BDP WEB

从输入URL到页面加载完成，发生了什么?

概括

输入URL

开启网络

HTTP请求

后台接收

HTTP返回前端

浏览器解析渲染

连接结束

输入URL → 开启网络

浏览器（现代）是多进程的。

浏览器每打开一个标签页，相当于新开启一个进程。

该进程内部由多线程组成。

每次网络请求，都需要开辟单独的线程。

浏览器会根据解析URL得出的协议，开辟一个新的网络线程，并请求资源。

多进程优势

浏览器内核

解析URL

输入URL → 开启网络

多进程的优势

避免单个页面的crash，进而影响整个浏览器

避免第三方插件crash，进而影响整个浏览器

多进程能充分利用计算机多核的优势，进而提高浏览器效率

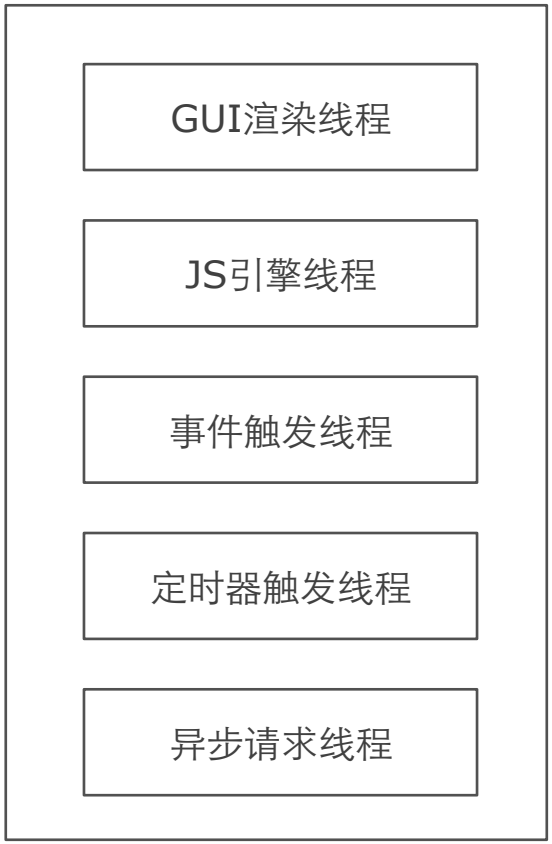
方便使用沙盒模型技术隔离可疑行为等进程，进而提高浏览器稳定性

输入URL → 开启网络

浏览器的内核

每个标签页面可以看作一个浏览器内核进程。

该进程是由多线程组成共同完成浏览器渲染。

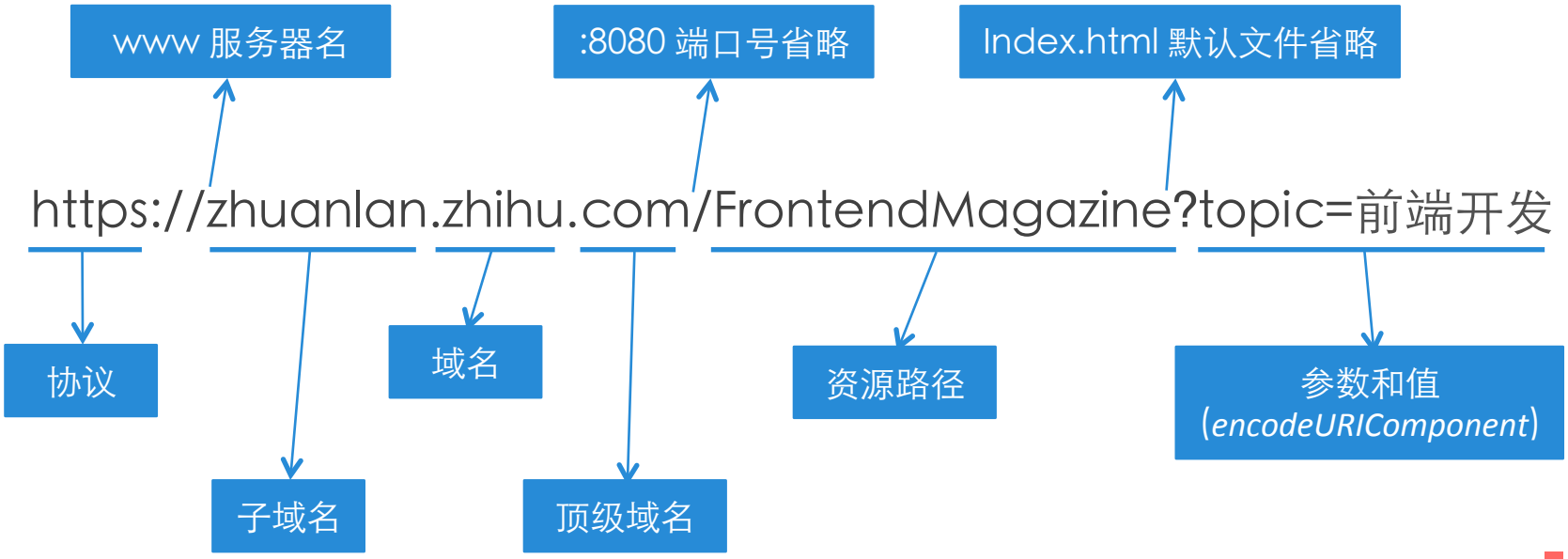


输入URL → 开启网络

解析URL

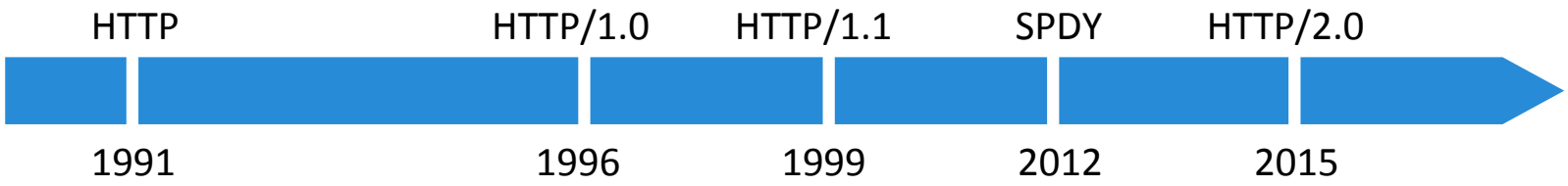
URL完整格式为

协议://用户名:密码@子域名.域名.顶级域名:端口/目录/文件名.文件后缀?参数=值#标志



#标志： hash值，一般用来定位

输入URL → 开启网络



网络应用层　　TCP协议　　三握四挥RTT

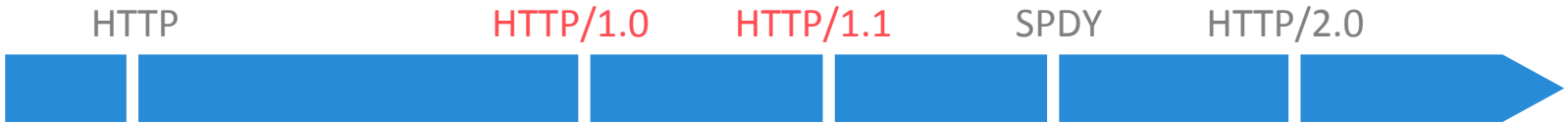
影响一个HTTP网络请求的因素

带宽

延迟

浏览器阻塞　　DNS　　连接建立

输入URL → 开启网络

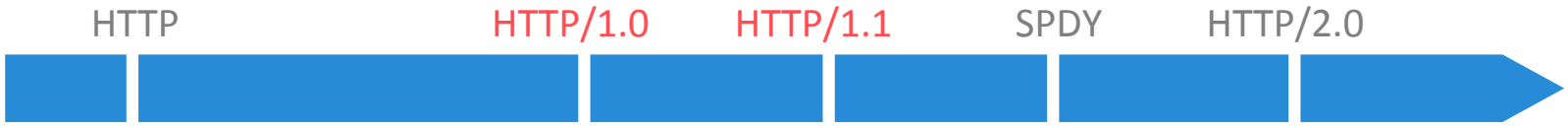


Differences

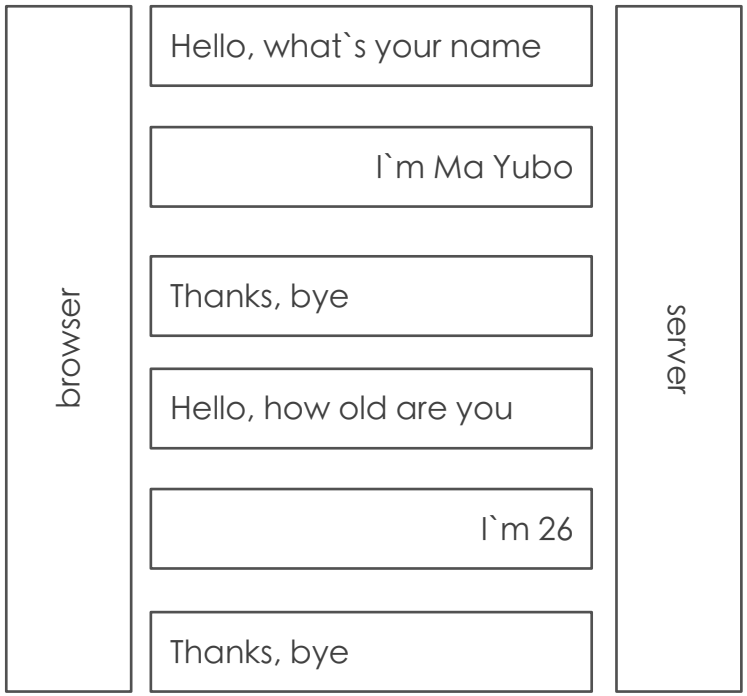
缓存	Expires, Entity tag, If-Unmodified-Since, If-Match, If-None-Match		
带宽优化连接利用	断点续传	Range头 Content-range头	Code：206
错误通知	24个错误状态响应码		
Host头处理	一物理，多虚拟　共享IP		
长连接	长连接和请求流水线，一个TCP多个HTTP请求和响应 默认开启Connection：keep-alive		

输入URL → 开启网络

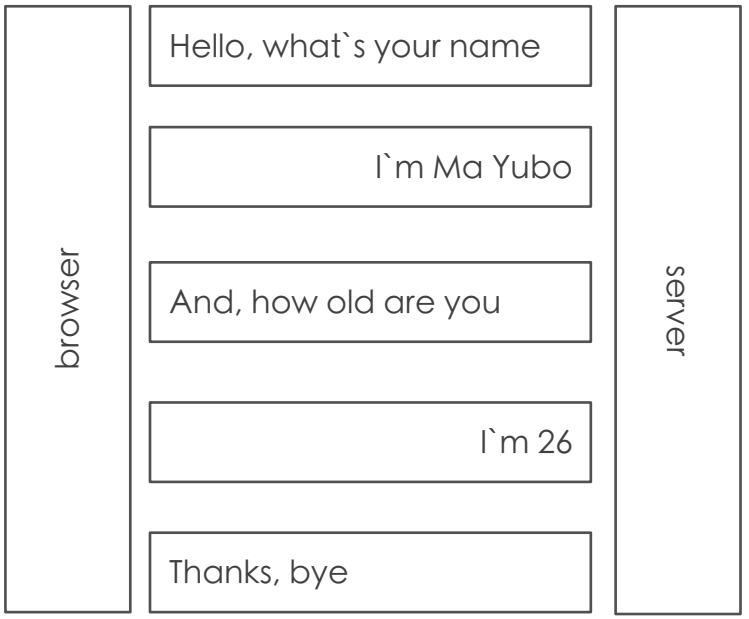
HTTP们



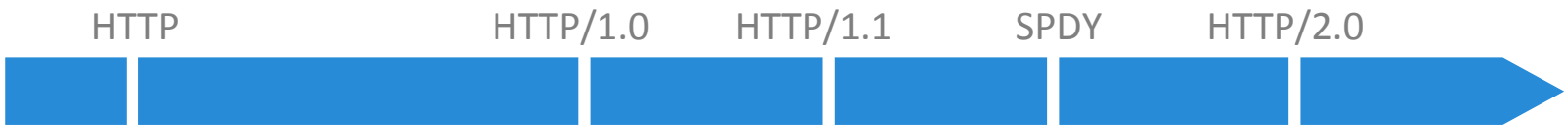
HTTP/1.0



HTTP/1.1



输入URL → 开启网络



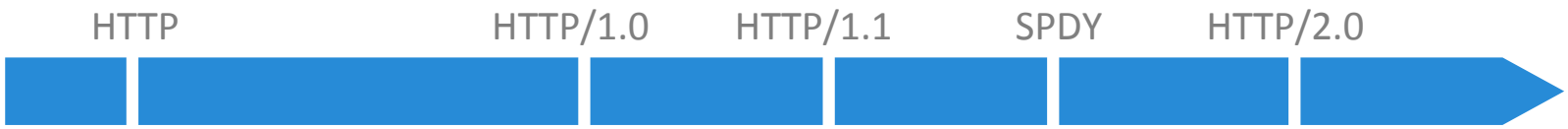
HTTP 1.X Problems

传输数据时，每次都要重新建立连接，增加大量延迟。

传输数据时，传输内容是明文，两端都无法互相验证身份，无法保证数据的安全性。

header里携带内容过大，增加了传输的成本。

大量使用keep-alive同样会给服务端带来巨大压力，尤其对于单文件被不断请求的服务(例如图片)。



HTTPS Differences

安全版的HTTP。

需要证书，传输内容加密，连接方式不同，防劫持，缺省端口443。

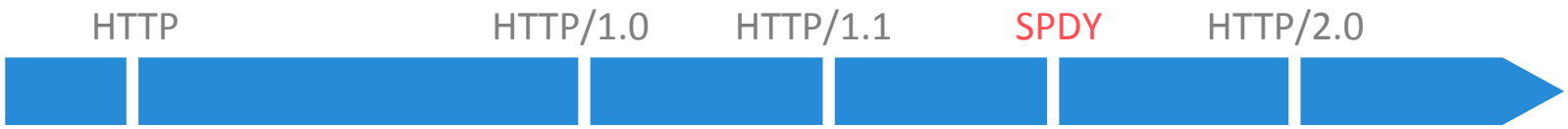
HTTPS Problems

证书费用。

SSL握手耗时，服务端解密钥。

2014年 OpenSSL Heartbleed安全漏洞

输入URL → 开启网络



SDPY Differences　　Google(-64%)

降低延迟：优雅地采取了多路复用，多个请求stream共享一个TCP连接。

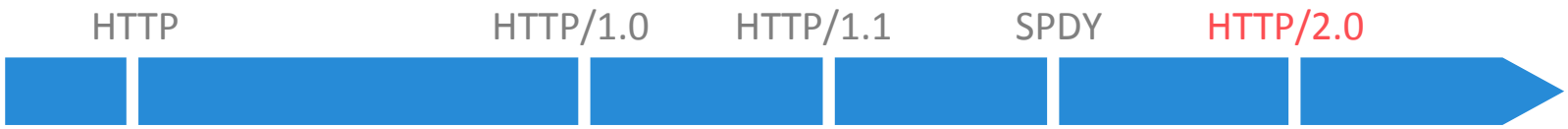
请求优先级：通过设置优先级解决共享连接造成的阻塞。

Header报头压缩：压缩并减少重复多余地header头。

基于HTTPS的加密协议传输。

服务端启动流：分发内容到客户端，而不需客户端发起请求。

输入URL → 开启网络



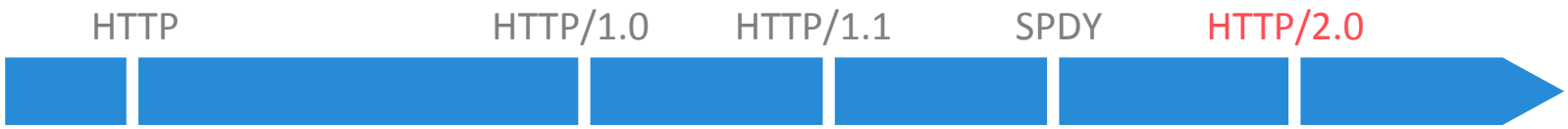
HTTP/2.0 Features

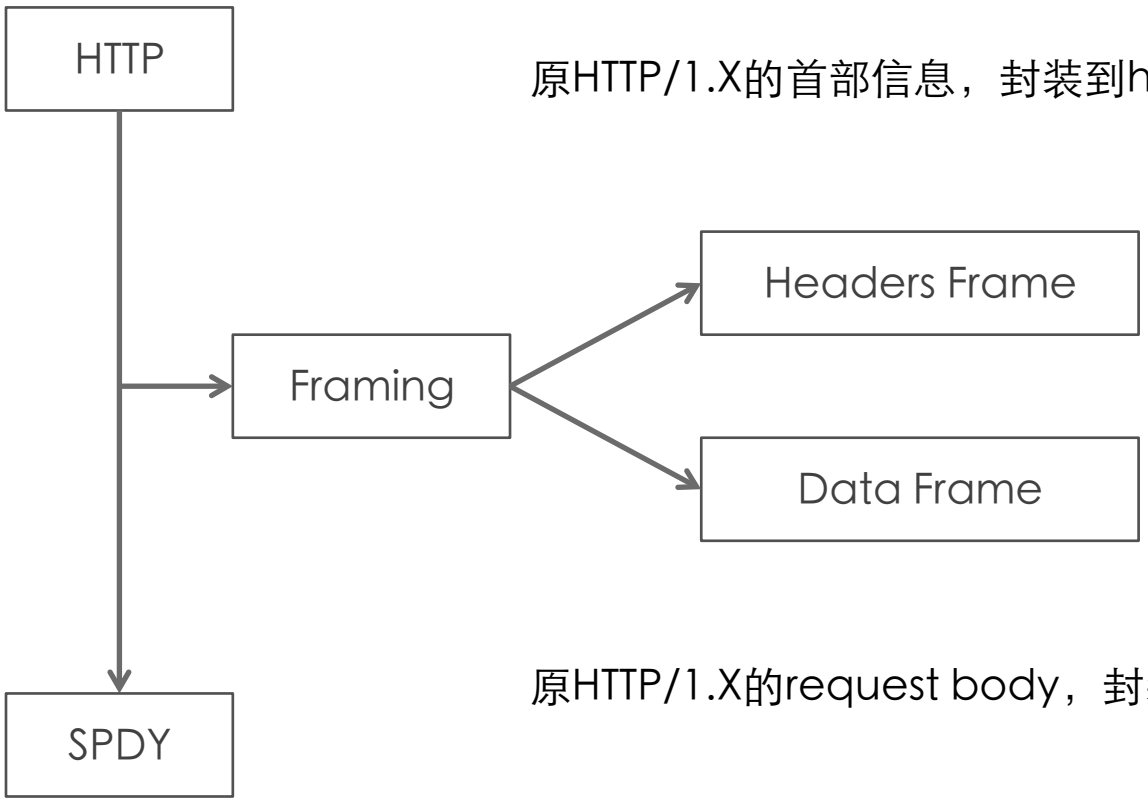
向下兼容，基于SPDY，但略有不同：

HTTP/2.0 支持明文，Header报头压缩算法不同。

- 二进制分帧
- header压缩
- 服务端推送
- 请求合并
- 请求优先级

输入URL → 开启网络



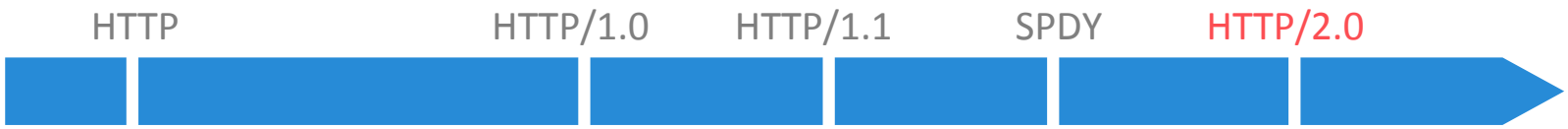


原HTTP/1.X的首部信息，封装到header帧

原HTTP/1.X的request body，封装到data帧

HTTP/2.0之二进制分帧

输入URL → 开启网络



Features

HTTP 2.0 通信都在一个连接上完成。

此连接可以承载任意数量的双向数据流。

每个数据流以消息的形式发送，而消息由一或多个帧组成。

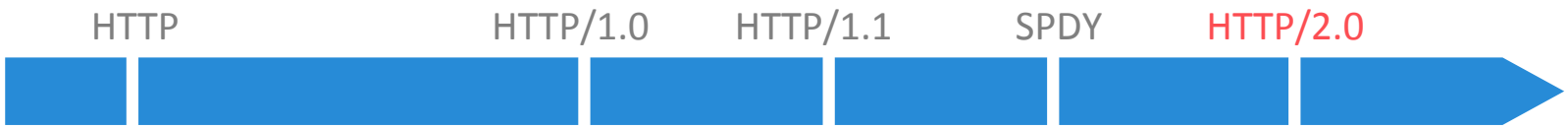
帧可以乱序，根据帧首部的流标识符重新组装。

保留HTTP 的语义、方法、状态码、URI 及首部字段等等这些核心概念。

突破上一代标准的性能限制，改进传输性能，实现低延迟和高吞吐量。

HTTP/2.0之二进制分帧

输入URL → 开启网络



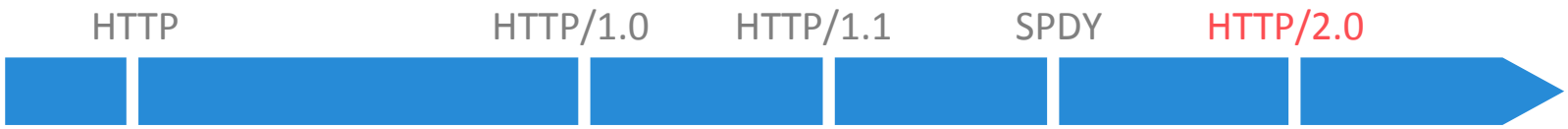
Features

客户端和服务器端，共同维护“首部表”（键-值对）。

相同的数据，只发一次。

不同的数据，渐进更新 。

输入URL → 开启网络



Features

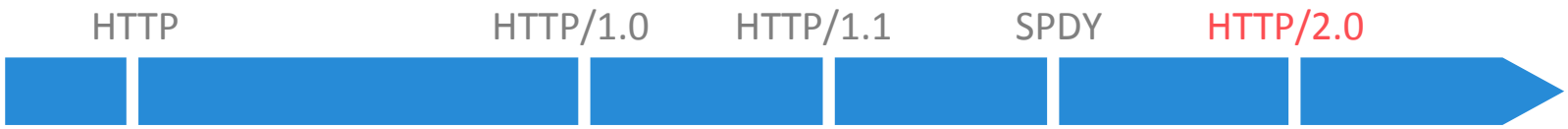
基本单位：独立帧　　并行双向交错发送消息　　单链接多资源

减少服务端压力，内存占用少，连接吞吐量大，TCP 连接减少，改善网络拥塞状况。

慢启动时间减少，拥塞和丢包恢复速度更快。

HTTP/2.0 之 One TCP

输入URL → 开启网络



Improvements

“资源合并减少请求”的优化手段对于HTTP2.0来说是没有效果

“域名分区”优化手段对于HTTP2.0是无用的，因为资源并行交错发送，且无限制

Problems

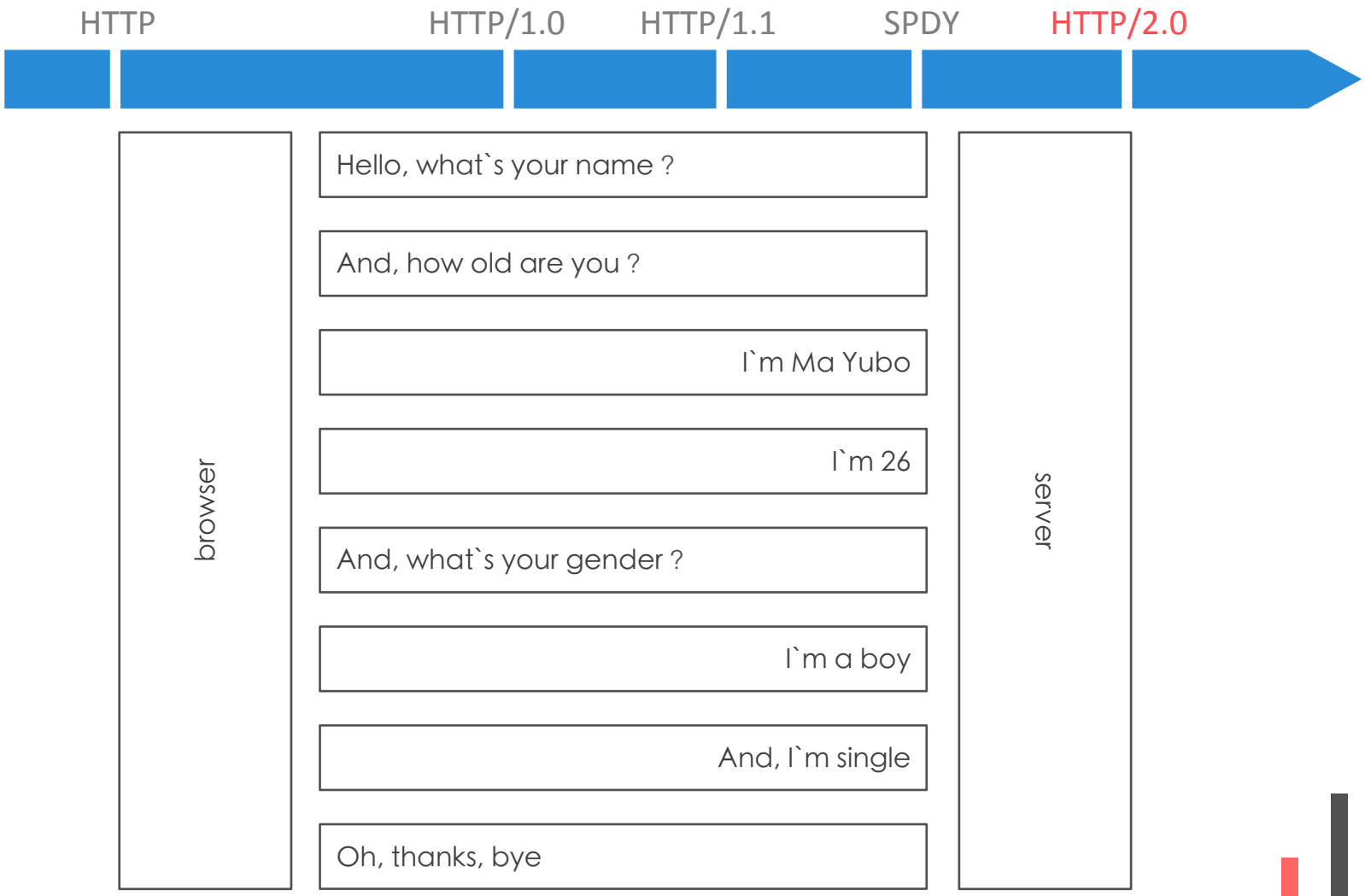
由于并行交错独立无限制，会不会导致应该加载JS资源时，却在加载图片？

请求优先级

HTTP/2.0 之 One TCP

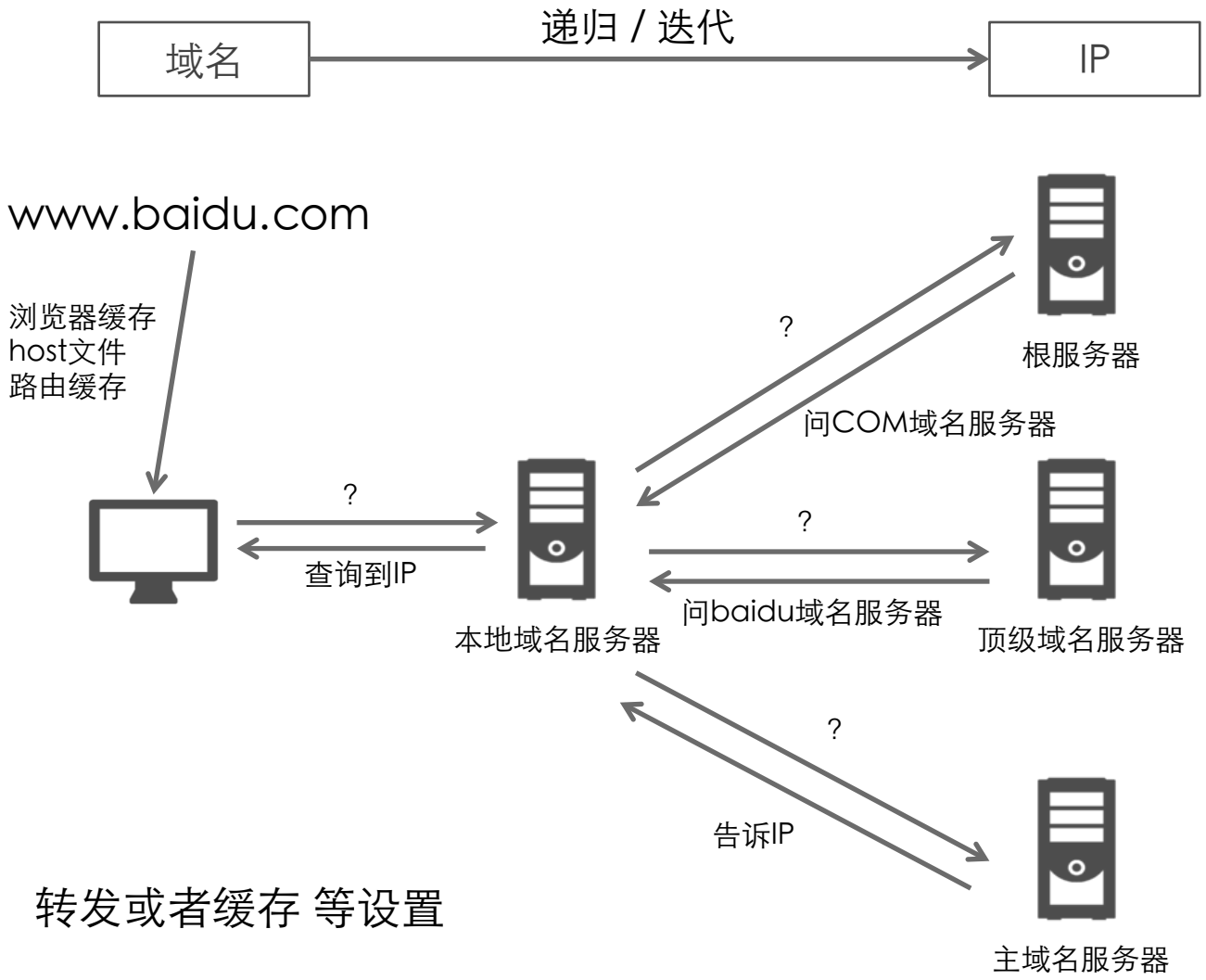
输入URL → 开启网络

HTTP/2.0之示例



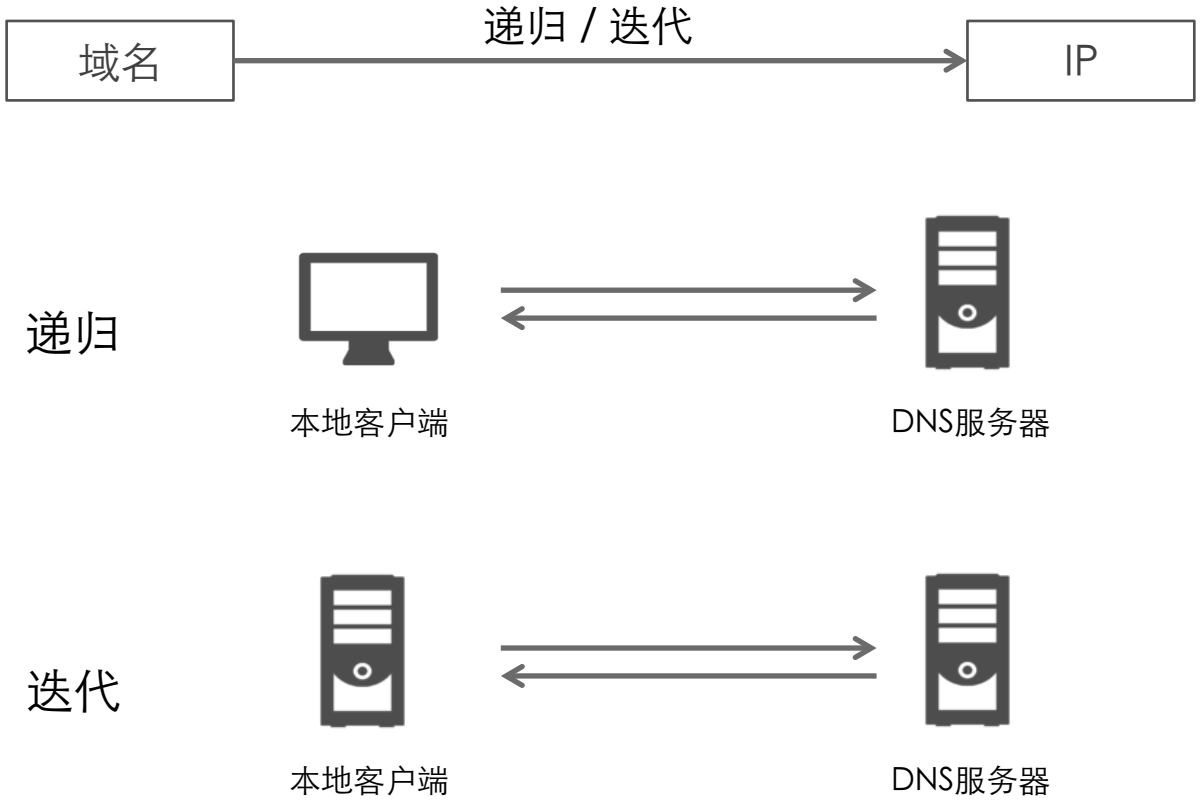
开启网络 → HTTP请求

DNS解析



转发或者缓存 等设置

开启网络 → HTTP请求



DNS缓存

负载均衡

开启网络 → HTTP请求

CDN

内容分发网络

广泛采用各种缓存服务器，依靠部署在各地的边缘服务器，通过中心平台的负载均衡、内容分发、调度等功能模块，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，降低网络拥塞，提高用户访问响应速度和命中率。

关键技术

内容存储发布

内容负载均衡（DNS等）

内容分发调度

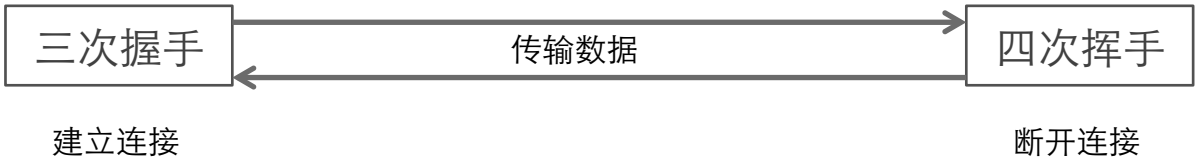
性能监听管理

https://www.zhihu.com/question/37353035/answer/71584314

开启网络 → HTTP请求

互联网内各网络设备间的通信都遵循TCP/IP协议。

HTTP本质是TCP/IP请求，使用TCP作为其传输层协议的



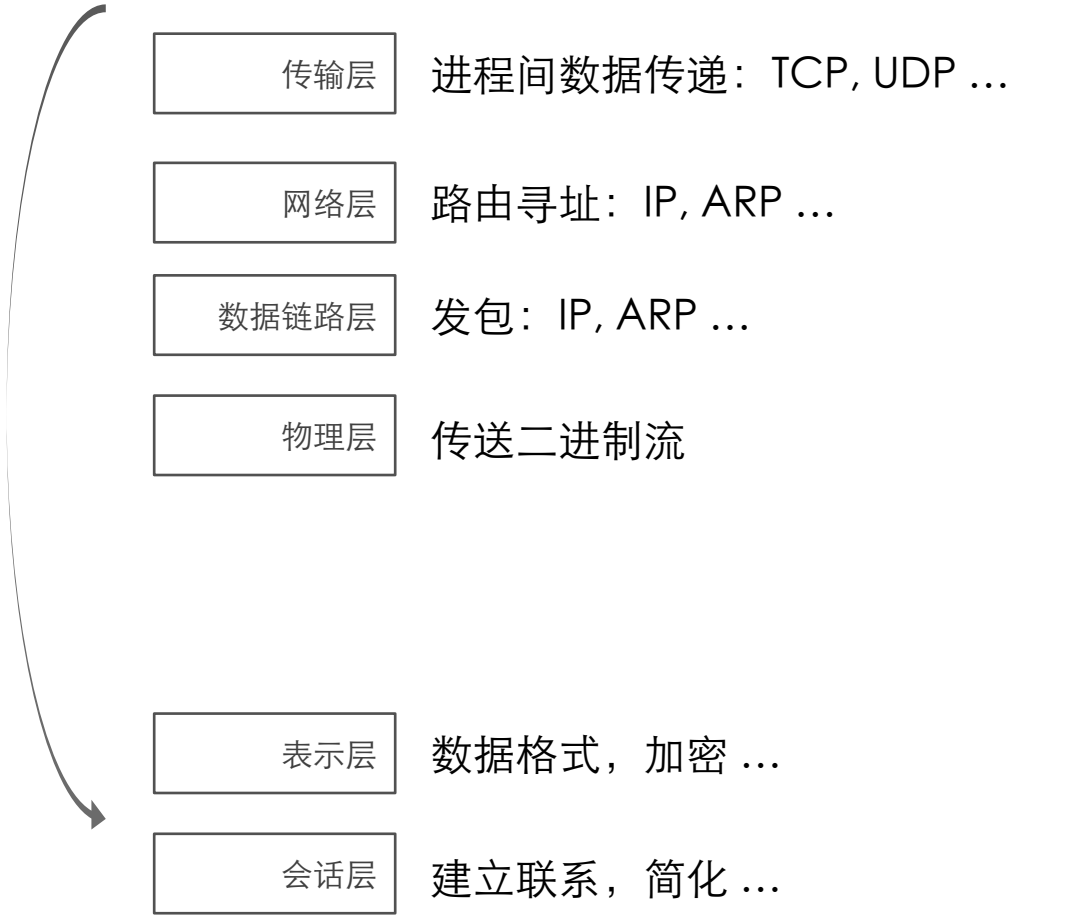
TCP/IP瓶颈及优化方案

GET	Headers 和 data 一并发出，服务器响应200，返回数据
POST	Headers 先发出，服务器响应100，data 再发出，响应200，返回数据

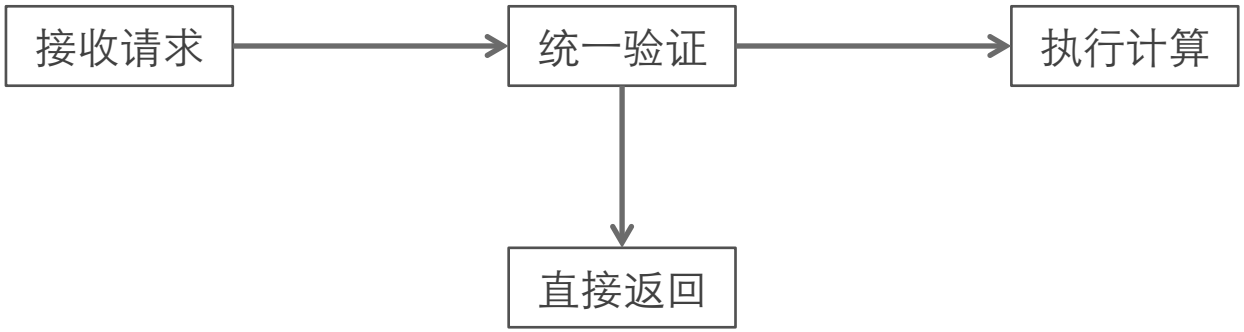
开启网络 → HTTP请求

因特网五层协议栈

OSI七层模型



HTTP请求→后台接收→返回前端



信息的载体：http报文

http报文一般结构

通用头部

请求/响应头部

请求/响应体

HTTP请求→后台接收→返回前端

HTTP 报文结构

通用头部 General

Request URL	请求web服务地址
Request Method	请求方式
Status Code	请求返回状态码
Remote Address	请求远程服务地址
Referrer Policy	请求来源的策略

HTTP请求→后台接收→返回前端

HTTP 报文结构

Request Method（GET 和 POST 的区别）

get	post
浏览器回退无害	浏览器回退再一次提交
URL地址可以被bookmark	URL地址不可被bookmark
会被浏览器主动cache	不会，除非手动设置
请求只可进行URL编码	请求支持多种编码
请求参数会被浏览器历史记录完整保留	请求参数不会被保留
传参长度有限 (IE浏览器URL为2K字节，或操作系统本身)	未做明显限制 (受限于服务器的限制)
只接收ASCII字符	未限制接收字符
不安全，参数直接暴露	相比之下更安全
参数通过URL传递	参数放在 Request body 中
1个TCP包	2个TCP包

本质无异（TCP），行为不同

HTTP请求→后台接收→返回前端

Referrer Policy

2014年 W3C Web应用安全组发布。

页面引入图片和JS 等资源，或者跳转页面，会产生新的 HTTP 请求。浏览器一般都会给这些请求头加上表示来源的 Referer 字段。

广泛用于分析用户来源，但可能包含用户敏感信息。

指定 Referrer Policy 方法

CSP（Content Security Policy）

<meta> 标签

<a> <img>等标签 referrer 属性

https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Headers/Referrer-Policy

HTTP请求→后台接收→返回前端

HTTP报文结构

常见部分请求头

Accept	浏览器支持的MIME类型
Accept-Encoding	浏览器支持的压缩类型
Accept-Language	浏览器可理解的自然语言，优先选择语言
Cache-control	指定遵循的缓存机制
Connection	通信时，对于长链接的处理，如keep-alive
Cookie	存在并且同域访问时携带
Content-Type	客户端发出的实体内容类型

https://developer.mozilla.org/zh-CN/docs/tag/HTTP

HTTP请求→后台接收→返回前端

HTTP报文结构

Expires	缓存控制
Host	请求服务器URL
If-Modify-Since	匹配文件是否变动（对应Last-Modified）
If-No-Match	匹配文件内容是否改变（对应ETag）
Max-age	资源缓存时间
Origin	初始请求来源
Pragma	No-cache 时禁用缓存（HTTP1.0）
Referer	页面来源信息
User-Agent	客户端必要信息

HTTP请求→后台接收→返回前端

HTTP报文结构

BDP示例

▼ **Request Headers** view source

Accept: application/json
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,zh-TW;q=0.8,en;q=0.7
Cache-Control: no-cache
Connection: keep-alive
Cookie: locale=zh; MEIQIA_EXTRA_TRACK_ID=b298d59cf7bbccab35b8a514fd49cf9d; theme_id=2; user_id=86607ce5e1cc7a478b311da072e8e131; user=mayubo; domain=haizhi; is_weak=0; yaxis_max=2416; yaxis_min=0; yaxis_max_right=114; yaxis_min_right=0; grid_index=0; select_folder_id=folder_root; ds_tb_id=tb_7c232dd49bad4059a04738034e92ef95; Hm_lvt_26f4ab258444603cf0bec2222ed39db4=1521775086,1522036118,1522723902,1523081635; bdp_data2017jssdkcross=%7B%22distinct_id%22%3A%2215e187a3f5f2ae-0b427e13371dfb-31627c01-2073600-15e187a3f627af%22%2C%22props%22%3A%7B%22%24latest_referrer%22%3A%22%22%2C%22%24latest_referrer_host%22%3A%22%22%2C%22check_params%22%3A%22isExpired%3Dtrue%26lastSessionTime%3D1510711225417%26bdpa_session_isExpired%3Dtrue%26sessionStore%3D%7B%5C%22session_time%5C%22%3A1510711225417%2C%5C%22session_id%5C%22%3A%5C%2215fbd67d2db1d9-0e1c97e620d94f-173f6d56-2073600-15fbd67d2dc1b0%5C%22%2C%5C%22session_hasBeenExpired%5C%22%3A0%2C%5C%22lastSend_sessonId%5C%22%3A%5C%2215fbd67d2db1d9-0e1c97e620d94f-173f6d56-2073600-15fbd67d2dc1b0%5C%22%7D%22%2C%22%24is_first_session%22%3A0%2C%22user_id%22%3A%2286607ce5e1cc7a478b311da072e8e131%22%2C%22ver%22%3A1804032586%7D%7D; Hm_lpvt_26f4ab258444603cf0bec2222ed39db4=1523081637; token=6743f2a17854d98efe241c6e43586605da6beef2691e4481; __bdpa_session_key__2017__=%7B%22session_time%22%3A1523081644644%2C%22session_id%22%3A%221629ebd63de45f-06484b5ec55dd6-336d7b05-2073600-1629ebd63df804%22%2C%22session_hasBeenExpired%22%3A0%2C%22lastSend_sessonId%22%3A%221629ebd63de45f-06484b5ec55dd6-336d7b05-2073600-1629ebd63df804%22%7D
Host: www.bdp.cn
Pragma: no-cache
Referer: https://www.bdp.cn/index.html?lang=zh
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36

▼ **Request Headers** view parsed

```
GET /api/user/info?_t=1523081656817 HTTP/1.1
Host: www.bdp.cn
Connection: keep-alive
Pragma: no-cache
Cache-Control: no-cache
Accept: application/json
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_13_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/65.0.3325.181 Safari/537.36
Referer: https://www.bdp.cn/index.html?lang=zh
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9,zh-TW;q=0.8,en;q=0.7
Cookie: locale=zh; MEIQIA_EXTRA_TRACK_ID=b298d59cf7bbccab35b8a514fd49cf9d; theme_id=2; user_id=86607ce5e1cc7a478b311da072e8e131; user=mayubo; domain=haizhi; is_weak=0; yaxis_max=2416; yaxis_min=0; yaxis_max_right=114; yaxis_min_right=0; grid_index=0; select_folder_id=folder_root; ds_tb_id=tb_7c232dd49bad4059a04738034e92ef95; Hm_lvt_26f4ab258444603cf0bec2222ed39db4=1521775086,1522036118,1522723902,1523081635; bdp_data2017jssdkcross=%7B%22distinct_id%22%3A%2215e187a3f5f2ae-0b427e13371dfb-31627c01-2073600-15e187a3f627af%22%2C%22props%22%3A%7B%22%24latest_referrer%22%3A%22%22%2C%22%24latest_referrer_host%22%3A%22%22%2C%22check_params%22%3A%22isExpired%3Dtrue%26lastSessionTime%3D1510711225417%26bdpa_session_isExpired%3Dtrue%26sessionStore%3D%7B%5C%22session_time%5C%22%3A1510711225417%2C%5C%22session_id%5C%22%3A%5C%2215fbd67d2db1d9-0e1c97e620d94f-173f6d56-2073600-15fbd67d2dc1b0%5C%22%2C%5C%22session_hasBeenExpired%5C%22%3A0%2C%5C%22lastSend_sessonId%5C%22%3A%5C%2215fbd67d2db1d9-0e1c97e620d94f-173f6d56-2073600-15fbd67d2dc1b0%5C%22%7D%22%2C%22%24is_first_session%22%3A0%2C%22user_id%22%3A%2286607ce5e1cc7a478b311da072e8e131%22%2C%22ver%22%3A1804032586%7D%7D; Hm_lpvt_26f4ab258444603cf0bec2222ed39db4=1523081637; token=6743f2a17854d98efe241c6e43586605da6beef2691e4481; __bdpa_session_key__2017__=%7B%22session_time%22%3A1523081644644%2C%22session_id%22%3A%221629ebd63de45f-06484b5ec55dd6-336d7b05-2073600-1629ebd63df804%22%2C%22session_hasBeenExpired%22%3A0%2C%22lastSend_sessonId%22%3A%221629ebd63de45f-06484b5ec55dd6-336d7b05-2073600-1629ebd63df804%22%7D
```

HTTP请求→后台接收→返回前端

常见部分响应头

Age	使用缓存时，表示经过多少时间
Content-Type	…
Content-Encoding	…
Cache-control	…
Connection	…
Date	消息发送时间
ETag	控制缓存的验证标签
Expires	…

HTTP请求→后台接收→返回前端

Keep-Alive	…
Last-Modified	请求资源的最后修改时间
Max-age	本地资源应该缓存时间范围
Server	服务器相关必要信息
Set-Cookie	服务器设置cookie
Transfer-Encoding	将entity安全传递给用户所采用的编码
Vary	服务器/缓存/CDN，判断请求是否一致的依据

HTTP请求→后台接收→返回前端

HTTP 报文结构

BDP示例

▼ **Response Headers** view parsed

```
HTTP/1.1 200 OK
Server: BLB/
Date: Sat, 07 Apr 2018 06:14:16 GMT
Content-Type: application/json;charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Vary: Accept-Encoding
Etag: W/"0e6d7c18c9a2bffddfcb983fc1e06ea9e49b82b4"
Content-Encoding: gzip
```

▼ **Response Headers** view source

Connection: keep-alive
Content-Encoding: gzip
Content-Type: application/json;charset=utf-8
Date: Sat, 07 Apr 2018 06:14:16 GMT
Etag: W/"0e6d7c18c9a2bffddfcb983fc1e06ea9e49b82b4"
Server: BLB/
Transfer-Encoding: chunked
Vary: Accept-Encoding

HTTP请求→后台接收→返回前端

HTTP 角度的性能优化

Cookie优化

安全角度： HTTPOnly　　RSA非对称加密

传输角度：多域名拆分（静态资源分组）
DNS-prefetch（空闲时提前解析dns）

GZIP压缩

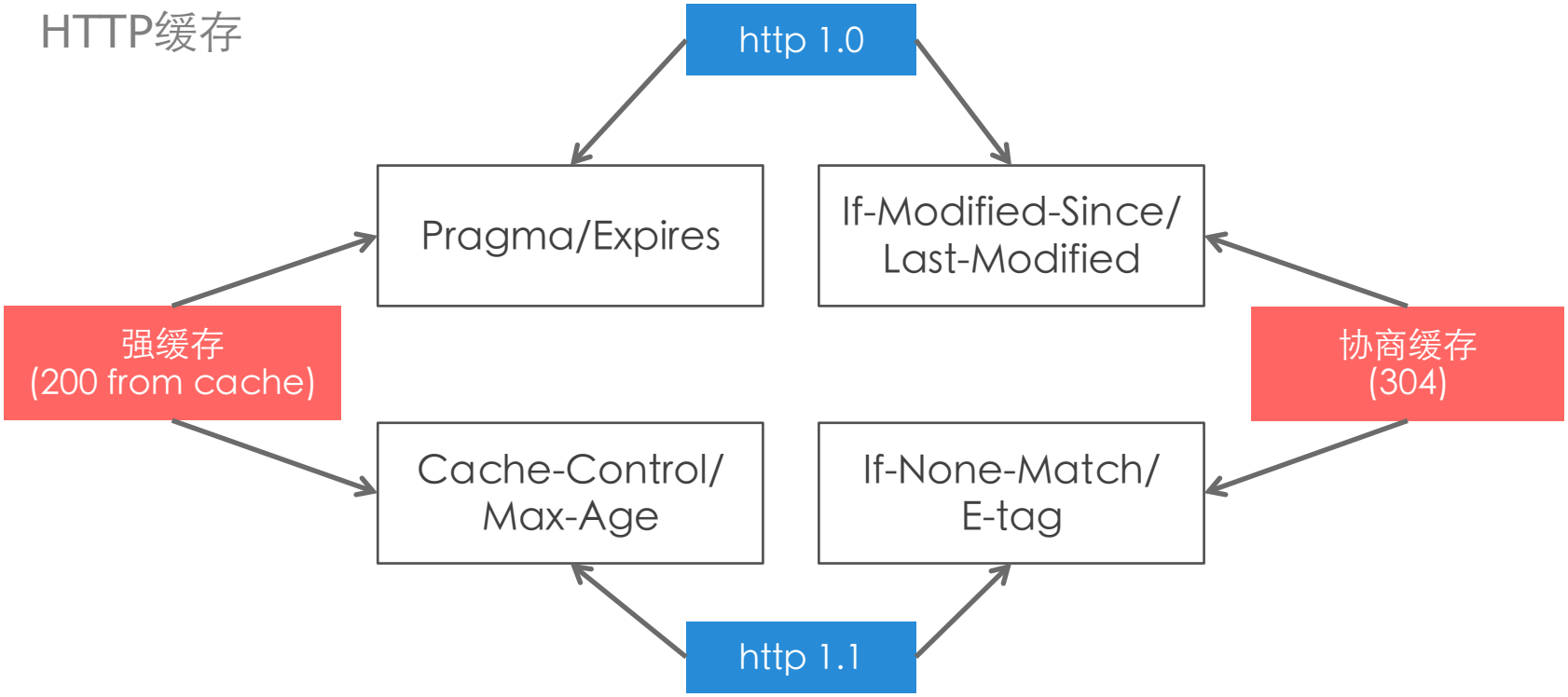
HTTP -> HTTPS -> HTTP2.0

HTTP缓存

强缓存(200 from cache)

协商缓存(304)

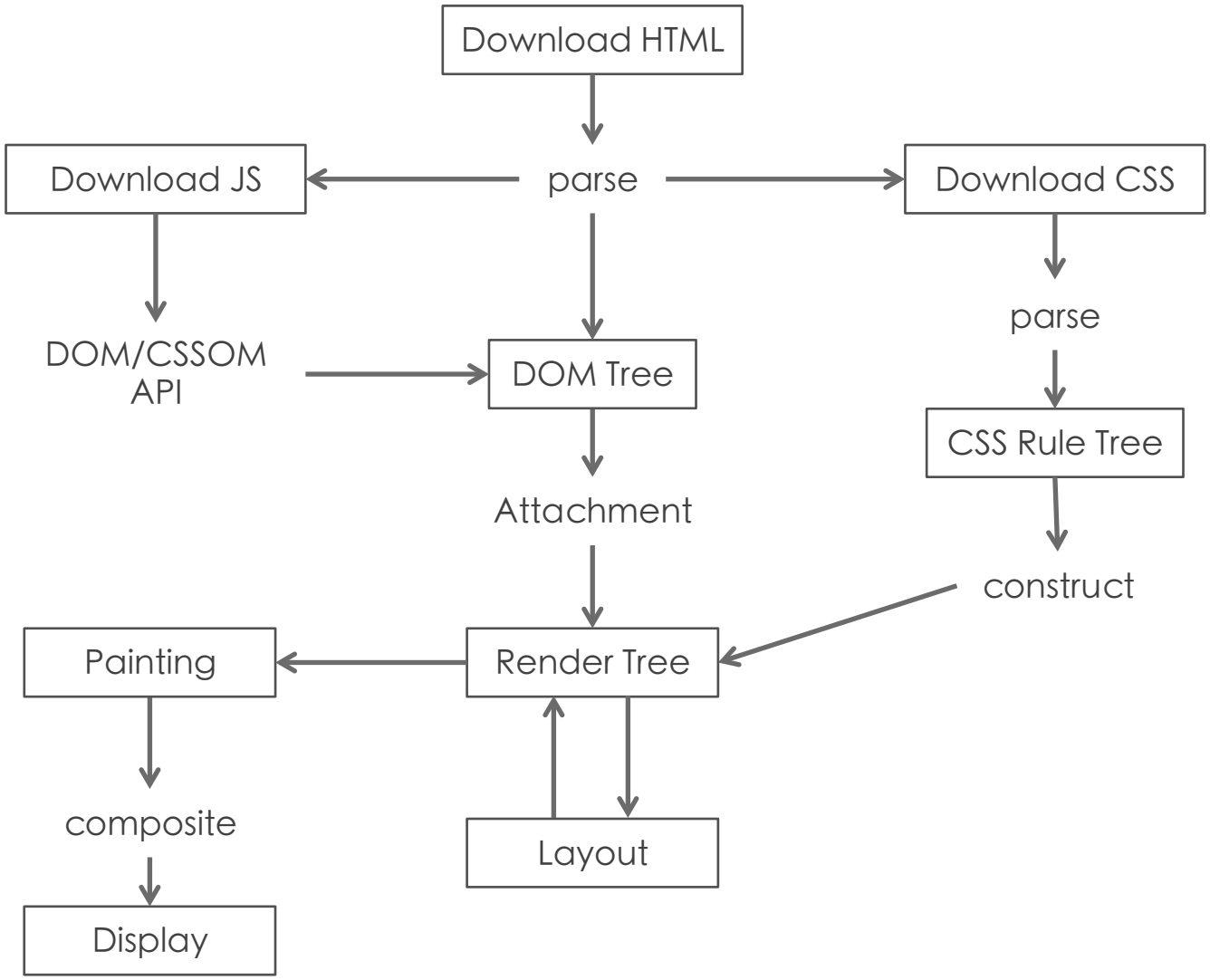
HTTP请求→后台接收→返回前端



Expires 服务器时间， Max-Age 客户端时间
Cache-Control 优先级高于 Expires

Last-Modified 校验时间，精确到秒， E-tag 校验文件无精确限制
E-tag优先级高于 Last-Modified

HTTP返回→浏览器解析渲染页面



http://taligarsiel.com/Projects/howbrowserswork1.htm

HTTP返回 → 浏览器解析渲染页面

解析页面基本流程

每一步逐步完成，为了更好的用户体验，渲染引擎将会尽可能早的将内容呈现到屏幕上，并不会等到所有的html都解析完成之后再去构建和布局render树。它是解析完一部分内容就显示一部分内容，同时，可能还在通过网络下载其余内容。

“自上而下”加载，加载中进行解析渲染

下载并解析资源文件

外部样式或者脚本可能会阻塞浏览器

DOM树，深度优先遍历，构建完触发 DOMContentLoaded

计算CSS

Render Tree 不同于 DOM Tree

按照节点样式布局页面

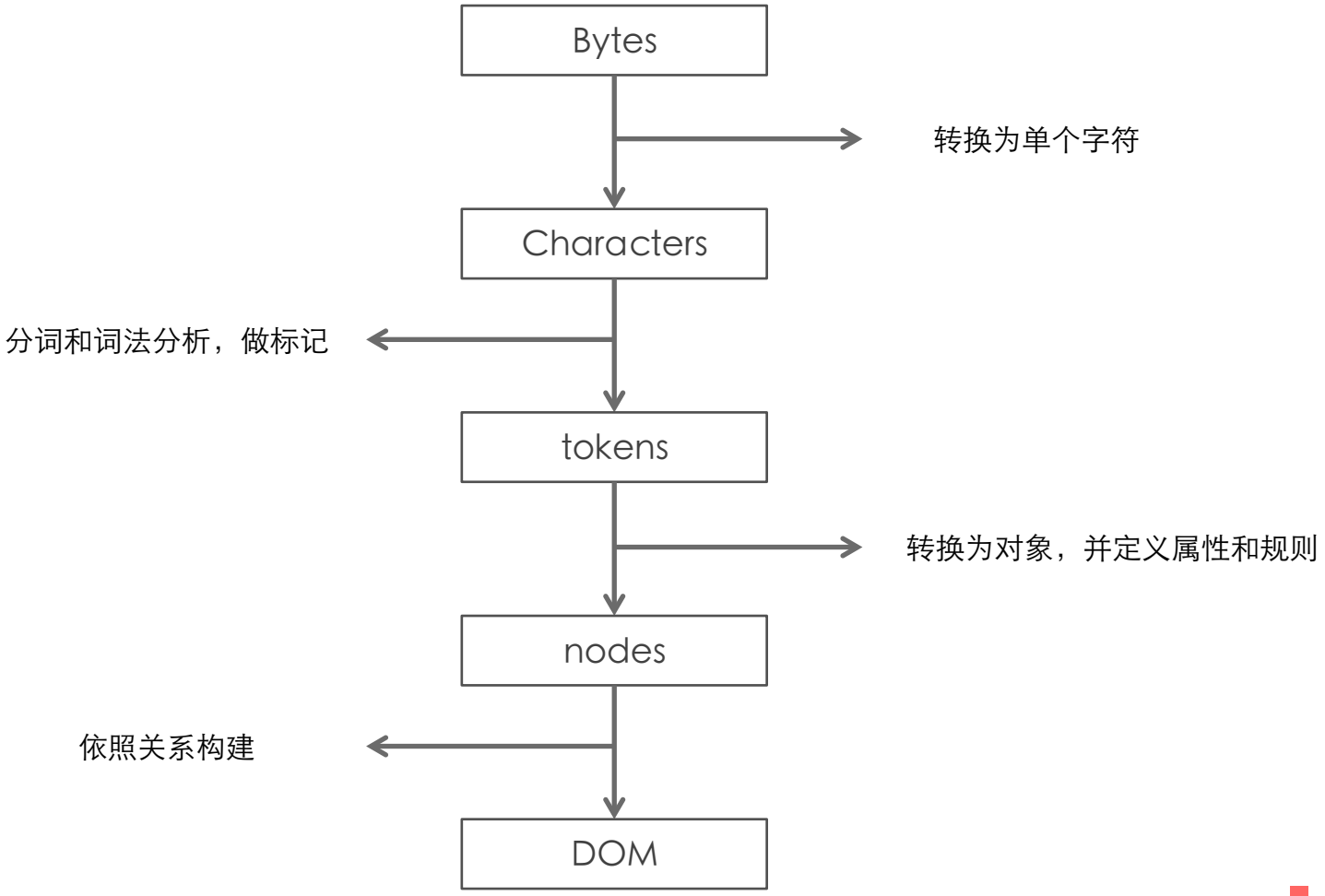
绘制页面

http://taligarsiel.com/Projects/howbrowserswork1.htm

HTTP返回 → 浏览器解析渲染页面

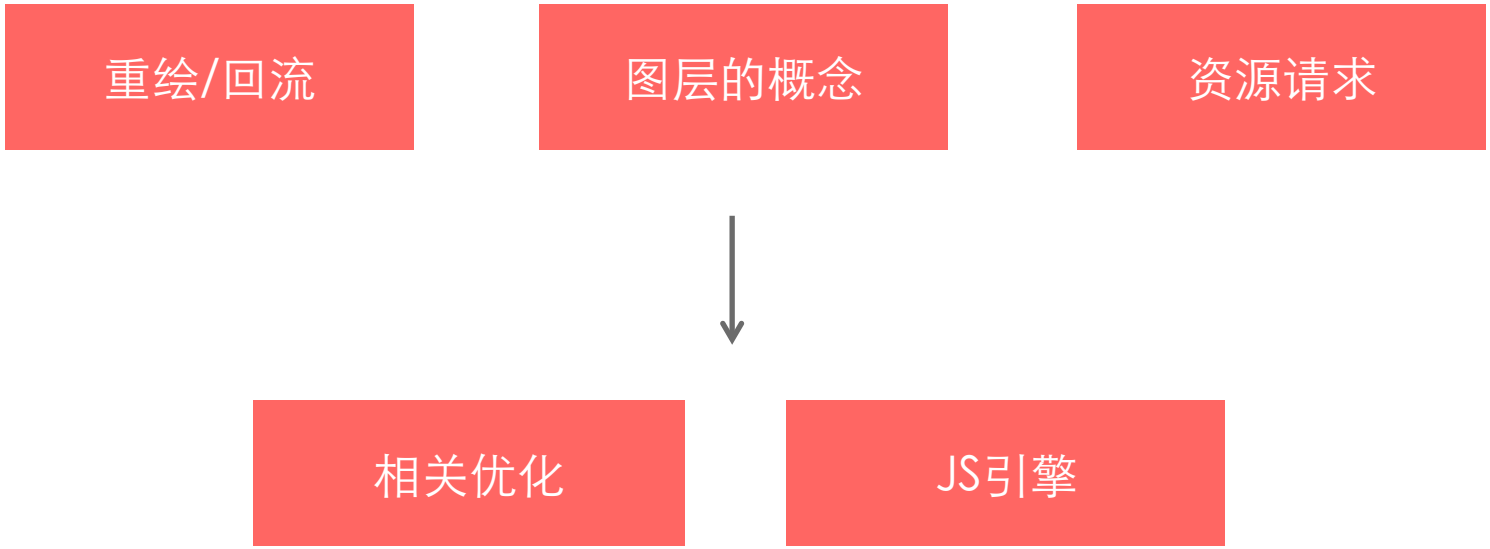
解析页面基本流程

解析 HTML 构建 DOM



HTTP返回 → 浏览器解析渲染页面

对 Render Tree 进行渲染



HTTP返回→浏览器解析渲染页面

重绘/回流

重绘

渲染树中的一些节点，改变了一些外观属性(如颜色，背景等)，
但并没有影响该元素的周边或内部布局，
会触发浏览器的重新绘制这些元素。

回流（重排）

渲染树中的一些节点，由于内容，尺寸，位置，结构等布局属性的改变
需要重新计算样式和渲染，
会触发浏览器的回流重排这些元素。

回流的成本开销高于重绘，而且一个节点的回流可能会导致子节点和同级节点的回流

回流必将引起重绘，但重绘并不一定引起回流

HTTP返回 → 浏览器解析渲染页面

常见触发回流的场景

页面渲染初始化

操作DOM，可见元素的增删

元素尺寸位置的改变

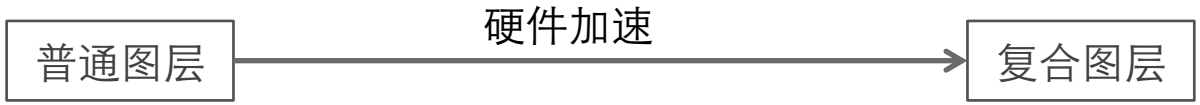
元素内容改变导致的计算宽高的改变

页面resize

浏览器针对触发回流的优化

维护一个队列，用于存储引起回流的操作，
达到时间限制或者存储长度后，批处理渲染

HTTP返回 → 浏览器解析渲染页面



图层的概念

How To ?

CSS属性 translate3d, translateZ, opacity

Advantage

GPU负责，独立于文档流，单独绘制不互相干扰，改变时不会触发重绘和回流

Attention

配合z-index使用，避免后续元素继承隐式使用复合图层

absolute 可脱离普通图层，但无法脱离复合图层

消耗资源，需适度使用

HTTP返回 → 浏览器解析渲染页面

JS 单线程

为了防止操纵DOM同时渲染界面（即JS线程和UI线程同时运行）时出现不可预期的结果，
浏览器设置GUI渲染线程与JS引擎为互斥的关系，
当JS引擎执行时，GUI线程会被挂起，GUI更新时，则会被保存在队列中，直至JS引擎线程空闲

Solution　HTML5 Web Worker

浏览器单独开辟子线程，不可操作DOM，不影响主线程

双方通过特定方式通信，postMessage API

阻塞问题

HTTP返回 → 浏览器解析渲染页面

阻塞问题

Load & DOMContentLoaded

DOMContentLoaded: 仅DOM加载完成，不包含样式图片的等异步资源

onLoad: 页面DOM，样式，图片，脚本等全部加载完成

静态资源加载

CSS JS IMG

HTTP1.1 中会会为每个外部资源开启一个请求

浏览器支持并行下载资源文件，但是并行数量有上限

HTTP返回 → 浏览器解析渲染页面

阻塞问题

CSS资源

异步加载，不阻塞DOM Tree，但会影响Render Tree

IMG资源

异步加载，不阻塞页面解析

JS资源

等待加载完成，阻塞HTML解析，页面渲染

defer & async　不阻塞页面渲染，<script>脚本中不可使用document.write

defer:　延迟执行，html解析完成后执行，按照加载顺序

async:　异步执行，资源异步下载之后执行，不按照加载顺序

HTTP返回→浏览器解析渲染页面

解析渲染相关优化

针对CSS的优化 （CSS 从右到左进行解析）

减少DOM树深度
减少inline引用的数量（若为了减少加载速度，还需使用inline）
减少修改浏览器定义好的默认样式

CSS代码的压缩合并，使用现代合法标准的的CSSS属性

避免选择符嵌套层级过深，
避免使用后代选择符，或尽量使用子选择符
避免使用通配符
避免使用计算表达式
避免使用filter属性
避免使用@import引入文件

不滥用浮动属性，不滥用web字体，不滥用高级CSS属性，不滥用@important
不使用id选择器，不保留空规则

学会使用CSS继承，学会不同属性之间有效或无效的配合

HTTP返回→浏览器解析渲染页面

解析渲染相关优化

针对JS的优化

JS代码的压缩合并，JS代码的位置，JS代码本身的编写规范

避免频繁进行DOM操作
避免使用过多的cookie

使用无阻塞的加载方式
使用动态加载脚本形式引入资源

针对IMG的优化

CSS-sprite合图
根据设定的标准改为base64内嵌
尺寸大小压缩，使用WebP
延迟加载，模糊图

HTTP返回 → 浏览器解析渲染页面

同步任务　主线程，执行栈（函数调用栈）

异步任务　事件触发线程，任务队列

异步过程概述

主线程发起一个异步请求，工作线程接收请求并告知主线程已收到(异步函数返回)

主线程继续执行后面代码，同时工作线程执行异步任务

工作线程完成工作后通知主线程

主线程收到通知后，执行某些动作(调用回调函数)

工作线程将消息放到消息队列中，由主线程通过事件循环过程去获取消息

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/EventLoop

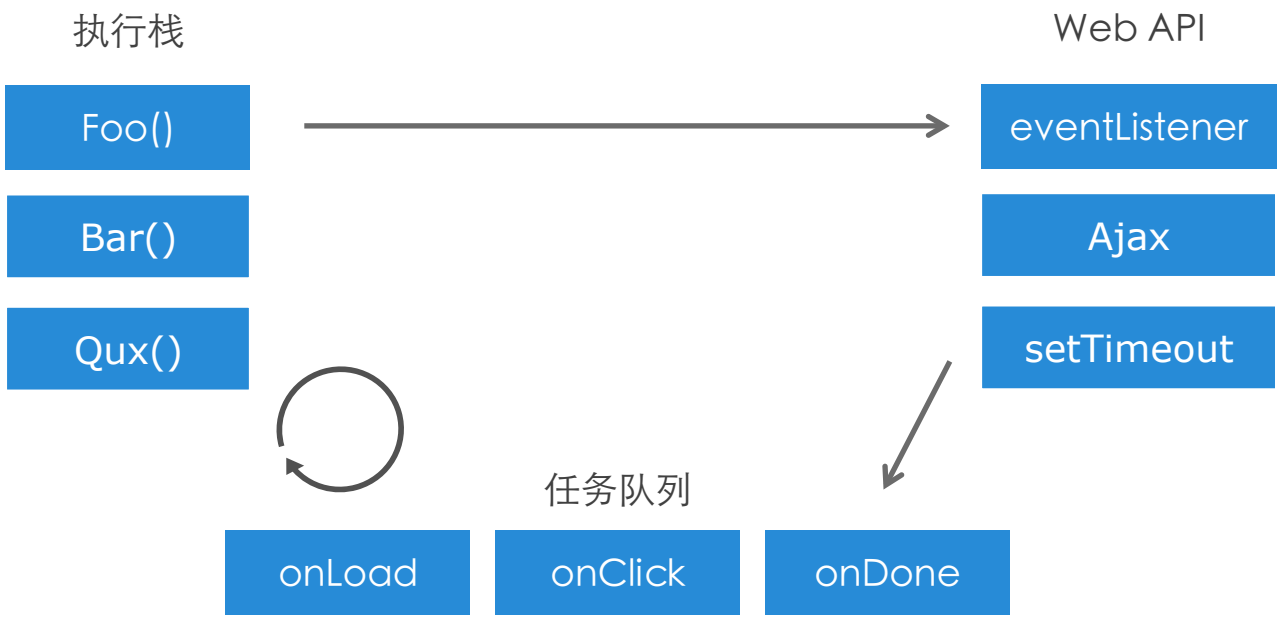
HTTP返回 → 浏览器解析渲染页面

JavaScript 事件循环

事件循环机制

主线程从消息队列里面读取取消息并执行消息。当消息队列为空时，就会等待直到消息队列变成非空。而且，主线程只有在执行完成当前的消息之后，才会再去获取取下一个消息。

队列消息：可以简单形象地认为是，在注册异步任务时所声明的回调函数



https://vimeo.com/96425312

HTTP返回 → 浏览器解析渲染页面

定时器

setTimeout / setInterval：开启定时器线程计时，计时完成后就会将特定的事件推入事件队列。

零延迟：并不意味着回调立即执行。其等待的时间基于队列里正在等待的消息数量。因为延迟是要求运行时 (runtime) 处理请求所需的最小时间，但不是有所保证的时间。

计时谁更准?

setTimeout	当次计时完成后，推送下次计时
setInterval	按照固定间隔时间，推送计时事件

JS引擎的优化，如果当前事件队列中有setInterval的回调，不会重复添加

HTTP返回 → 浏览器解析渲染页面

定时器

动画谁更适合？ requestAnimationFrame()

充分利用显示器的刷新机制，60帧/s

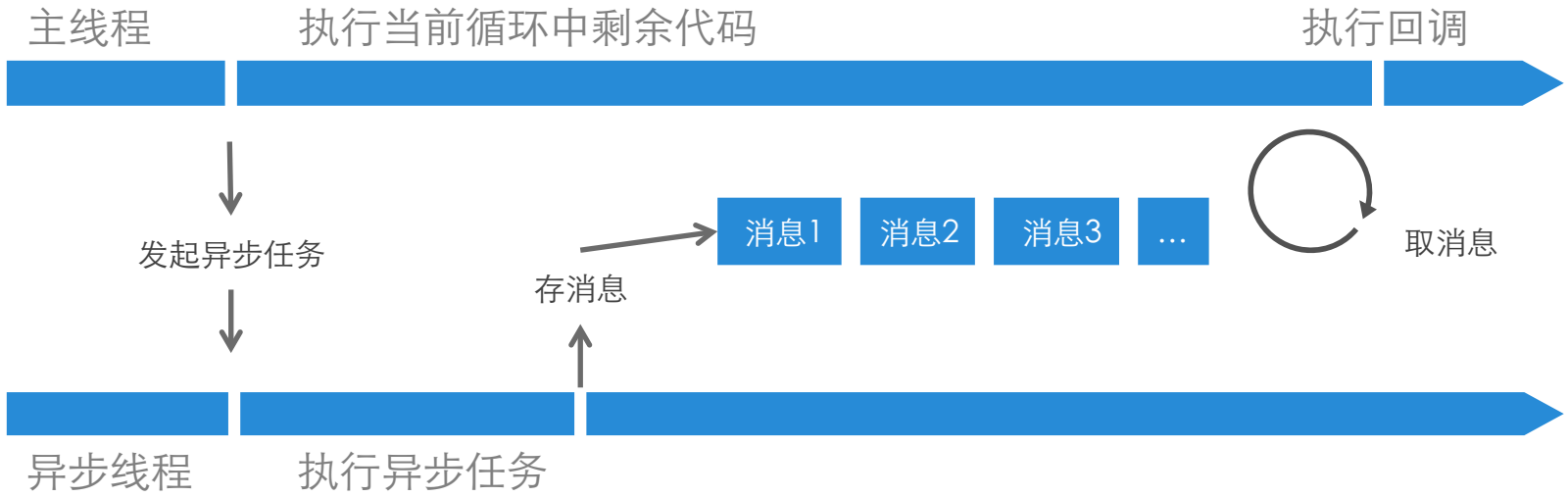
每帧中所有DOM操作集中在一次重绘或回流中就完成，并且重绘或回流时间间隔紧跟浏览器的刷新频率

对于隐藏或不可见的元素，将不会被进行重绘或回流

页面未激活状态下，动画暂停刷新，有效节省了CPU开销

在主线程上完成。如果主线程非常繁忙，requestAnimationFrame的动画效果会受影响。

HTTP返回→浏览器解析渲染页面



工作线程是生产者，主线程是消费者(只有一个)。
工作线程执行异步任务，执行完成后把对应的回调函数封装成一条消息放到消息队列中
主线程不断地从消息队列中取消息并执行，当消息队列空时，主线程阻塞，直到消息队列再次非空

浏览器解析渲染页面 → 连接结束

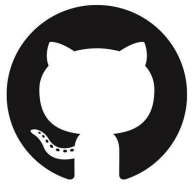
消息的持续性

Long poll / Ajax轮询	不断地建立HTTP连接，被动等待服务端处理推送消息
WebSocket	通过第一个 HTTP request 建立了 TCP 连接，之后的交换数据都不需要再发 HTTP request了 真正的长连接 双通道的连接，在同一个 TCP 连接上既可发也可收 具有 multiplexing 功能，不同的 URI 可以复用同一个 WebSocket 连接

总结

参考

简书



阮一峰的网络日志

http://www.alloyteam.com/2016/07/httphttp2-0spdyhttps-reading-this-is-enough

https://segmentfault.com/a/1190000012925872

http://www.dailichun.com/2018/03/12/whenyouenteraurl.html

《你不知道的Javascript》（中卷）

好多，好多，好多 ... ...