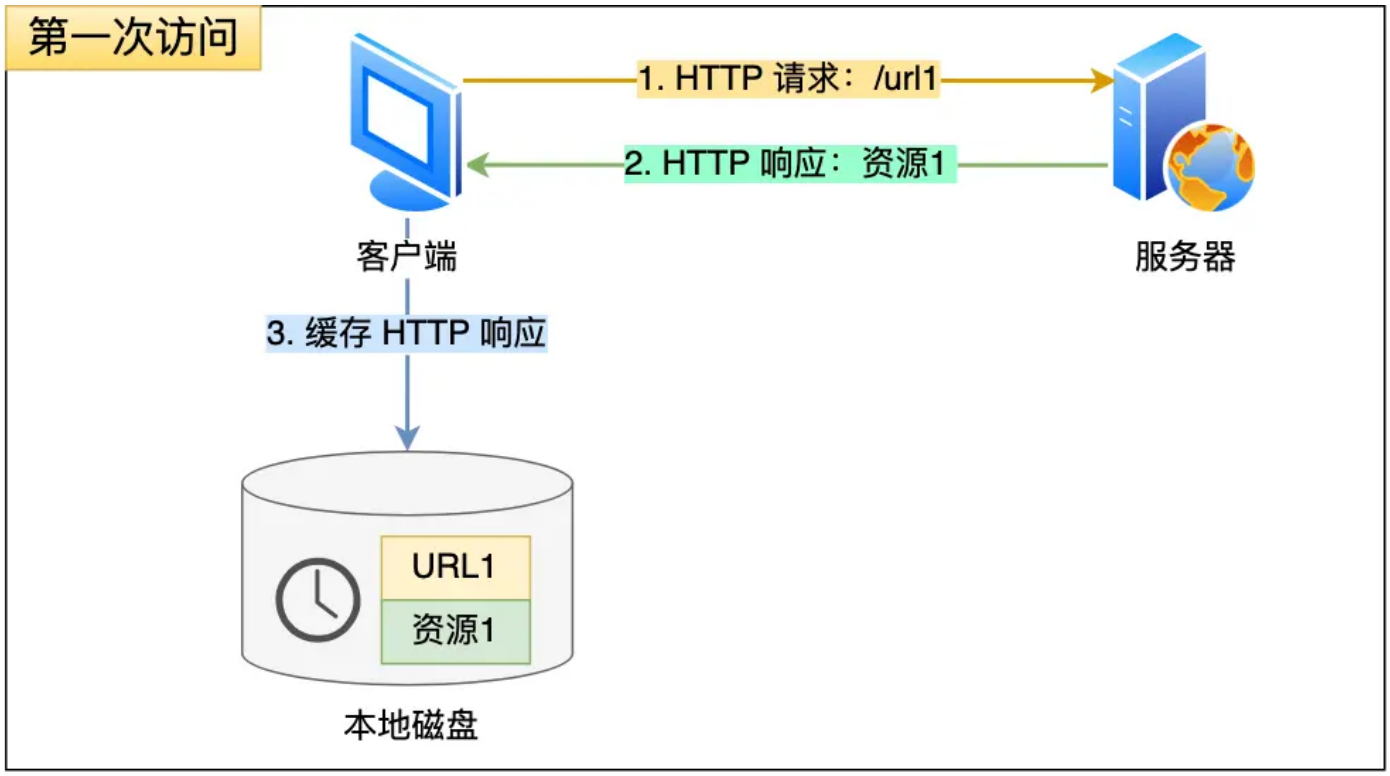
流程如下:

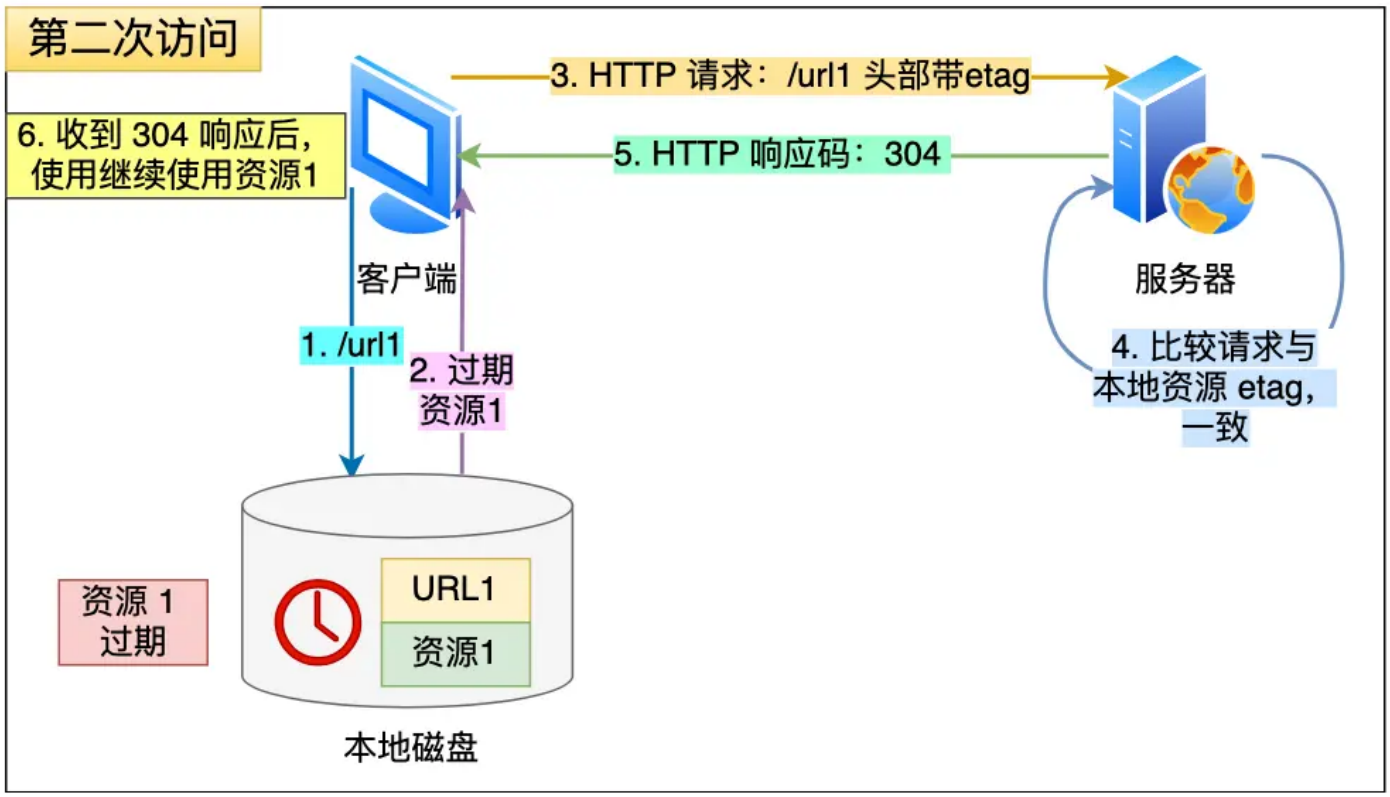
当浏览器第一次请求访问服务器资源时，服务器会在返回这个资源的同时，在 Response 头部加上 Cache-Control，Cache-Control 中设置了过期时间大小;

浏览器再次请求访问服务器中的该资源时，会先通过请求资源的时间与 Cache-Control 中设置的过期时间大小，来计算出该资源是否过期，如果没有，则使用该缓存，否则重新请求服务器;服务器再次收到请求后，会再次更新 Response 头部的 Cache-Control。

什么是协商缓存?

当我们在浏览器使用开发者工具的时候，你可能会看到过某些请求的响应码是 304 ，这个是告诉浏览器可以使用本地缓存的资源，通常这种通过服务端告知客户端是否可以使用缓存的方式被称为协商缓存。





上图就是一个协商缓存的过程，所以协商缓存就是与服务端协商之后，通过协商结果来判断是否使用本地缓存。

协商缓存可以基于两种头部来实现。

第一种:请求头部中的`If-Modified-Since`字段与响应头部中的`Last-Modified`字段实现，这两个字段的意思是:

- 响应头部中的`Last-Modified`：标示这个响应资源的最后修改时间;
- 请求头部中的`If-Modified-Since`：当资源过期了，发现响应头中具有 Last-Modified 声明，则再次发起请求的时候带上 Last-Modified 的时间，服务器收到请求后发现有 If-Modified-Since 则与被请求资源的最后修改时间进行对比(Last-Modified)，如果最后修改时间较新(大)，说明资源又被改过，则返回最新资源，HTTP 200 OK:如果最后修改时间较旧(小)，说明资源无新修改，响应 HTTP 304 走缓存。

第二种:请求头部中的`If-None-Match`字段与响应头部中的`ETag`字段，这两个字段的意思是

·响应头部中`Etag`：唯一标识响应资源;

·请求头部中的`If-None-Match`：当资源过期时，浏览器发现响应头里有 Etag，则再次向服务器发起请求时，会将请求头 If-None-Match 值设置为 Etag 的值。服务器收到请求后进行比对，如果资源没有变化返回 304，如果资源变化了返回 200。

第一种实现方式是基于时间实现的，第二种实现方式是基于一个唯一标识实现的，相对来说后者可以更加准确地判断文件内容是否被修改，避免由于时间篡改导致的不可靠问题。

如果在第一次请求资源的时候，服务端返回的 HTTP 响应头部同时有 Etag 和 Last-Modified 字段，那么客户端再下一次请求的时候，如果带上了 ETag 和 Last-Modified 字段信息给服务端，这时 Etag 的优先级更高，也就是服务端先会判断 Etag 是否变化了，如果 Etag 有变化就不用在判断 Last-Modified 了，如果Etaq 没有变化，然后再看 Last-Modified。

为什么 ETag 的优先级更高?

这是因为 ETag 主要能解决 Last-Modified 几个比较难以解决的问题:

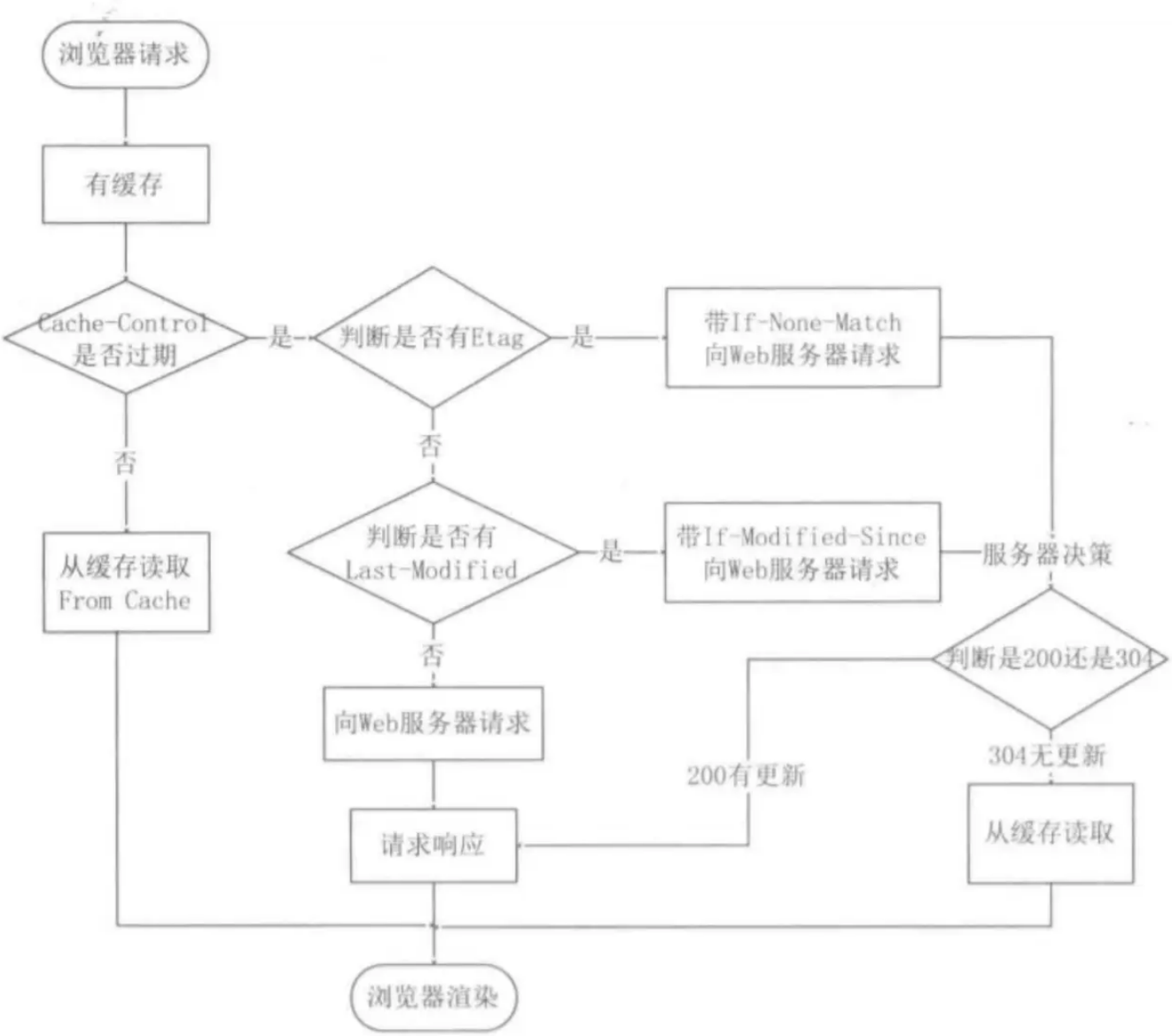
1.在没有修改文件内容情况下文件的最后修改时间可能也会改变，这会导致客户端认为这文件被改动了从而重新请求;

2.可能有些文件是在秒级以内修改的，`If-Modified-Since`能检查到的粒度是秒级的，使用 Etag 就能够保证这种需求下客户端在1秒内能刷新多次;

3.有些服务器不能精确获取文件的最后修改时间。

注意，协商缓存这两个字段都需要配合强制缓存中 Cache-Control 字段来使用，只有在未能命中强制缓存的时候，才能发起带有协商缓存字段的请求。

下图是强制缓存和协商缓存的工作流程:



当使用 ETag 字段实现的协商缓存的过程:

- 当浏览器第一次请求访问服务器资源时，服务器会在返回这个资源的同时，在 Response 头部加上 ETag唯一标识，这个唯一标识的值是根据当前请求的资源生成的;
- 当浏览器再次请求访问服务器中的该资源时，首先会先检查强制缓存是否过期:

  如果没有过期，则直接使用本地缓存;

  如果缓存过期了，会在 Request 头部加上 If-None-Match 字段，该字段的值就是 ETag 唯一标识;
- 服务器再次收到请求后，会根据请求中的 If-None-Match 值与当前请求的资源生成的唯一标识进行比较:

  如果值相等，则返回 304 Not Modified，不会返回资源;

  如果不相等，则返回 200 状态码和返回资源，并在 Response 头部加上新的 ETag 唯一标识;
- 如果浏览器收到 304 的请求响应状态码，则会从本地缓存中加载资源，否则更新资源。

HTTP 特性

到目前为止，HTTP 常见到版本有 HTTP/1.1，HTTP/2.0，HTTP/3.0，不同版本的 HTTP 特性是不一样的。

HTTP/1.1 的优点有哪些?

HTTP 最突出的优点是「简单、灵活和易于扩展、应用广泛和跨平台」。

1. *简单*

HTTP 基本的报文格式就是 `header + body`，头部信息也是 `key-value` 简单文本的形式，易于理解，降低了学习和使用的门槛。

2. *灵活和易于扩展*

HTTP 协议里的各类请求方法、URI/URL、状态码、头字段等每个组成要求都没有被固定死，都允许开发人员自定义和扩充。

同时 HTTP 由于是工作在应用层（ `OSI` 第七层），则它下层可以随意变化，比如：

OSI 网络模型

- 应用层，给用户提供应用功能；
- 表示层，负责把数据转换成兼容另一个系统能识别的格式；
- 会话层，负责建立、维持、同步会话；
- 传输层，负责端到端的数据传输；
- 网络层，寻址、路由；
- 数据链路层，负责数据的封帧和差错检测，以及 MAC 寻址；
- 物理层，负责在物理网络中传输数据帧；

- HTTPS 就是在 HTTP 与 TCP 层之间增加了 SSL/TLS 安全传输层；

3. *应用广泛和跨平台*

互联网发展至今，HTTP 的应用范围非常的广泛，从台式机的浏览器到手机上的各种 APP，从看新闻、刷贴吧到购物、理财、吃鸡，HTTP 的应用遍地开花，同时天然具有跨平台的优越性。

HTTP/1.1 的缺点有哪些?

HTTP 协议里有优缺点一体的双刃剑，分别是「无状态、明文传输」，同时还有一大缺点「不安全」。

1. *无状态双刃剑*

无状态的好处，因为服务器不会去记忆 HTTP 的状态，所以不需要额外的资源来记录状态信息，这能减轻服务器的负担，能够把更多的 CPU 和内存用来对外提供服务。

无状态的坏处，既然服务器没有记忆能力，它在完成有关联性的操作时会非常麻烦。

例如登录->添加购物车->下单->结算->支付，这系列操作都要知道用户的身份才行。但服务器不知道这些请求是有关联的，每次都要问一遍身份信息。

对于无状态的问题，解法方案有很多种，其中比较简单的方式用 Cookie 技术

`Cookie` 通过在请求和响应报文中写入 Cookie 信息来控制客户端的状态。

相当于，在客户端第一次请求后，服务器会下发一个装有客户信息的「小贴纸」，后续客户端请求服务器的时候，带上「小贴纸」，服务器就能认得了了

2. *明文传输双刃剑*

明文意味着在传输过程中的信息，是可方便阅读的，比如 Wireshark 抓包都可以直接肉眼查看，为我们调试工作带了极大的便利性。

但是这正是这样，HTTP 的所有信息都暴露在了光天化日下，相当于信息裸奔。在传输的漫长的过程中，信息的内容都毫无隐私可言，很容易就能被窃取，那你号没了。

3. *不安全*

HTTP 比较严重的缺点就是不安全：

- 通信使用明文（不加密），内容可能会被窃听。比如，账号信息容易泄漏，那你号没了。
- 不验证通信方的身份，因此有可能遭遇伪装。比如，访问假的淘宝、拼多多，那你钱没了。
- 无法证明报文的完整性，所以有可能已遭篡改。比如，网页上植入垃圾广告，视觉污染，眼没了。

HTTP 的安全问题，可以用 HTTPS 的方式解决，也就是通过引入 SSL/TLS 层，使得在安全上达到了极致。

HTTP 与 HTTPS

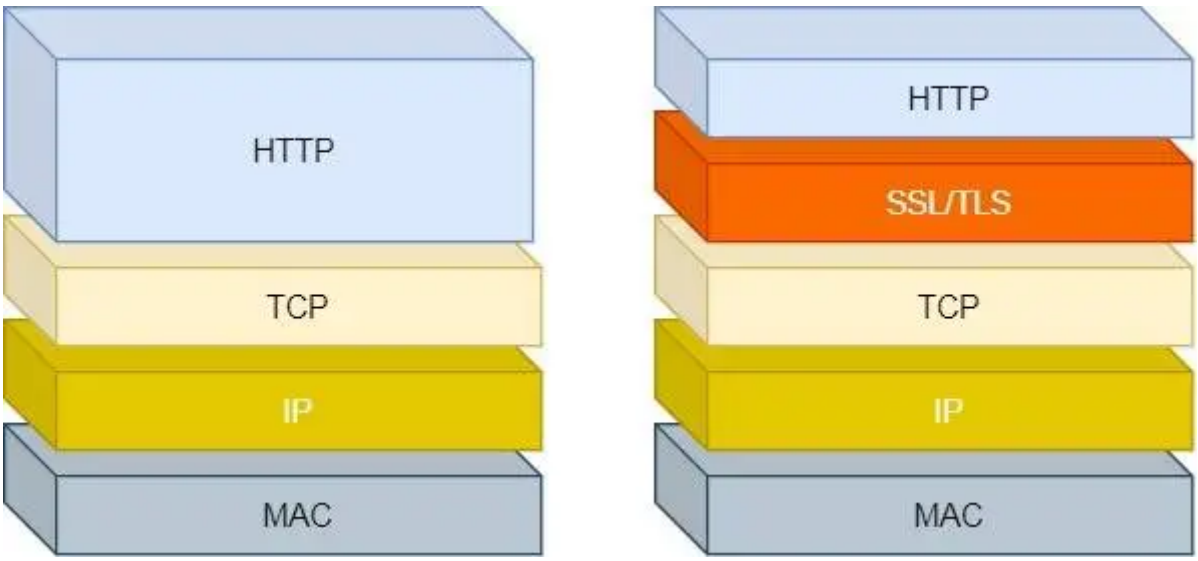
HTTP 与 HTTPS 有哪些区别?

- HTTP 是超文本传输协议，信息是明文传输，存在安全风险的问题。HTTPS 则解决 HTTP 不安全的缺陷，在 TCP 和 HTTP 网络层之间加入了 SSL/TLS 安全协议，使得报文能够加密传输。
- HTTP 连接建立相对简单，TCP 三次握手之后便可进行 HTTP 的报文传输。而 HTTPS 在 TCP 三次握手之后，还需进行 SSL/TLS 的握手过程，才可进入加密报文传输。
- 两者的默认端口不一样，HTTP 默认端口号是 80，HTTPS 默认端口号是 443。.
- HTTPS 协议需要向 CA(证书权威机构)申请数字证书，来保证服务器的身份是可信的。

HTTPS 解决了 HTTP 的哪些问题?

HTTP 由于是明文传输，所以安全上存在以下三个风险:

- 窃听风险，比如通信链路上可以获取通信内容，用户号容易没。
- 篡改风险，比如强制植入垃圾广告，视觉污染，用户眼容易瞎。
- 冒充风险，比如冒充淘宝网站，用户钱容易没。



HTTPS 在 HTTP 与 TCP 层之间加入了 SSL/TLS 协议，可以很好的解决了上述的风险.

- 信息加密:交互信息无法被窃取。
- 校验机制:无法篡改通信内容，篡改了就不能正常显示。
- 身份证书:证明淘宝是真的淘宝网。.

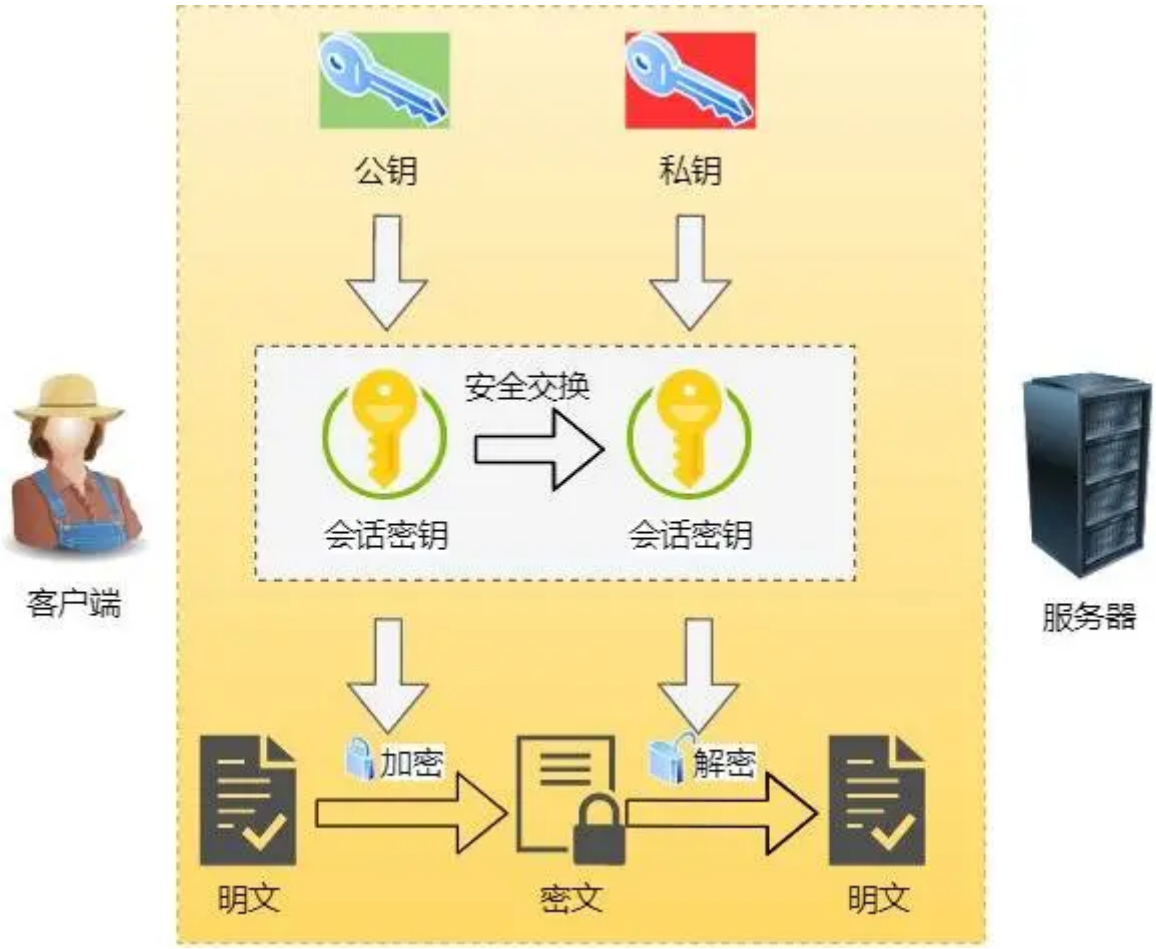
可见，只要自身不做「恶」，SSL/TLS 协议是能保证通信是安全的。

> HTTPS 是如何解决上面的三个风险的?

- 混合加密的方式实现信息的机密性，解决了窃听的风险。
- 摘要算法的方式来实现完整性，它能够为数据生成独一无二的「指纹」，指纹用于校验数据的完整性，解决了篡改的风险。
- 将服务器公钥放入到数字证书中，解决了冒充的风险。

1.混合加密

通过混合加密的方式可以保证信息的机密性，解决了窃听的风险。



HTTPS 采用的是对称加密和非对称加密结合的「混合加密」方式:

- 在通信建立前采用非对称加密的方式交换「会话秘钥」，后续就不再使用非对称加密。
- 在通信过程中全部使用对称加密的「会话秘钥」的方式加密明文数据。

采用「混合加密」的方式的原因:

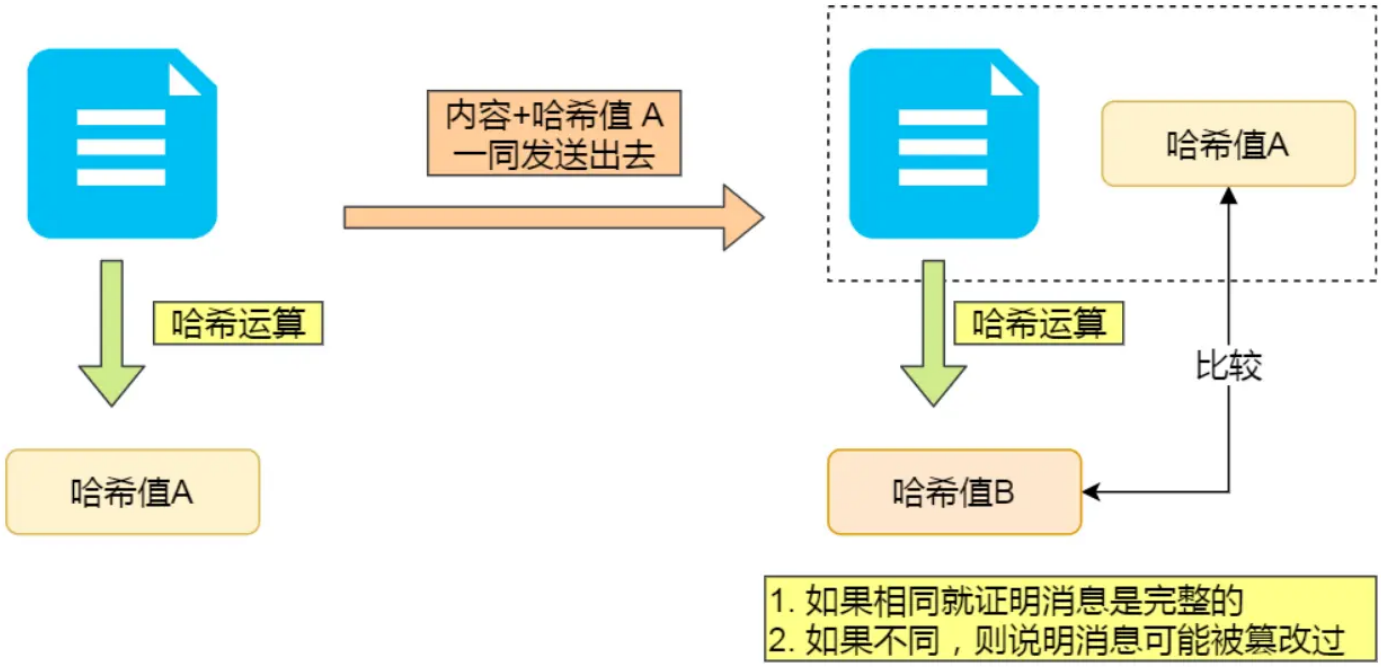
- 对称加密只使用一个密钥，运算速度快，密钥必须保密，无法做到安全的密钥交换。
- 非对称加密使用两个密钥:公钥和私钥，公钥可以任意分发而私钥保密，解决了密钥交换问题但速度慢。

2.摘要算法 + 数字签名

为了保证传输的内容不被篡改，我们需要对内容计算出一个「指纹」，然后同内容一起传输给对方。

对方收到后，先是对内容也计算出一个「指纹」，然后跟发送方发送的「指纹」做一个比较，如果「指纹」相同，说明内容没有被篡改，否则就可以判断出内容被篡改了。

那么，在计算机里会用摘要算法(哈希函数)来计算出内容的哈希值，也就是内容的「指纹」，这个哈希值是唯一的，且无法通过哈希值推导出内容。



通过哈希算法可以确保内容不会被篡改，但是并不能保证「内容+哈希值」不会被中间人替换，因为这里缺少对客户端收到的消息是否来源于服务端的证明。

举个例子，你想向老师请假，一般来说是要求由家长写一份请假理由并签名，老师才能允许你请假。

但是你有模仿你爸爸字迹的能力，你用你爸爸的字迹写了一份请假理由然后签上你爸爸的名字，老师一看到这个请假条，查看字迹和签名，就误以为是你爸爸写的，就会允许你请假。

那为了避免这种情况，计算机里会用非对称加密算法来解决，共有两个密钥:

- 一个是公钥，这个是可以公开给所有人的;
- 一个是私钥，这个必须由本人管理，不可泄露。

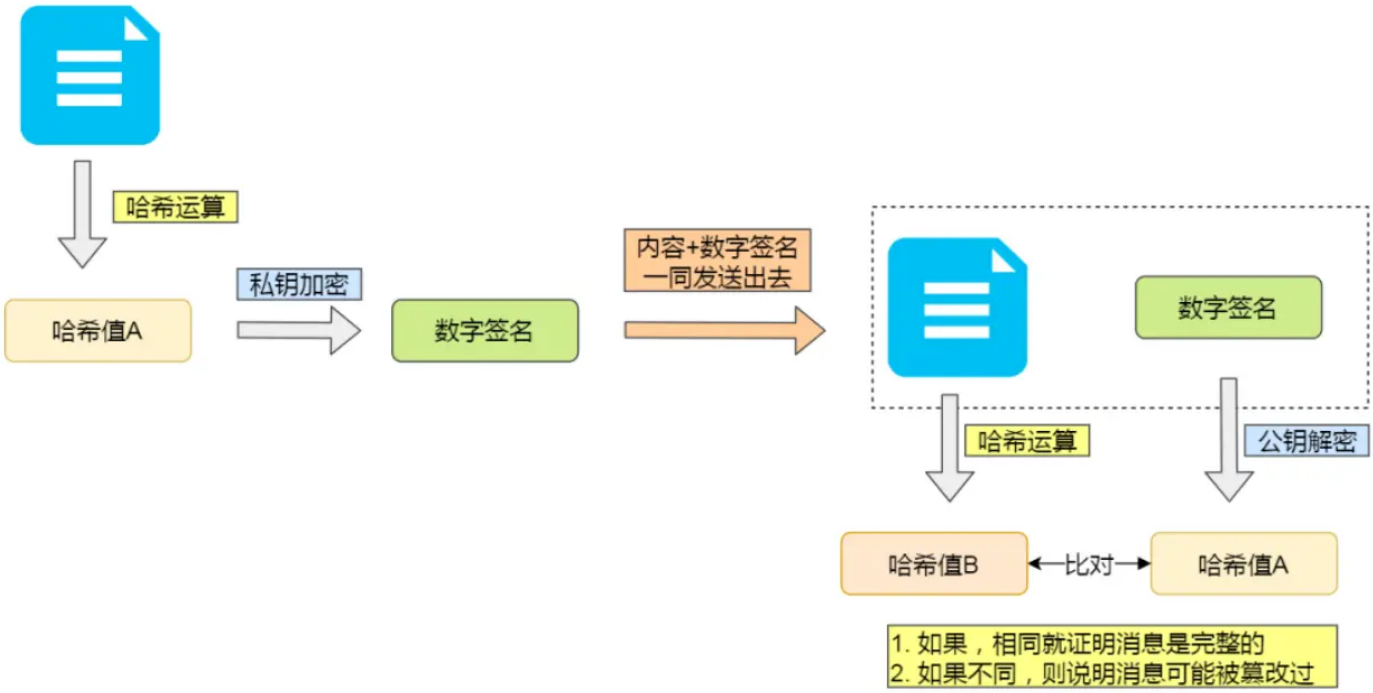
这两个密钥可以双向加解密的，比如可以用公钥加密内容，然后用私钥解密，也可以用私钥加密内容，公钥解密内容。

流程的不同，意味着目的也不相同:

- 公钥加密，私钥解密。这个目的是为了保证内容传输的安全，因为被公钥加密的内容，其他人是无法解密的，只有持有私钥的人，才能解密出实际的内容;
- 私钥加密，公钥解密。这个目的是为了保证消息不会被冒充，因为私钥是不可泄露的，如果公钥能正常解密出私钥加密的内容，就能证明这个消息是来源于持有私钥身份的人发送的。

一般我们不会用非对称加密来加密实际的传输内容，因为非对称加密的计算比较耗费性能的。

所以非对称加密的用途主要在于通过「私钥加密，公钥解密」的方式，来确认消息的身份，我们常说的数字签名算法，就是用的是这种方式，不过私钥加密内容不是内容本身，而是对内容的哈希值加密。



私钥是由服务端保管，然后服务端会向客户端颁发对应的公钥。如果客户端收到的信息，能被公钥解密，就说明该消息是由服务器发送的。

引入了数字签名算法后，你就无法模仿你爸爸的字迹来请假了，你爸爸手上持有着私钥，你老师持有着公钥。

这样只有用你爸爸手上的私钥才对请假条进行「签名」，老师通过公钥看能不能解出这个「签名」，如果能解出并且确认内容的完整性，就能证明是由你爸爸发起的请假条，这样老师才允许你请假，否则老师就不认。

3.数字证书

前面我们知道:

- 可以通过哈希算法来保证消息的完整性;
- 可以通过数字签名来保证消息的来源可靠性(能确认消息是由持有私钥的一方发送的):

但是这还远远不够，还缺少身份验证的环节，万一公钥是被伪造的呢?

还是拿请假的例子，虽然你爸爸持有私钥，老师通过是否能用公钥解密来确认这个请假条是不是来源你父亲的。

但是我们还可以自己伪造出一对公私钥啊!

你偷偷把老师桌面上和你爸爸配对的公钥，换成了你的公钥，那么下次你在请假的时候，你继续模仿你爸爸的字迹写了个请假条，然后用你的私钥做个了「数字签名」。

但是老师并不知道自己的公钥被你替换过了，所以他还是按照往常一样用公钥解密，由于这个公钥和你的私钥是配对的，老师当然能用这个被替换的公钥解密出来，并且确认了内容的完整性，于是老师就会以为是你父亲写的请假条，又允许你请假了。

后面你的老师和父亲发现了你伪造公私钥的事情后，决定重新商量一个对策。

既然伪造公私钥那么随意，所以你爸把他的公钥注册到警察局，警察局用他们自己的私钥对你父亲的公钥做了个数字签名，然后把你爸爸的「个人信息+公钥+数字签名」打包成一个数字证书，也就是说这个数字证书包含你爸爸的公钥。

这样，你爸爸如果因为家里确实有事要向老师帮你请假的时候，不仅会用自己的私钥对内容进行签名，还会把数字证书给到老师。

老师拿到了数字证书后，首先会去警察局验证这个数字证书是否合法，因为数字证书里有警察局的数字签名，警察局要验证证书合法性的时候，用自己的公钥解密，如果能解密成功，就说明这个数字证书是在警察局注册过的，就认为该数字证书是合法的，然后就会把数字证书里头的公钥(你爸爸的)给到老师。

由于通过警察局验证了数字证书是合法的，那么就能证明这个公钥就是你父亲的，于是老师就可以安心的用这个公钥解密出请假条，如果能解密出，就证明是你爸爸写的请假条。

正是通过了一个权威的机构来证明你爸爸的身份，所以你的伪造公私钥这个小伎俩就没用了。

在计算机里，这个权威的机构就是 CA(数字证书认证机构)，将服务器公钥放在数字证书(由数字证书认证机构颁发)中，只要证书是可信的，公钥就是可信的。

通过数字证书的方式保证服务器公钥的身份，解决冒充的风险。