前端需要了解的HTTP知识

> 本文主要讲述一些与前端开发相关的HTTP知识，扩展得有点泛。

简述

HTTP协议最初只是为了获取基于文本的静态资源而开发的简单协议，但后来人们以各种形式扩展和利用它，使其能够支持如今常见的复杂分布式应用程序。

HTTP使用基于消息的模型：客户端发送一条请求消息，服务器返回一条响应消息。该协议基本上不需要连接，虽然HTTP使用TCP协议作为传输机制，但每次请求和响应都是无状态的，可能使用不同的TCP连接。

HTTP请求响应

HTTP请求格式为：一行或几行单行显示的消息头（`header`），加上一个强制空白行，以及可选的消息主题（`body`）。例如百度首页的请求如下所示：

> 关于`header`的详细内容，参考HTTP/1.1 Header Field Definitions

HTTP请求响应示例

```
Remote Address:115.239.210.27:80          # 远程主机IP地址
Request URL:http://www.baidu.com/         # 请求的URL地址
Request Method:GET                        # 说明HTTP方法的动词
Status Code:200 OK                        # HTTP响应状态码和响应状
态内容

Request Headers
GET / HTTP/1.1                            # 使用的HTTP动词和HTTP
版本，最常见的是`GET`和`POST`。
```

> 因特网常用HTTP版本为1.0和1.1，多数浏览器默认使用1.1版本。在攻击WEB应用时可能遇到的唯一差异是1.1版本必须使用Host请求头。实际上目前所有我们用到的浏览器都在使用HTTP1.1了，而且，POST请求也依赖HTTP1.1。记住，HTTP2(基于SPDY)也在蓄势待发中咯。

> [参考链接](http://stackoverflow.com/questions/535053/which-webbrowsers-use-http-1-1-by-default)

```
Host: www.baidu.com                         # HTTP/1.1必需的请求头信息，如果基于HTTP/1.1的服务器收到的请求没有该请求头，会返回`400(Bad Request)`报错
Connection: keep-alive                      # HTTP/1.1默认为持久连接

Cache-Control: max-age=0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_4) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/35.0.1916.153 Safari/537.36     # 因为历史原因，大多数浏览器都包含`Mozilla`前缀。这是因为当初的“老大”Netscape使用了`User-Agent`字符串，而其他浏览器也希望让网站相信他们兼容这种标准。
DNT: 1
Referer: http://baidu.com/                  # 也是一个重要的安全请求头，参考[HTTP Referrer详解](http://www.atatech.org/articles/17771)
Accept-Encoding: gzip,deflate,sdch
```

> HTTP压缩方法：对应请求头中的`Content-Encoding`，有的安全软件会通过代理强制使用非压缩方法，从而使得网络资源加载速度变慢。关于`deflate`方法，微软的服务器和客户端因为历史原因是以`raw`（未处理）的`deflated stream`来实现的，因此，一些软件，包括Apache服务器，只实现了`gzip`压缩编码的方法。[详情](http://en.wikipedia.org/wiki/HTTP_compression)

> 关于这个请求头出现过比较严重的安全问题：[BREACH](http://en.wikipedia.org/wiki/BREACH)和[CRIME](http://en.wikipedia.org/wiki/CRIME)**注意：影响包括TLS协议以及SPDY和HTTP协议。**

> 关于`sdch`（`Shared Dictionary Compression over HTTP`）可以参考[google group的讨论](https://groups.google.com/forum/#!forum/SDCH)
> [qq空间](http://i.qq.com)使用了`sdch`压缩，可以去围观下。

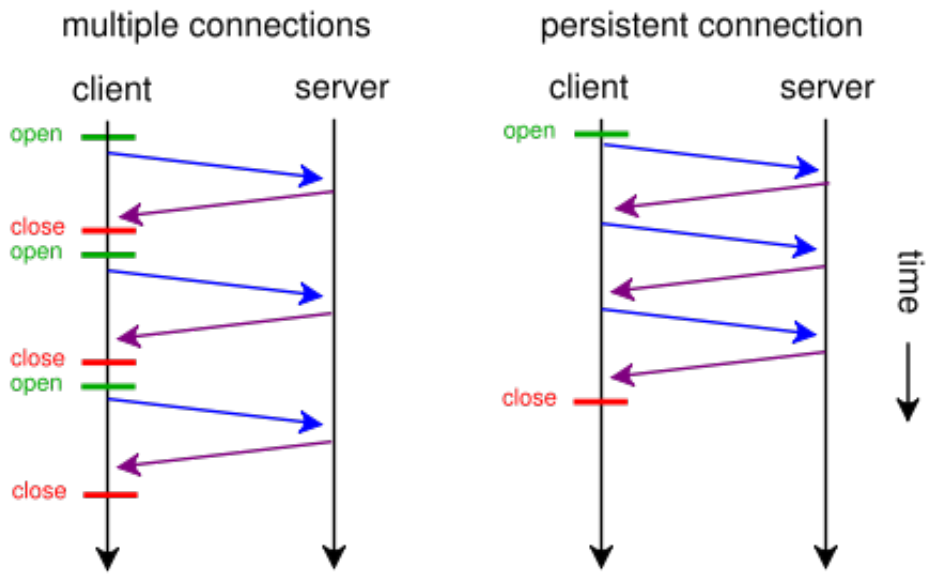
```
Accept-Language: zh-CN,zh;q=0.8,en;q=0.6
Cookie: BAIDUID=2E8A983DC66A95B9B3B42D17908D11FB:FG=1; H_PS
_PSSID=7544_7618_1468_7571_5225_6997_7447_7541_7532_6506_60
17_7202_6930_7253_6887_7649_7595_7474        # 让无状态的HTTP
变成有状态的功臣，详情可参考[Cookie与安全](http://www.atatech.org
/articles/13455)以及[HTTP Cookie](http://www.atatech.org/art
icles/13453)

Response Headers
HTTP/1.1 200 OK
Date: Sat, 12 Jul 2014 06:59:33 GMT
Content-Type: text/html; charset=utf-8     #  响应内容类型，`t
ext/html`表明是HTML文件的网页
Transfer-Encoding: chunked
Connection: Keep-Alive
Vary: Accept-Encoding
Cache-Control: private
Cxy_all: baidu+021bdc3e988aa98d3b0dee965c6c73b4
Expires: Sat, 12 Jul 2014 06:59:27 GMT
X-Powered-By: HPHP
Server: BWS/1.1
BDPAGETYPE: 1
BDQID: 0xad31e807000690c2
BDUSERID: 0
Set-Cookie: BDSVRTM=0; path=/      向浏览器发送一个Cookie，在随
后的请求中由Cookie消息头返回。
Set-Cookie: H_PS_PSSID=7544_7618_1468_7571_5225_6997_7447_7
541_7532_6506_6017_7202_6930_7253_6887_7649_7595_7474; path
=/; domain=.baidu.com
Content-Encoding: gzip
```

> 还有例如 `pragma: no-cache` 指示浏览器不要缓存响应内容。 `Expires:` 消息头指示响应内容的过期时间等。

`HTTP Persistent Connection`（HTTP持久连接）

`HTTP Persistent Connection` 也成为 `HTTP Keep-alive` 或者 `HTTP Connection Resue` 。也就是用一个TCP连接来发送和接收多个HTTP请求响应。SPDY协议用到了相同的理念，但更进一步支持在一个连接上多路(`multiplex`)传输多个并发请求/响应。示意图如下：



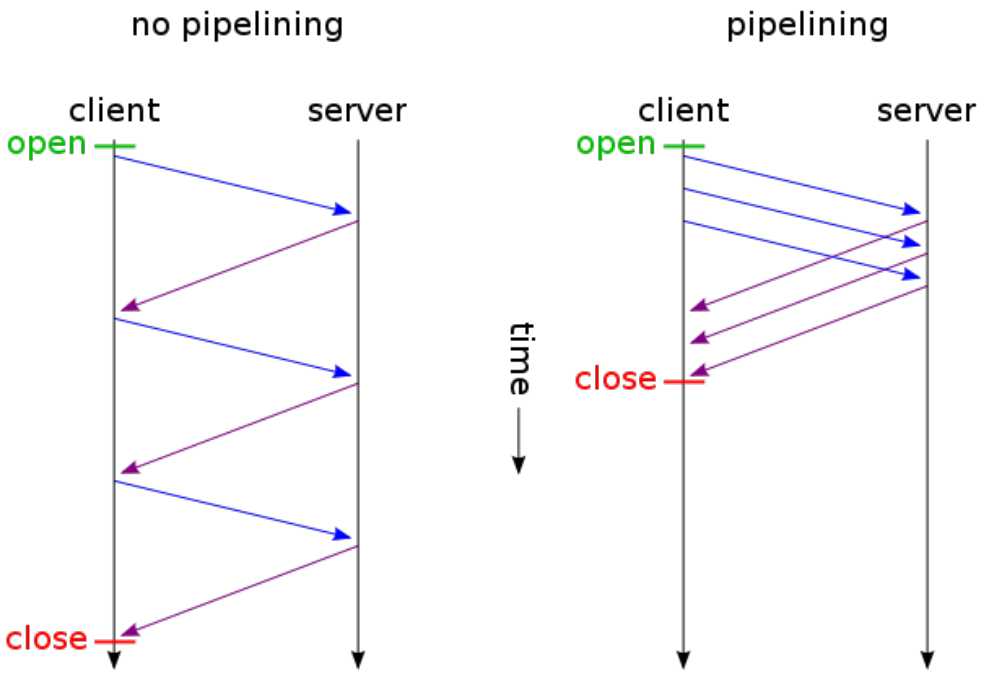
支持 `HTTP Persistent Connection` 的客户端和服务器在请求和响应头中都会带上：

```
Connection: Keep-Alive
```

`HTTP Pipelining`（HTTP流水线技术）

`HTTP Pipelining` 是一种在单个TCP连接中发送多个HTTP，而不用等待相关响应的技术。但是在实际宽带网络里，因为服务器发送响应的顺序必须和接收请求的顺序一致，因此整个连接仍然是先进先出，而且可能发生 `HOL blocking`（`Head-of-line blocking`，线头阻塞）。
HTTP2.0（或者SPDY）的异步操作可能可以解决这个问题。

示意图：



`HTTP Pipelining` 需要客户端和服务器的支持。支持HTTP/1.1的服务器必须支持流水线技术。但是这并不是说服务器必须要将响应流水线化，而是说在客户端选择 `pipelining` 时服务器不会响应失败。

> 注意： **`HTTP Pipelining`** **只有HTTP/1.1支持，1.0版不支持。**

SPDY

主要由Google开发的一种web内容传输协议。SPDY会操作HTTP流量，以减少Web页面加载延迟，增强web安全。SPDY通过压缩、多工和优先化来降低延迟。

SPDY的实现基本上都用到了TLS加密，并且传输内容头都使用了 `gzip` 或者 `DEFLATE` 进行压缩（HTTP正好相反，头信息都是以人类可读的文本方式来传输的）。此外，服务器还可以提示甚至推送内容，而不是等待页面上的资源请求。

SPDY要求使用SSL/TLS，而且不支持通过普通的TCP操作。SSL是为了安全考虑，而且也可以避免在通过代理通讯时不兼容。

SPDY和HTTP没有关系，也不会替代HTTP，它只是修改了HTTP请求和响应的发送方式。这意味着可以加上一层SPDY兼容的传输层，而不用修改已有的服务

器端应用。

HTTP2.0

HTTP2.0基于SPDY协议。因此它支持 `header` （头信息）压缩以及服务器推送。HTTP2.0完全兼容HTTP1.1，两种版本的协议可以一起使用。

其实从上面提到的持久连接和流水线化，我们可以了解到HTTP2.0从这两方面都做了改进。比如，支持通过相同的TCP连接来提供多帧数据流（ `multiple-frame streams` ）。这样服务器就可以先发送最重要的数据。

另外的重大不同是服务器可以通过已经建立的连接来推送更多的信息。这与网页关系最大（**也就是说与我们前端开发关系更大**）。高效的站点都会通过压缩图片、脚本等资源来减少渲染页面所需要的请求数目。HTTP2.0提供的信息更多，使得服务器可以告诉用户他们所需要的用于页面渲染的资源。

HTTP2.0d的目标是“通过启用完全多工的请求和响应、通过高效的压缩来最小化协议开销来减少网页加载所花费的时间”。然而，要优化HTTP，还需要加入新的流控制(`flow control`)，错误处理(`error handling`)和升级机制等。

HTTP2.0没有修改HTTP1.1的方法——`methods`，状态码——`status code`，头部域——`header fields` 以及 `URI` 等高级语法，而是修改了底层的，例如数据在帧里的组织方式、客户端和服务器的传输方式等内容。

WebSocket

`WebSocket` 是一种在单个TCP连接上提供了全双工——`full-duplex` 通讯通道的协议。IETF在 `RFC 6455` 规范中对WebSocket协议进行了标准化，W3C组织也正在推进WebSocketAPI的标准化工作。

WebSocket协议基于TCP协议，它和HTTP唯一的联系在于WebSocket的握手——`handshake` 会被HTTP服务器解析成 `Upgrade Request` 。WebSocket协议使得浏览器和服务器可以有更多的互动，能够支持实时内容和实时游戏等应用。这是因为服务器可以不用请求浏览器就能在打开的连接上来回发送内容，从而使得服务器和浏览器能够双向会话。一种非标准的实现浏览器与服务器的双向会话的方法是 `comet` 。

> WebSocket最常见的一种实现方案是socket.io，在Node.js中使用socket.io的入门文章可以参考这篇文章。 还有这篇和这篇

WebSocket握手

一个典型的握手（建立连接过程）请求如下所示：

客户端请求：

```
GET /chat HTTP/1.1
Host: server.example.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Protocol: chat, superchat
Sec-WebSocket-Version: 13
Origin: http://example.com
```

服务器响应：

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sMlYUkAGmm5OPpG2HaGWk=
Sec-WebSocket-Protocol: chat
```

WebSocket的握手过程和HTTP相同，而且WebSocket也工作在80端口。

其他：QUIC（Quick UDP Internet Connections）

链接

参考资料

- HTTP/1.0 RFC1945
- HTTP/1.1 RFC2616
- HTTP/1.1 Header Field Definitions
- List of useful HTTP headers
- HTTP Compression
- BREACH

- SPDY
- HTTP Persistent Connection
- HTTP Pipelining
- Head-of-line blocking
- HTTP 2.0
- WebSocket
- `comet`
- QUIC
- Shared Dictionary Compression Over HTTP
- SDCH: Shared Dictionary Compression over HTTP

> Google真是一家受人尊敬的公司，单看它的Chrome Dev Tools，单看它在本文提到的与HTTP协议相关的领域所做的探索和努力，难怪它能称霸互联网。