# HTTP

* HTTP构建于TCP/IP协议之上，默认端口号是80。
* HTTP是 **无连接无状态** 的。

无连接的含义是 **限制每次连接只处理一个请求**。服务器处理完客户的请求，并收到客户的应答后，即断开连接。后来使用了Keep-Alive技术。

无状态是指 **协议对于事务处理没有记忆能力，服务器不知道客户端是什么状态**。即我们给服务器发送 HTTP 请求之后，服务器根据请求，会给我们发送数据过来，但是，发送完，不会记录任何信息。

HTTP 协议这种特性有优点也有缺点，优点在于解放了服务器，每一次请求“点到为止”不会造成不必要连接占用，**缺点在于每次请求会传输大量重复的内容信息**。

为了解决HTTP无状态的缺点，两种用于保持 HTTP 连接状态的技术就应运而生了，一个是 Cookie，而另一个则是 Session。Cookie在客户端记录状态，比如登录状态。Session在服务器记录状态。

## Http的报文结构

### HTTP 请求报文头部

* User-Agent：产生请求的浏览器类型。
* Accept：客户端可识别的响应内容类型列表;
* Accept-Language：客户端可接受的自然语言;
* Accept-Encoding：客户端可接受的编码压缩格式;
* Accept-Charset：可接受的应答的字符集;
* Host：请求的主机名，允许多个域名同处一个IP 地址，即虚拟主机;（必选）
* Connection：连接方式(close 或 keep-alive);
* Cookie：存储于客户端扩展字段，向同一域名的服务端发送属于该域的cookie;
* 请求包体：在POST方法中使用。
* Referer：包含一个URL，用户从该URL代表的页面出发访问当前请求的页面。
* If-Modified-Since：文档的最后改动时间

### HTTP 响应头

* Allow 服务器支持哪些请求方法（如GET、POST等）。
* Content-Encoding 文档的编码（Encode）方法。
* Content-Length 表示内容长度。只有当浏览器使用持久HTTP连接时才需要这个数据。
* Content-Type 表示后面的文档属于什么MIME类型。
* Date 当前的GMT时间。你可以用setDateHeader来设置这个头以避免转换时间格式的麻烦。
* Expires 应该在什么时候认为文档已经过期，从而不再缓存它。
* Last-Modified 文档的最后改动时间。
* Refresh 表示浏览器应该在多少时间之后刷新文档，以秒计。
* Server 服务器名字。
* Set-Cookie 设置和页面关联的Cookie。
* ETag：被请求变量的实体值。ETag是一个可以与Web资源关联的记号（MD5值）。
* Cache-Control：这个字段用于指定所有缓存机制在整个请求/响应链中必须服从的指令。

max-age：表示当访问此网页后的 x 秒内再次访问不会去服务器；no-cache，实际上Cache-Control: no-cache是会被缓存的，只不过每次在向客户端（浏览器）提供响应数据时，缓存都要向服务器评估缓存响应的有效性；no-store，这个才是响应不被缓存的意思；

Last-Modified与If-Modified-Since都是用来记录页面的最后修改时间。当客户端访问页面时，服务器会将页面最后修改时间通过 Last-Modified 标识由服务器发往客户端，客户端记录修改时间，再次请求本地存在的cache页面时，客户端会通过 If-Modified-Since 头将先前服务器端发过来的最后修改时间戳发送回去，服务器端通过这个时间戳判断客户端的页面是否是最新的，如果不是最新的，则返回新的内容，如果是最新的，则返回 304。

## Http的状态码含义。

* 1\*\* 信息，服务器收到请求，需要请求者继续执行操作
* 2\*\* 成功，操作被成功接收并处理
* 3\*\* 重定向，需要进一步的操作以完成请求
  + 301 Moved Permanently。请求的资源已被永久的移动到新URI，返回信息会包括新的URI，浏览器会自动定向到新URI。今后任何新的请求都应使用新的URI代替
  + 302 Moved Temporarily。与301类似。但资源只是临时被移动。客户端应继续使用原有URI
  + 304 Not Modified。所请求的资源未修改，服务器返回此状态码时，不会返回任何资源。**客户端通常会缓存访问过的资源，通过提供一个头信息指出客户端希望只返回在指定日期之后修改的资源**。
* 4\*\* 客户端错误，请求包含语法错误或无法完成请求
  + 400 Bad Request 由于客户端请求有语法错误，不能被服务器所理解。
  + 401 Unauthorized 请求未经授权。这个状态代码必须和WWW-Authenticate报头域一起使用
  + 403 Forbidden 服务器收到请求，但是拒绝提供服务。服务器通常会在响应正文中给出不提供服务的原因
  + 404 Not Found 请求的资源不存在，例如，输入了错误的URL
* 5\*\* 服务器错误，服务器在处理请求的过程中发生了错误
  + 500 Internal Server Error 服务器发生不可预期的错误，导致无法完成客户端的请求。
  + 503 Service Unavailable 服务器当前不能够处理客户端的请求，在一段时间之后，服务器可能会恢复正常。

## Http request的几种类型。

* GET 请求指定的页面信息，并返回实体主体。
* POST 向指定资源提交数据进行处理请求（例如提交表单或者上传文件）。数据被包含在请求体中。POST请求可能会导致新的资源的建立和/或已有资源的修改。
* PUT 从客户端向服务器传送的数据取代指定的文档的内容。
* DELETE 请求服务器删除指定的页面。

GET可提交的数据量受到URL长度的限制，HTTP协议规范没有对URL长度进行限制。这个限制是特定的浏览器及服务器对它的限制

理论上讲，POST是没有大小限制的，HTTP协议规范也没有进行大小限制，出于安全考虑，服务器软件在实现时会做一定限制

## 条件 GET

HTTP条件GET 是 HTTP 协议为了减少不必要的带宽浪费，提出的一种方案。实际上就是利用If-Modified-Since做浏览器缓存。

## 持久连接

我们知道 HTTP 协议采用请求-应答模式，当使用普通模式，即非 Keep-Alive 模式时，每个请求/应答客户和服务器都要新建一个连接，完成之后立即断开连接（HTTP协议为无连接的协议）；当使用 Keep-Alive 模式（又称持久连接、连接重用）时，Keep-Alive 功能使客户端到服务器端的连接持续有效，当出现对服务器的后继请求时，Keep-Alive 功能避免了建立或者重新建立连接。

在 HTTP 1.0 中, 没有官方的 keep alive 的操作。通常是在现有协议上添加一个指数。如果浏览器支持 keep-alive，它会在请求的包头中添加：

Connection: Keep-Alive

然后当服务器收到请求，作出回应的时候，它也添加一个头在响应中：

Connection: Keep-Alive

这样做，连接就不会中断（超过 Keep-Alive 规定的时间--服务器设置，意外断电等情况除外），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。这一直继续到客户端或服务器端认为会话已经结束，其中一方中断连接。

在 HTTP 1.1 版本中，默认情况下所有连接都被保持，如果加入 "Connection: close" 才关闭。

HTTP Keep-Alive 简单说就是保持当前的TCP连接，避免了重新建立连接。

**HTTP 长连接不可能一直保持，例如 Keep-Alive: timeout=5, max=100，表示这个TCP通道可以保持5秒，max=100，表示这个长连接最多接收100次请求就断开**。

HTTP是一个无状态协议，这意味着每个请求都是独立的，Keep-Alive没能改变这个结果。另外，**Keep-Alive也不能保证客户端和服务器之间的连接一定是活跃的**，在HTTP1.1版本中也如此。唯一能保证的就是当连接被关闭时你能得到一个通知，所以不应该让程序依赖于Keep-Alive的保持连接特性，否则会有意想不到的后果。

使用长连接之后，客户端、服务端怎么知道本次传输结束呢？两部分：1. 判断传输数据是否达到了Content-Length 指示的大小；2. 动态生成的文件没有 Content-Length ，它是分块传输（chunked），这时候就要根据 chunked 编码来判断，chunked 编码的数据在最后有一个空 chunked 块，表明本次传输数据结束。

## 跨站攻击

CSRF（Cross-site request forgery，跨站请求伪造）伪造请求，冒充用户在站内的正常操作，比如爬虫。

### 防范的方法

* 关键操作只接受POST请求
* 验证码
* 检测 Referer
* Token
  + Token 要足够随机——只有这样才算不可预测
  + Token 是一次性的，即每次请求成功后要更新Token——这样可以增加攻击难度，增加预测难度
  + Token 要注意保密性——敏感操作使用 post，防止 Token 出现在 URL 中

## 断点续传

要实现断点续传的功能，通常都需要客户端记录下当前的下载进度，并在需要续传的时候通知服务端本次需要下载的内容片段。

HTTP1.1协议中定义了断点续传相关的HTTP头 Range 和 Content-Range 字段，一个最简单的断点续传实现大概如下： 1. 客户端下载一个1024K的文件，已经下载了其中512K 2. 网络中断，客户端请求续传，因此需要在HTTP头中申明本次需要续传的片段：Range:bytes=512000-，这个头通知服务端从文件的512K位置开始传输文件。 3. 服务端收到断点续传请求，从文件的512K位置开始传输，并且在HTTP头中增加：Content-Range:bytes 512000-/1024000，并且此时服务端返回的HTTP状态码应该是206，而不是200。

但是在实际场景中，会出现一种情况，即在终端发起续传请求时，URL对应的文件内容在服务端已经发生变化，此时续传的数据肯定是错误的。如何解决这个问题了？显然此时我们需要有一个标识文件唯一性的方法。在RFC2616中也有相应的定义，比如 **实现Last-Modified来标识文件的最后修改时间，这样即可判断出续传文件时是否已经发生过改动**。同时RFC2616中还定义有一个ETag的头，可以使用ETag头来放置文件的唯一标识，比如文件的MD5值。

客户端在发起续传请求时应该在HTTP头中申明If-Match 或者 If-Modified-Since 字段，帮助服务端判别文件变化。

## 一次HTTP请求

1. 域名解析
   1. 浏览器缓存
   2. 系统缓存
   3. hosts
   4. ISP DNS 缓存
   5. DNS 服务器搜索
2. 浏览器发送 HTTP 请求到目标服务器
3. 服务器永久重定向
4. 浏览器跟踪重定向地址
5. 服务器“处理”请求
6. 服务器发回一个HTML响应
7. 浏览器开始显示HTML
8. 浏览器请求获取嵌入在 HTML 中的对象（图片&脚本等）
9. 浏览器发送异步（AJAX）请求