前端训练营

模块一 前端基本功

林溪

第一课预习资料

Object

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects/Object

CSS Flex

https://www.ruanyifeng.com/blog/2015/07/flex-grammar.html

CSS grid

https://www.ruanyifeng.com/blog/2019/03/grid-layout-tutorial.html

一、JavaScript基础

二、BOM

三、DOM

四、CSS

五、底层技术

六、正则表达式

七、数据结构与算法基础

一、JavaScript基础

JavaScript

基本类型

undefined | null | boolean | number | string

object | symbol

引用类型

Boolean | Number | String | Symbol | Object

Array | Map | Set

基本类型

typeof("aa") "string"

typeof(null) "object"

typeof(undefined) "undefined"

typeof([]) "object"

typeof(object) "undefined"

typeof(Object) "function"

typeof(Symbol("aa")) "symbol"

typeof(symbol) "undefined"

引用类型

new String("aa") instanceof String

[] instanceof Array

String

String("haha") === "haha"

new String("haha") === "haha"

new String("haha") === new String("haha")

```
String.prototype.haha = function(){
    console.log(this.toString());
}
```

"hehe".haha() //hehe

new String("hoho").haha() //hoho

Boolean

```
Boolean(false) === false
```

```
Boolean(false) === Boolean(false)
```

```
new Boolean(false) === new Boolean(false)
```

Number

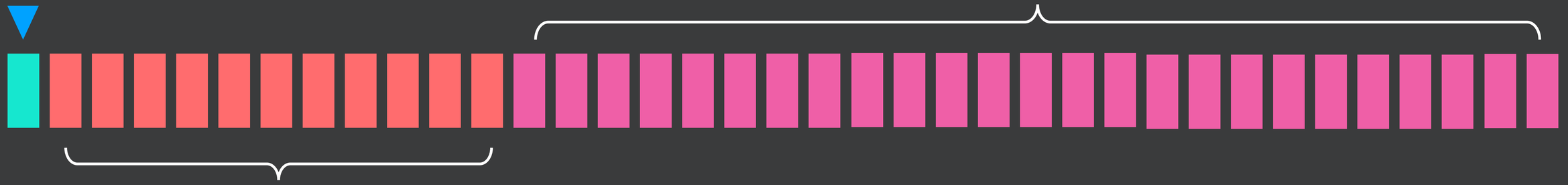
双精度浮点数

IEEE 754

$$V = (-1)^S \times 2^{E-1023} \times (M+1)$$

符号

尾数M（52）



指数E（11位） 0-1022表示为负，1024-2047表示为正

Number.MIN_VALUE	Math.pow(2, -51) * Math.pow(2, 0-1023)
Number.MAX_VALUE	Math.pow(2, 2047-1023) - 1

Number

安全数　　连续

Number.MAX_SAFE_INTEGER　　Number.MIN_SAFE_INTEGER

Math.pow(2, 53)-1　　-Math.pow(2, 53)+1

```
Math.pow(2, 53) === Math.pow(2, 53) + 1
Math.pow(2, 1024) === Infinity
```

小数转二进制

4.12

整数部分 4

除2取余

4 / 2 = 2，余数0
2 / 2 = 1，余数0
1 / 2 = 0，余数1
除数为0，结束

[0,0,1] 逆转 [1,0,0]

100

小数转二进制

4.12 小数部分 0.12 乘2取整

0.12 * 2 = 0.24，取整0
0.24 * 2 = 0.48，取整0
0.48 * 2 = 0.96，取整0
0.96 * 2 = 1.92，取整1，取小数继续乘
0.92 * 2 = 1.84，取整1，取小数继续乘
0.84 * 2 = 1.68，取整1，取小数继续乘
0.68 * 2 = 1.36，取整1，取小数继续乘
0.36 * 2 = 0.72，取整0，取小数继续乘

0.36 * 2 = 0.72，取整0
0.72 * 2 = 1.44，取整1
0.44 * 2 = 0.88，取整0
0.88 * 2 = 1.76，取整1，取小数继续乘
0.76 * 2 = 1.52，取整1，取小数继续乘
0.52 * 2 = 1.04，取整1，取小数继续乘
0.04 * 2 = 0.08，取整0，取小数继续乘
0.08 * 2 = 0.16，取整0，取小数继续乘

0.0001111001011100。。。

位数精度足够或者小数部分为0结束

二进制转小数

100. 0.0001111001011100 。。

整数部分

100

小数部分

0. 0001111001011100

1* 2^2 + 0 * 2^1 + 0 * 2^0

4

1 * 2^-4 + 1* 2^-5 + 1*2^-6 + 1* 2^-7

0.1171875 = 0.0625 + 0.03125 + 0.015625 + 0.0078125

结果 4.1171875

二进制存储

100.0001111001011100 。。

1.00000011110010111100 * 2^2

符号位+指数位+小数部分 (阶数)

符号位: 0

指数位: 1023 +2 =1025
二进制： 10000000001

小数部分: 00000011110010111100..000 补0，补52位

0 + 10000000001 + 00000011110010111100...0

十进制小数据部分转二进制

因为计算机最多只能存 52 位，除了 0.5，其余在转换过程中都会进行截断，于是就出现了精度缺失

Number

0.1 + 0.2 === 0.3  浮点运算精度缺失

0.1 0.0001100110011001100110011001100110011001100110011001101

0.2 0.0011001100110011001100110011001100110011001100110011001101

0.0100110011001100110011001100110011001100110011001100111

0.30000000000000004

parseFloat((0.1+0.2).toPrecision(12)) === 0.3

https://github.com/MikeMcl/bignumber.js/

方法

极客时间 训练营

- isInteger()
- isSafeInteger()
- toExponential()
- toFixed()
- toPrecision()

toFixed 与 toPrecision 区别

toPrecision 是处理精度，精度是从左至右第一个不为0的数开始数起

toFixed 是小数点后指定位数取整，从小数点开始数起

1.005.toFixed(2) === 1.00

0.1.toPrecision(21) = 0.100000000000000005551

进制 10 0b101 0755 0xA

1.005.toFixed(2) === 1.00

```
function toFixed(number, precision) {
    var str = number + ''
    var len = str.length
    var last = str.substring(len - 1, len)
    if (last == '5') {
        last = '6'
        str = str.substring(0, len - 1) + last
        return (str - 0).toFixed(precision)
    } else {
        return number.toFixed(precision)
    }
}
```

先扩大再缩小

```
function toFixed(num, s) {
    var times = Math.pow(10, s)
    // 0.5 为了舍入
    var des = num * times + 0.5
    // 去除小数
    des = parseInt(des, 10) / times
    return des + ''
}
```

BigInt

提供了一种方法来表示大于 Math.pow(2, 53)-1的整数

```
Const big = BigInt(Number.MAX_SAFE_INTEGER);
// 9007199254740991n

typeof 1n === 'bigint';

0n === 0   // false
```

作业

进制转换，将10进制浮点数转换成64进制

Symbol

```
let a = Symbol("haha")
console.log(a.toString());
```

```
Symbol( “bar” ) === Symbol("bar");

//false
```

```
let a = Symbol.for("haha")
console.log(a.toString());
```

```
Symbol.for("bar") === Symbol.for("bar");

//true
```

Object

Object.assign()	Object.create()
Object.freeze()	Object.seal()
Object.defineProperty()	Object.deleteProperty()
Object.getOwnPropertyNames()	Object.is()
Object.isExtensible()	Object.isFrozen()
Object.isSealed()	Object.values()

原型

原型编程语言

行为重用是通过复制已经存在的原型对象的实现的

类编程语言

行为重用是通过继承实现的

__proto__

隐式原型，指向构造该对象的构造函数（constructor）的原型（prototype）

prototype

显式原型，指向函数的原型对象

1994年，网景公司（Netscape）发布了Navigator浏览器0.9版，这个版本的浏览器只能用来浏览，不具备与访问者互动的能力，网景公司急需一种网页脚本语言，使得浏览器可以与网页互动。

工程师Brendan Eich负责开发这种新语言，他觉得，没必要设计得很复杂，这种语言只要能够完成一些简单操作就够了，比如判断用户有没有填写表单。

1994年正是面向对象编程（object-oriented programming）最兴盛的时期，C++是当时最流行的语言，而Java语言的1.0版即将于第二年推出，Sun公司正在大肆造势。Brendan Eich无疑受到了影响，Javascript里面所有的数据类型都是对象（object）

如果真的是一种简易的脚本语言，其实不需要有"继承"机制。但是，Javascript里面都是对象，必须有一种机制，将所有对象联系起来。所以，Brendan Eich最后还是设计了"继承"。

考虑不引入类，因为一旦有了"类"，Javascript就是一种完整的面向对象编程语言了，这好像有点太正式了，而且增加了初学者的入门难度（在 ES2015/ES6 中引入了 class 关键字，但那只是语法糖，JavaScript 仍然是基于原型的）

原型与构造函数

Brendan Eich把new命令引入了Javascript，用来从原型对象生成一个实例对象，C++和Java使用new命令时，都会调用"类"的构造函数（constructor）。他就做了一个简化的设计，在Javascript语言中，new命令后面跟的不是类，而是构造函数。

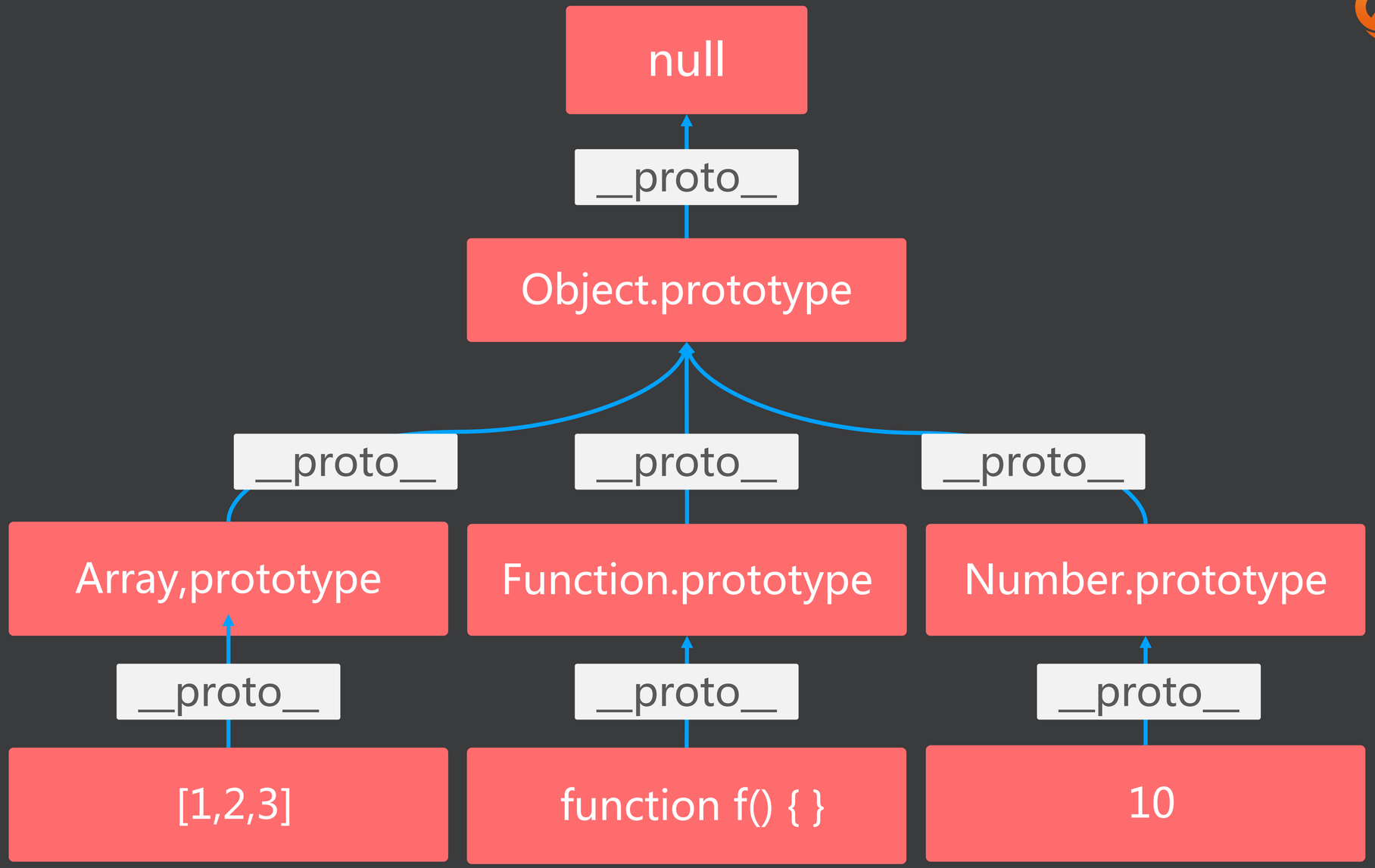
new运算符的缺点

无法共享属性和方法，每一个实例对象，都有自己的属性和方法的副本。这不仅无法做到数据共享，也是极大的资源浪费。

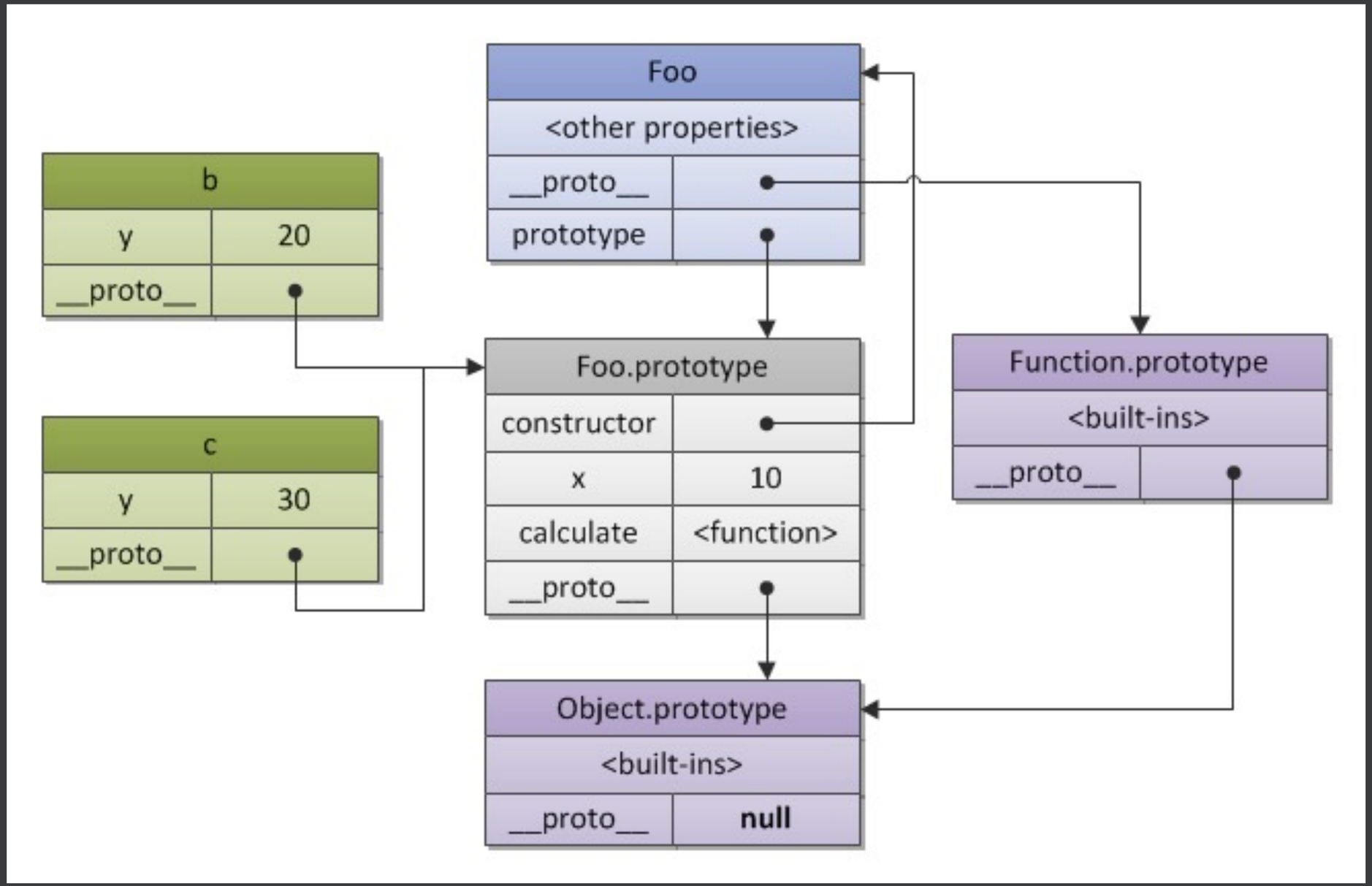
prototype属性的引入

为了解决new的缺点，Brendan Eich决定为构造函数设置一个prototype属性。
这个属性包含一个对象（以下简称"prototype对象"），所有实例对象需要共享的属性和方法，都放在这个对象里面；那些不需要共享的属性和方法，就放在构造函数里面。
实例对象一旦创建，将自动引用prototype对象的属性和方法。也就是说，实例对象的属性和方法，分成两种，一种是本地的，另一种是引用的。

原型是为了共享多个对象之间的一些共有特性（属性或方法）



Object.prototype是原型链的顶端，所有对象从它继承了包括toString等等方法和属性。


Foo
<other properties>
__proto__
prototype
b
y
20
__proto__
c
y
30
__proto__
Foo.prototype
constructor
x
10
calculate
<function>
__proto__
Function.prototype
<built-ins>
__proto__
Object.prototype
<built-ins>
__proto__
null

总结

prototype是函数的属性，指向该构造函数的原型对象
__proto__是对象的属性，指向该对象的构造函数的原型对象

函数也是对象，函数也有__proto__，是构造函数Function()的实例对象
所有构造函数的__proto__属性都指向Function构造函数的原型对象

prototype放公用属性方法
__proto__告诉你可以去那里找公用属性和方法

对象创建方法

字面量创建的对象

```
var a = {};
a.__proto__ === a.constructor.prototype;
a.__proto__ === Object.prototype;
Object.getPrototypeOf(a) === Object.prototype
```

```
var b = 'b';
b.__proto__ === b.constructor.prototype;
b.__proto__ === String.prototype;
Object.getPrototypeOf(a) === String.prototype;
```

对象创建方法

构造器方式

```
var a = new Array();
```

```
a.__proto__ === Array.prototype;
```

```
a.__proto__ === a.constructor.prototype;
```

```
function Person(){}
var p = new Person();
```

```
p.__proto__ === p.constructor.prototype;
```

```
p.__proto__ === Person.prototype;
```

对象创建方法

Object.create

```
var a = {name: "LIN"};
var b= Object.create(a);
```

```
b.__proto__ === b.constructor.prototype;
```

```
b.__proto__ === Object.prototype;
```

```
b.__proto__ === a
```

```
Object.create =  function (o) {
    var F = function () {};
    F.prototype = o;
    return new F();
};
```

构造函数缺失

instanceof

isPrototypeOf

```
a.isPrototypeOf(b);
```

原型链遍历

JavaScript引擎会先在对象的属性里去寻找属性，如果它找不到,就会再进一步往该对象的__proto__里面去寻找，直到某个对象的原型为null为止

hasOwnProperty()
Object.keys()
JavaScript 中唯一2个处理属性并且不会遍历原型链的方法。

Array

isArray()	判断是否是数组
sort()	V8 引擎 sort 函数只给出了两种排序 InsertionSort 和 QuickSort， 数量小于10的数组使用 InsertionSort， 比10大的数组则使用 QuickSort。
队列 & 栈	队列 和 栈

Array

练习

ArraySort

ArraySort 与 QuickSort 性能对比

Map

键可以是任意值

key 是有序的

```
const data = {};
const element = document.getElementById('myDiv');
data[element] = 'metadata';
data['[object HTMLDivElement] ']    // "metadata"

const m = new Map();
const o = {p: 'Hello World'  };
m.set(o, 'content')
```

Map

Map和Object区别

	key	元素顺序	数据访问	迭代	Size	性能
Map	复杂类型	有序	.get()	是	是	好
Object	简单类型	无序	[]	否	否	差

WeakMap

WeakMap只接受对象作为键名（null除外），
不接受其他类型的值作为键名。

WeakMap的键名所指向的对象，不计入垃圾回收机制

不能遍历

```
new WeakMap().set(1， 2)
```

```
new WeakMap().set(null， 2)
```

```
new WeakMap().set(Symbol()， 2)
```

Set

去重

```
[...new Set([1,2,3,3,4,4,4,4])]
```

```
Array.from(new Set('ababbc'))
```

- add()
- delete()
- has()
- clear()
- keys()
- values()
- entries()
- forEach()

WeakSet

WeakSet 的成员只能是对象，而不能是其他类型的值

WeakSet 不能遍历，是因为成员都是弱引用，随时可能消失

WeakSet 中的对象都是弱引用，即垃圾回收机制不考虑 WeakSet 对该对象的引用

```
new WeakSet().add(1)
```

二、BOM

location

hash 设置或返回从井号 (#) 开始的 URL
host 设置或返回主机名和当前 URL 的端口号
hostname 设置或返回当前 URL 的主机名
href 设置或返回完整的 URL
pathname 设置或返回当前 URL 的路径部分
port 设置或返回当前 URL 的端口号
protocol 设置或返回当前 URL 的协议
search 设置或返回从问号 (?) 开始的 URL

assign()

reload(true/false)

replace()

location.assign("http://baidu.com")
location.href = "http://baidu.com"

location.replace("http://baidu.com")
不会留history记录

navigation

userAgent

Mozilla/5.0 (Linux; Android 10; ELS-AN00 Build/HUAWEIELS-AN00; wv) AppleWebKit/537.36 (KHTML, like Gecko) Version/4.0 Chrome/77.0.3865.120 MQQBrowser/6.2 TBS/045709 Mobile Safari/537.36 MMWEBID/7800 MicroMessenger/8.0.6.1900(0x28000653) Process/tools WeChat/arm64 Weixin NetType/WIFI Language/zh_CN ABI/arm64

通过useragent获取手机品牌型号

https://github.com/fex-team/ua-device

navigation

clipboard

获取复制内容

设置剪贴板内容

如何粘贴图片

ClipboardData

getData

setData

navigation

clipboard

粘贴图片

navigation

keyboard

方法

getLayoutMap()

lock()

unlock()

全屏模式下可以监听ctrl+n，ctrl+w，esc等快捷键

navigation

keyboard

Keyboard API

navigation

geolocation

H5地理定位

Navigation.Geolocation.getCurrentPosition()

IP定位

获取IP

https://www.taobao.com/help/getip.php

https://pv.sohu.com/cityjson?ie=utf-8

IP解析

https://api.map.baidu.com/location/ip

navigation

geolocation

H5混合定位

navigation

mediaDevices

WebRTC Real-Time Communications

实时通讯技术，它允许网络应用或者站点，在不借助中间媒介的情况下，建立浏览器之间点对点（Peer-to-Peer）的连接，实现视频流和（或）音频流或者其他任意数据的传输

P2P

navigation

mediaDevices

Web P2P视频Demo

navigation

serviceWorker

基于web worker，独立于JavaScript主线程的独立线程，不会堵塞主线程

在web worker的基础上增加了离线缓存的能力

充当服务器与浏览器之间的代理服务器，可以拦截全站的请求，
并作出相应的动作

支持推送

可以控制管理缓存的内容以及版本

screen

- orientation
  - landscape-primary
  - landscape-secondary
  - portrait-secondary
  - portrait-primary
- availLeft
- availWidth
- width

history

- length
- scrollRestoration
- state

- back()
- forward()
- go()
- pushState()
- replaceState()

performance

性能API

重定向时间	redirectEnd - redirectStart
DNS查询耗时	domainLookupEnd - domainLookupStart
DNS缓存耗时	domainLookupEnd - fetchStart
TCP连接耗时	connectEnd - connectStart
Https连接耗时	
onload时间	loadEventEnd - navigationStart
domready时间	domContentLoadedEventEnd - navigationStart
TTFB 首字节时间	responseEnd - requestStart
下载时间	responseEnd - responseStart
解析dom树耗时	domComplete- domInteractive
白屏时间	domloadng - navigationStart

window

```
window.btoa("hello")
window.atob("aGVsbG8=")
```

Base64编码

```
window.createImageBitmap
```

Sprite图合并

```
window.getComputedStyle(element)
```

```
window.getSelection()
```

```
window.postMessage()
```

```
window.queueMicrotask()
```

生成随机数

```
var arr = new Uint32Array(2);
window.crypto.getRandomValues(arr);
```

```
Math.random()
```

Uint32Array(2) [1386750091, 412288628]

密码学安全的随机数

密码学不安全的随机数

伪随机数的评价标准:
1.可以通过所有随机性统计检验
2.不可预测
3.不能可靠地重复产生

$$X_{n+1} = (aX_n + b) \bmod c$$

线性拟合

Proxy

Proxy 对象用于创建一个对象的代理，从而实现基本操作的拦截和自定义（如属性查找、赋值、枚举、函数调用等）

- Reflect.apply
- Reflect.get
- Reflect.construct
- Reflect.set
- Reflect.defineProperty
- Reflect.has
- Reflect.deleteProperty

Reflect

Reflect.apply

Reflect.construct

Reflect.defineProperty

Reflect.deleteProperty

Reflect.get

Reflect.set

Reflect.has

indexedDB

浏览器提供的本地数据库，支持事务、索引

存储空间

Cookie	4kb
LocalStorage	2.5m ~ 10mb
IndexedDB	可用磁盘空间的50%

特性

键值对储存
异步
支持事务
同源限制
储存空间大
支持二进制储存

indexedDB

练习

indexedDB Demo

三、DOM

尺寸和定位

尺寸

- style.width
- getComputedStyle
- offsetWidth
- clientWidth
- getBoudingClientRect

定位

- offsetTop
- offsetLeft

隐藏元素的宽度如何获取？

尺寸和定位

图片懒加载

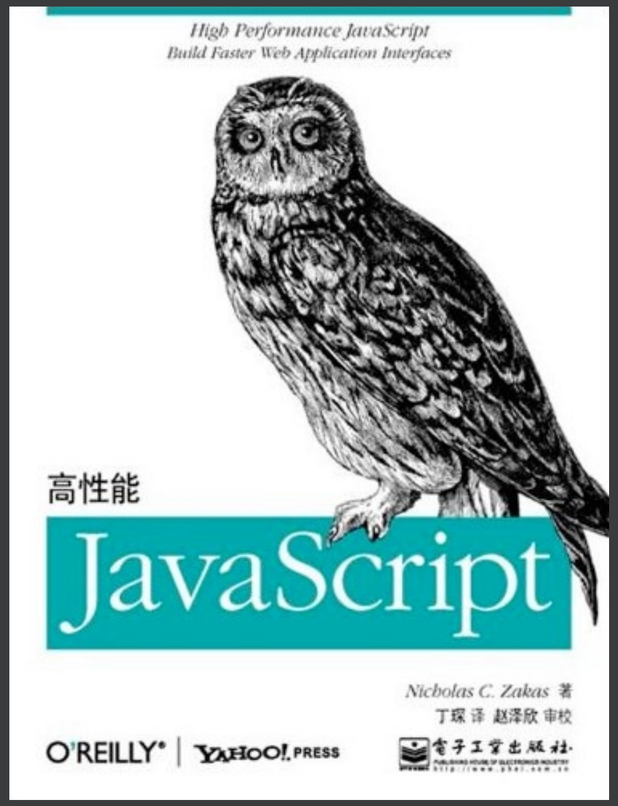
Fragment

```
document.createDocumentFragment()
```

避免频繁大量的dom操作，提升性能

Fragment上的元素有宽度吗？

高性能DOM编程



减少重排

批量修改DOM

事件委托
（Event Delegation）

引起重排的操作：

1. 浏览器窗口大小发生改变
2. 元素尺寸或位置发生改变
3. 元素内容变化（文字数量或图片大小等等）
4. 添加或者删除可见的DOM元素
5. 激活CSS伪类（例如：:hover）
6. 查询某些属性或调用某些方法

引起回流的属性和方法：
clientWidth、clientHeight、clientTop、clientLeft
offsetWidth、offsetHeight、offsetTop、offsetLeft
scrollWidth、scrollHeight、scrollTop、scrollLeft
scrollIntoView()、scrollIntoViewIffNeeded()
getComputedStyle()
getBoundingClientRect()
scrollTo()

优化：

1. 避免频繁操作样式
2. 避免频繁操作DOM（fragment，Vitual Dom）
3. 避免频繁读取引发reflow的属性
4. 创建图层（transfer，will-change）
5. 不要使用table布局

四、CSS

Flex

容器

- flex-direction
- flex-wrap
- flex-flow
- justify-content
- align-items
- align-content

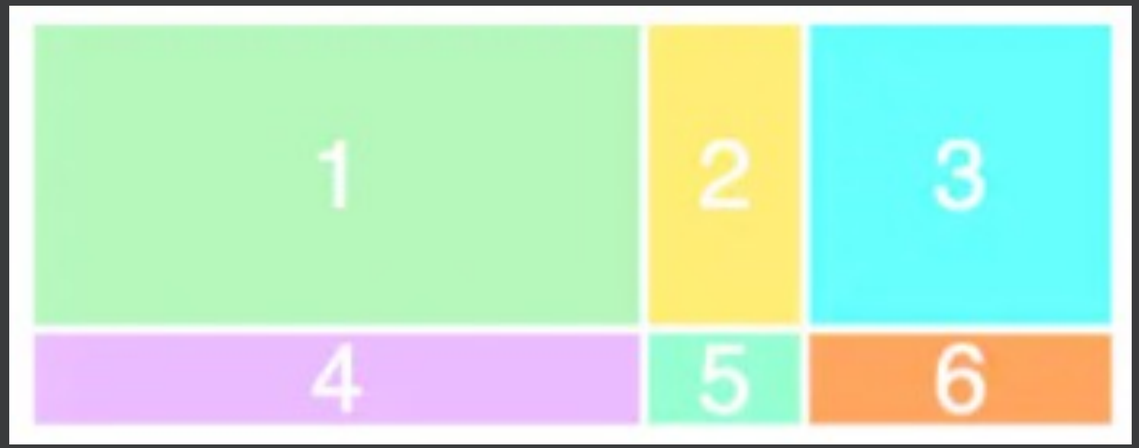
项目

- order
- flex-grow
- flex-shrink
- flex-basis
- flex
- align-self

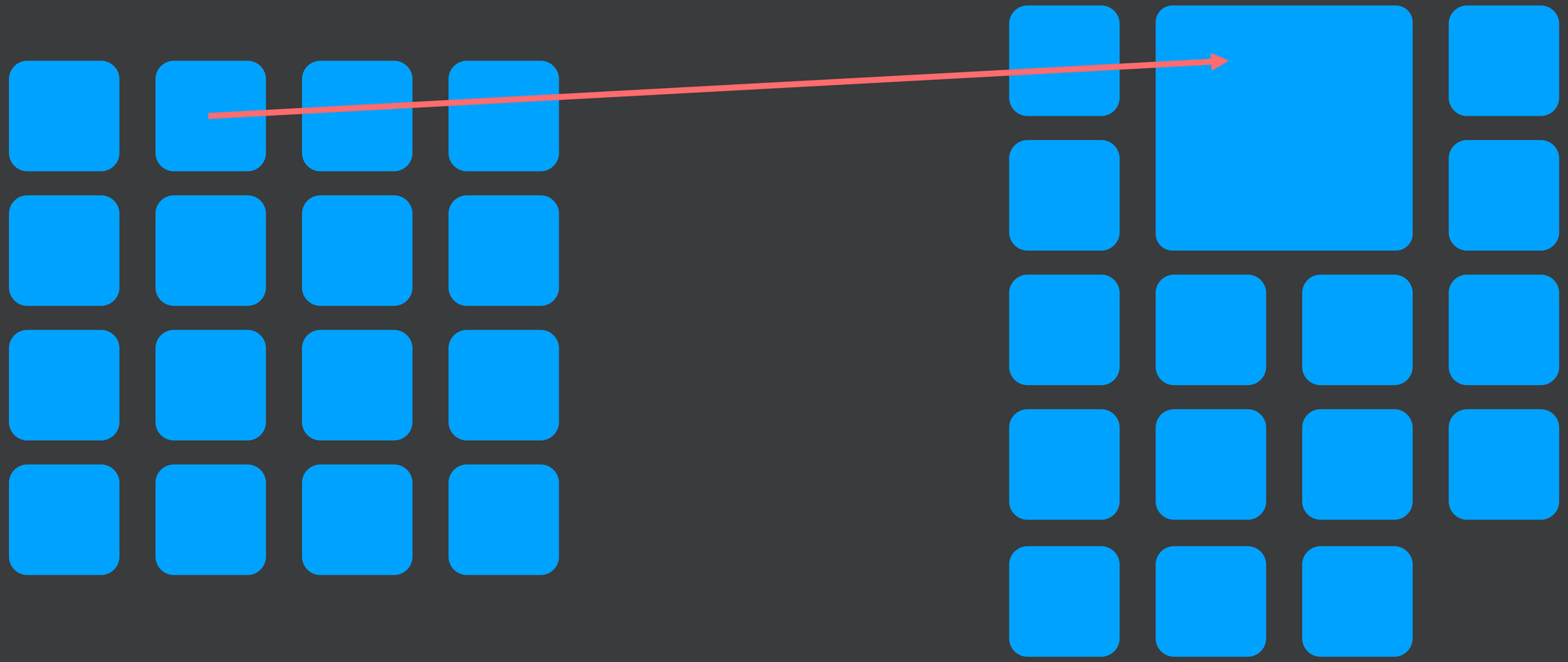
Grid

两维布局

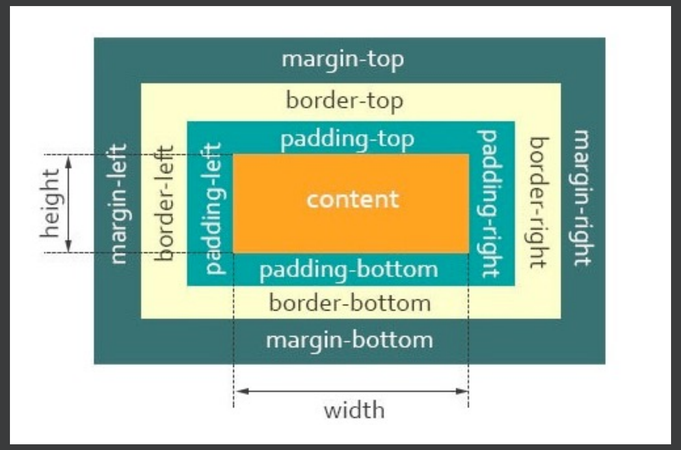
```
.wrapper {
    display: grid;
    grid-template-columns: 200px 50px 100px;
    grid-template-rows: 100px 30px;
}
```



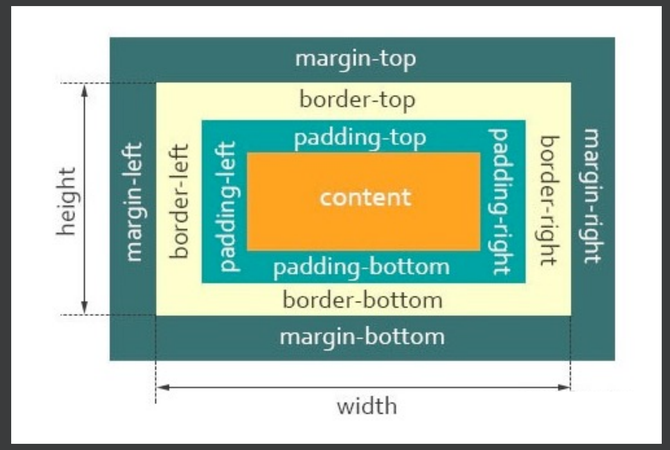
练习



盒模型



W3C 标准盒模型



怪异盒模型

box-sizing: content-box

box-sizing: border-box

外边距合并

外边距合并只针对块级元素
而且是顶部或底部的外边距

BFC

块格式化上下文，解决外边距重合

浮动元素：float 除 none 以外的

绝对定位元素：position (absolute、fixed)

display 为 inline-block、table-cells、flex

overflow 除了 visible 以外的值 (hidden、auto、scroll

预处理器技术

变量，嵌套，混合（mixin），继承（extend），运算，函数

scss	less	stylus
Ruby	Javascript	缩进式风格

功能更强大

安装较麻烦

激进，简练

预处理器技术

多主题

变量定义

$--color-primary: #409EFF;

```
/* Color
-------------------------- */
/// color|1|Brand Color|0
$--color-primary: #409EFF !default;
/// color|1|Background Color|4
$--color-white: #FFFFFF !default;
/// color|1|Background Color|4
$--color-black: #000000 !default;
$--color-primary-light-1: mix($--color-white, $--color-primary, 10%)
!default; /* 53a8ff */
$--color-primary-light-2: mix($--color-white, $--color-primary, 20%)
!default; /* 66b1ff */
$--color-primary-light-3: mix($--color-white, $--color-primary, 30%)
!default; /* 79bbff */
$--color-primary-light-4: mix($--color-white, $--color-primary, 40%)
!default; /* 8cc5ff */
$--color-primary-light-5: mix($--color-white, $--color-primary, 50%)
!default; /* a0cfff */
$--color-primary-light-6: mix($--color-white, $--color-primary, 60%)
!default; /* b3d8ff */
$--color-primary-light-7: mix($--color-white, $--color-primary, 70%)
!default; /* c6e2ff */
$--color-primary-light-8: mix($--color-white, $--color-primary, 80%)
!default; /* d9ecff */
$--color-primary-light-9: mix($--color-white, $--color-primary, 90%)
```

预处理器技术

mixin

include

extend

```scss
@mixin button-size($padding-vertical, $padding-horizontal, $font-size,
$border-radius) {
  padding: $padding-vertical $padding-horizontal;
  font-size: $font-size;
  border-radius: $border-radius;
  &.is-round {
    padding: $padding-vertical $padding-horizontal;
  }
}
```

```scss
@include button-size($--button-padding-vertical,
$--button-padding-horizontal, $--button-font-size,
$--button-border-radius);
```

```scss
%size {
  width: 100%;
  height: 100%;
}

@include b(image) {
  @extend %size;
  position: relative;
  display: inline-block;
  overflow: hidden;
}
```

颜色

```
mix(#000, #fff, 10%)          //按比例权重混合颜色
adjust-hue(#f36,150deg)       //改变#f36颜色的色相值为150deg
lighten(#f36,50%)             //把#f36颜色亮度提高50%
darken(#f36,50%)              //把#f36颜色亮度降低50%
saturate(#f36,50%)            //把#f36颜色饱和度提高50%
desaturate(#f36,50%)          //把#f36颜色饱和度降低50%
opacify((#f36, 0.5)           //把颜色的透明度变成50%
grayscale(#f36)               //把#f36颜色变成灰色
```

Polyfill

```
#photofade{
    -webkit-animation: photoFade 12s infinite;
    -moz-animation:photoFade 12s infinite;
    -ms-animation: photoFade 12s infinite;
    animation: photoFade 12s infinite;
 }
```

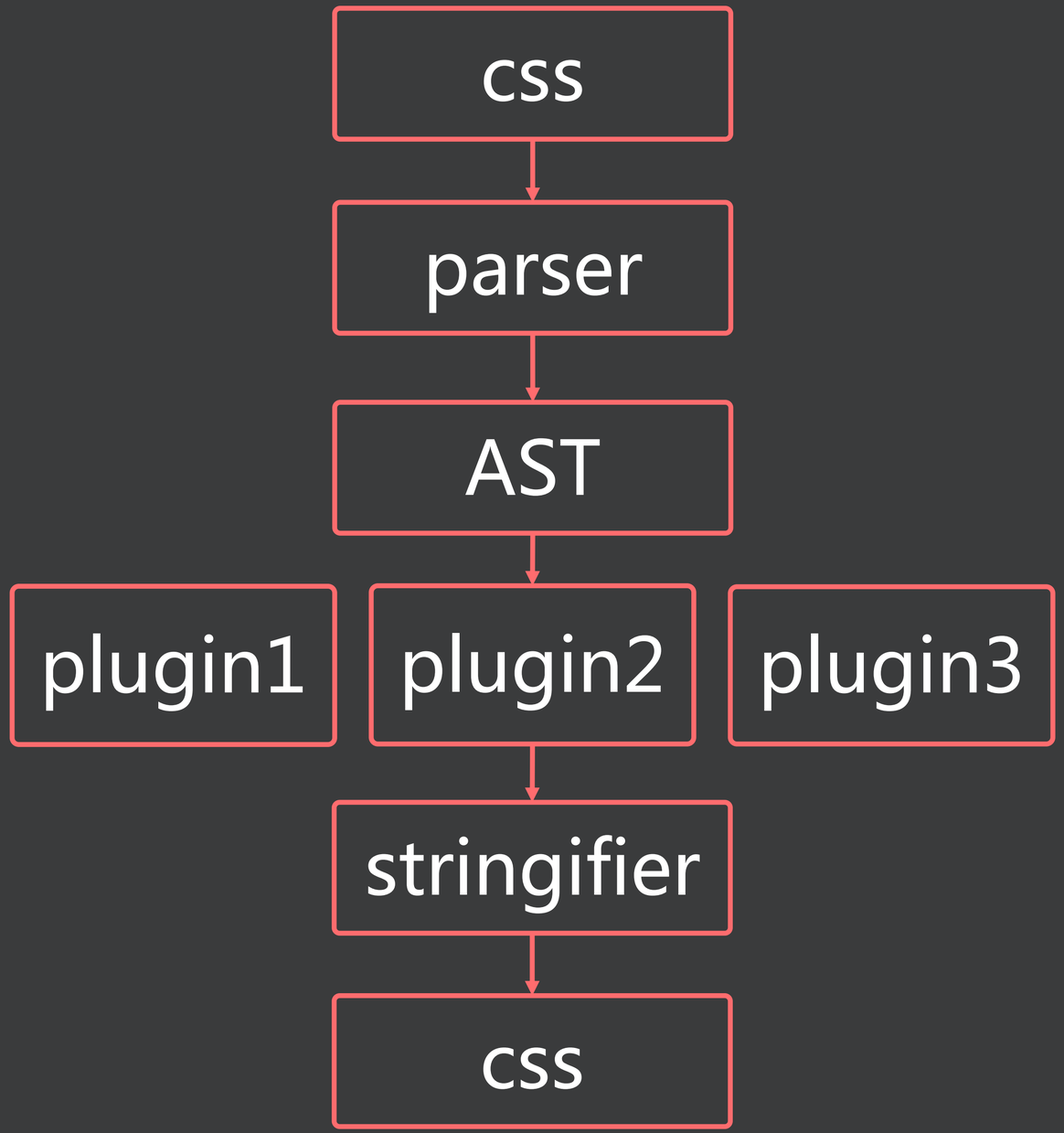
Autoprefixer

```
"browserslist": [
  "> 5%",
  "last 2 versions"
],
```

Postcss

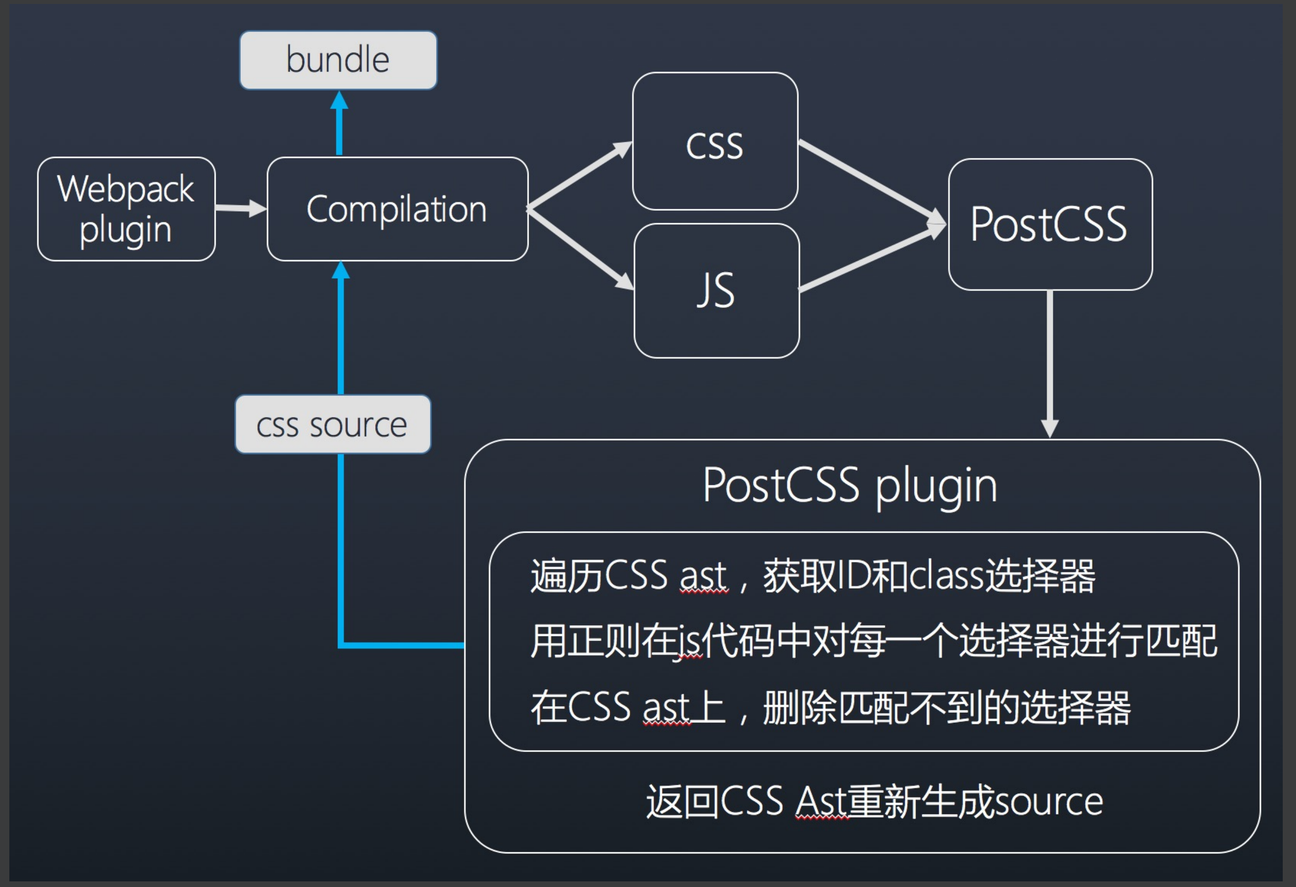
CSS后处理器框架

- Autoprefixer
- postcss-pxtorem



Postcss

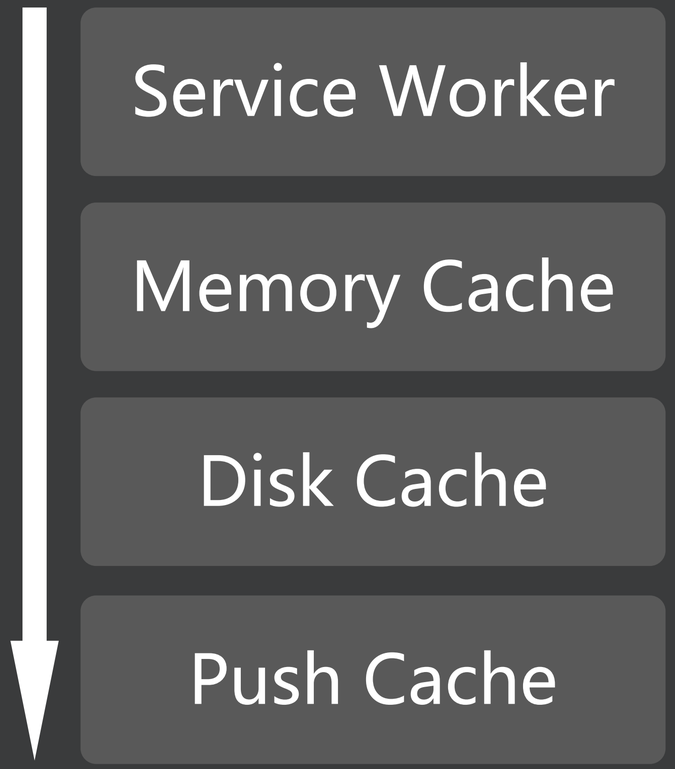
Postcss插件



五、底层技术

浏览器缓存

缓存顺序



缓存分类

强制缓存	Expires	Cache-Control
协商缓存	Last-modify	Etag

浏览器缓存

Service Worker

缓存

```
this.addEventListener('install', function (event) {

    event.waitUntil(
        caches.open('v1').then(function (cache) {
            return cache.addAll([
                './index.html'
            ]);
        })
    );
});
```

浏览器缓存

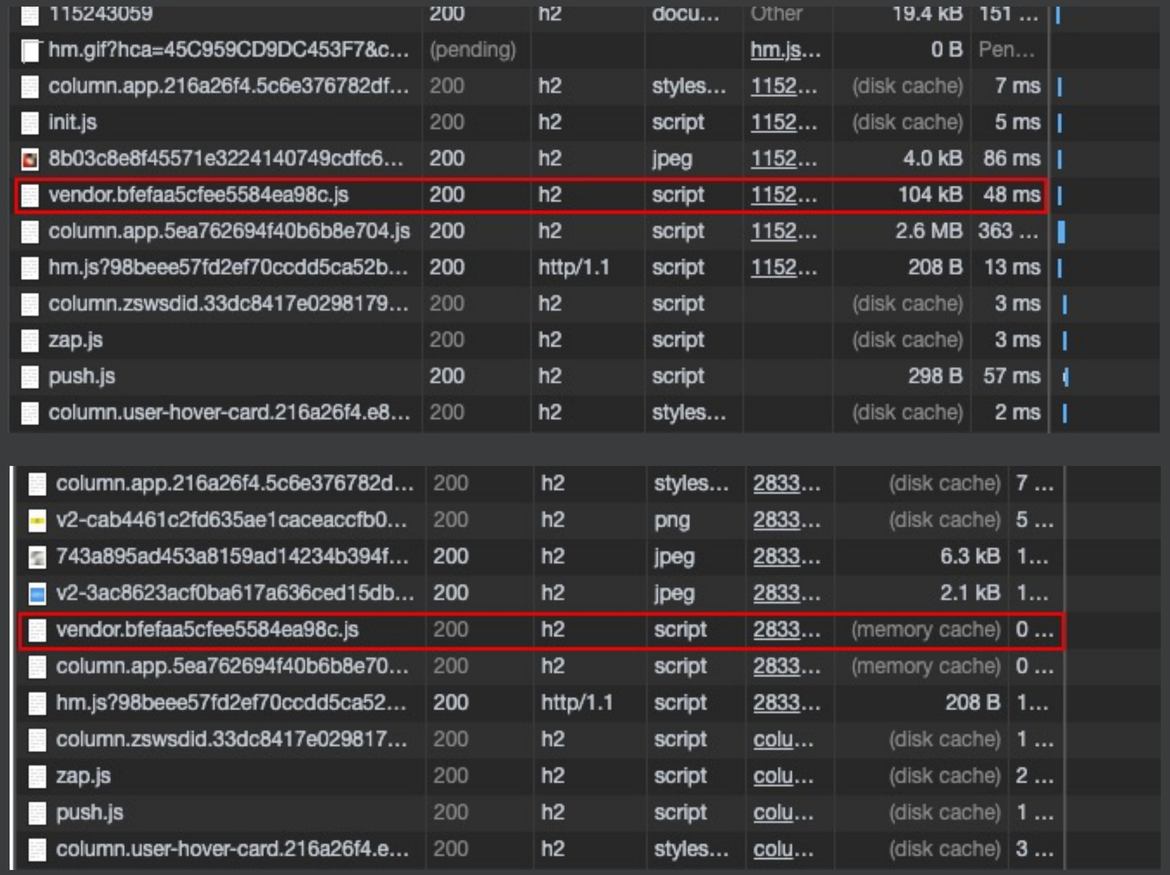
Memory Cache

缓存网页用到的子资源：图片 js css, 由浏览器自己控制

优化

webpack vendor公共资源打包

```
entry: {
  vendor: ["vue"],
  app: "./entry"
}
new CommonsChunkPlugin({
  name: "vendor",
  minChunks: Infinity
})
```



浏览器缓存

Disk Cache

Expires

版本：**HTTP/1.0**
来源：存在于服务端返回的响应头中
语法：**Expires: Wed, 22 Nov 2019 08:41:00 GMT**
缺点：服务器的时间和浏览器的时间可能并不一致导致失效

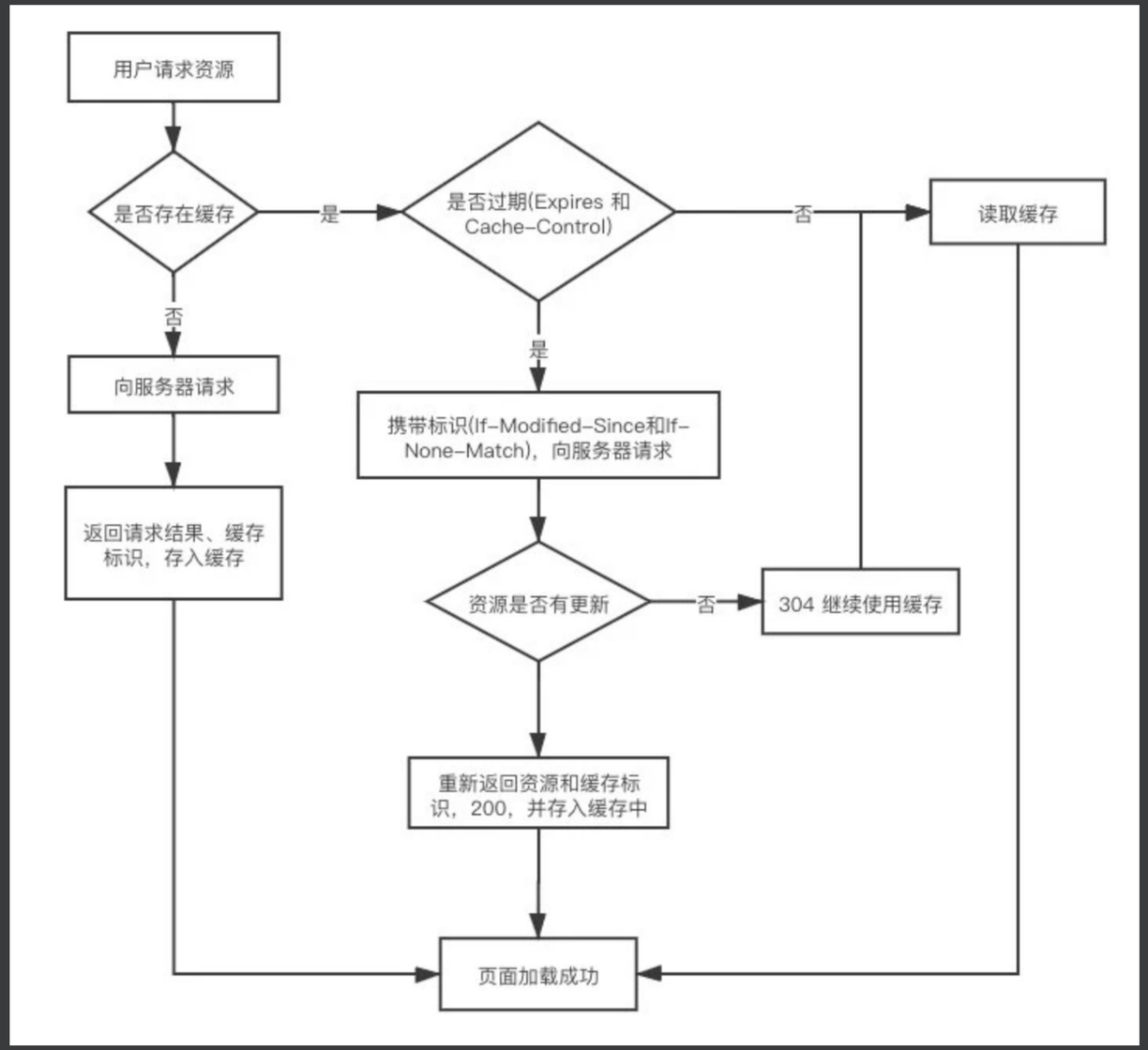
版本：HTTP/1.1
来源：响应头和请求头
语法：Cache-Control:max-age=3600
缺点：时间最终还是会失效

Cache-Control

public：所有内容都将被缓存
private：所有内容只有客户端可以缓存，默认值
no-cache：客户端缓存内容，但是是否使用缓存则需要经过协商缓存来验证决定
no-store：所有内容都不会被缓存，即不使用强制缓存，也不使用协商缓存
max-age=xxx (xxx is numeric)：缓存内容将在xxx秒后失效
must-revalidate: 强制浏览器严格遵守你设置的cache规则
proxy-revalidate: 强制proxy严格遵守你设置的cache规则

浏览器缓存

强制缓存



- Expires
- Cache-Control

浏览器缓存

协商缓存

服务端响应头

Last-Modified

文件最后修改时间

ETag

文件内容MD5
Last-modify + Size

浏览器请求头

If-Modified-Since

上次返回的Last-Modified

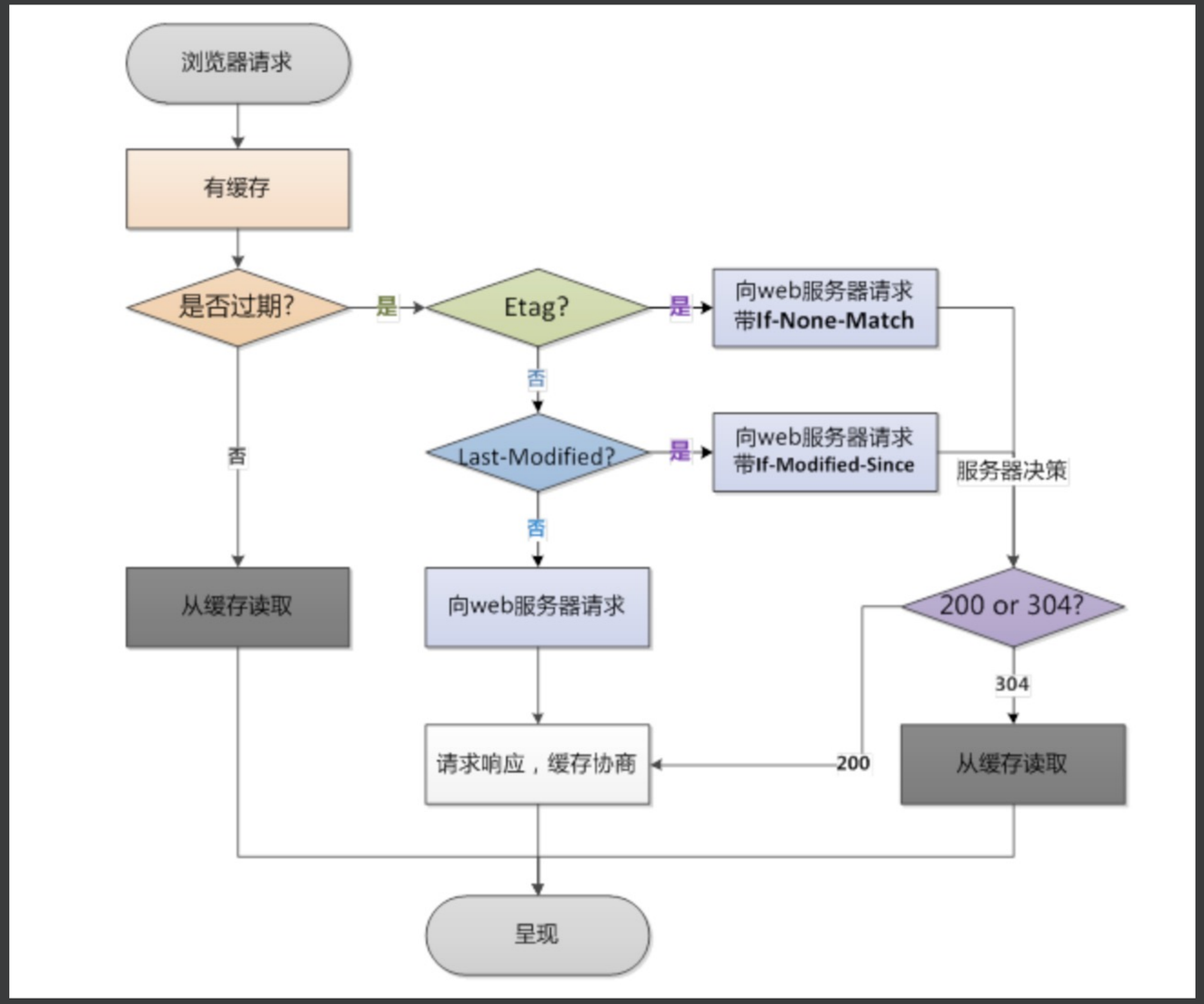
If-None-Match

上次返回的ETag

Last-modify **ETag**

浏览器缓存

协商缓存



Last-modify　ETag

服务器会优先验证ETag
一致的情况下才会继续比对
Last-Modified

浏览器缓存

Push Cache

Push Cache（推送缓存）是 HTTP/2 中的内容，当以上三种缓存都没有命中时，它才会被使用。它只在会话（Session）中存在，一旦会话结束就被释放，并且缓存时间也很短暂，在Chrome浏览器中只有5分钟左右，同时它也并非严格执行HTTP头中的缓存指令

一旦连接被关闭，Push Cache 就被释放
多个页面可以使用同一个HTTP/2的连接，也就可以使用同一个Push Cache。这主要还是依赖浏览器的实现而定，出于对性能的考虑，有的浏览器会对相同域名但不同的tab标签使用同一个HTTP连接。
Push Cache 中的缓存只能被使用一次

浏览器缓存

网络

影响网络传输效率的因素：

- TCP连接耗时
- 浏览器域名并发限制
- 数据传输冗余

优化

- 减少HTTP请求
- Cookie
- HTTP/2

HTTP/2

二进制传输	HTTP/1.x 纯文本报文，HTTP/2 二进制
Header压缩	HPACK算法
多路复用	一个域名，一个连接，双向数据流
多流并行	并行交错地发送多个请求/响应， 请求/响应之间互不影响
Server Push	主动把JS和CSS文件推送给客户端， 减少等待的延迟

浏览器缓存

网络

HTTP/2 缺点

TCP + TLS 建立连接，增加两个握手延迟

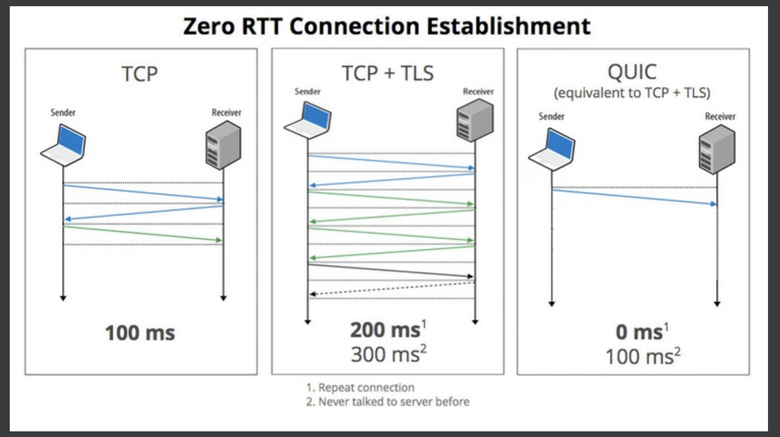
TCP丢包重传，阻塞所有请求，还不如HTTP/1.1

HTTP/3

基于UDP的QUIC协议，快速建立连接

多个stream，相互无依赖，解决TCP行头阻塞

更好的错误处理能力，向前纠错机制



浏览器渲染

浏览器

主进程

插件进程1

GPU进程

渲染进程1

插件进程n

渲染进程n

渲染进程

GUI
渲染线程

JS
引擎线程

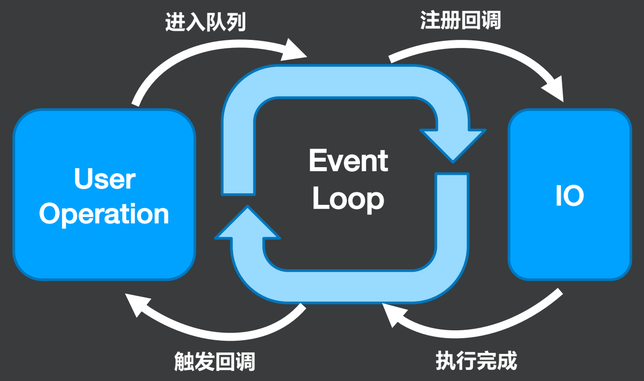
事件
触发线程

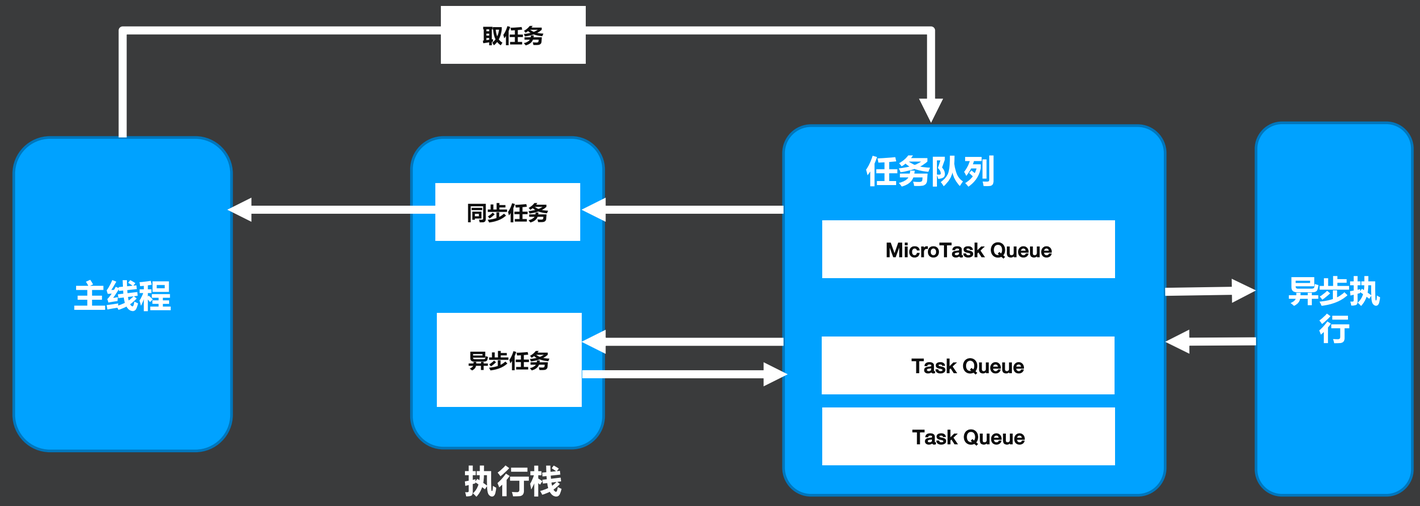
定时器
触发线程

异步HTTP
请求线程

浏览器渲染

Event Loop & 任务队列





浏览器渲染

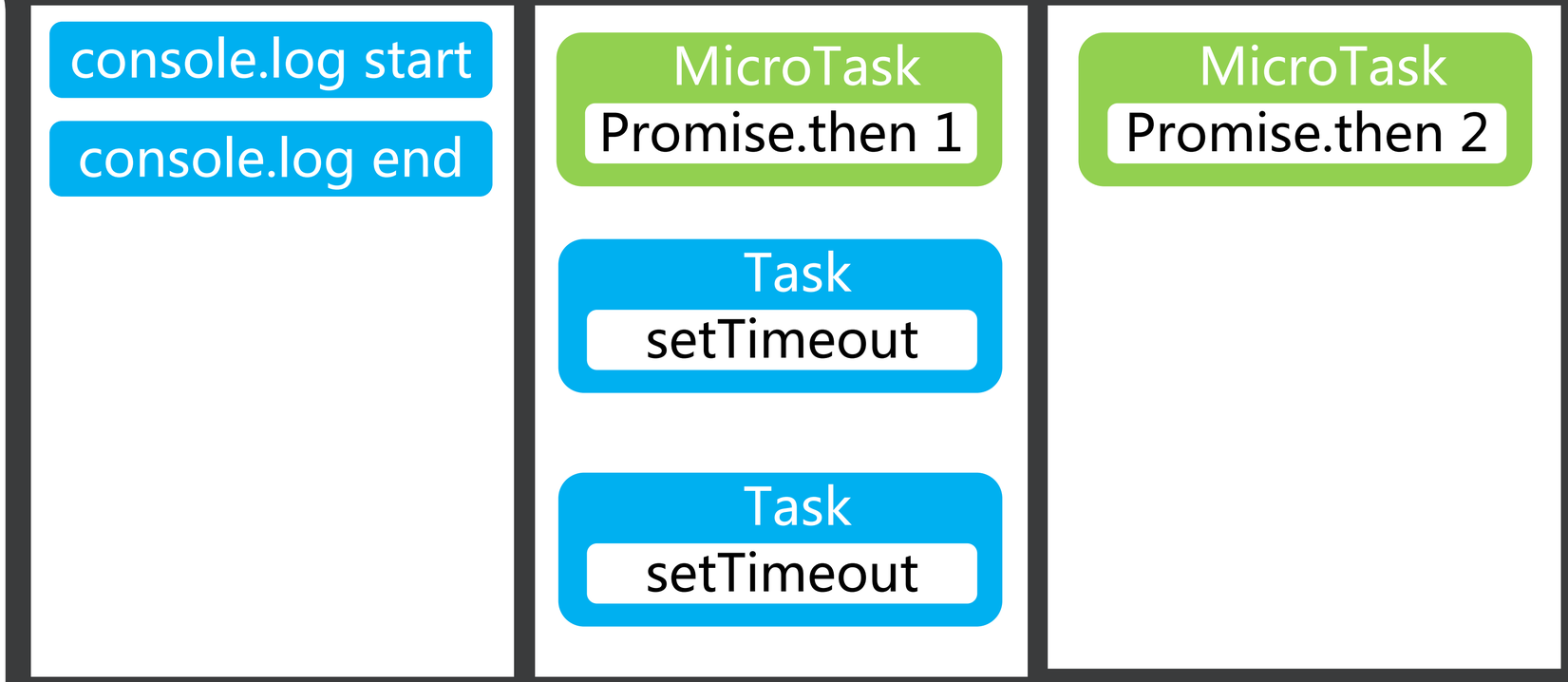
```
console.log('script start');

setTimeout(function(){
    console.log('setTimeout');
}, 0);

Promise.resolve().then(function() {
  console.log('promise1');
}).then(function() {
  console.log('promise2');
});

console.log('script end');
```

任务队列



浏览器渲染

任务队列

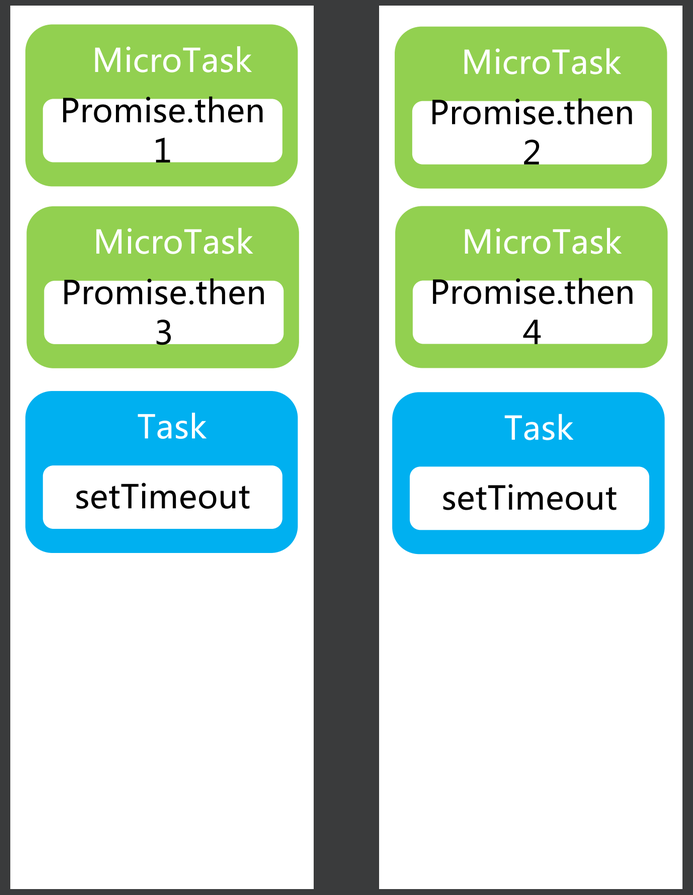
```
console.log('script start');

Promise.resolve().then(function() {
  console.log('promise1');
}).then(function() {
  console.log('promise2');
});

setTimeout(function() {
  console.log('setTimeout');
}, 0);

Promise.resolve().then(function() {
  console.log('promise3');
}).then(function() {
  console.log('promise4');
});
console.log('script end');
```

任务队列 Task Queue



任务队列

```
while (eventLoop.waitForTask()) {
  const taskQueue = eventLoop.selectTaskQueue()
  if (taskQueue.hasNextTask()) {
    taskQueue.processNextTask()
  }

  const microtaskQueue = eventLoop.microTaskQueue
  while (microtaskQueue.hasNextMicrotask()) {
    microtaskQueue.processNextMicrotask()
  }
}
```

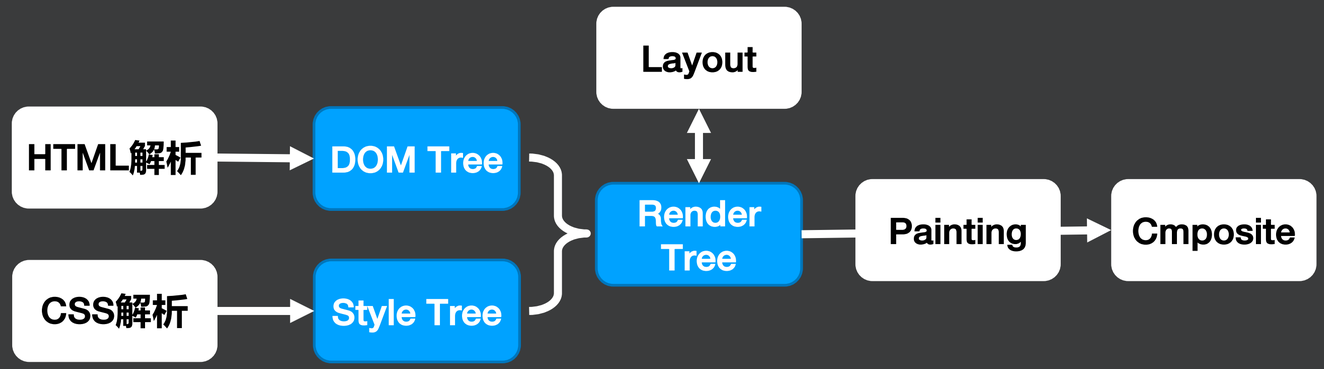
1. 从任务队列中选择任务
2. 执行任务
3. 执行微任务队列任务，只对队列为空
4. 重复1

Micro Task

#	浏览器	Node
process.nextTick	❌	✅
MutationObserver	✅	❌
Promise.then catch finally	✅	✅

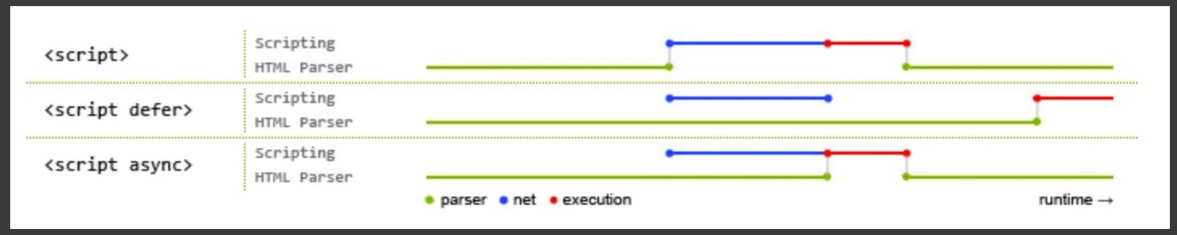
浏览器渲染

HTML渲染



遇到<script>，如果没有async和defer，停止解析HTML，加载执行JS代码

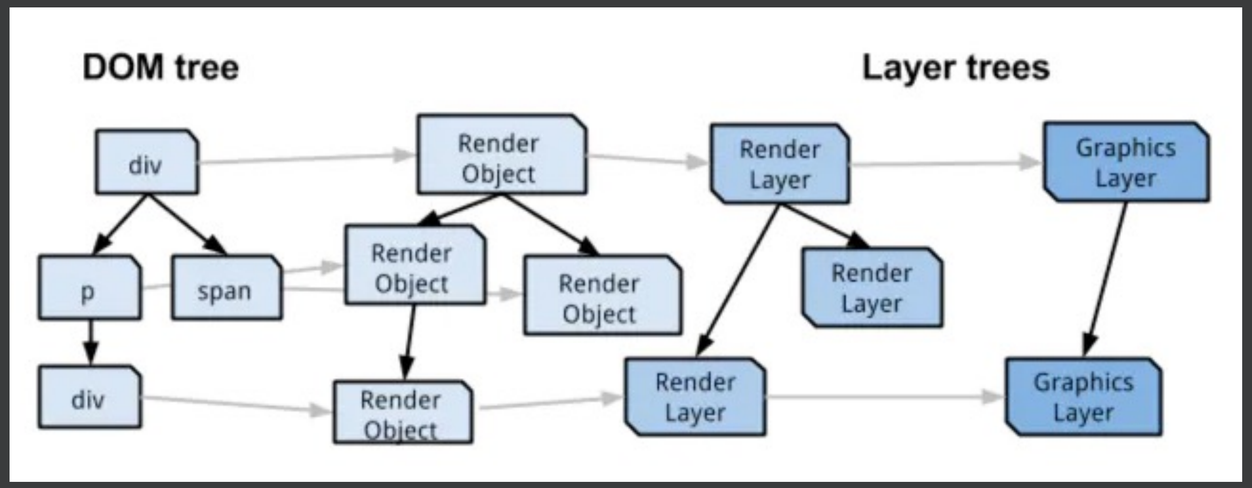
```
<script async src="http://www.baidu.com/a.js" />
<script defer src="http://www.baidu.com/a.js" />
```



async无序　　defer有序，DOMContentLoaded 之前执行

浏览器渲染

HTML渲染



Composite Layer

1. 3D或透视变换CSS属性（perspective，transform）
2. video标签
3. canvas标签
4. css opacity
5. ccs filter
6.will-change
7. z-index

浏览器渲染

HTML渲染

reflow

引起回流：
1.页面首次渲染
2.浏览器窗口大小发生改变
3.元素尺寸或位置发生改变
4.元素内容变化（文字数量或图片大小等等）
5.元素字体大小变化
6.添加或者删除可见的DOM元素
7.激活CSS伪类（例如：:hover）
8.查询某些属性或调用某些方法

引起回流的属性和方法：
clientWidth、clientHeight、clientTop、clientLeft
offsetWidth、offsetHeight、offsetTop、offsetLeft
scrollWidth、scrollHeight、scrollTop、scrollLeft
scrollIntoView()、scrollIntoViewIffNeeded()
getComputedStyle()
getBoundingClientRect()
scrollTo()

优化

引起回流：
1. 避免频繁操作样式
2. 避免频繁操作DOM（fragment，Vitual Dom）
3. 避免频繁读取引发reflow的属性
4. 创建图层（transfer，will-change）
5. 不要使用table布局

六、正则表达式

元字符

\	转义符 反向引用
^	匹配输入字符串的开始位置。如果设置了RegExp对象的Multiline属性，^也匹配“\n”或“\r”之后的位置。
$	匹配输入字符串的结束位置。如果设置了RegExp对象的Multiline属性，$也匹配“\n”或“\r”之前的位置。

元字符

\b	匹配一个单词边界，也就是指单词和空格间的位置（即正则表达式的“匹配”有两种概念，一种是匹配字符，一种是匹配位置，这里的\b就是匹配位置的）。例如，“er\b”可以匹配“never”中的“er”，但不能匹配“verb”中的“er”。
\d	匹配一个数字字符。等价于[0-9]。grep 要加上-P，perl正则支持
\D	匹配一个非数字字符。等价于[^0-9]。grep要加上-Pperl正则支持
\n	匹配一个换行符。等价于\x0a和\cJ。
\r	匹配一个回车符。等价于\x0d和\cM。
\s	匹配任何不可见字符，包括空格、制表符、换页符等等。等价于[\f\n\r\t\v]。
\S	匹配任何可见字符。等价于[^ \f\n\r\t\v]。
\w	匹配包括下划线的任何单词字符。类似但不等价于“[A-Za-z0-9_]”，这里的"单词"字符使用Unicode字符集。
\W	匹配任何非单词字符。等价于“[^A-Za-z0-9_]”。

字符集合

[xyz]	字符集合。匹配所包含的任意一个字符。例如，“[abc]”可以匹配“plain”中的“a”。
[^xyz]	负值字符集合。匹配未包含的任意字符。例如，“[^abc]”可以匹配“plain”中的“plin”。
[a-z]	字符范围。匹配指定范围内的任意字符。例如，“[a-z]”可以匹配“a”到“z”范围内的任意小写字母字符。注意:只有连字符在字符组内部时,并且出现在两个字符之间时,才能表示字符的范围; 如果出字符组的开头,则只能表示连字符本身.
[^a-z]	负值字符范围。匹配任何不在指定范围内的任意字符。例如，“[^a-z]”可以匹配任何不在“a”到“z”范围内的任意字符。

量词

*	匹配前面的子表达式任意次。例如，zo*能匹配“z”，也能匹配“zo”以及“zoo”。
+	匹配前面的子表达式一次或多次(大于等于1次）。例如，“zo+”能匹配“zo”以及“zoo”，但不能匹配“z”。+等价于{1,}。
?	匹配前面的子表达式零次或一次。例如，“do(es)?”可以匹配“do”或“does”中的“do”。?等价于{0,1}。
{n}	n是一个非负整数。匹配确定的n次。例如，“o{2}”不能匹配“Bob”中的“o”，但是能匹配“food”中的两个o。
{n,}	n是一个非负整数。至少匹配n次。例如，“o{2,}”不能匹配“Bob”中的“o”，但能匹配“foooood”中的所有o。“o{1,}”等价于“o+”。“o{0,}”则等价于“o*”。
{n,m}	m和n均为非负整数，其中n<=m。最少匹配n次且最多匹配m次。例如，“o{1,3}”将匹配“fooooood”中的前三个o。“o{0,1}”等价于“o?”。请注意在逗号和两个数之间不能有空格。

分支

|	或，条件选择，例如：ab|cd 能匹配 abefcd 中的ab 和 cd

分支

忽略大小写	全局	多行
i	g	m

重复

手机号验证：15588888888

```
/^1\d{10}$/
```

选择

身份证验证：420000200001011100X

```
/^\d{6}[1-2]\d{3}[01]\d[0-3]\d{4}(\d|X)$/i
```

字符集合

邮箱验证：xlin@a.com

```
/^\w@\w+\.\w+/i
```

反义

匹配标签名称： hello,world

```
/<([^>]+)>/i
```

分组

ip地址验证：127.0.0.1

```
/(\d{1,3}\.){3}\d{1,3}/i
```

贪婪与懒惰

```
let str= “I want you to (match the content) between the bracket)”;

/((.*))/.exec(str);
/((.*?))/.exec(str);
/(([^(]*))/.exec(str);
```

后向引用

匹配标签里的内容： hello,world

```
/<(\w+)>([^<]+)<\/\1>/g
```

后向引用

匹配标签里的内容： hello,world

```
/<(\w+)\s*(.*?)>([^<]+)<\/\1>/g
```

后向引用

过滤html标签，只留下文字内容：

<li class="item">莫问归期

情断之时冷暖自知，谁诀别相思成疾莫问天涯


```
/<(\w+)(.*?)>([^<]+)<\/\1>/g
```

后向引用

(?=exp)	正向前瞻	匹配exp前面的位置
(?!exp)	负向前瞻	匹配后面不是exp的位置
(?<=exp)	正向后瞻	匹配exp后面的位置不支持
(?<!exp)	负向后瞻	匹配前面不是exp的位置不支持

JavaScript不支持后瞻

后向引用

```
var str1 = "bedroom";
var str2 = "bedding";
var reg1 = /(bed(?=room))   //后面是room
alert(reg1.test(str1));   //true
alert(RegExp.$1)    //bed


var reg2 = /(bed(?!room))/  //后面不能是room
alert(reg2.test(str1))   //false
alert(reg2.test(str2))   //true
```

零宽断言

给金额加逗号，1234567890.34

```
let a = 1234567890.34;
let b = String(a).replace(/(\d)(?=(\d{3})+\.)/g, '$1,');
```

零宽断言

模版解析

```
{{#if 3>2}}
    {{#if 5>0}}
        沉默有时念想有时
    {{#else}}
        谁诀别相思成疾莫问天涯
    {{/if}}
{{#else}}
     也莫问归期
{{/if}}
```

```
/{{#if\s+([^}]])}}((?!{{#if).)?{{\/if}}/
```

平衡组

平衡组用于匹配左右两边开始，结束符号相等数量的内容
例如，对于字符串"xx <aa <bbb> <bbb> aa> yy>" 左右两边的< >
是不等的，如果简单的<.+>匹配到的是最外层的开始括号<与结束括号>之间的内容，但是开始和封闭的括号数量不一致。如果你希望匹配到
的是左右括号正常结束的字符串，那么就需要用到平衡组了。

```
/{{#if\s+([^}]*)}}((?!{{#if).)*?{{\/if}}/
```

平衡组

匹配嵌套的div标签

```
<div[^>]*>[^<>]*(((?'Open'<div[^>]*>)[^<>]*)+((?'-Open'</div>)[^<>]*)+)*(?(Open)(?!))</div>
```

```
<                          #最外层的左括号
[^<>]*                     #最外层的左括号后面的不是括号的内容
( ( (?‘Open’<)             #碰到了左括号，在黑板上写一个"Open"
[^<>]*                     #匹配左括号后面的不是括号的内容
)+ ( (?‘-Open’>)           #碰到了右括号，擦掉一个"Open"
[^<>]*                     #匹配右括号后面不是括号的内容
)+ )* (?(Open)(?!))        #在遇到最外层的右括号前面，判断黑板上还有没有没擦掉的"Open"；如果还有，则匹配失败
>                          #最外层的右括号
```

JavaScript不支持平衡组

作业

编程计算四则运算的值 如：f(5+(4/(9+1)))= 5.4

七、数据结构与算法基础

排序算法

快速排序

归并排序

堆排序

树的遍历

前序

中序

后序

层序

作业

链表反转

数组中第K大的元素

不重复最长子串

位运算/位掩码

THANKS

极客时间 | 训练营