url到页面加载完成的过程及优化

1.写入URL，浏览器解析url，开辟单独的网络线程

2.dns解析，

2.1 浏览器检查缓存，域名缓存的时间限制是TTL属性来设置的

2.2如果用户浏览器缓存中没有数据，浏览器会查找操作系统缓存中是否有这个域名对应的DNS解析结果 在Windows中可以通过C:\Windows\System32\drivers\etc\hosts文件来设置，在Linux中可以通过/etc/hosts文件来设置

2.3这个域名发送给这个LDNS，也就是本地区的域名服务器。这个DNS通常也会在用户所在城市的某个角落，不会很远。 Windows环境下通过命令行输入ipconfig，Linux环境下通过cat /etc/resolv.conf就可以查询配置的DNS服务器了。

2.4如果LDNS仍然没有命中，就直接到Root Server域名服务器请求解析

2.5根域名服务器返回给本地域名服务器一个所查询的主域名服务器（gTLD Server）地址。gTLD是国际顶级域名服务器，如.com、.cn、.org等，全球只有13台左右

2.6本地域名服务器LDNS再向上一步返回的gTLD服务器发送请求

2.7、接受请求的gTLD服务器查找并返回此域名对应的Name Server域名服务器的地址，这个Name Server通常就是用户注册的域名服务器，例如用户在某个域名服务提供商申请的域名，那么这个域名解析任务就由这个域名提供商的服务器来完成

2.8、Name Server域名服务器会查询存储的域名和IP的映射关系表，在正常情况下都根据域名得到目标IP地址，连同一个TTL值返回给DNS Server域名服务器

2.9、返回该域名对应的IP和TTL值，LDNS会缓存这个域名和IP的对应关系，缓存时间由TTL值控制

2.10、把解析的结果返回给用户，用户根据TTL值缓存在本地系统缓存中，域名解析过程结束

dns解析优化，可以考虑dns-prefetch （dns预解析）

1. 用meta信息来告知浏览器, 当前页面要做DNS预解析:<meta http-equiv="x-dns-prefetch-control" content="on" />

2. 在页面header中使用link标签来强制对DNS预解析: <link rel="dns-prefetch" href="http://bdimg.share.baidu.com" />

3.tcp/ip请求

三次握手

1.client：你是server？

2.server：我是，你是client么？

3.client：我是

四次挥手

1主动方：我要关了，我只能收东西了

2被动方：收到

3.被动方：我也关了

4主动方：收到，无法通信

浏览器对同一域名下并发的tcp连接是有限制的（2-10个不等）

get和post的区别

1.GET是通过URL方式请求，可以直接看到，明文传输。

2.POST是通过请求header请求，可以开发者工具或者抓包可以看到，同样也是明文的。

3.GET请求会保存在浏览器历史纪录中，还可能会保存在Web的日志中。

五层因特网协议栈

1.应用层(dns,http) DNS解析成IP并发送http请求

2.传输层(tcp,udp) 建立tcp连接（三次握手）

3.网络层(IP,ARP) IP寻址

4.数据链路层(PPP) 封装成帧

5.物理层(利用物理介质传输比特流) 物理传输（然后传输的时候通过双绞线，电磁波等各种介质）

4 后台收到请求

负载均衡

5后台和前台的http交互

http报文结构 通用头部，请求/响应头部，请求/响应体

通用头部

Request URL:

https://segmentfault.com:9443/socket.io/?EIO=3&transport=polling&t=MErp0ou&sid=ZmuxEXsi3yl\_VaUfIGAW

1. Request Method:

GET

1. Status Code:

200 OK

1. Remote Address:

127.0.0.1:51931

1. Referrer Policy: //用于过滤 Referrer 报头内容

参考https://www.cnblogs.com/caixw/p/referrer-policy.html

no-referrer-when-downgrade //当发生降级（比如从 https:// 跳转到 http:// ）时，不传递 Referrer 报头。但是反过来的话不受影响。通常也会当作浏览器的默认安全策略。

请求/响应头部

常用的请求头部

Accept: 接收类型，表示浏览器支持的MIME类型

（对标服务端返回的Content-Type）

Accept-Encoding：浏览器支持的压缩类型,如gzip等,超出类型不能接收

Content-Type：客户端发送出去实体内容的类型

Cache-Control: 指定请求和响应遵循的缓存机制，如no-cache

If-Modified-Since：对应服务端的Last-Modified，用来匹配看文件是否变动，只能精确到1s之内，http1.0中

Expires：缓存控制，在这个时间内不会请求，直接使用缓存，http1.0，而且是服务端时间

Max-age：代表资源在本地缓存多少秒，有效时间内不会请求，而是使用缓存，http1.1中

If-None-Match：对应服务端的ETag，用来匹配文件内容是否改变（非常精确），http1.1中

Cookie: 有cookie并且同域访问时会自动带上

Connection: 当浏览器与服务器通信时对于长连接如何进行处理,如keep-alive

Host：请求的服务器URL

Origin：最初的请求是从哪里发起的（只会精确到端口）,Origin比Referer更尊重隐私

Referer：该页面的来源URL(适用于所有类型的请求，会精确到详细页面地址，csrf拦截常用到这个字段)

User-Agent：用户客户端的一些必要信息，如UA头部等

常用的响应头部

Access-Control-Allow-Headers: 服务器端允许的请求Headers

Access-Control-Allow-Methods: 服务器端允许的请求方法

Access-Control-Allow-Origin: 服务器端允许的请求Origin头部（譬如为\*）

Content-Type：服务端返回的实体内容的类型

Date：数据从服务器发送的时间

Cache-Control：告诉浏览器或其他客户，什么环境可以安全的缓存文档

Last-Modified：请求资源的最后修改时间

Expires：应该在什么时候认为文档已经过期,从而不再缓存它

Max-age：客户端的本地资源应该缓存多少秒，开启了Cache-Control后有效

ETag：请求变量的实体标签的当前值

Set-Cookie：设置和页面关联的cookie，服务器通过这个头部把cookie传给客户端

Keep-Alive：如果客户端有keep-alive，服务端也会有响应（如timeout=38）

Server：服务器的一些相关信息

CokIE以及优化

cookie是浏览器的一种本地存储方式，一般用来帮助客户端和服务端通信的，常用来进行身份校验，结合服务端的session使用

优化

将静态资源分组，分别放到不同的域名下（如static.base.com）

* 而page.base.com（页面所在域名）下请求时，是不会带上static.base.com域名的cookie的，所以就避免了浪费

长连接与短连接

长连接：一个tcp/ip连接上可以连续发送多个数据包，在tcp连接保持期间，如果没有数据包发送，需要双方发检测包以维持此连接，一般需要自己做在线维持（类似于心跳包）

Connection: keep-alive

keep-alive不会永远保持，它有一个持续时间，一般在服务器中配置（如apache），另外长连接需要客户端和服务器都支持时才有效

* 短连接：通信双方有数据交互时，就建立一个tcp连接，数据发送完成后，则断开此tcp连接

http2.0

一个tcp/ip请求可以请求多个资源，也就是说，只要一次tcp/ip请求，就可以请求若干个资源，分割成更小的帧请求，速度明显提升。

服务器端推送（服务端可以对客户端的一个请求发出多个响应，可以主动通知客户端）

首部压缩（http头部压缩，减少体积）

二进制分帧（在应用层跟传送层之间增加了一个二进制分帧层，改进传输性能，实现低延迟和高吞吐量）

https

在请求前，会建立ssl链接，确保接下来的通信都是加密的，无法被轻易截取分析

SSL/TLS的握手流程

1. 浏览器请求建立SSL链接，并向服务端发送一个随机数–Client random和客户端支持的加密方法，比如RSA加密，此时是明文传输。

2. 服务端从中选出一组加密算法与Hash算法，回复一个随机数–Server random，并将自己的身份信息以证书的形式发回给浏览器

（证书里包含了网站地址，非对称加密的公钥，以及证书颁发机构等信息）

3. 浏览器收到服务端的证书后

- 验证证书的合法性（颁发机构是否合法，证书中包含的网址是否和正在访问的一样），如果证书信任，则浏览器会显示一个小锁头，否则会有提示

- 用户接收证书后（不管信不信任），浏览会生产新的随机数–Premaster secret，然后证书中的公钥以及指定的加密方法加密`Premaster secret`，发送给服务器。

- 利用Client random、Server random和Premaster secret通过一定的算法生成HTTP链接数据传输的对称加密key-`session key`

- 使用约定好的HASH算法计算握手消息，并使用生成的`session key`对消息进行加密，最后将之前生成的所有信息发送给服务端。

4. 服务端收到浏览器的回复

- 利用已知的加解密方式与自己的私钥进行解密，获取`Premaster secret`

- 和浏览器相同规则生成`session key`

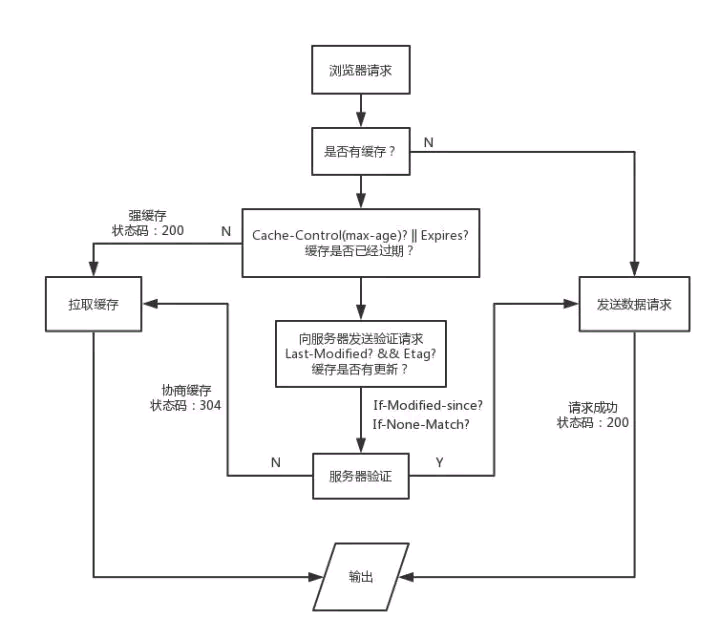
- 使用`session key`解密浏览器发来的握手消息，并验证Hash是否与浏览器发来的一致

- 使用`session key`加密一段握手消息，发送给浏览器

5. 浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，

之后所有的https通信数据将由之前浏览器生成的session key并利用对称加密算法进行加密

http的缓存



浏览器总是优先使用cache-control，如果没有cache-control才考虑Expires

Req中的If-Modified-Since 和res中的Last-Modified

为什么使用ETag呢？ 主要是为了解决Last-Modified 无法解决的一些问题。[·](http://caibaojian.com/http-cache.html)

1. 某些服务器不能精确得到文件的最后修改时间， 这样就无法通过最后修改时间来判断文件是否更新了。

2. 某些文件的修改非常频繁，在秒以下的时间内进行修改. Last-Modified只能精确到秒。

3. 一些文件的最后修改时间改变了，但是内容并未改变。 我们不希望客户端认为这个文件修改了。

If-None-Match 和etag比较

200 from memory chche 不访问服务器，直接读缓存，从内存中读取缓存。此时的数据时缓存到内存中的，当kill进程后，数据将不存在

200 from disk cache不访问服务器，直接读缓存，从磁盘中读取缓存，当kill进程时，数据还是存在。

浏览器会先从memory查，在从disk中查。（感觉都存了）

网络传输性能优化

1.缓存，nginx配置

etag on; //开启etag验证

location ~\* \.(gif|jpg|png|bmp)$ { expires 10d; } 这里是设置图片的过期时间为10天。

Koa2

koa-static-cache中间件

app.use(staticCache(path.join(\_\_dirname, 'public'), {

  maxAge: 365 \* 24 \* 60 \* 60

}))

2 压缩

webpack

1.new webpack.optimize.UglifyJsPlugin() w3 optimization.minimize w4 //js

2.HtmlWebpackPlugin //html

3ExtractTextPlugin //分离css，压缩

3图片资源优化

1.不要缩放图片

2.css sprite https://www.toptal.com/developers/css/sprite-generator

3使用字体图标 [www.iconfont.cn/](https://link.juejin.im/?target=https%3A%2F%2Flink.zhihu.com%2F%3Ftarget%3Dhttp%253A%2F%2Fwww.iconfont.cn%2F)

4.网络传输性能检测工具——Page Speed

5 cdn

6 页面渲染

1. 解析HTML，构建DOM树

2. 解析CSS，生成CSS规则树

3. 合并DOM树和CSS规则，生成render树

4. 布局render树（Layout/reflow），负责各元素尺寸、位置的计算

5. 绘制render树（paint），绘制页面像素信息

6. 浏览器会将各层的信息发送给GPU，GPU会将各层合成（composite），显示在屏幕上

7Css可视化模型

css中规定每个元素都有自己的盒子模型（相当于规定这个元素如何显示）

可视化模型就是把这些盒子按规则放到页面上

 CSS的可视化格式模型就是规定了浏览器在页面中如何处理文档树

7.1 包含块（containing Block）

一个元素的盒子的定位和尺寸，会与某一矩形框架有关，这个就是包含块

元素会为它的子孙创建包含块，所以元素的包含块与它的祖先元素样式有关系

特点

根元素没有父节点，它的包含块就是初始包含块

static和relative的包含块由它最近的块级，单元格或者行内块祖先元素的内容框（content）创建

fixed的包含块是当前可视窗口

absolute的包含块由它最近的static，relative，fixed创建

* 如果其祖先元素是行内元素，则包含块取决于其祖先元素的direction特性
* 如果祖先元素不是行内元素，那么包含块的区域应该是祖先元素的内边距边界

7.2 控制框（controlling Box）

块框：

块级元素会产生一个占据一整行的块框

块框同时也是一个块包含框（Containing Box），里面要么只包含块框，要么只包含行内框（不能混杂），如果块框内部有块级元素也有行内元素，那么行内元素会被匿名块框包围

如果一个块框在其中包含另外一个块框，那么我们强迫它只能包含块框，因此其它文本内容生成出来的都是匿名块框（而不是匿名行内框）

### BFC（Block Formatting Context）

FC是格式上下文，定义了内部的元素渲染规则

块级框对应BFC