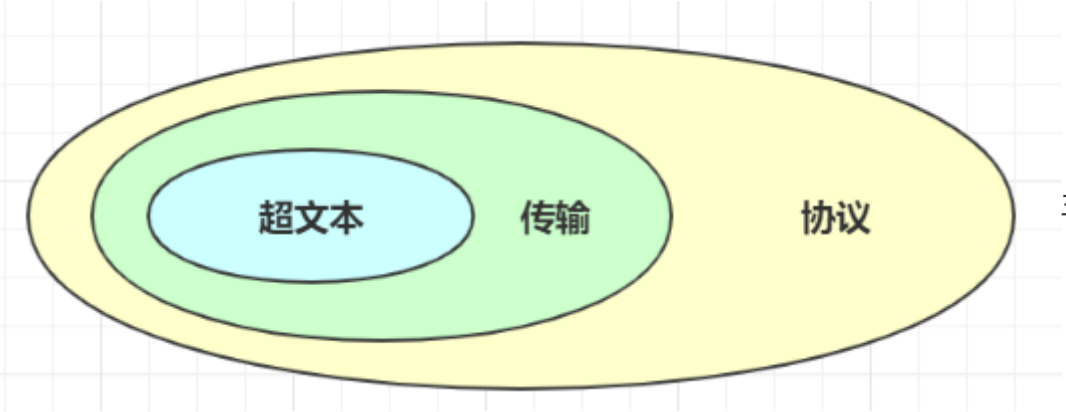
应用层 - HTTP

HTTP是什么

HTTP 是超文本传输协议，也就是**H**yper**T**ext **T**ransfer **P**rotocol。

HTTP的名字「超文本协议传输」，可以拆成三个部分：

- 超文本
- 传输
- 协议



三个部分

1. 「协议」

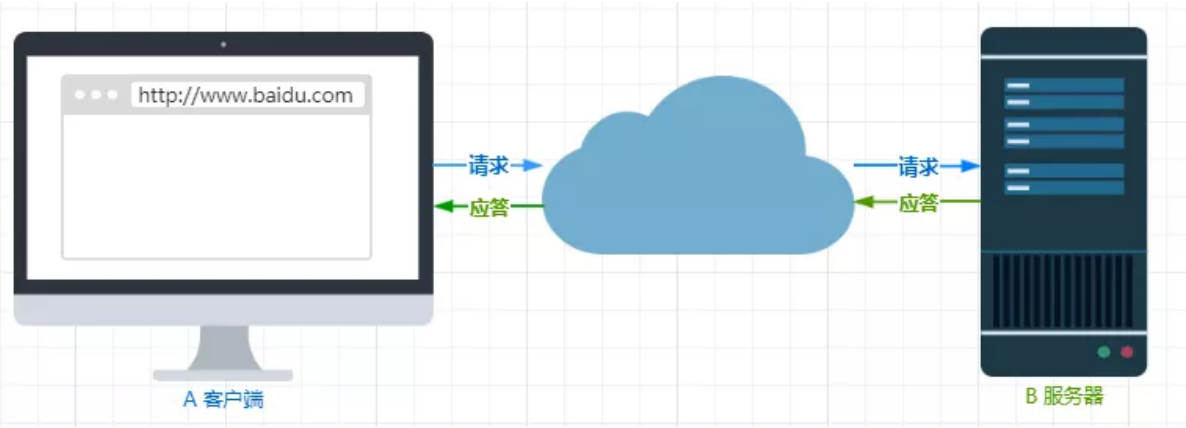
针对 HTTP **协议**，我们可以这么理解。

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」

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。



请求 - 应答

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」。

我们先来理解「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 眼里这些都算做「文本」。

再来理解「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，在经过浏览器的解释，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使用TCP而不是UDP

HTTP 使用 TCP 作为它的支撑运输协议（而不是在 UDP 上运行）。 HTTP 客户首先发起一个与服务器的 TCP 连接。 一旦连接建立，该浏览器和服务器进程就可以通过套接字接口访问 TCP。TCP 为HTTP 提供可靠数据传输服务 。这意味着， 一个客户进程发出的每个 HTTP 请求报文最终能完整地到达服务器；类似地，服务器进程发出的每个 HTTP 响应报文最终能完整地到达客户 。

这里我们看到了分层体系结构最大的优点，即 HTTP协议不用担心数据丢失，也不关注 TCP 从网络的数据丢失和乱序故障中恢复的细节 。 那是 TCP 以及协议栈较低层协议的工作。

HTTP是无状态协议

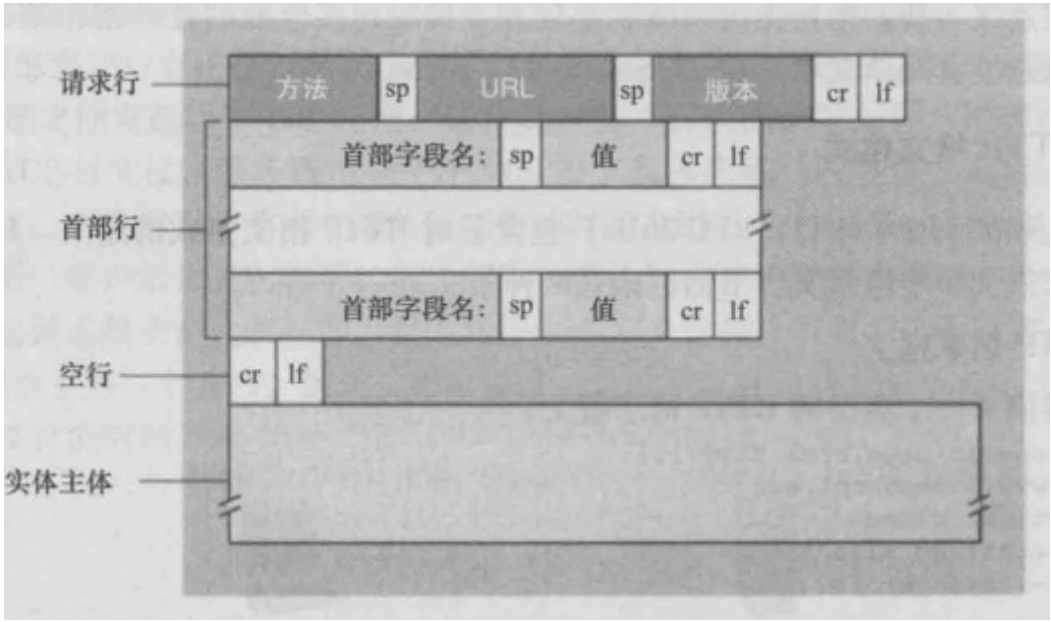
注意到下列现象很重要：服务器向客户发送被请求的文件，而不存储任何关于该客户的状态信息 。 假如某个特定的客户在短短的几秒钟内两次请求同一个对象，服务器并不会因为刚刚为该客户提供了该对象就不再做出反应，而是重新发送该对象，就像服务器已经完全忘记不久之前所做过的事一样 。 因为 HTTP 服务器并不保存关于客户的任何信息，所以我们说 HTTP 是一个**无状态协议（ stateless protocol ）** 。 我们同时也注意到 Web 使用了客户 - 服务器应用程序体系结构。

Web 服务器总是打开的，具有一个固定的 IP 地址，且它服务于可能来自数以百万计的不同客户端的请求。

HTTP报文格式

客户端发送一个请求报文给服务器，服务器根据请求报文中的信息进行处理，并将处理结果放入响应报文中返回给客户端。每一行数据必须通过"\r\n"分割，这里可以理解为行末标识符。

请求报文结构



HTTP请求报文格式

- 请求行（只有一行）

  结构：method、 URL、version

  - method：HTTP的请求方法，一共有9种，但GET和POST占了99%以上的使用频次。
  - URL：用来指代请求的文件。
  - version：HTTP协议的版本，该字段有HTTP/1.0和HTTP/1.1两种。
- 请求头（多行）

  在HTTP/1.1中，请求头除了Host都是可选的。包含的头五花八门，这里只介绍部分。

  - Host：指定请求资源的主机和端口号。端口号默认80。
  - Connection：值为keep-alive和close。keep-alive使客户端到服务器的连接持续有效，不需要每次重连，此功能为HTTP/1.1预设功能。
  - Accept：浏览器可接收的MIME类型。假设为text/html表示接收服务器回发的数据类型为text/html，如果服务器无法返回这种类型，返回406错误。
  - Cache-control：缓存控制，Public内容可以被任何缓存所缓存，Private内容只能被缓存到私有缓存，non-cache指所有内容都不会被缓存。
  - Cookie：将存储在本地的Cookie值发送给服务器，实现无状态的HTTP协议的会话跟踪。
  - Content-Length：请求消息正文长度。

  另有User-Agent、Accept-Encoding、Accept-Language、Accept-Charset、Content-Type等请求头这里不一一罗列。由此可见，请求报文是告知服务器请求的内容，而请求头是为了提供服务器一些关于客户机浏览器的基本信息，包括编码、是否缓存等。
- 空行（一行）
- 实体体（ entity body ）（多行）： 使用 GET 方法时实体体为空，而使用POST 方法时才使用该实体体 。 当用户提交表单时， HTTP 客户常常使用 POST 方法，例如当用户向搜索引擎提供搜索关键词时。 使用 POST 报文时，用户仍可以向服务器请求一个Web 页面，但 Web 页面的特定内容依赖于用户在表单字段中输入的内容 。 如果方法字段的值为 POST 时，则实体体中包含的就是用户在表单字段中的输人值。

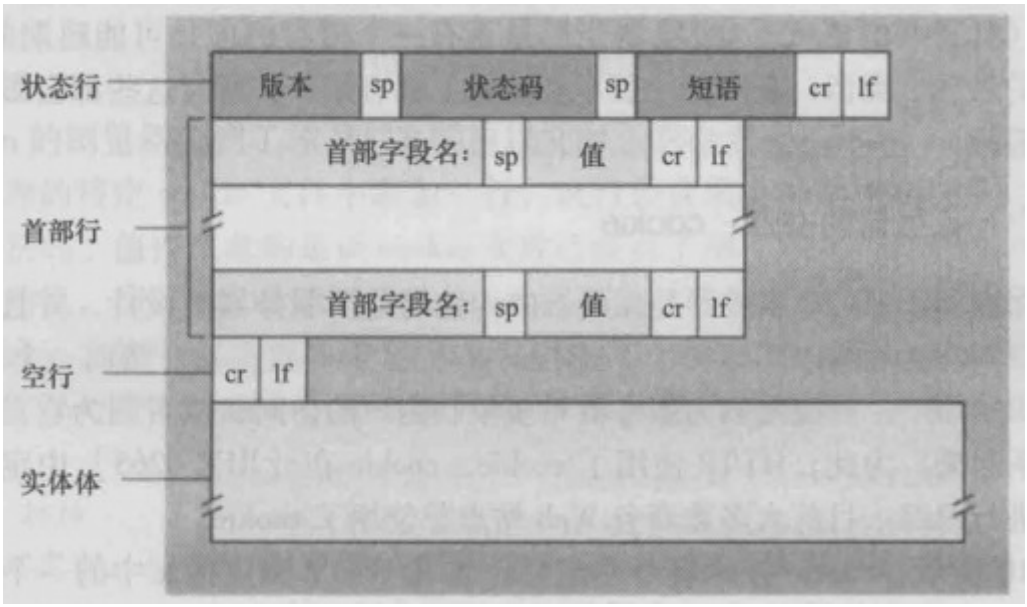
> **用表单生成的请求报文不是必须使用 POST 方法,也可以使用GET。**HTML表单经常使用 GET 方法，并在（表单字段中）所请求的 URL 中包括输入的数据 。
>
> 例如，一个表单使用 GET 方法，它有两个字段，分别填写的是“ monkeys “和“ bananas”，这样，该 URL 结构为www.somesite. com/animalsearcb?monkeys&bananas。又或者百度搜索HTTP链接为：https://www.baidu.com/s?&wd=HTTP，其中”wd=“后接的就是搜索的内容”HTTP“。

```
GET http://www.example.com/ HTTP/1.1
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9,en;q=0.8
Cache-Control: max-age=0
Host: www.example.com
If-Modified-Since: Thu, 17 Oct 2019 07:18:26 GMT
If-None-Match: "3147526947+gzip"
Proxy-Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 xxx

entity body
```

响应报文结构

响应报文是服务器对请求资源的响应，同样地，数据也是以"\r\n"来分割。



HTTP响应报文格式

- 报文头（一行）

  结构：version status_code status_message

  - version：描述所遵循的HTTP版本。
  - status_code 状态码 status_message 相应状态信息（与状态码对应）：

    状态码及其相应的短语指示了请求的结果，常见的见后文。
- 响应头（多行）
  - Date：表示信息发送的时间。
  - Server：Web服务器用来处理请求的软件信息。
  - Content-Encoding：Web服务器表明了自己用什么压缩方法压缩对象。
  - Content-Length：服务器告知浏览器自己响应的对象长度。
  - Content-Type：告知浏览器响应对象类型。
- 空行（一行）
- 实体体（多行）

  实际有效数据，通常是HTML格式的文件，该文件被浏览器获取到之后解析呈现在浏览器中。

```
HTTP/1.1 200 OK
```

```
Age: 529651
Cache-Control: max-age=604800
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 648
Content-Type: text/html; charset=UTF-8
Date: Mon, 02 Nov 2020 17:53:39 GMT
Etag: "3147526947+ident+gzip"
Expires: Mon, 09 Nov 2020 17:53:39 GMT
Keep-Alive: timeout=4
Last-Modified: Thu, 17 Oct 2019 07:18:26 GMT
Proxy-Connection: keep-alive
Server: ECS (sjc/16DF)
Vary: Accept-Encoding
X-Cache: HIT

<!doctype html>
<html>
<head>
    <title>Example Domain</title>
    // 省略...
</body>
</html>
```

HTTP 方法

客户端发送的 **请求报文** 第一行为请求行，包含了方法字段。

DELETE

> 删除文件

与 PUT 功能相反，并且同样不带验证机制。

```
DELETE /file.html HTTP/1.1
```

OPTIONS

> 查询支持的方法

查询指定的 URL 能够支持的方法。

会返回 `Allow: GET, POST, HEAD, OPTIONS` 这样的内容。

TRACE

> 追踪路径

服务器会将通信路径返回给客户端。

发送请求时，在 Max-Forwards 首部字段中填入数值，每经过一个服务器就会减 1，当数值为 0 时就停止传输。

通常不会使用 TRACE，并且它容易受到 XST 攻击（Cross-Site Tracing，跨站追踪）。

GET（最常用）

> 获取资源

当前网络请求中，绝大部分使用的是 GET 方法。

POST（常用）

> 传输实体主体

POST 主要用来传输数据，而 GET 主要用来获取资源。

GET和POST的区别

本质

- GET ： 通过URL**从服务器获取资源**，这个资源可以是静态的文本、页面、图片视频等。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）。
- POST：**向服务器提交数据**，数据就放在报文的 body 里，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）。

区别

- GET 通过 url 传递参数， POST 放在 request body 中。因此GET不适合用来传递敏感信息。
- GET 提交的数据最大是2k（ 限制实际上取决于浏览器）， POST 理论上没有限制。
- GET 请求会浏览器主动 cache， 而 POST 不会。
- GET 和 POST 本质上就是 TCP 链接， 并无差别。 但是由于 HTTP 的规定和浏览器/服务器的限制，导致他们在应用过程中体现出一些不同。
- GET 产生一个 TCP 数据包； POST 产生两个 TCP 数据包。
- 安全性：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。
    - GET安全，GET是「只读」操作，不修改服务器上的数据
    - POST不安全，POST是「新增或提交数据」的操作，会修改服务器上的资源。
- 幂等性：多次执行相同的操作，结果都是「相同」的。
    - GET幂等，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。
    - POST不幂等，POST多次提交数据就会创建多个资源，服务端多次处理。

HTTP 首部

有 4 种类型的首部字段：通用首部字段、请求首部字段、响应首部字段和实体首部字段。

各种首部字段及其含义如下（不需要全记，仅供查阅）：

通用首部字段

首部字段名	说明
Cache-Control	控制缓存的行为
Connection	控制不再转发给代理的首部字段、管理持久连接
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

请求首部字段

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

响应首部字段

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

实体首部字段

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
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

HTTP 状态码

服务器返回的 **响应报文** 中第一行为状态行，包含了状态码以及原因短语，用来告知客户端请求的结果。

状态码	类别	含义
1XX	Informational（信息性状态码）	接收的请求正在处理
2XX	Success（成功状态码）	请求正常处理完毕
3XX	Redirection（重定向状态码）	需要进行附加操作以完成请求
4XX	Client Error（客户端错误状态码）	服务器无法处理请求
5XX	Server Error（服务器错误状态码）	服务器处理请求出错

1XX 信息

> `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。

- **100 Continue** ： 表明到目前为止都很正常，客户端可以继续发送请求或者忽略这个响应。

2XX 成功

> `2xx` 类状态码表示服务器**成功**处理了客户端的请求。

- **200 OK**
- **204 No Content** ： 请求已经成功处理，但是返回的响应报文不包含实体的主体部分。一般在只需要从客户端往服务器发送信息，而不需要返回数据时使用。
- **206 Partial Content** ： 表示客户端进行了范围请求，响应报文包含由 Content-Range 指定范围的实体内容。

3XX 重定向

> `3xx` 类状态码表示客户端请求的资源发送了变动，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** ： 永久性重定向
- **302 Found** ： 临时性重定向
- **303 See Other** ： 和 302 有着相同的功能，但是 303 明确要求客户端应该采用 GET 方法获取资源。
- 注：虽然 HTTP 协议规定 301、302 状态下重定向时不允许把 POST 方法改成 GET 方法，但是大多数浏览器都会在 301、302 和 303 状态下的重定向把 POST 方法改成 GET 方法。
- **304 Not Modified** ： 如果请求报文首部包含一些条件，例如：If-Match，If-Modified-Since，If-None-Match，If-Range，If-Unmodified-Since，如果不满足条件，则服务器会返回 304 状态码。
- **307 Temporary Redirect** ： 临时重定向，与 302 的含义类似，但是 307 要求浏览器不会把重定向请求的 POST 方法改成 GET 方法。

4XX 客户端错误

> `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** ： 请求报文中存在语法错误。
- **401 Unauthorized** ： 该状态码表示发送的请求需要有认证信息（BASIC 认证、DIGEST 认证）。如果之前已进行过一次请求，则表示用户认证失败。
- **403 Forbidden** ： 请求被拒绝。
- **404 Not Found**

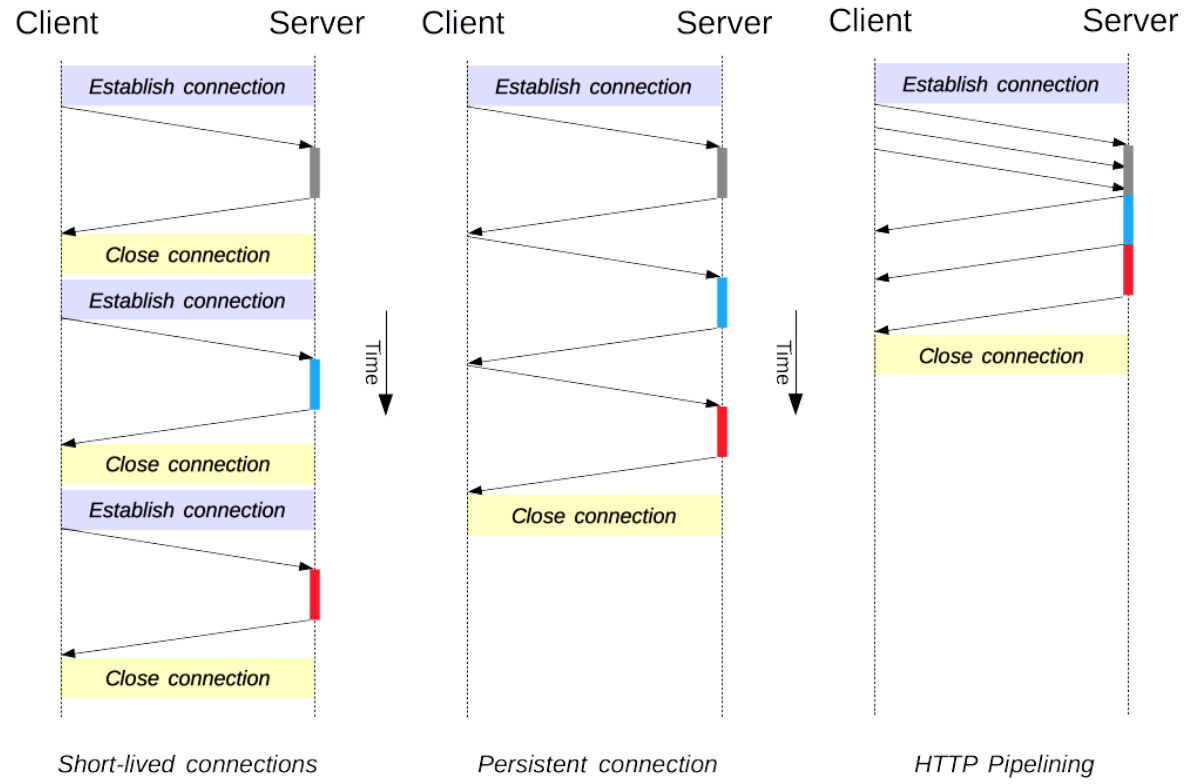
5XX 服务器错误

> `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** ： 服务器正在执行请求时发生错误。
- **503 Service Unavailable** ： 服务器暂时处于超负载或正在进行停机维护，现在无法处理请求。

HTTP连接与状态

短连接与长连接



概念

客户 - 服务器的交互是经 TCP 进行的，应用程序的研制者就需要做一个重要决定，即每个请求 / 响应对是经一个单独的 TCP 连接发送，还是所有的请求及其响应经相同的 TCP 连接发送呢?

采用前一种方法，该应用程序被称为使用非持续连接（短连接 ）；

采用后一种方法，该应用程序被称为使用持续连接（长连接） 。

短连接的劣势

> 往返时间（ Round- Trip Time, RTT）的定义．该时间是指一个短分组从客户到服务器然后再返回客户所花费的时间 。 RTT 包括分组传播时延、分组在中间路由器和交换机上的排队时延以及分组处理时延。

非持续连接有一些缺点 。 首先，必须为每一个请求的对象建立和维护一个全新的连接 。 对于每个这样的连接，在客户和服务器中都要分配 TCP 的缓冲区和保持 TCP 变量，这给 Web 服务器带来了严重的负担，因为一 台 Web 服务器可能同时服务于数以百计不同的客户的请求 。 第二 ， 就像我们刚描述的那样，每一个对象经受两倍 RTT 的交付时延，即一个 RTT 用于创建 TCP ， 另一个 RTT 用于请求和接收一个对象 。

长连接只需要建立一次 TCP 连接就能进行多次 HTTP 通信。

流水线

默认情况下，HTTP 请求是按顺序发出的，下一个请求只有在当前请求收到响应之后才会被发出。由于受到网络延迟和带宽的限制，在下一个请求被发送到服务器之前，可能需要等待很长时间。

流水线是在同一条长连接上连续发出请求，而不用等待响应返回，这样可以减少延迟。

带流水线的长连接

在采用持续连接的情况下，服务器在发送响应后保持该 TCP 连接打开。在相同的客户与服务器之间的后续请求和响应报文能够通过相同的连接进行传送 。 一般来说，如果一条连接经过一定时间间隔（ 一个可配置的超时间隔）仍未被使用， HTTP 服务器就关闭该连接。**HTTP 的默认模式是使用带流水线的长连接（持续连接）。**

- 在 HTTP/1.1 之前默认是短连接的，如果需要使用长连接，则使用 `Connection : Keep-Alive`。
- 从 HTTP/1.1 开始默认是长连接的，如果要断开连接，需要由客户端或者服务器端提出断开，使用 `Connection : close`；

Cookie与Session

HTTP作为无状态协议，必然需要在某种方式保持连接状态。这里简要介绍一下Cookie和Session。

Cookie

Cookie是**客户端**保持状态的方法。

Cookie简单的理解就是存储由服务器发至客户端并由客户端保存的一段字符串。为了保持会话，服务器可以在响应客户端请求时将Cookie字符串放在Set-Cookie下，客户机收到Cookie之后保存这段字符串，之后再请求时候带上Cookie就可以被识别。

除了上面提到的这些，Cookie在客户端的保存形式可以有两种，一种是会话Cookie一种是持久Cookie，会话Cookie就是将服务器返回的Cookie字符串保持在内存中，关闭浏览器之后自动销毁，持久Cookie则是存储在客户端磁盘上，其有效时间在服务器响应头中被指定，在有效期内，客户端再次请求服务器时都可以直接从本地取出。需要说明的是，存储在磁盘中的Cookie是可以被多个浏览器代理所共享的。

Session

Session是**服务器**保持状态的方法。

首先需要明确的是，Session保存在服务器上，可以保存在数据库、文件或内存中，每个用户有独立的Session用户在客户端上记录用户的操作。我们可以理解为每个用户有一个独一无二的Session ID作为Session文件的Hash键，通过这个值可以锁定具体的Session结构的数据，这个Session结构中存储了用户操作行为。

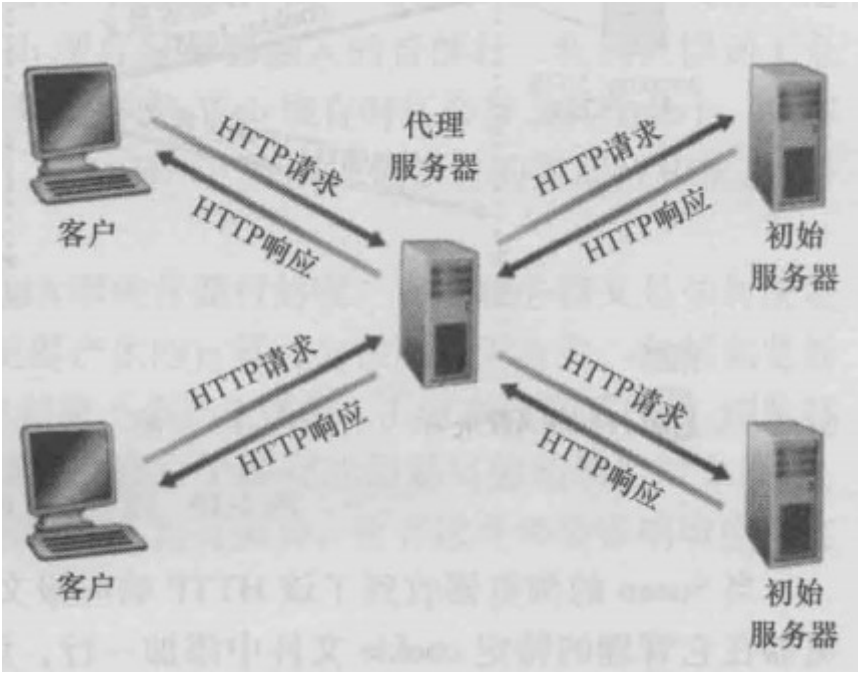
综合

当服务器需要识别客户端时就需要结合Cookie了。每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用Cookie来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在Cookie里面记录一个Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了。如果客户端的浏览器禁用了Cookie，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如sid=xxxxx这样的参数，服务端据此来识别用户，这样就可以帮用户完成诸如用户名等信息自动填入的操作了。

Web缓存与条件GET

概念

Web 缓存器 （ Web cache ）也叫 **代理服务器 （ proxy server）**，它是能够代表初始 Web 服务器来满足 HTTP 请求的网络实体 。 Web 缓存器有自己的磁盘存储空间，并在存储空间中保存最近请求过的对象的副本 。



客户通过 Web 援存器请求对象

客户通过 Web 援存器请求对象的过程：

假设浏览器正在请求对象 http://www.someschool.edu/campus.gif，将会发生如下情况：

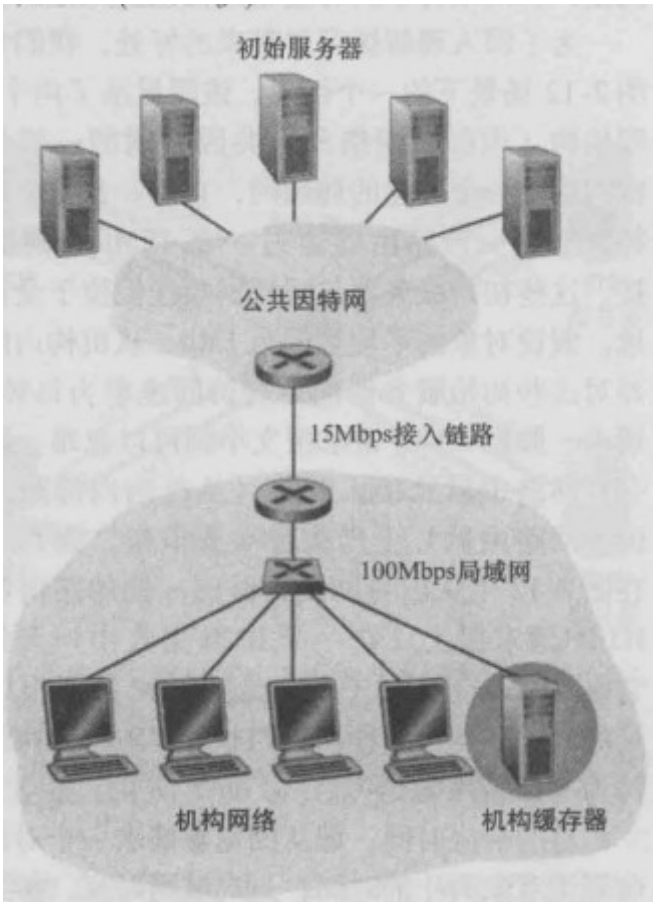
- 浏览器建立一个到 Web 缓存器的 TCP 连接，并向 Web 缓存器中的对象发送一个HTTP 请求 。
- Web 缓存器进行检查，看看本地是否存储了该对象副本 。 如果有， Web 缓存器就向客户浏览器用 HTTP 响应报文返回该对象。
- 如果 Web 缓存器中没有该对象，它就打开一个与该对象的初始服务器（如www. someschool. edu ）的 TCP 连接 。 Web 缓存器则在这个缓存器到服务器的 TCP连接上发送一个对该对象的 HTTP 请求 。 在收到该请求后，初始服务器向该 Web缓存器发送具有该对象的 HTTP响应 。
- 当 Web 缓存器接收到该对象时，它在本地存储空间存储一份副本，并向客户的浏览器用HTTP 响应报文发送该副本（通过现有的客户浏览器和 Web 缓存器之间的TCP 连接）。

部署 Web 缓存器的原因：

- Web 缓存器可以大大减少对客户请求的响应时间，特别是当客户与初始服务器之间的瓶颈带宽远低于客户与 Web 缓存器之间的瓶颈带宽时更是如此。

  > 如果在客户与 Web 缓存器之间有一个高速连接（情况常常如此），并且如果用户 Yr请求的对象在 Web 缓存器上，则 Web 缓存器可以迅速将该对象交付给用户。

- Web 缓存器能够大大减少一个机构的接入链路到因特网的通信量 。 通过减少通信量，该机构（如一家公司或者一所大学）就不必急于增加带宽，因此降低了费用 。 此外， Web 缓存器能从整体上大大减低因特网上的 Web流量，从而改善了所有应用的性能。



机构网络中添加Web缓存器

内容分发网络CDN

通过使用**内容分发网络（Content Distribulion Network, CDN)**, Web 缓存器正在因特网中发挥着越来越重要的作用。 CDN 公司在因特网上安装了许多地理上分散的缓存器，因而使大量流量实现了本地化。

条件GET

尽管高速缓存能减少用户感受到的响应时间，但也引人了一个新的问题，即存放在缓存器中的对象副本可能是陈旧的。换句话说，保存在服务器中的对象自该副本缓存在客户上以后可能已经被修改了。幸运的是， HTTP 协议有一种机制，允许缓存器证实它的对象是最新的。 这种机制就是**条件GET (conditional GET)** 方法。

条件 GET 请求报文：请求报文使用 **GET** 方法；请求报文中包含一个**" If- Modified-Since：”首部行。**

条件GET的例子

> 首先，一个代理缓存器（proxy cache）代表一个请求浏览器，向某 Web 服务器发送一个请求报文：
>
> ```
> GET /frilit/kiwi.gif HTTP/1.1
> Host: www.exotiquecuisine.com
> ```
>
> 其次，该 Web 服务器向缓存器发送具有被请求的对象的响应报文：

```
HTTP/1.1 200 OK
Date: Sat, 8 Oct 2011 15:39:29
Server: Apache/ 1. 3 . 0 (Unix)
Last-Modified: Wed, 7 Sep 2011 09:23:24
Content-Type: image/gif

(data data data data data . .. )
```

该缓存器在将对象转发到请求的浏览器的同时，也在本地缓存了该对象 。 重要的是，缓存器在存储该对象时也存储了最后修改日期 。 第三，一个星期后，另一个用户经过该缓存器请求同一个对象，该对象仍在这个缓存器中 。 由于在过去的一个星期巾位于 Web 服务器上的该对象可能已经被修改了，该缓存器通过发送一个条件 GET 执行最新检查 。 具体说来，该缓存器发送：

```
GET /fruit/kiwi.git HTTP/1.1
Host: www.exotiquecuisine.com
If-Modified-Since: Wed, 7 Sep 2011 09:23:24

(empty entity body)
```

值得注意的是 ff-Modified-Since： 首部行的值正好等于一星期前服务器发送的响应报文中的 La.st- Modified： 首部行的值 。 该条件 GET 报文告诉服务器，仅当自指定日期之后该对象被修改过，才发送该对象。 假设该对象自 2011 年 9 月 7 日 09: 23: 24 后没有被修改 。接下来的第四步， Web 服务器向该缓存器发送一个响应报文：

```
HTTP/1.1 304 Not Modified
Date: Sat, 15 Oct 2011 15:39:29
Server: Apache/ 1. 3. 0 (Unix)

(empty entity body)
```

我们看到，作为对该条件 GET 方法的响应，该 Web 服务器仍发送一个响应报文，但并没有在该响应报文中包含所请求的对象。 包含该对象只会浪费带宽，并增加用户感受到的响应时间，特别是如果该对象很大的时候更是如此。 值得注意的是在最后的响应报文中，状态行中为 304 Not Modified ，它告诉缓存器可以使用该对象，能向请求的浏览器转发它（该代理缓存器）缓存的该对象副本 。

HTTP 特性

优点

HTTP 最凸出的优点是「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。**

HTTPS 也就是在 HTTP 与 TCP 层之间增加了 SSL/TLS 安全传输层，HTTP/3 甚至把 TCP 层换成了基于 UDP 的 QUIC。

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**片地开花**，同时天然具有**跨平台**的优越性。

缺点

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。

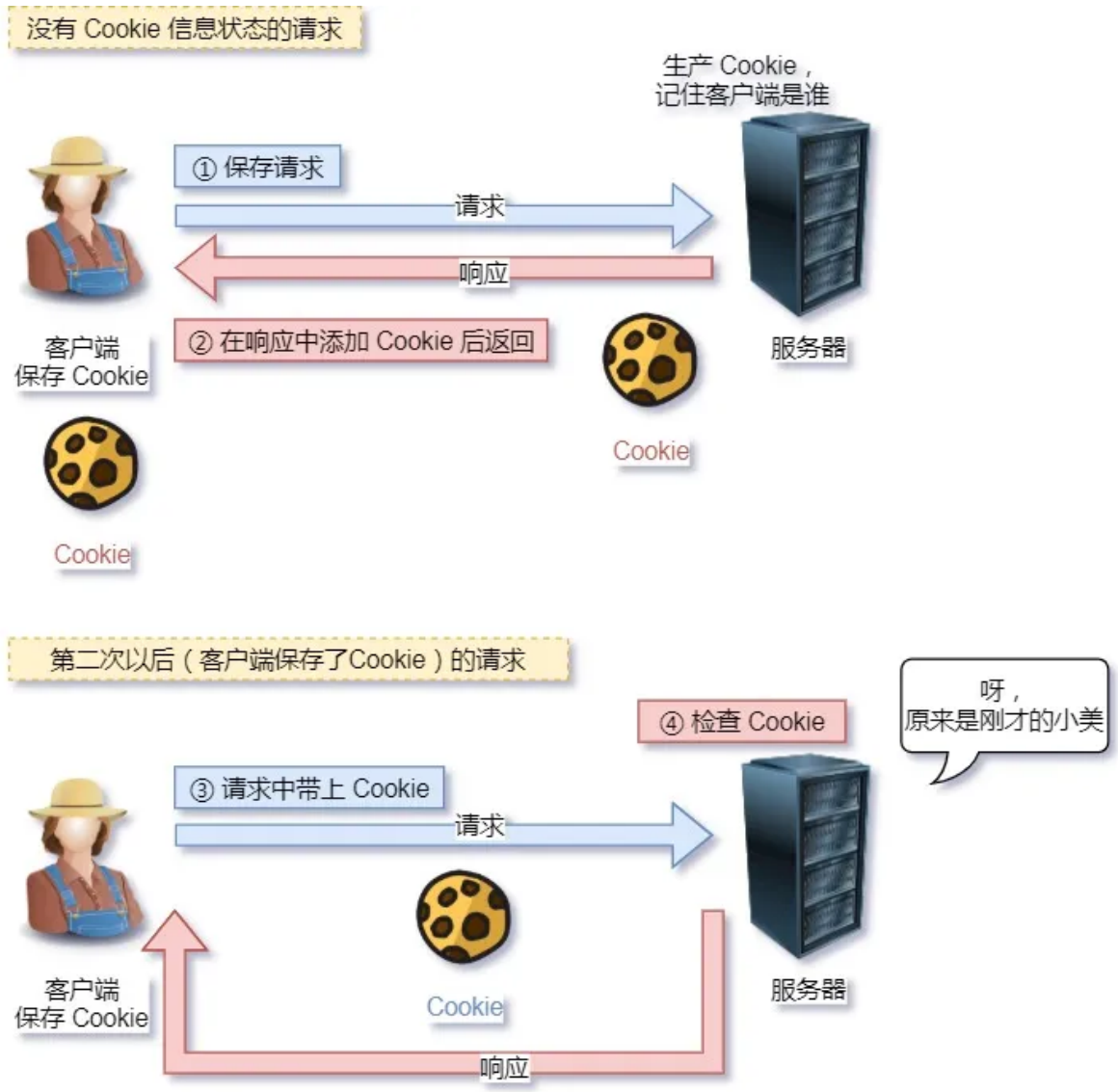
例如登录->添加购物车->下单->结算->支付，这系列操作都要知道用户的身份才行。但服务器不知道这些请求是有关联的，每次都要问一遍身份信息。

这样每操作一次，都要验证信息，这样的购物体验还能愉快吗？别问，问就是**酸爽**！

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**，



Cookie 技术

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，通过浏览器的 F12 控制台或 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，如果里面有你的账号密码信息，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。