

# 1.1 Web系统应用的发展

## 1. Web应用的发展

### Web 1.0

传统的主要为单向向用户传递信息的Web应用

Web 2.0 核心:互动、分享与关系

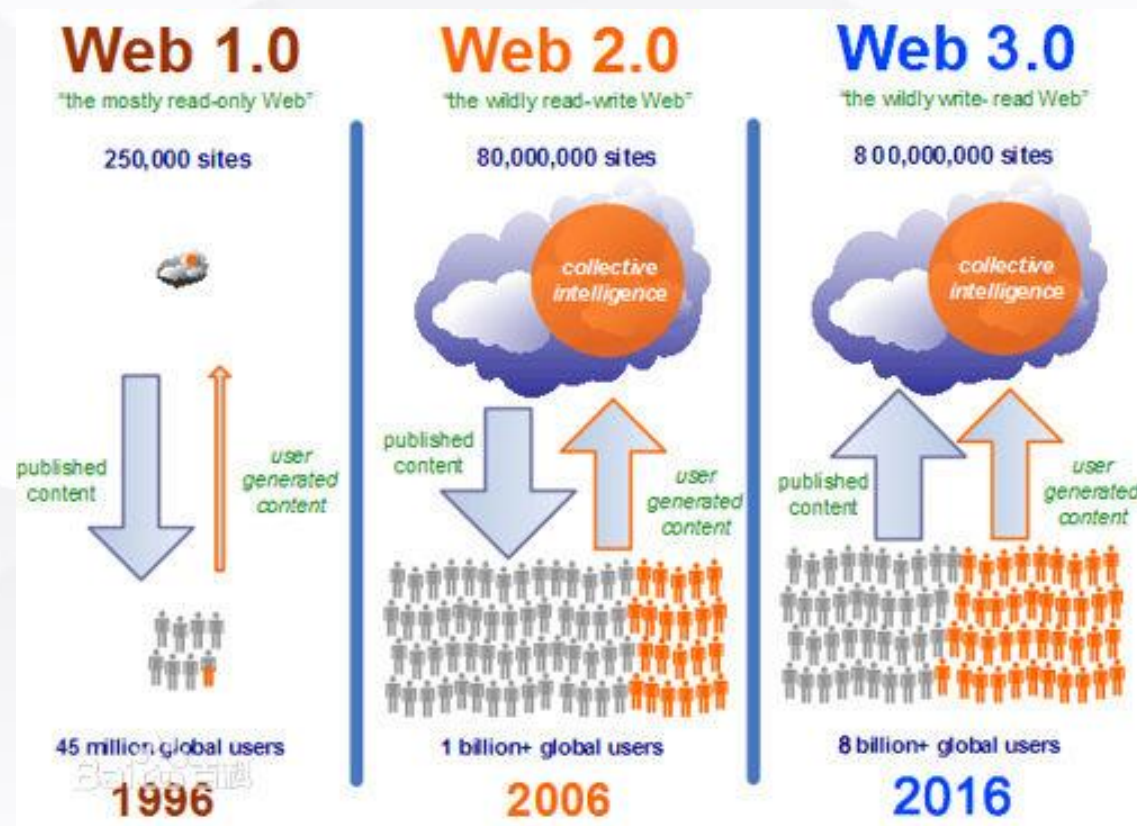
博客、百科、社交网站、P2P、IM

### Web 3.0

无处不联网: 移动互联网

与传统行业结合: 互联网+

网络计算: SaaS软件即服务, 云计算



# 1.1 Web系统应用的发展

## 现代Web应用的特点

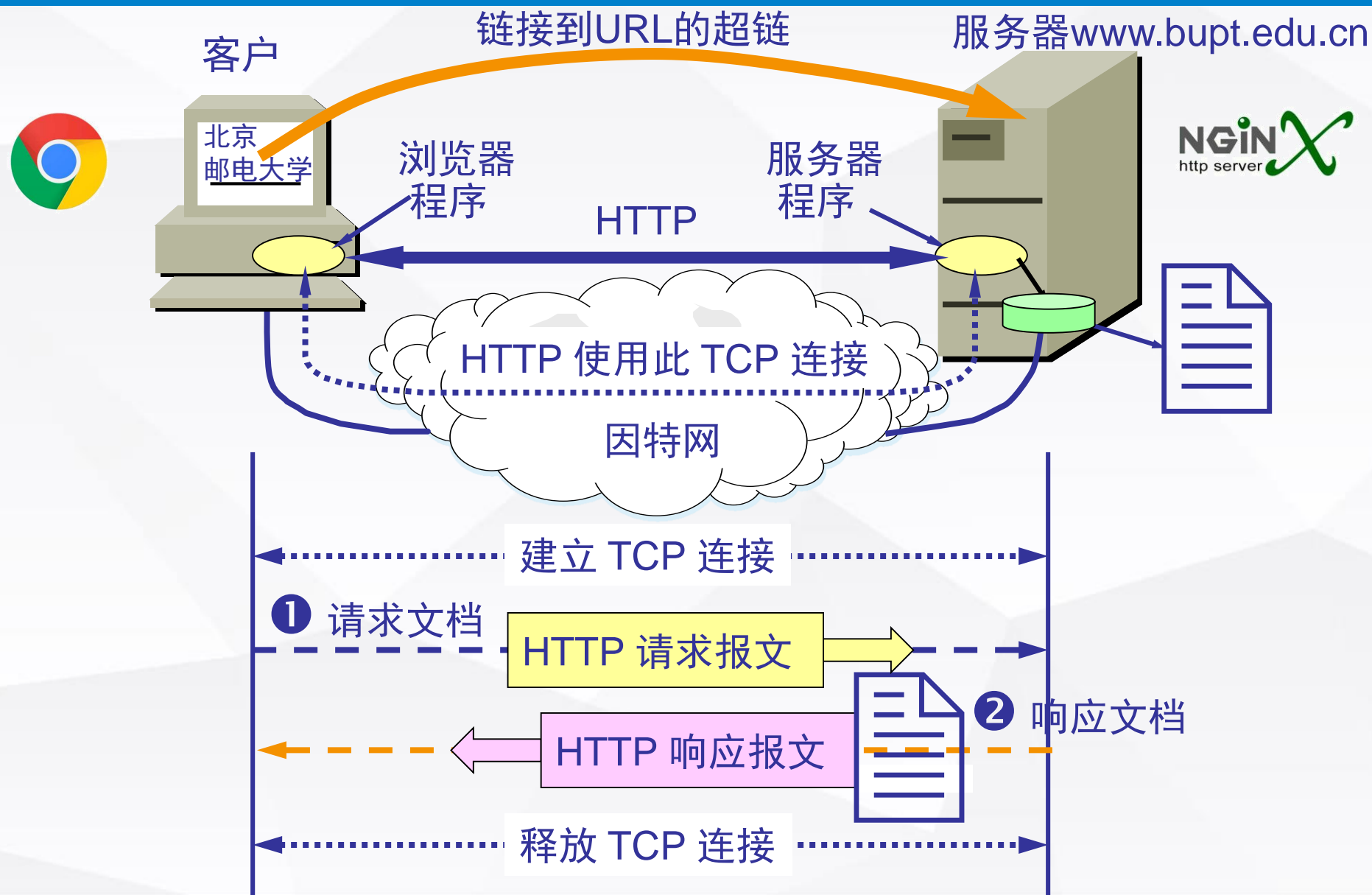
- 深入到人们的生活、办公、娱乐等各个方面，承载的功能需求多，涵盖的业务系统多，是**现代软件开发**的重要组成部分
- 强调页面表现、交互性能、运行流畅等多方位的用户体验
- 有移动应用需求（4G、5G）
- 高并发，高可用，高性能
- 高安全性
- 易于扩展和维护
- .....

## 1.2 Web系统基本构成和工作过程

- **Web系统的基本构成**
  - Web服务器
  - Web浏览器
  - 超文本传输协议HTTP (HyperText Transfer Protocol)
  - 超文本标记语言HTML(HyperText Markup Language)



## 1.2 Web系统基本构成和工作过程



## 1.2 Web系统基本构成和工作过程

用户点击鼠标后所发生的事件

- (1) 浏览器分析超链指向页面的 URL。
- (2) 浏览器向 DNS 请求解析 `www.bupt.edu.cn` 的 IP 地址。
- (3) 域名系统 DNS 解析出北京邮电大学服务器的 IP 地址。
- (4) 浏览器与服务器建立 TCP 连接
- (5) 浏览器发出取文件命令：  
GET `index.html`。
- (6) 服务器给出响应，把文件 `index.html` 发给浏览器。
- (7) TCP 连接释放。
- (8) 浏览器显示“北京邮电大学主页”文件 `index.html` 中的所有文本。

## 1.2 Web系统基本构成和工作过程

怎样标志分布在整个因特网上的万维网文档？

- 使用**统一资源标识符** URI (Uniform Resource Identifier)来唯一标志万维网上的信息资源。  
URI包括URL和URN（统一资源名，和资源所在地无关）两种形式。
- 使用**统一资源定位符** URL (Uniform Resource Locator)来描述信息资源的特定位置，是URI的子集。

<协议>://<主机>:<端口>/<路径>?<参数>#<片段>

ftp —— 文件传送协议 FTP

http —— 超文本传送协议 HTTP

https——建构在SSL/TLS之上的 http协议

## 1.2 Web系统基本构成和工作过程

使用 HTTP 的 URL 的一般形式

`http://<主机>:<端口>/<路径>?<参数>#<片段>`

HTTP 的默认端口号是 80，HTTPS 的默认端口号是 443，通常可省略

路径：服务器上资源的本地名

参数：某些方案会用此组件来传递参数，不同参数用&分割

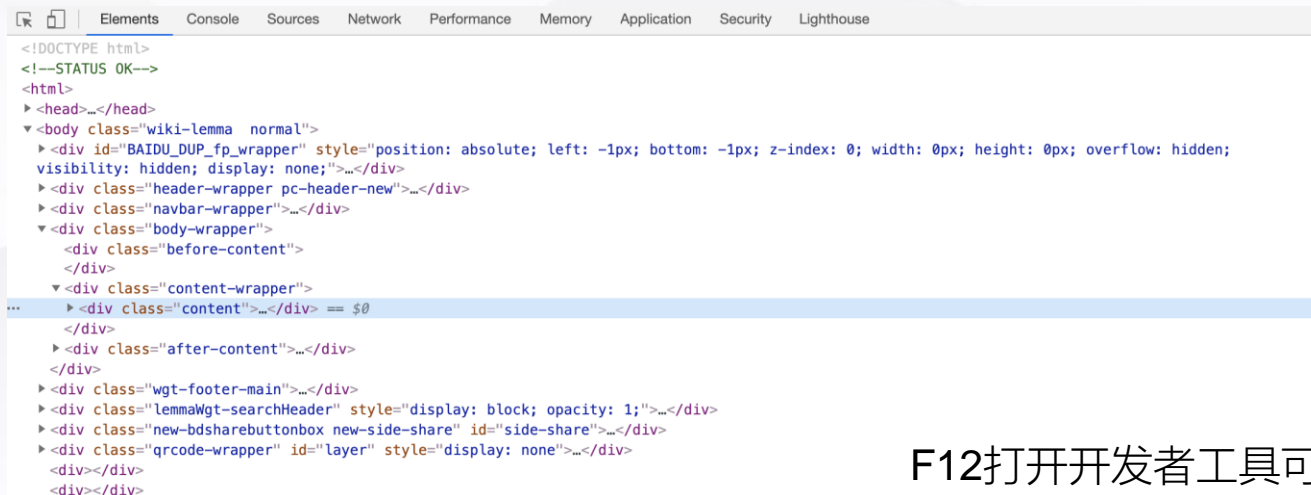
片段：代表网页中的一个位置。其右面的字符，就是该位置的标识符。#是用来指导浏览器动作的，对服务器端完全无用。所以，HTTP请求中不包括#。

<https://baike.baidu.com/item/web前端开发工程师?fromtitle=web开发&fromid=16011377#3>

## 1.2 Web系统基本构成和工作过程

### 超文本标记语言 HTML (HyperText Markup Language)

- 能够格式化的把内容显示出来
- 可以很方便地用一个超链从本页面的某处链接到因特网上的任何一个万维网页面
- HTML 定义了许多用于排版的命令（即标签）。使用这些标签把内容组织在一个文件中，就构成了所谓的 HTML 文档。
- HTML 文档是一种可以用任何文本编辑器创建的文本文件。



```
<!DOCTYPE html>
<!--STATUS OK-->
<html>
  <head>...</head>
  <body class="wiki-lemma normal">
    <div id="BAIDU_DUP_fp_wrapper" style="position: absolute; left: -1px; bottom: -1px; z-index: 0; width: 0px; height: 0px; overflow: hidden; visibility: hidden; display: none;">...</div>
    <div class="header-wrapper pc-header-new">...</div>
    <div class="navbar-wrapper">...</div>
    <div class="body-wrapper">
      <div class="before-content">
        <div class="content-wrapper">
          <div class="content">...</div>
        </div>
      <div class="after-content">...</div>
    </div>
    <div class="wgt-footer-main">...</div>
    <div class="LemmaWgt-searchHeader" style="display: block; opacity: 1;">...</div>
    <div class="new-bdsharebuttonbox new-side-share" id="side-share">...</div>
    <div class="qrcode-wrapper" id="layer" style="display: none;">...</div>
  </div>
</body>
</html>
```

F12打开开发者工具可以查看页面元素对应的html



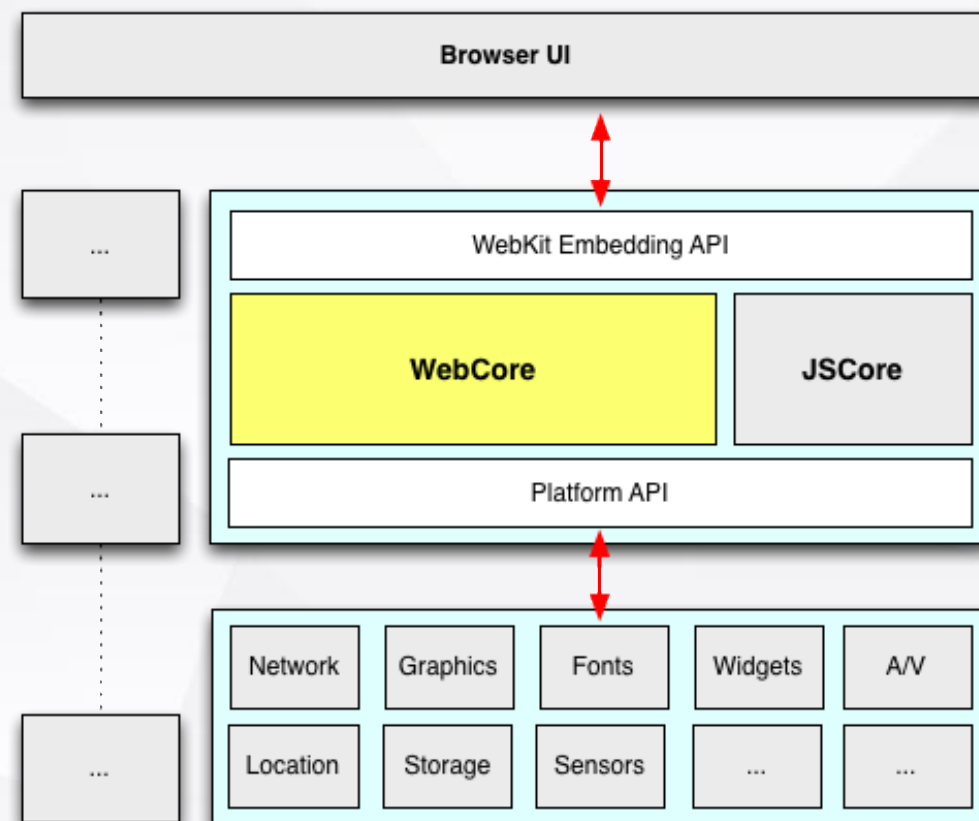
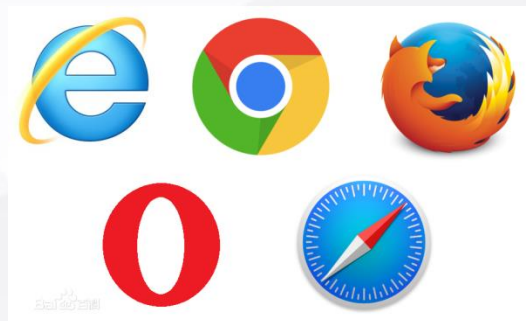
## 1.2 Web系统基本构成和工作过程

- Web浏览器核心功能

- 渲染引擎**(layout engine或者Rendering Engine), 对语法进行解析, 并渲染网页
- JS引擎**, 对JavaScript进行解释、编译和执行

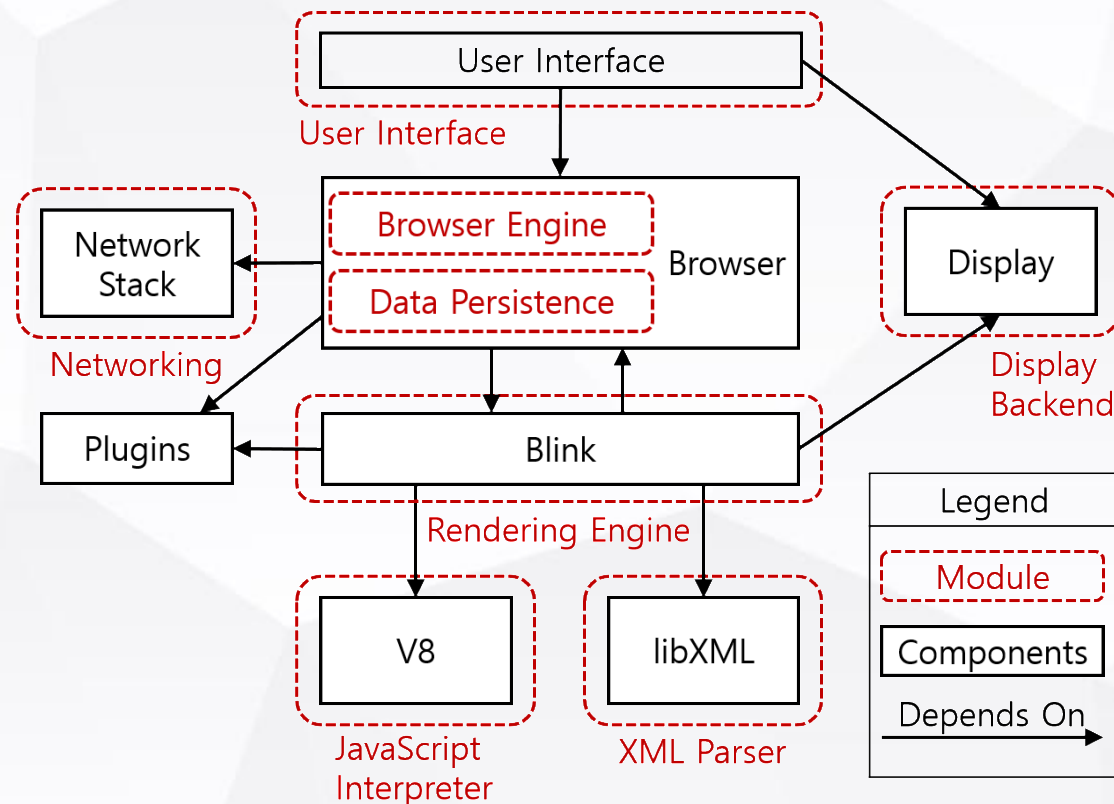
- Web浏览器产品

- Microsoft Edge /IE (微软)
- Firefox (Mozilla)
- Safari (Apple)
- Chrome (Google)
- Opera (Opera)
- QQ浏览器、360浏览器、UC浏览器.....

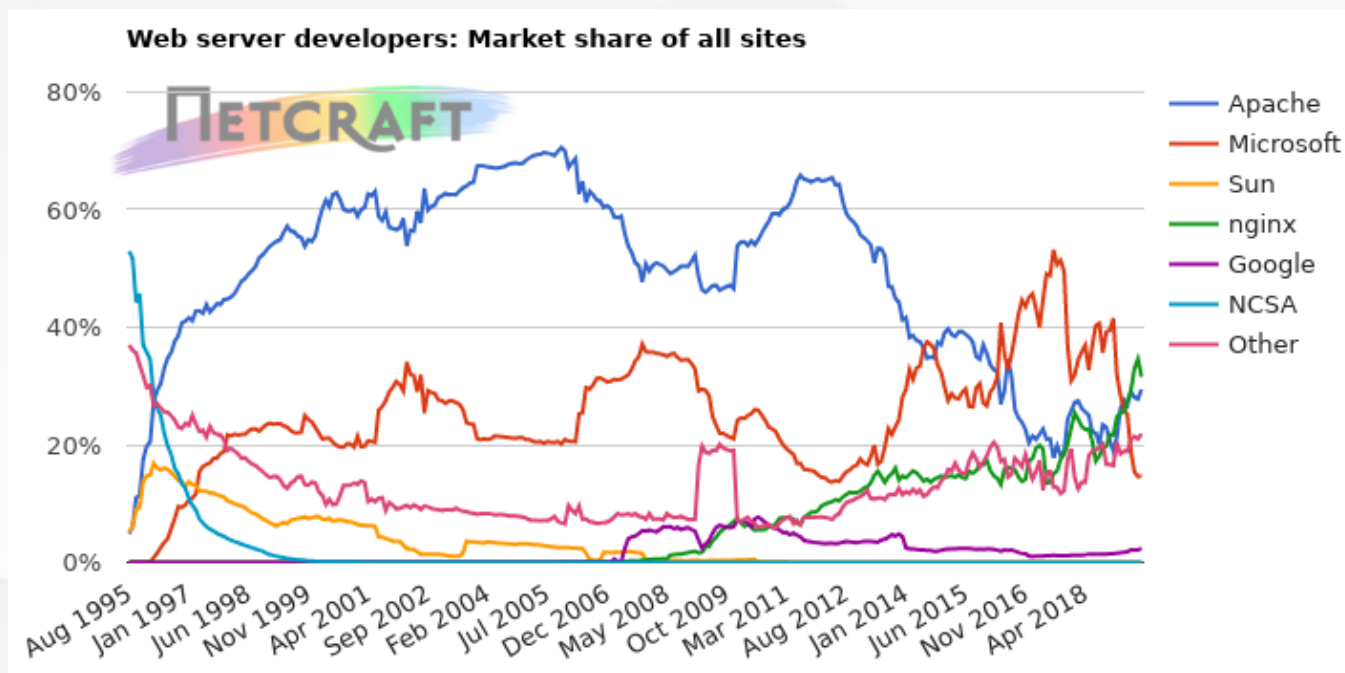


## 1.2 Web系统基本构成和工作过程

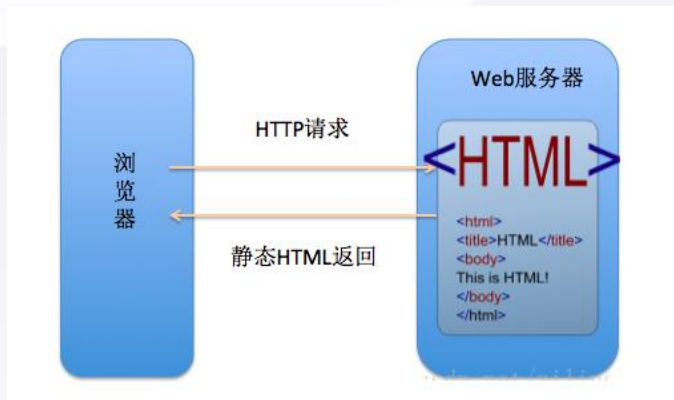
- Web浏览器内核/引擎
  - Trident 也叫IE内核，使用该内核的浏览器包括IE以及众多国产浏览器
  - Gecko 也叫Firefox内核，使用该内核的浏览器是Firefox
  - Webkit 使用该内核的浏览器有Safari、360、搜狗、Chrome 的早期版本
  - Blink源自Webkit，使用该内核的浏览器有Chrome（28以后版本）、Opera、Edge



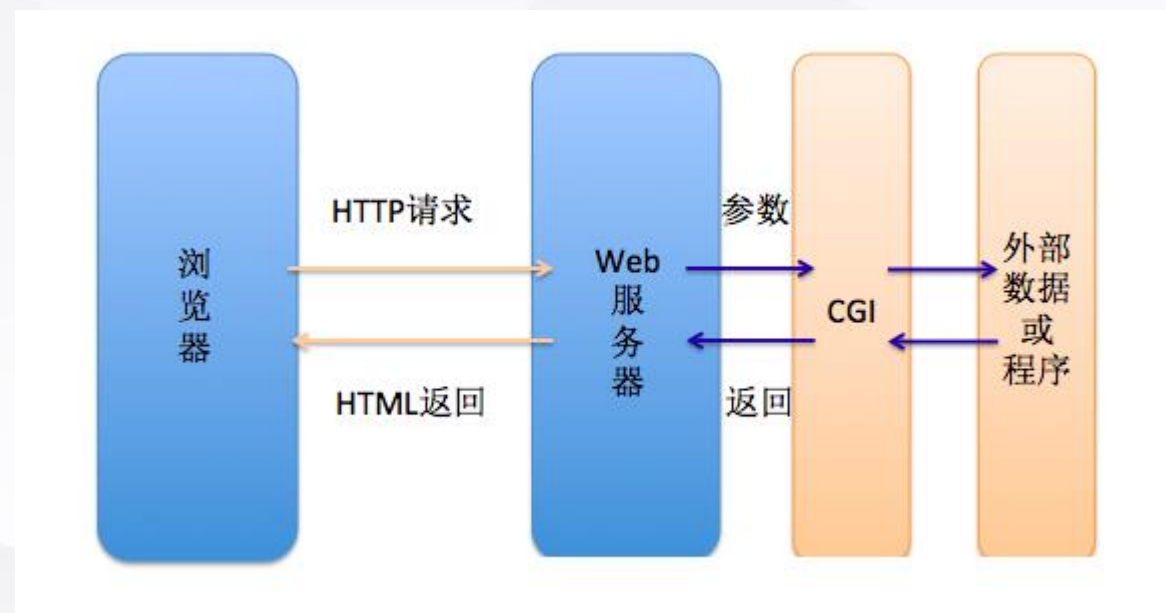
<http://www.chromium.org/blink>



## 1.3 Web应用架构演化



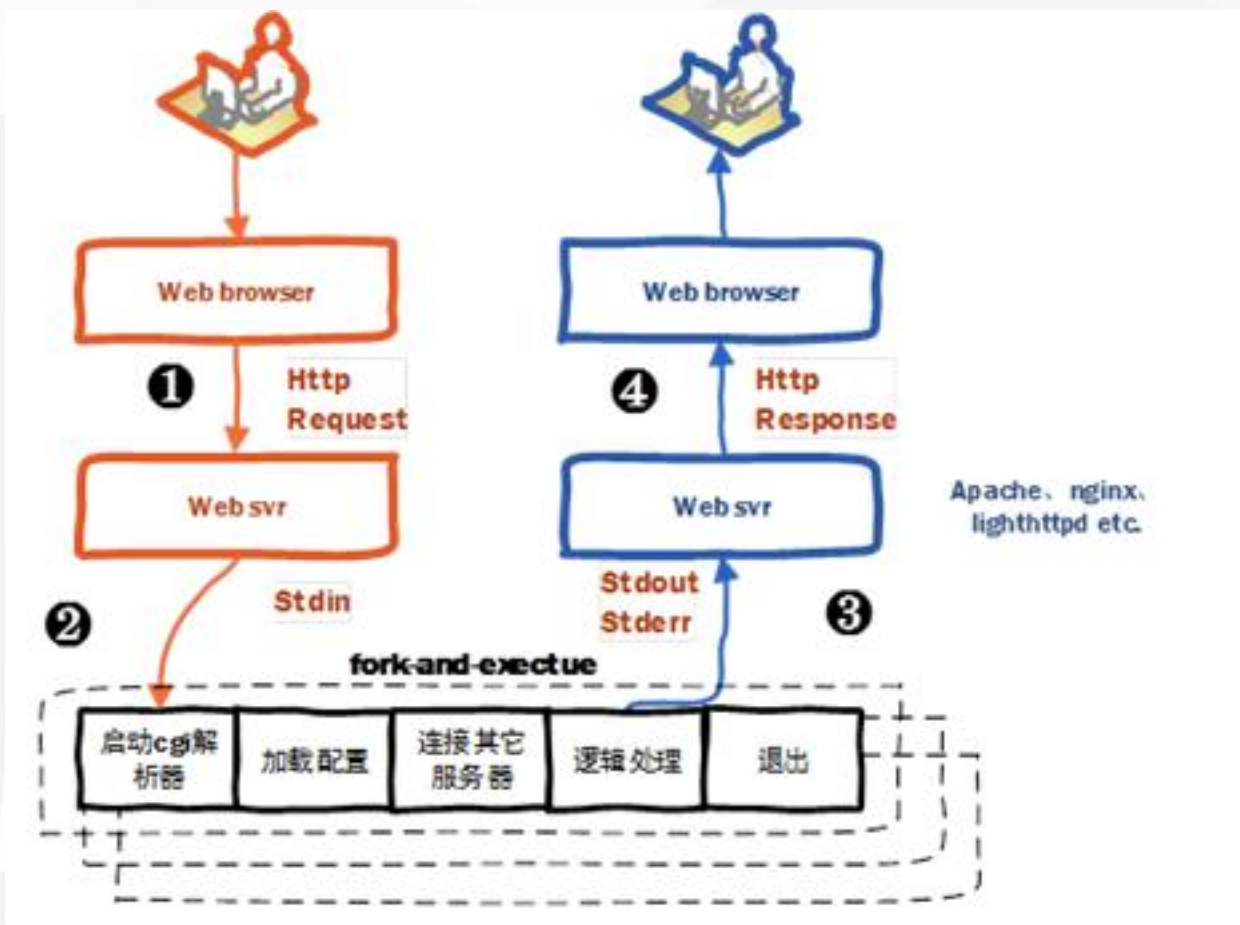
静态内容



早期的动态内容实现方案

1993年，CGI（Common Gateway Interface）定义了Web服务器与外部应用程序之间的通信接口标准，因此Web服务器可以通过CGI执行外部程序。

## 1.3 Web应用架构演化



没有专门的web开发

- 浏览器通过HTML表单或超链接请求指向一个CGI应用程序的URL。
- 服务器收发到请求。
- 服务器执行所指定的CGI应用程序。
- CGI应用程序执行所需要的操作，通常是基于浏览者输入的内容。
- CGI应用程序把结果格式化为网络服务器和浏览器能够理解的文档（通常是HTML网页）。
- 网络服务器把结果返回到浏览器中。

## 1.3 Web应用架构演化

1994年，PHP诞生，1995年PHP1.0发布，2004年PHP 5发布。

1996年，ASP1.0发布，2000年ASP3.0发布，2001年ASP.net。

1998年，JSP诞生，1999年JSP1.0发布。



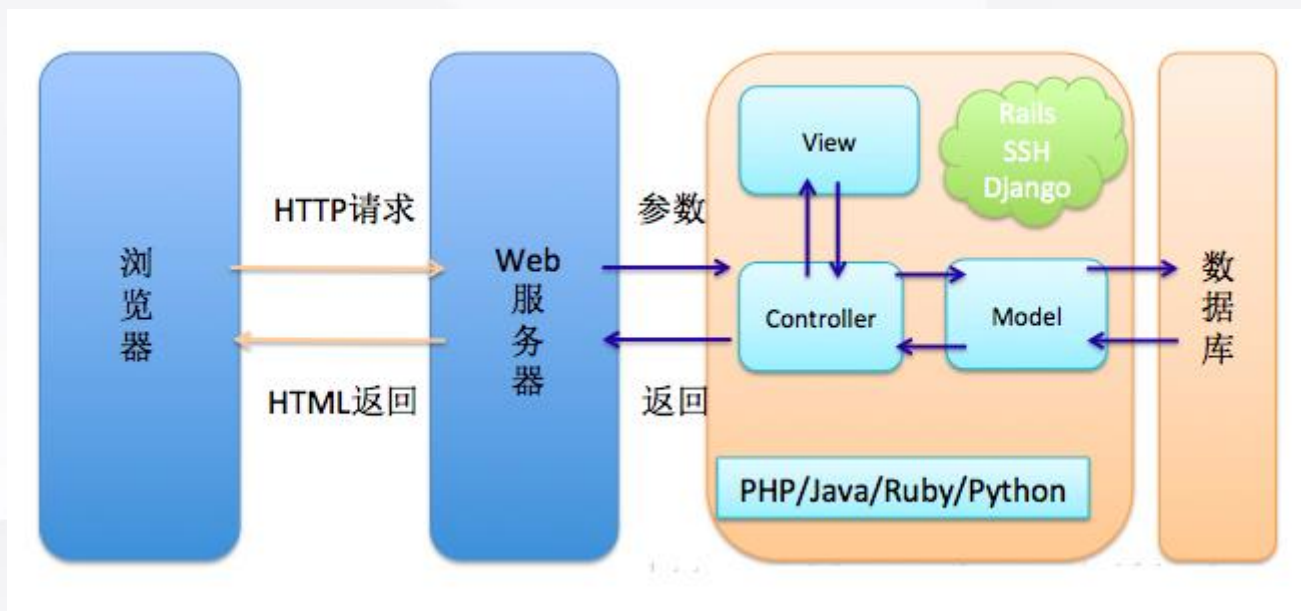


## 1.3 Web应用架构演化

Web系统日益复杂，MVC、ORM等概念被引入到Web开发中来。

2004年Ruby on Rails，2004年Struts，2005年 Django，2006年ThinkPHP.....

2001年Hibernate，2001年iBATIS（MyBatis的前身）



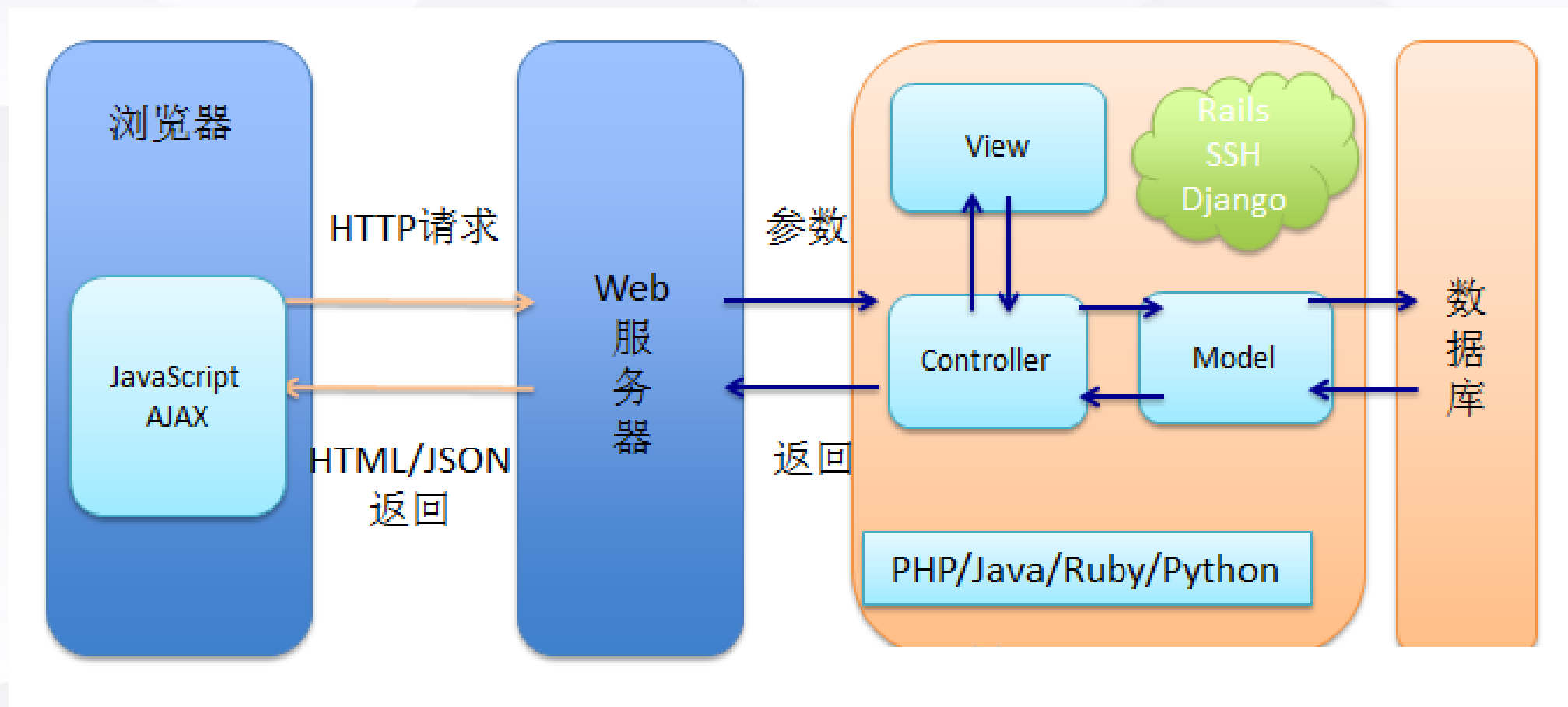
后端MVC框架时代

后端开发工程师

+

前端开发 or 美工

## 1.3 Web应用架构演化



前后端半分离的方案 (**AJAX+JSON**)



## 1.3 Web应用架构演化

- AJAX概念
  - Asynchronous Javascript And XML（异步JavaScript和XML）
  - AJAX不是一种新的编程语言，而是一种用于创建更好更快交互性更强的Web应用程序的技术。
  - 使用Javascript向服务器提出请求并处理响应而不阻塞用户，即可以在不重新加载整个网页的情况下，对网页的某部分进行更新。
  - 传统的网页（不使用AJAX）如果需要更新内容，必须重载整个网页页面。
  - AJAX的核心对象是XMLHttpRequest。
  - 示例：[http://www.runoob.com/try/try.php?filename=tryajax\\_first](http://www.runoob.com/try/try.php?filename=tryajax_first)

## 1.3 Web应用架构演化

- JSON概念<https://www.runoob.com/json/json-tutorial.html>
  - JSON: **J**ava**S**cript **O**bject **N**otation(JavaScript 对象表示法)
  - JSON使用文本数据来描述数据对象, 类似 XML。
  - JSON 比 XML 更小、更快, 更易解析。

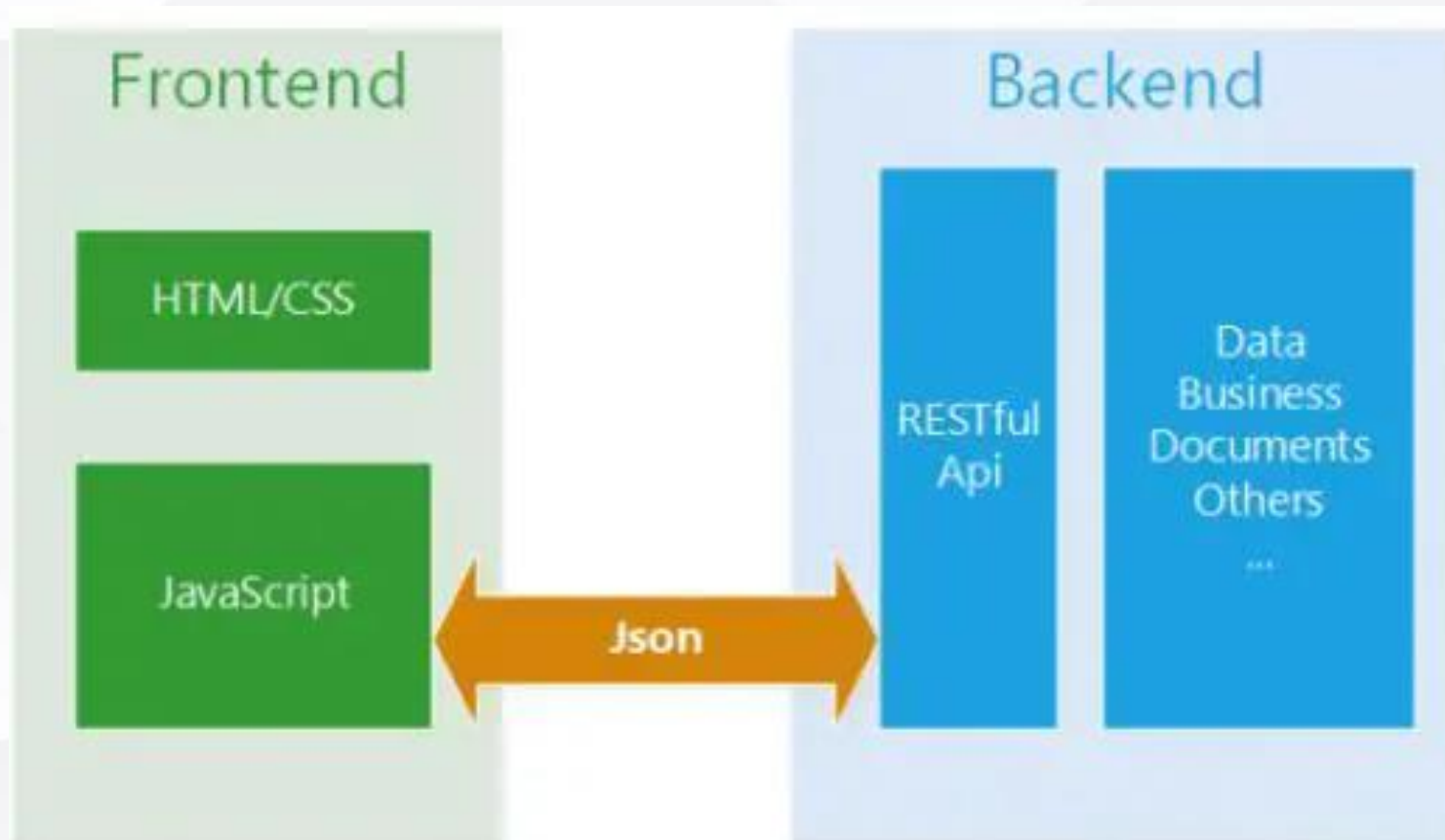
```
{
  "employees": [
    { "firstName":"John" , "lastName":"Doe" },
    { "firstName":"Anna" , "lastName":"Smith" },
    { "firstName":"Peter" , "lastName":"Jones" }
  ]
}
```

## 1.3 Web应用架构演化

- AJAX仅解决了不重新加载整个网页的情况下，向服务器发起请求获取数据，然后对网页的局部进行更新。
- 但现代Web应用对前端还有更高要求：
  - 如何方便的修改DOM?
  - 如何方便的编写DOM事件响应?
  - 如何方便的实现DOM元素之间的逻辑联动?
  - 有没有可能不经过后端自己切换显示的“页面”，即页面内部路由?
  - 如何实现负责前端页面的多人协作工程化开发?
  - .....

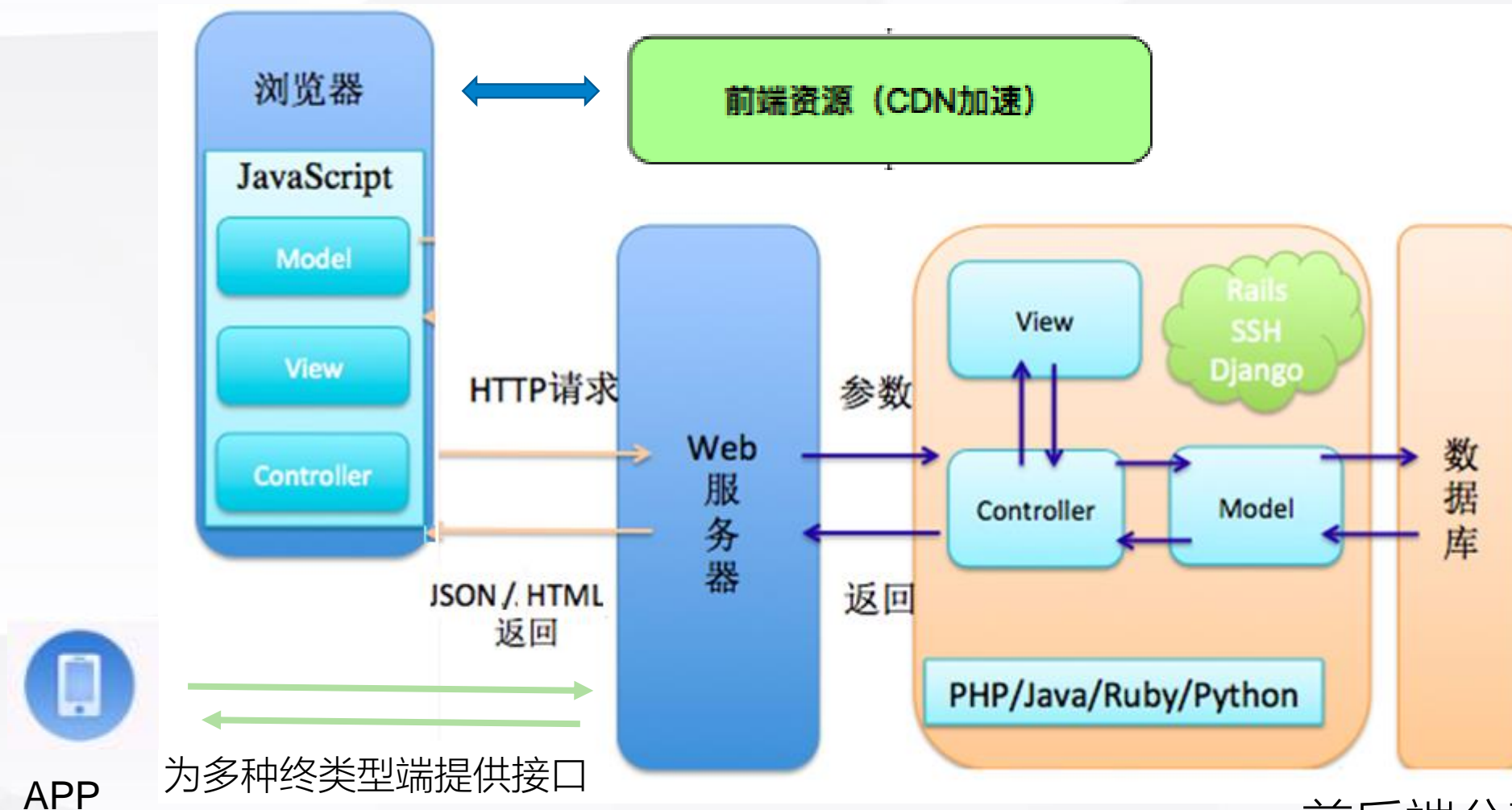


## 1.3 Web应用架构演化



## 1.3 Web应用架构演化

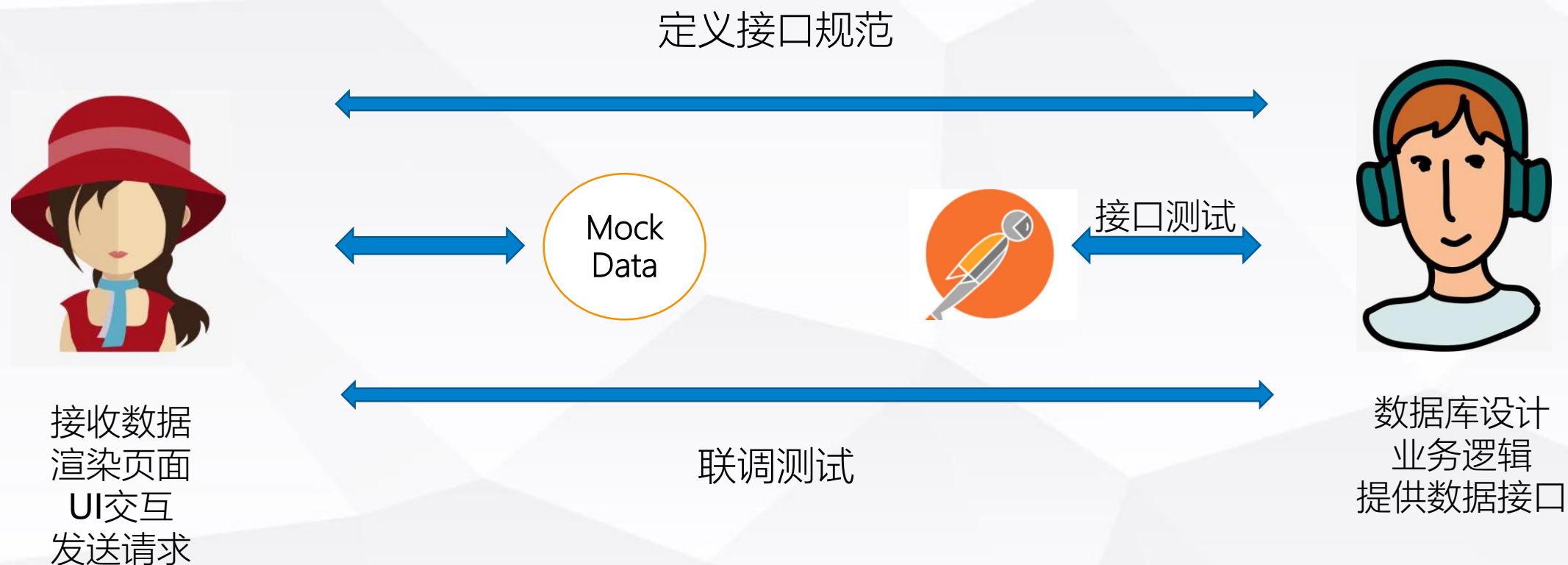
从静态文件Web服务器或CDN加载开发好的静态页面文件



前后端分离的方案

## 1.3 Web应用架构演化

- 前后端分离后的开发模式



## 1.3 Web应用架构演化

- 前后端分离的好处

- 前端引入了分层框架并且实现了组件化，为开发复杂的交互性页面提高了基础，web前端技术已经逐步具备了开发替代各类客户端应用程序的能力。
- 把页面渲染的操作由后端转移到浏览器，降低了服务器压力，也改善了用户体验。
- 后端从用户灵活多变的UI需求实现工作中解脱出来，专注于后台的业务逻辑、数据处理层以及分布式高性能方案。
- 后端可以一套接口为多种类型的接入终端提供服务。
- 前后端职责清晰，按约定的接口进行开发，有利于团队协作，并行开发。

## 1.3 Web应用架构演化

后端关注大型Web系统的非功能性需求

- **高并发**: 支持高的并发用户数、并发请求数 (QPS)、并发事务数 (TPS)
- **高性能**: 支持在高并发的同时具备短的响应时间, 运行效率高
- **高可用**: 保证系统可以持续提供服务, 即使在出现故障的情况下, 也尽可能对外提供服务, 冗余备份、降级、限流
- **可伸缩**: 面对高并发访问和数据量的增长, 可通过扩展节点满足业务需求的增长
- **可扩展**: 能够快速响应需求变化以及业务拓展
- **安全性**: 互联网是开放的, 需要保护系统不受恶意攻击破坏, 保护重要数据不被窃取
- **易运维**



## 1.3 Web应用架构演化

### 云开发

为开发者提供高可用、自动弹性扩缩的后端云服务，包含计算、存储、托管等 **Serverless** 化能力，可用于云端一体化开发多种端应用（小程序、公众号、Web 应用、Flutter 客户端等），帮助开发者统一构建和管理后端服务和云资源，避免了应用开发过程中繁琐的服务器搭建及运维，开发者可以专注于业务逻辑的实现，开发门槛更低，效率更高。

<https://cloud.tencent.com/product/tcb>



## 1.4 HTTP协议

- HTTP协议

- RFC2616, HTTP1.1
- RFC7540, HTTP/2
- HTTP 是一个应用层协议，它使用 TCP 连接进行可靠的传送。
- HTTP 有两类报文：
  - 请求报文——从客户向服务器发送请求报文。
  - 响应报文——从服务器到客户的回答。
- HTTP 是面向正文的(text-oriented)，在报文中的每一个字段都是一些 ASCII 码串，因而每个字段的长度都是不确定的。

## 1.4 HTTP协议

- HTTP协议

- HTTP协议是无状态

- 同一个客户端的这次请求和上次请求是没有对应关系，对http服务器来说，它并不知道这两个请求来自同一个客户端。为了解决这个问题，Web程序引入了Session和Cookie机制来维护状态
    - 无状态不代表HTTP不能保持TCP连接，从HTTP/1.1起，默认都开启了Keep-Alive，保持连接特性，即当一个网页打开完成后，客户端和服务端之间用于传输HTTP数据的TCP连接不会关闭，如果客户端再次访问这个服务器上的网页，会继续使用这一条已经建立的连接。

## 1.4 HTTP协议

- HTTP协议

- 打开一个网页一般需要浏览器发送很多次Request
  - 浏览器分析Response中的 HTML，发现其中引用了很多其他文件，比如图片，CSS文件，JS文件。
  - 浏览器会自动再次发送Request去获取图片，CSS文件，或者JS文件。
  - 等所有的文件都下载成功后。网页就被显示出来了。

## 1.4 HTTP协议

- HTTP请求消息结构:



## 1.4 HTTP协议

### ▣ Hypertext Transfer Protocol

#### ✦ GET / HTTP/1.1\r\n

```
Host: www.bupt.edu.cn\r\n
Connection: keep-alive\r\n
Cache-Control: max-age=0\r\n
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8\r\n
Upgrade-Insecure-Requests: 1\r\n
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome,
Accept-Encoding: gzip, deflate, sdch\r\n
Accept-Language: zh-CN,zh;q=0.8\r\n
\r\n
```

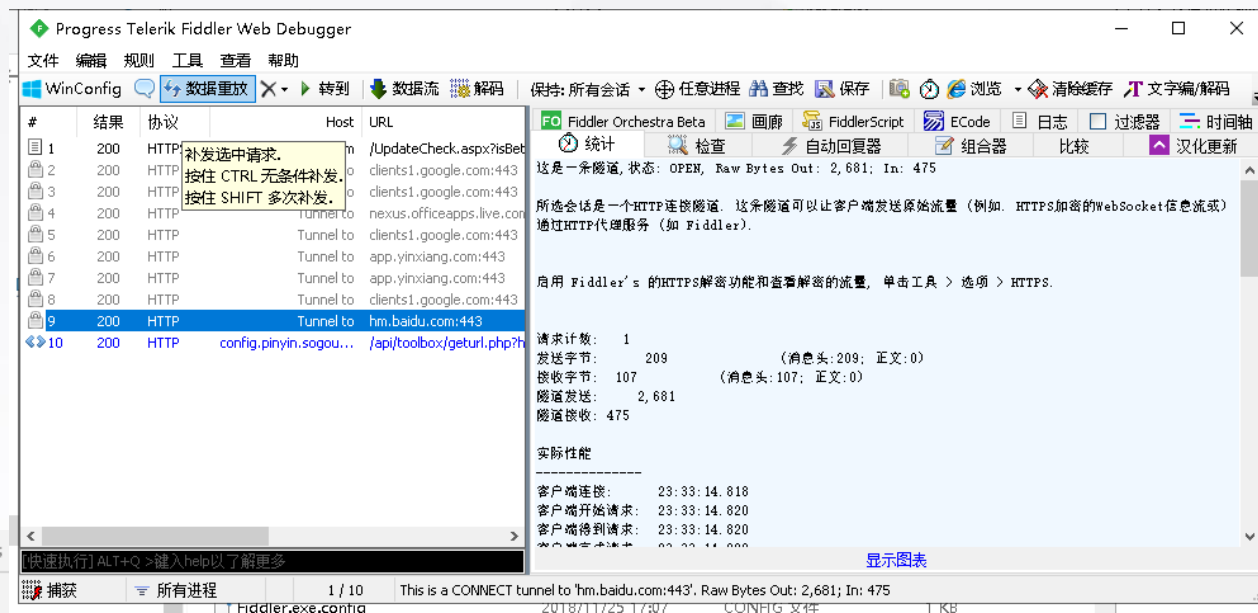
这个消息共有9行(每行以一个回车符和一个换行符结束)，最后一行后面还有额外的一个回车符和换行符。

# 1.4 HTTP协议

补充：协议分析工具

- 1、wireshark
- 2、浏览器开发者工具network
- 3、fiddler, http协议调试代理工具

Name	Status	Type
activity_start_52498d2c.js	200	script
mt_min_2247e202.css	200	stylesheet
nsguide_a8cbc2e7.css	200	stylesheet
super_ext_5031d813.css	200	stylesheet
tipspluslist?indextype=manht&_req_seqid=0x89a82955...id=29634_1437_21095_29523_29720_2956...	200	xhr
blank_3ec625ab.png	200	png
personalcontent?num=8&indextype=manht&_req_seqid=0...id=29634_1437_21095_29523_29720_2...	200	xhr
v.gif?mod=mancard%3AAskeleton&submod=presenter&utyp...Show&logactid=1100000000&showT...	200	gif
ps_default.gif?t=1568907121197	200	gif
ps_default.gif?t=1568907121197	200	gif
ps_default.gif?t=1568907121197	200	gif



## 1.4 HTTP协议

1、请求方法：指出客户请求服务器执行的一般操作。

HTTP1.0 定义了三种请求方法：GET, POST 和 HEAD方法。

HTTP1.1 新增了六种请求方法：OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT

GET：请求获取Request-URI所标识的资源，“获取/查操作”

POST：在Request-URI所标识的资源后附加新的数据，“增操作”，不是幂等

HEAD：请求获取由Request-URI所标识的资源的响应消息报头

PUT：请求服务器存储一个资源，并用Request-URI作为其标识，“改/增操作”，幂等

PATCH：是对 PUT 方法的补充，用来对已知资源进行局部更新，“改操作”

DELETE：请求服务器删除Request-URI所标识的资源“删操作”

TRACE：请求服务器回送收到的请求信息，主要用于测试或诊断

CONNECT：保留将来使用

OPTIONS：请求查询服务器的性能，或者查询与资源相关的选项和需求



## 1.4 HTTP协议

2、URI:统一资源标识, 简单地讲被请求资源所处的地址, 如: /

3、HTTP版本: HTTP/1.1, 高版本的服务器接受低版本客户的请求, 并向客户发送同样版本的应答; 高版本的客户接受低版本服务器的应答。

**幂等**: 任意多次执行所产生的影响均与一次执行的影响相同, 如

GET <http://www.news.com/article/123456>, 执行多次也不会改变资源的状态。

DELETE <http://www.news.com/article/4231>, 删除id为4231的文章

POST <http://www.news.com/articles>, 语义是在articles下面创建一篇文章, 重复执行会创建多篇文章, 不幂等

PUT <http://www.news.com/articles/4231> 的语义是创建或更新ID为4231的文章。对同一URI进行多次PUT的副作用和一次PUT是相同的; 因此, PUT方法具有幂等性。

## 1.4 HTTP协议

### 报头

- 1、Connection: close是在告知服务器本浏览器不想使用永久连接方式（HTTP/1.0使用非永久连接，HTTP/1.1默认使用永久连接）。
- 2、User-agent: Mozilla/5.0指定用户浏览器的类型。
- 3、Accept-Encoding: gzip,compress指出发送此请求的浏览器支持哪些压缩编码方式。
- 4、Accept-language: en指出客户浏览器支持的语言是英语（english），

## 1.4 HTTP协议



- 状态行示例： HTTP/1.1 200 OK
- 状态行由协议版本、状态码、原因短语3个元素组成

## 1.4 HTTP协议

```
⊞ Hypertext Transfer Protocol
  ⊞ HTTP/1.1 200 OK\r\n
    Date: Sun, 27 Sep 2015 13:56:39 GMT\r\n
    Server: Apache/2.2.29 (Unix) PHP/5.3.29\r\n
    X-Powered-By: PHP/5.3.29\r\n
    X-Frame-Options: SAMEORIGIN\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Transfer-Encoding: chunked\r\n
    Content-Type: text/html\r\n
    \r\n
```

- 1、Date: Sun 27 Sep 2015 13:56:39 GMT指出服务器创建并发送本响应消息的日期和时间。
- 2、Server: Apache/2.2.29 (Unix)指出本消息是由Apache服务器产生的，服务器版本为2.2.29
- 3、Content—Type: text/html指出包含在实体中的对象是HTML文本。

## 1.4 HTTP协议

状态代码有三位数字组成，第一个数字定义了响应的类别

**1xx**: 指示信息--表示请求已接收，继续处理

**2xx**: 成功--表示请求已被成功接收、理解、接受

**3xx**: 重定向--要完成请求必须进行更进一步的操作

**4xx**: 客户端错误--请求有语法错误或请求无法实现

**5xx**: 服务器端错误--服务器未能实现合法的请求

## 1.4 HTTP协议

常见状态代码、状态描述、说明：

**200 OK** //客户端请求成功

**400 Bad Request** //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

**403 Forbidden** //服务器收到请求，但是拒绝提供服务

**404 Not Found** //请求资源不存在， eg：输入了错误的URL

**500 Internal Server Error** //服务器发生不可预期的错误

**503 Server Unavailable** //服务器当前不能处理客户端的请求



## 1.4 HTTP协议

### HTTP消息报头

HTTP消息报头包括普通报头、请求报头、响应报头、实体报头。

每一个报头域都是由名字+“: ”+空格+值 组成，消息报头域的名字是大小写无关的

#### 1、普通报头

**Date**普通报头域表示消息产生的日期和时间

**Connection**普通报头域允许发送指定连接的选项。例如指定连接是连续，或者指定“close”选项，通知服务器，在响应完成后，关闭连接

**Cache-Control** 用于指定缓存指令

## 1.4 HTTP协议

缓存的优点：

1. 减少了冗余的数据传输
2. 缓解了网络瓶颈问题
3. 降低了对原始服务器的要求
4. 降低了传输时延

### ▼ Response Headers

**accept-ranges:** bytes

**age:** 45436

**cache-control:** max-age=2592000

**content-length:** 2910

**content-type:** image/png

**date:** Sun, 01 Sep 2019 02:18:52 GMT

**etag:** "5c386154-b5e"

**expires:** Mon, 30 Sep 2019 13:41:36 GMT

**last-modified:** Fri, 11 Jan 2019 09:26:44 GMT

**ohc-cache-hit:** bj2un82 [4]

**ohc-response-time:** 1 0 0 0 0 0

**server:** JSP3/2.0.14

**status:** 304

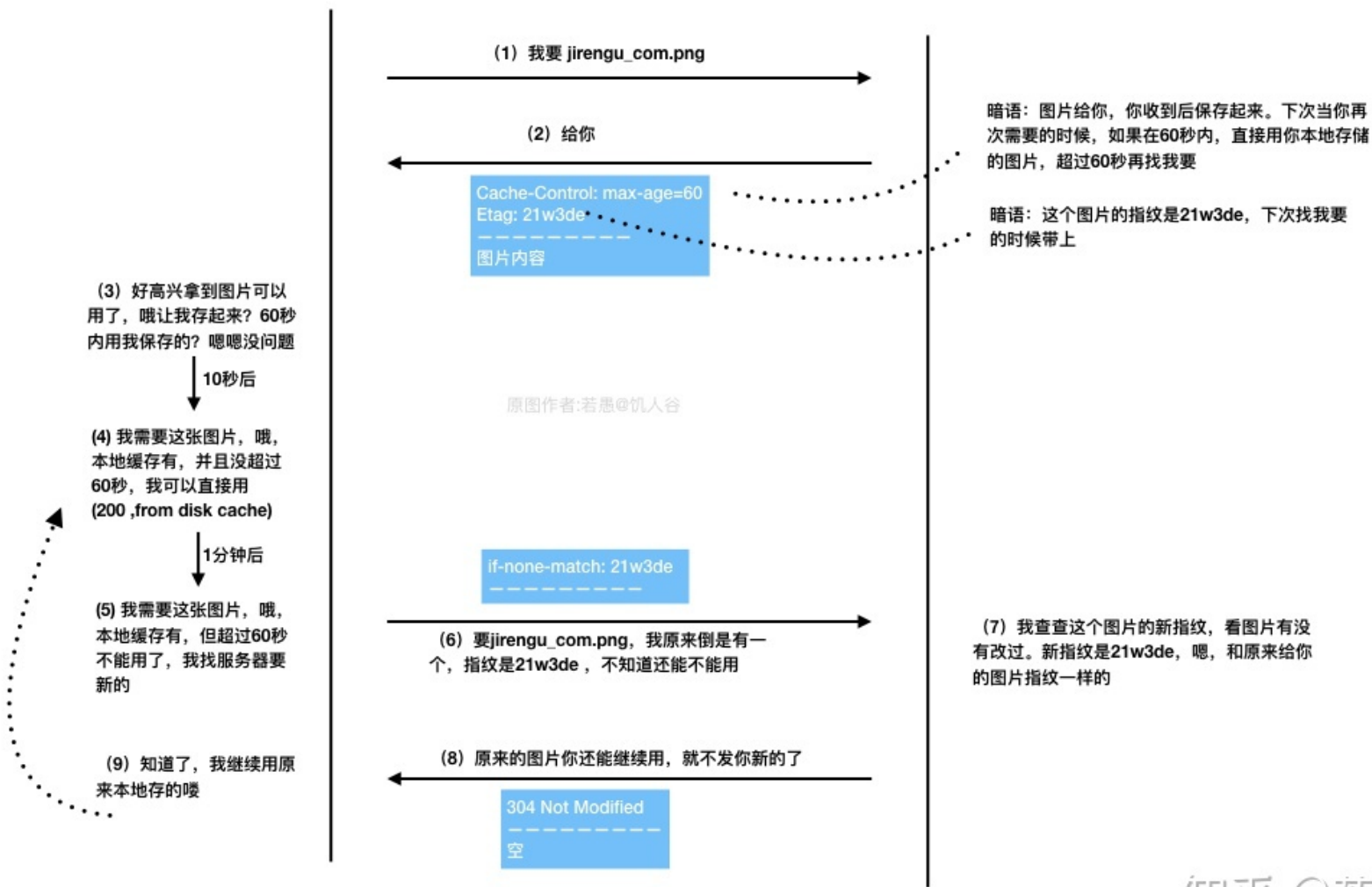


# 1.4 HTTP协议

浏览器

服务器

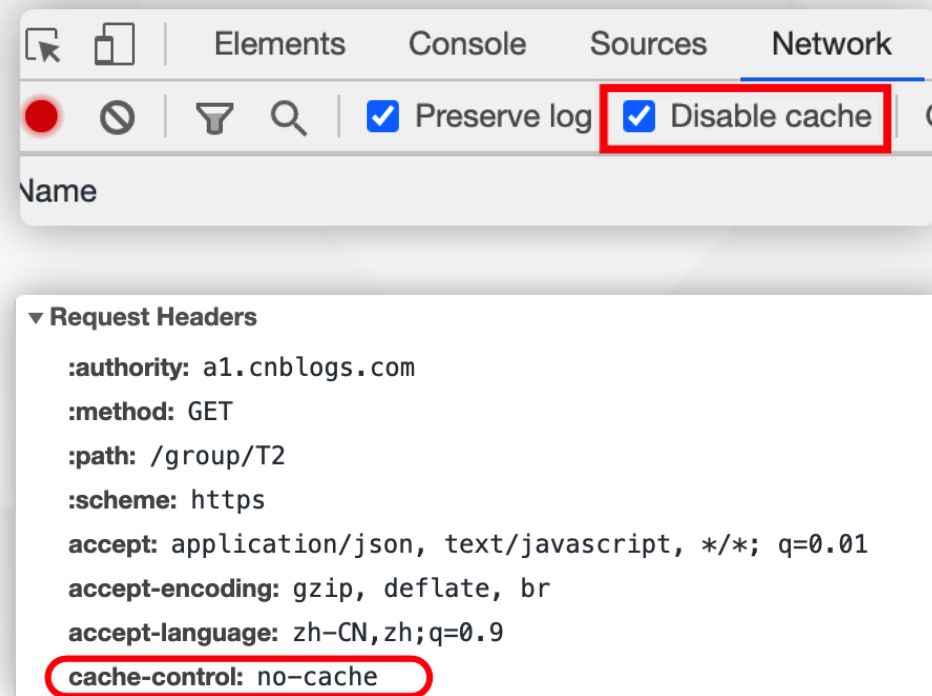
<https://zhuanlan.zhihu.com/p/55623075>



## 1.4 HTTP协议

**缓存控制：**在http中，控制缓存开关的字段有两个：Pragma 和 Cache-Control。Pragma逐渐被弃用。在请求中使用Cache-Control 时，它可选的值有：

字段名称	说明
no-cache	告知（代理）服务器不直接使用缓存，要求向原服务器发起请求 使用前需进行有效验证
no-store 彻底不缓存	所有内容都不会被保存到缓存或 Internet 临时文件中
max-age=delta-seconds	告知服务器客户端希望接收一个存在时间（Age）不大于 delta-seconds 秒的资源
max-stale [=delta-seconds]	告知（代理）服务器客户端愿意接收一个超过缓存时间的资源，若有定义 delta-seconds 则为 delta-seconds 秒，若没有则为任意超出的时间
min-fresh=delta-seconds	告知（代理）服务器客户端希望接收一个在小于 delta-seconds 秒内被更新过的资源
no-transform	告知（代理）服务器客户端希望获取实体数据没有被转换（比如压缩）过的资源
only-if-cached	告知（代理）服务器客户端希望获取缓存的内容（若有），而不用向原服务器发去请求
cache-extension	自定义扩展值，若服务器不识别该值将被忽略掉12375924



在开发者工具中勾选disable cache后，请求头会增加no-cache

## 1.4 HTTP协议

在响应中使用Cache-Control 时，它可选的值有：

字段名称	说明
public	表明任何情况下都得缓存该资源（即使是需要HTTP 认证的资源）
Private [= "field-name"]	表明返回报文中全部或部分（若指定了field-name 则为field-name 的字段数据）仅开放给某些用户（服务器指定的share-user，如代理服务器）做缓存使用，其他用户则不能缓存这些数据
no-cache	不直接使用缓存，要求向服务器发起（新鲜度校验）请求
no-store	所有内容都不会被保存到缓存或 Internet 临时文件中
no-transform	告知客户端缓存文件时不得对实体数据做任何改变
only-if-cached	告知（代理）服务器客户端希望获取缓存的内容（若有），而不用向原服务器发去请求
must-revalidate	当前资源一定是向原服务器发去验证请求的，若请求失败会返回 504（而非代理服务器上的缓存）
proxy-revalidate	与 must-revalidate 类似，但仅能应用于共享缓存（如代理）
max-age=delta-seconds	告知客户端该资源在 delta-seconds 秒内是新鲜的，无需向服务器发请求
s-maxage=delta-seconds	同 max-age，但仅应用于共享缓存（如代理）
cache-extension	自定义扩展值，若服务器不识别该值将被忽略掉

## 1.4 HTTP协议

### 缓存校验：

Last-Modified：资源的最后更改时间，客户端下次请求时通过If-Modified-Since或者If-Unmodified-Since带上Last-Modified，服务端检查该时间是否与服务器的最后修改时间一致：如果一致，则返回304状态码，不返回资源

etag：服务器通过某个算法对资源进行计算，取得一串值



## 1.4 HTTP协议

2、请求报头：允许客户端向服务器端传递请求的附加信息以及客户端自身的信息

**Host:** 主要用于指定被请求资源的Internet主机和端口号（**必须**）

**Accept:** 用于指定客户端接受哪些类型的信息。

**Accept-Charset:** 用于指定客户端接受的字符集。

**Accept-Encoding:** 用于指定可接受的内容编码。

**Accept-Language:** 用于指定一种自然语言。

**Authorization:** 用于证明客户端有权查看某个资源。

**User-Agent:** 浏览器的名称和版本

**Referer:** 表示从哪儿连结到目前的网页，采用的格式是URL。

**Cookie:** 将cookie的值发送给HTTP 服务器

## 1.4 HTTP协议

3、响应报头：允许服务器传递不能放在状态行中的附加响应信息，以及关于服务器的信息和对Request-URI所标识的资源进行下一步访问的信息。

**Set-Cookie:** 把cookie 发送到客户端浏览器， 每一个写入cookie都会生成一个Set-Cookie.

**Location:** 用于重定向接受者到一个新的位置。

**Server:** 包含了服务器用来处理请求的软件信息。与User-Agent请求报头域是相对应的。

**WWW-Authenticate:** 必须被包含在401（未授权的）响应消息中，客户端收到401响应消息时候，并发送Authorization报头域请求服务器对其进行验证时，服务端响应报头就包含该报头域。

## 1.4 HTTP协议

4、实体报头：请求和响应消息都可以传送一个实体。一个实体由实体报头域和实体正文组成，但并不是说实体报头域和实体正文要在一起发送，可以只发送实体报头域。实体报头定义了关于实体正文和请求所标识的资源的元信息。

**Content-Encoding**: 用作媒体类型的修饰符，它的值指示了已经被应用到实体正文的附加内容的编码。

**Content-Language**: 描述了资源所用的自然语言。

**Content-Length**: 指明实体正文的长度。

**Content-Type**: 指明实体正文的媒体类型（MIME type）。text/html、image/gif、application/javascript

**Last-Modified**: 指示资源的最后修改日期和时间。

**Expires**: 给出响应过期的日期和时间。



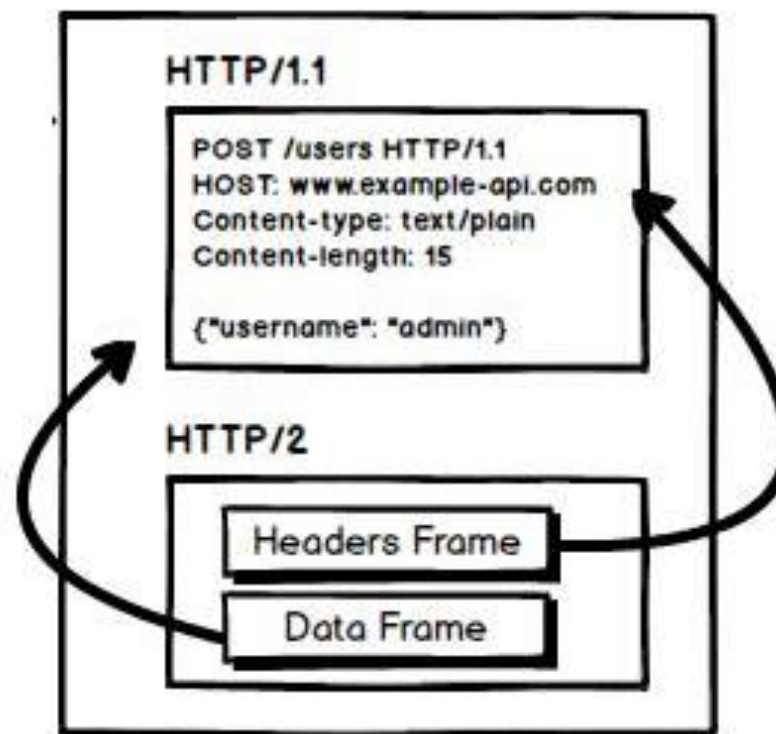
## 1.4 HTTP协议

### HTTP/2的新特性

1. 二进制协议：使用二进制替代明文，更容易解析，但可读性却不如HTTP/1.x。 HTTP 消息是由一个或多个帧组成的。

HEADERS帧承载了元数据

DATA帧则承载了内容。

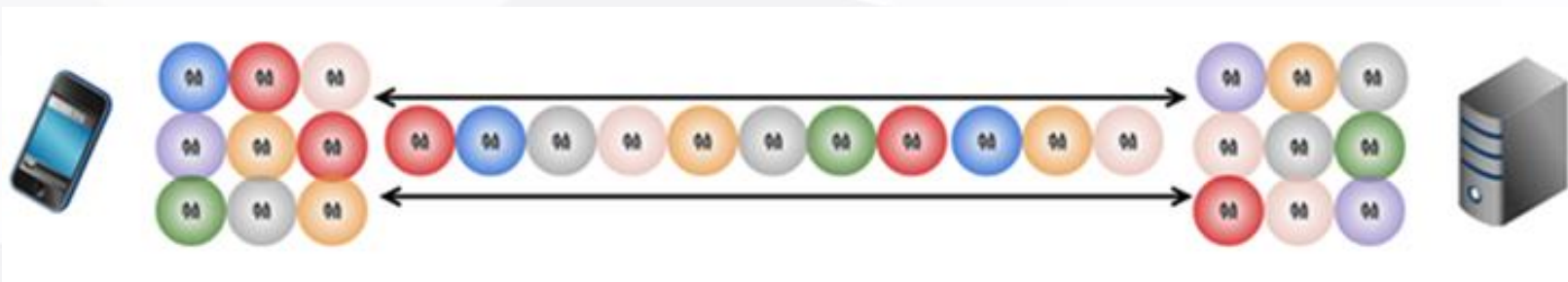




## 1.4 HTTP协议

### HTTP/2的新特性

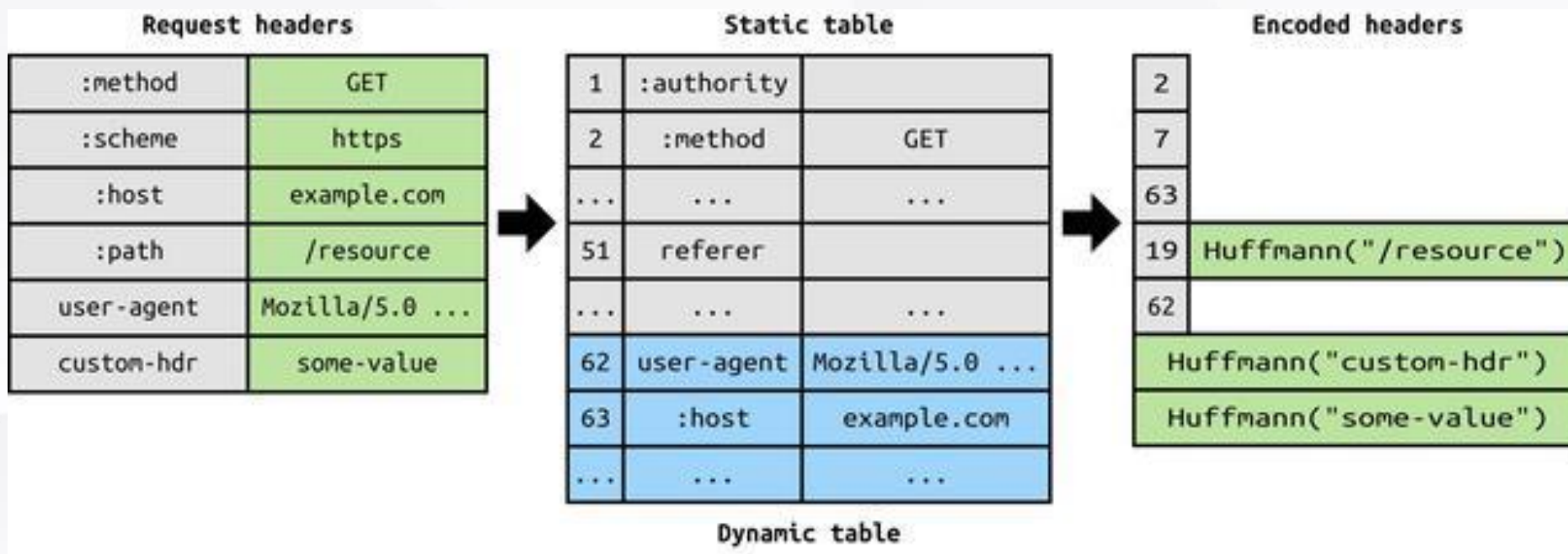
2. 多路传输：一旦建立了 **TCP** 连接，相同连接内的所有流都可以同过这个 **TCP** 连接异步发送，而不用另外打开连接。服务器也可以使用同样的异步方式返回响应，也就是说这些响应可以是无序的，客户端使用分配的流 **ID** 来识别数据包所属的流。解决了 HTTP/1.x 中请求管道被阻塞的问题，即客户端不必等待占用时间的请求而其他请求仍然可以被处理。



## 1.4 HTTP协议

### HTTP/2的新特性

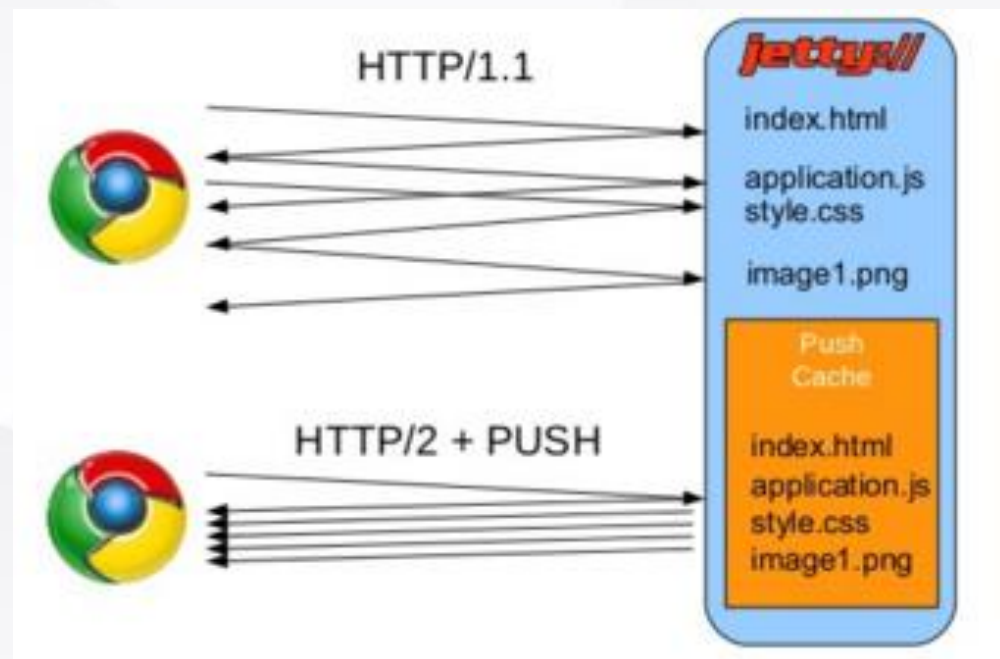
3. HPACK 请求头部压缩：当我们在同一客户端上不断地访问服务器时，许多冗余数据在头部中被反复发送，为了解决这个问题，HTTP/2引入了头信息压缩。



## 1.4 HTTP协议

### HTTP/2的新特性

4. 服务器推送：对于服务器来说，当它知道客户端需要一定的资源后，它可以把数据推送到客户端，即使客户端没有请求它。服务器通过发送一个名字为 **PUSH\_PROMISE** 特殊的帧通知到客户端



## 1.4 HTTP协议

### HTTP/2的新特性

5. 请求优先级：客户端可以在 **HEADERS** 帧中包含优先级信息来为流指定优先级。在任何时候，客户端都可以发送 **PRIORITY** 帧来改变流的优先级。
6. 安全性：HTTP/2定义了TLS的轮廓，包括版本、密码套件和用到的扩展。

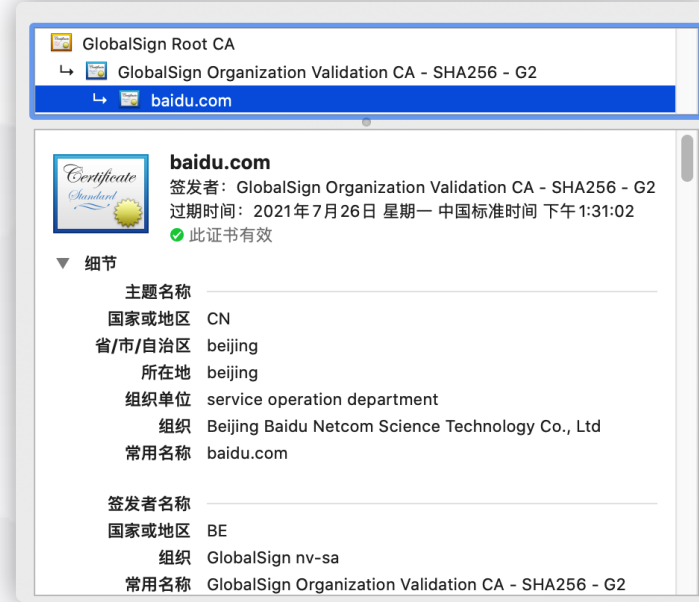
http2测试网址

<https://http2.akamai.com/demo>

## 1.4 HTTP协议

HTTPS:基于HTTP协议，通过SSL或TLS提供以下三种能力：

- 加密处理数据: 采用混合加密技术，中间者无法直接查看明文内容
- 验证对方身份: 通过证书认证客户端访问的是自己的服务器
- 数据完整性保护: 防止传输的内容被中间人冒充或者篡改



## 1.4 HTTP协议

- 1.客户端向服务器发起HTTPS请求
- 2.服务器端有用于非对称加密的密钥对：公钥和私钥
- 3.服务器将证书（含有公钥）发送给客户端
- 4.客户端收到证书后，验证其合法性，如合法则生成随机的用于对称加密的客户端密钥
- 5.客户端将客户端密钥使用公钥非对称加密后发送给服务器。
- 6.服务器用私钥进行非对称解密，得到客户端密钥。
- 7.服务器用客户端密钥对数据进行对称加密，并将加密后的密文发送给客户端。
- 8.客户端收到密文，用客户端密钥对其进行对称解密，得到服务器发送的数据。

