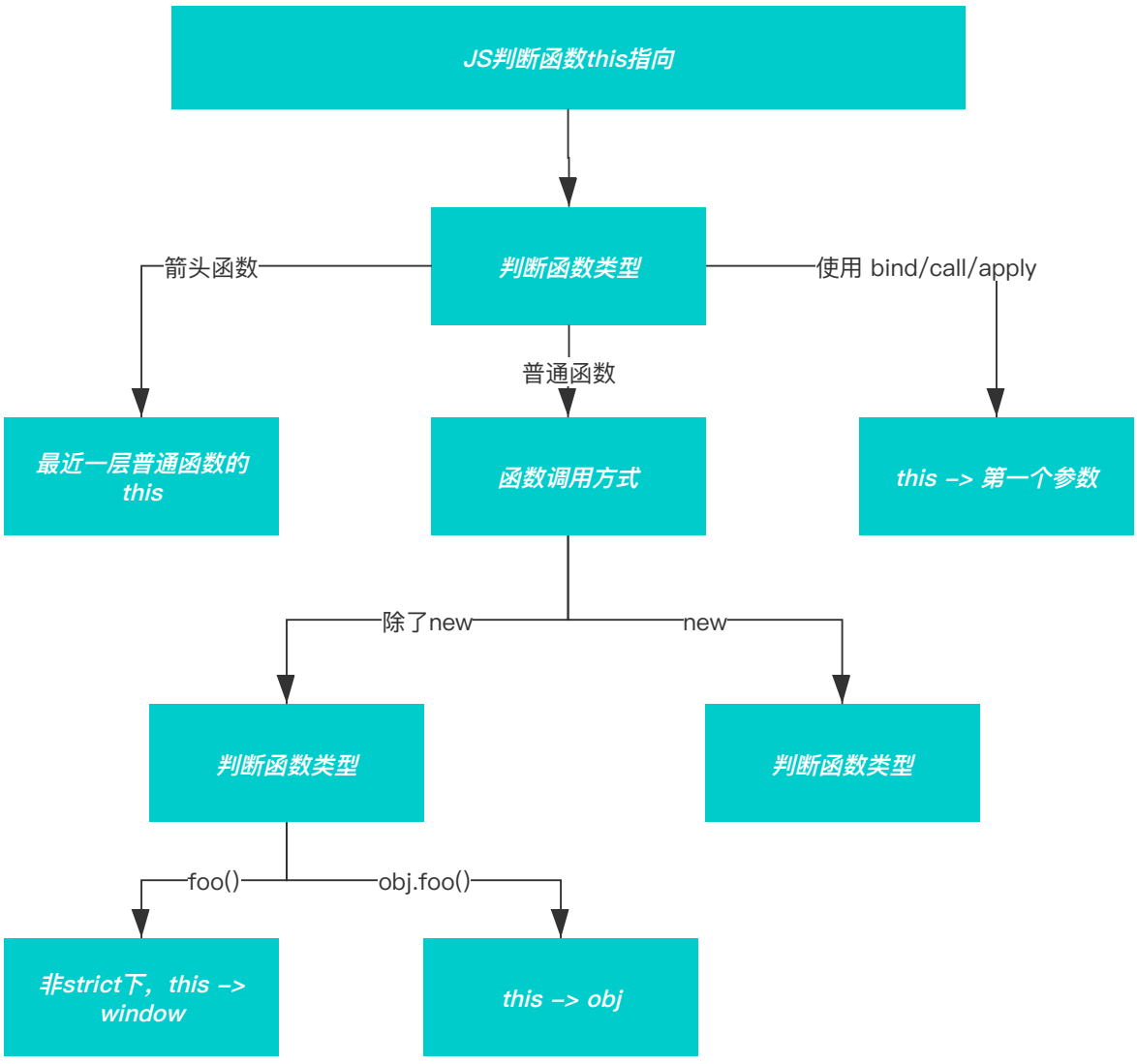
我在2019，找实习

JS基础

转Boolean

- Boolean(undefined) = Boolean(null) = Boolean('') = Boolean(0) = Boolean(-0) = Boolean(NaN) => false
- 其他的都返回true

this指向



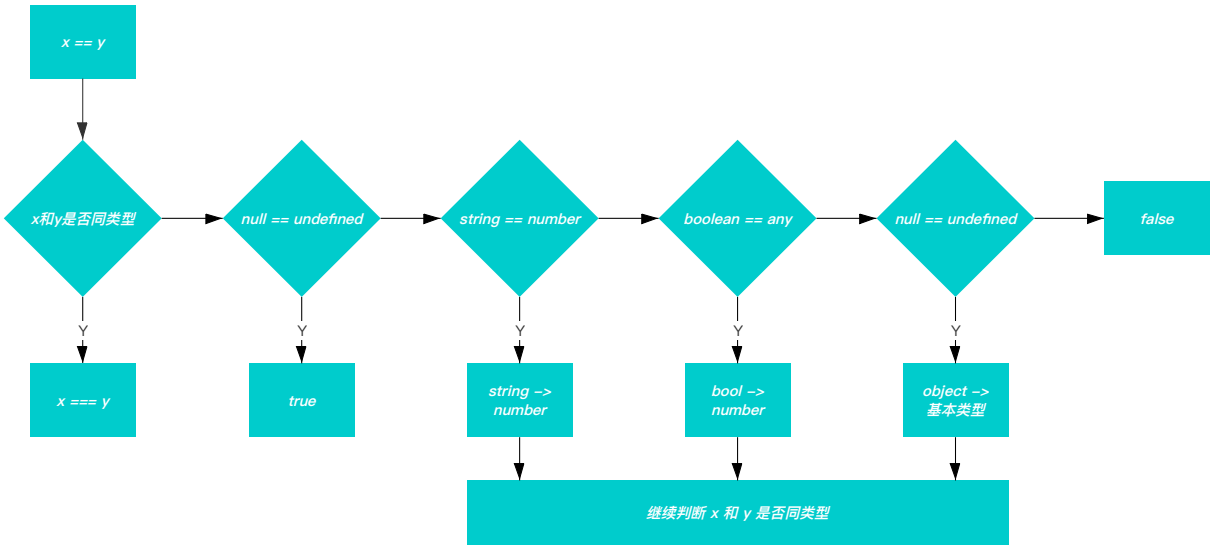
JS中的相等性判断

===

==

Object.is()

== 运算符的比较规则



delete操作符

- 如果你试图删除的属性不存在，那么delete将不会起任何作用，但仍会返回true
- 如果对象的原型链上有一个与待删除属性同名的属性，那么删除属性之后，对象会使用原型链上的那个属性（也就是说，delete操作只会在自身的属性上起作用）
- 任何使用 `var` 声明的属性不能从全局作用域或函数的作用域中删除。
    - 这样的话，delete操作不能删除任何在全局作用域中的函数（无论这个函数是来自于函数声明或函数表达式）
    - 除了在全局作用域中的函数不能被删除，在对象(object)中的函数是能够用delete操作删除的。
    - 任何用 `let` 或 `const` 声明的属性不能够从它被声明的作用域中删除。
- 不可设置的(Non-configurable)属性不能被移除。这意味着像 `Math`, `Array`, `Object` 内置对象的属性以及使用Object.defineProperty()方法设置为不可设置的属性不能被删除。

new发生了什么

描述1:

1. 创建继承子原型的新对象
2. 让构造函数被调用（有参数则传入参数）
3. 并将this绑定到新创建的对象
4. 注意：构造函数的返回的对象就是new表达式的结果。如果没有显式返回，则使用 1. 创建的对象（一般也不会去返回）

描述2:

1. 新生成了一个对象
2. 链接到原型
3. 绑定 this
4. 返回新对象

js迭代器

迭代器	缺点
forEach	不能break/continue
for...in...	适用于对象遍历，会输出手动添加的键值；只迭代自身属性
for...of...	不能直接遍历原生对象
for	编码繁琐，易出错

深浅拷贝

- Object.assign()
  - 浅拷贝,不能拷贝引用类型的数据
- 展开运算符
  - 浅拷贝,类似于Object.assign()
- var obj_2 = JSON.parse(JSON.stringify(obj_1));
  - 深拷贝

方法	深/浅	特点
Object.assign()	浅	不能拷贝引用类型的数据
展开运算符 ...	浅	不能拷贝引用类型的数据
JSON.parse(JSON.stringify(obj_1));	深	1. 会忽略 `undefined` 2. 会忽略 `symbol` 3. 不能序列化函数 4. 不能解决循环引用的对象
自己实现	深	考虑大量的边界情况

箭头函数：

- 箭头函数不会创建自己的this,它只会从自己的作用域链的上一层继承this
- 通过 call 或 apply 调用：由于 箭头函数没有自己的this指针，通过 call() 或 apply() 方法调用一个函数时，只能传递参数，他们的第一个参数会被忽略
- 箭头函数不绑定 `Arguments` 对象
- 箭头函数不能用作构造器，和 `new` 一起用会抛出错误
- 箭头函数没有prototype属性

关键字 let

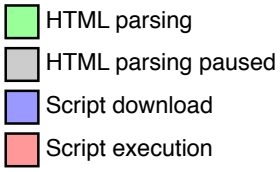
- 块级作用域
- 不会变量提升（var 的 undefine 不等于没有申明）
- for循环体和循环头有自己独立的块

JS字符串

- 不可变
- 注意什么时候是String 和 Object
- JS会自动将基本类型String转换为Object

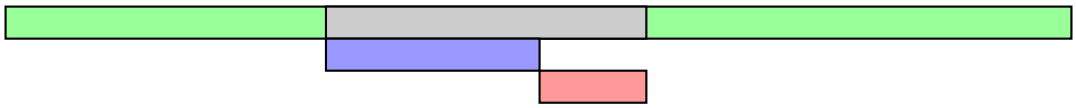
关于 `defer` `async`（图解）

> async vs defer attributes



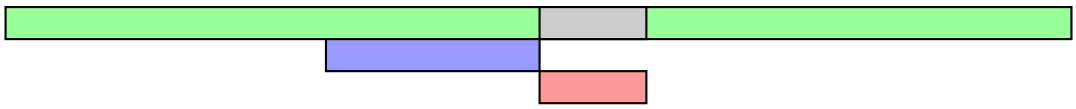
`<script>`

Let's start by defining what `<script>` without any attributes does. The HTML file will be parsed until the script file is hit, at that point parsing will stop and a request will be made to fetch the file (if it's external). The script will then be executed before parsing is resumed.



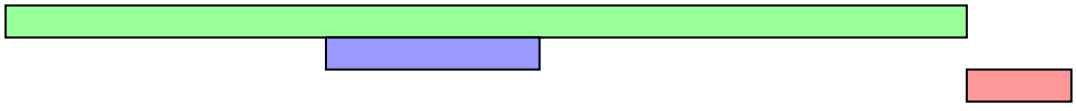
`<script async>`

`async` downloads the file during HTML parsing and will pause the HTML parser to execute it when it has finished downloading.



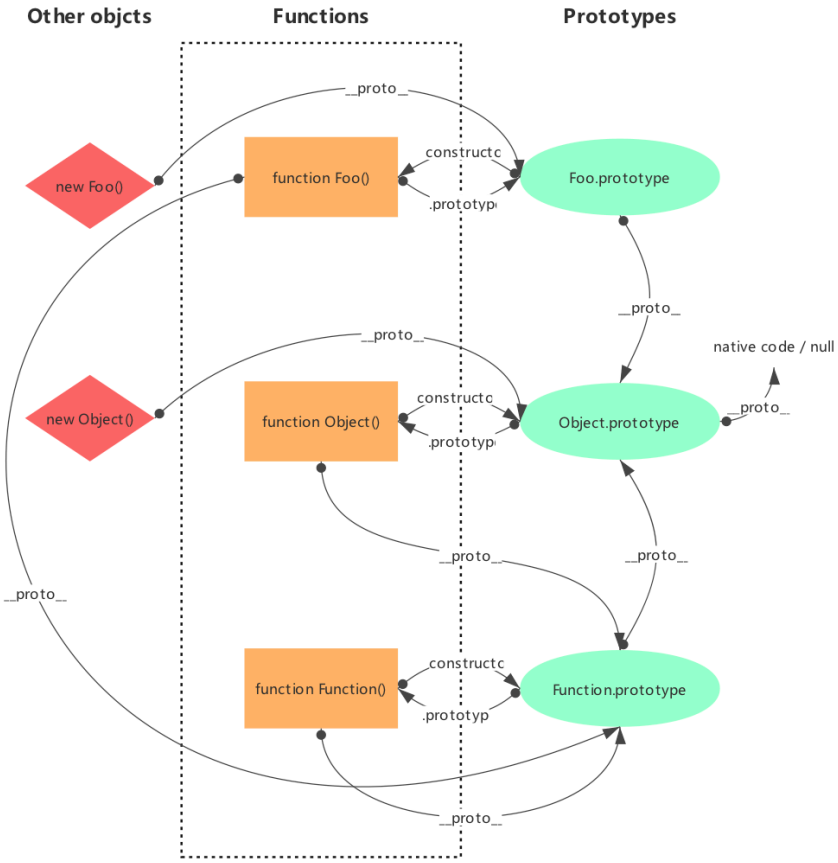
`<script defer>`

`defer` downloads the file during HTML parsing and will only execute it after the parser has completed. `defer` scripts are also guaranteed to execute in the order that they appear in the document.

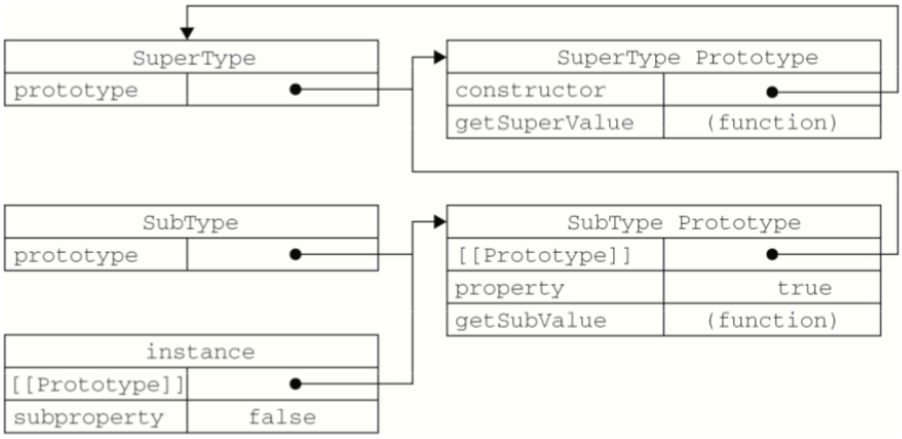


原型链

> 深度解析原型中的各个难点




```
function funName() // 内部调用 new Function
let objName = {} // 内部调用 new Object
// 等都是语法糖，其内部调用了各自的构造函数
```



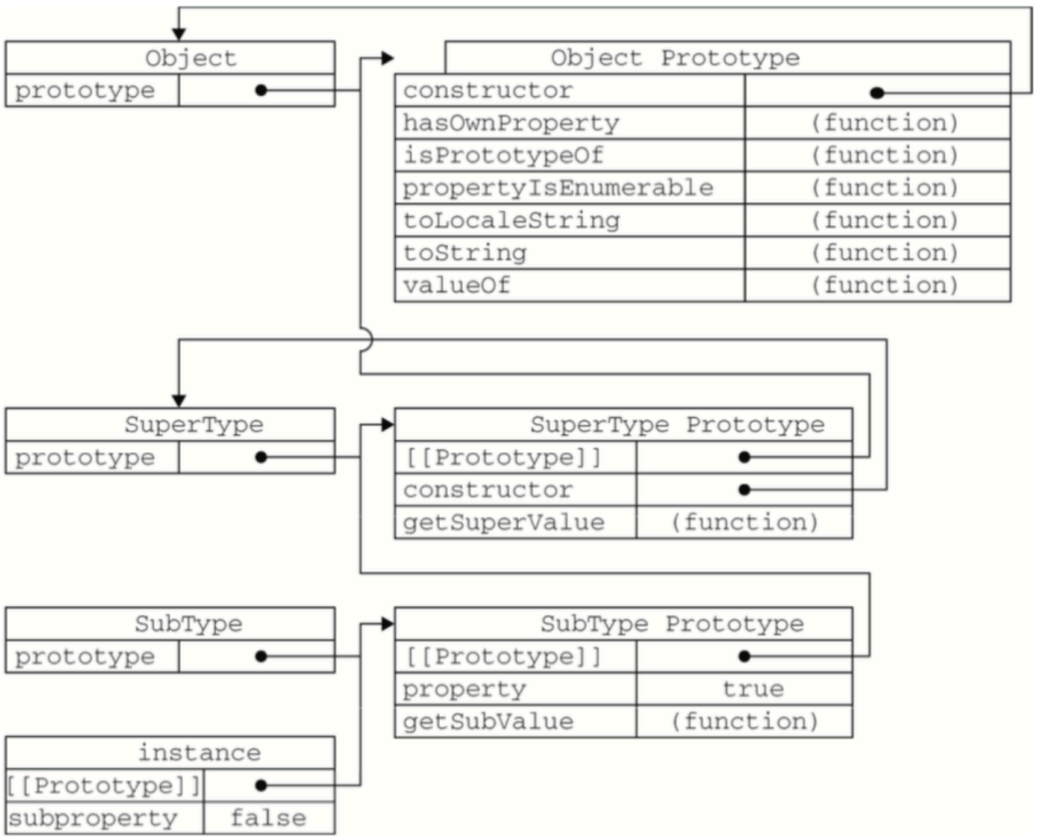


图 6-5

- 实例上的每一次属性查找都是一次遍历
- 判断实例和原型的关系
  - `prototypeObject.isPrototypeOf()`：方法用于测试一个对象是否存在于另一个对象的原型链上
  - `instanceof`：`instanceof`运算符用于测试**构造函数**的`prototype`属性是否出现在对象的原型链中的任何位置
  - `hasOwnProperty()`:方法会返回一个布尔值，指示对象**自身**属性中是否具有指定的属性
  - `in`: 如果指定的属性在指定的对象或其原型链中，则**in 运算符**返回`true`
- `__proto__`
  - 它是一个能够通过实例访问到实例原型的引用
  - 不推荐使用，取而代之的是Object.getPrototypeOf
  - 而且有性能问题!

闭包

> 你不懂JS：作用域与闭包

- 闭包就是函数能够记住并访问它的词法作用域，即使当这个函数在它的词法作用域之外执行时。
- 计时器、事件处理器、Ajax请求、跨窗口消息、web worker、或者任何其他的异步（或同步！）任务，当你传入一个 回调函数，你就在它周围悬挂了一些闭包!

JS异步编程

> JS 异步编程六种方案

1. 回调函数Callback
2. 事件监听
3. 发布订阅
4. Promise
5. async/await

Promise

> Promises/A+规范
>
> 关于Promise你可能不知道的6件事

promise 规范要求所有回调都是**异步**的

- 优点：
  - 解决了地狱回调
  - 统一了异步API
- 缺点：
  - 无法取消Promise，一旦新建就无法中途取消
  - 不设置回调函数内部的错误状态无法捕捉
  - 无法获取pengding当前的状态

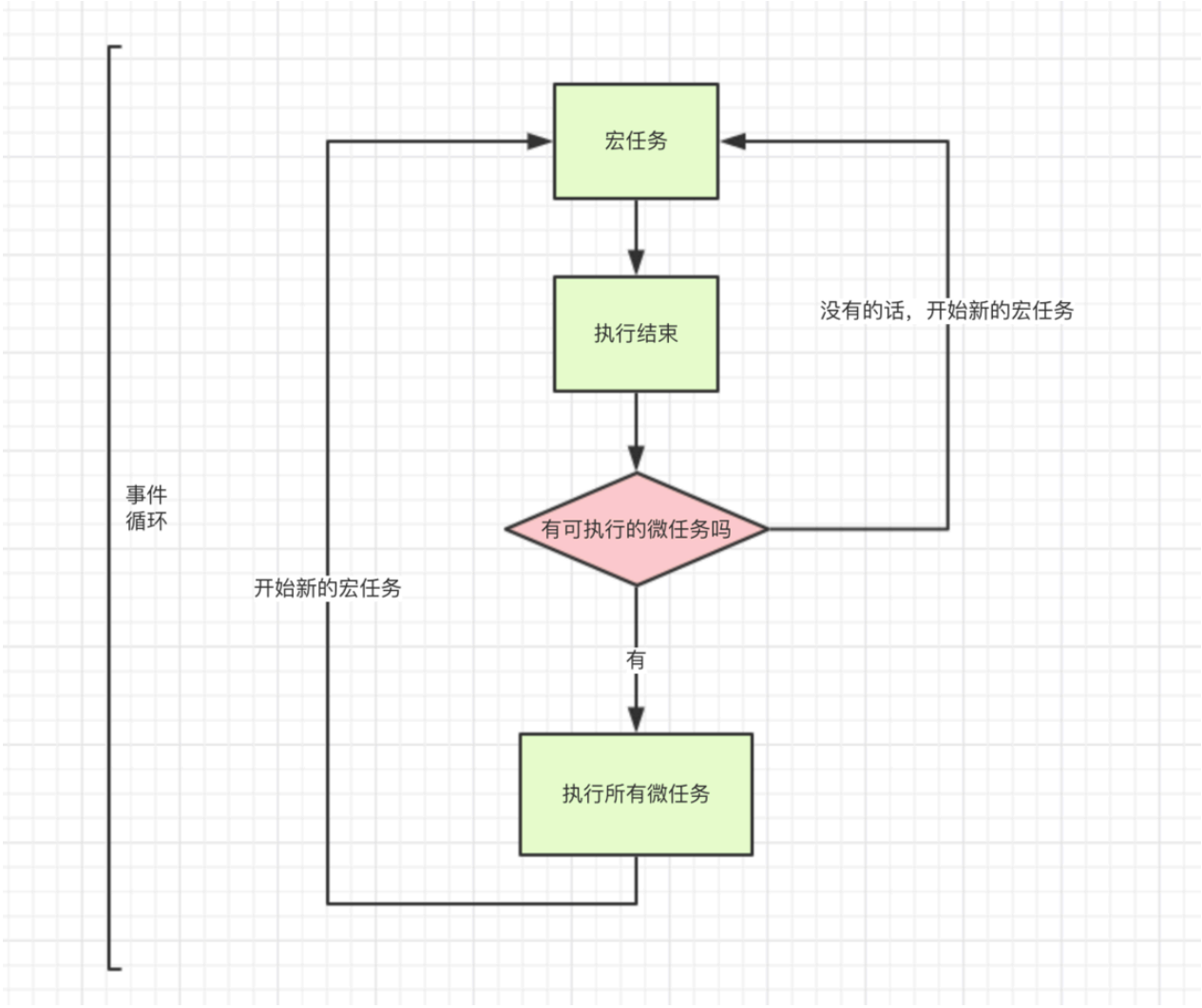
Event Loop

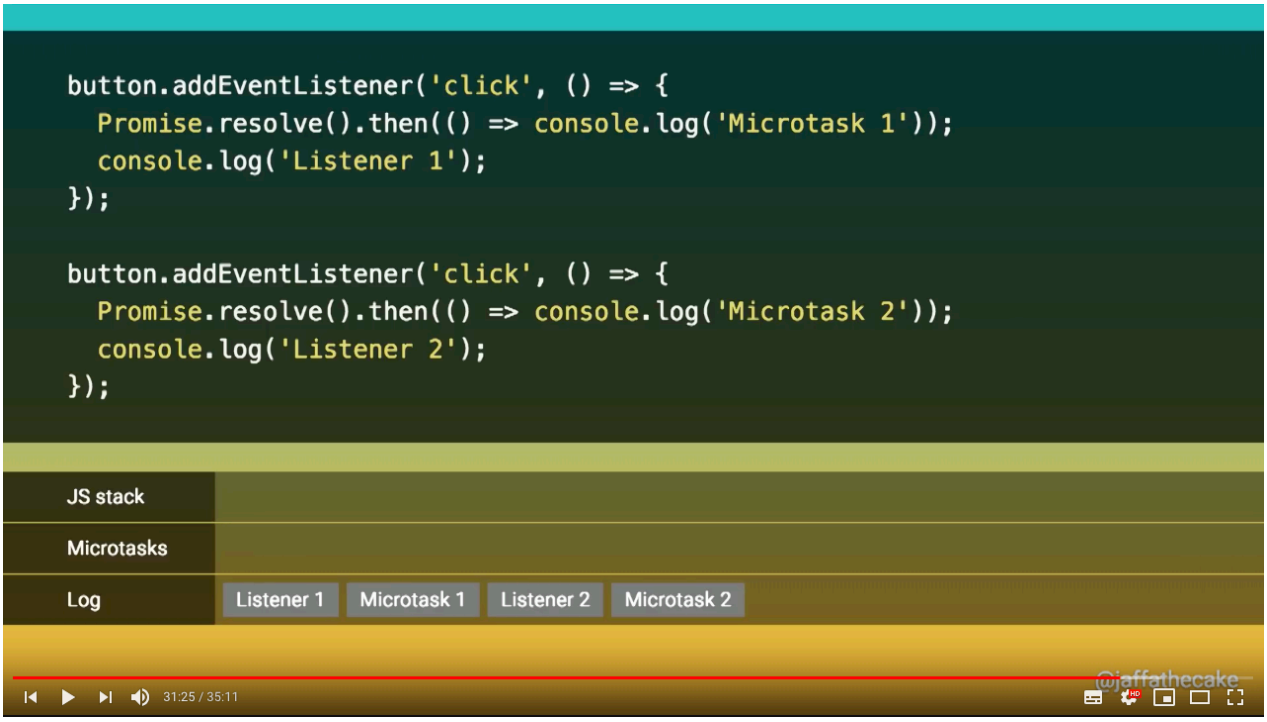
> 这一次，彻底弄懂 JavaScript 执行机制 理解event loop（浏览器环境与nodejs环境） Jake Archibald: In The Loop - JSConf.Asia 2018

- 概述：Event Loop 是 Javascript 的执行机制，即js是如何运行的。而且Javascript是单线程的！
- microtask / macrotask

	包括哪些	每一轮事件循环
宏任务(macrotast)	`script`,`setTimeout`,`setInterval`,`setImmediate`,`I/O`,`UI rendering`	只取任务队列中第一个(如果不为空)
微任务(microtast)	`Promise`,`process.nextTick`,`Object.observe`,`MutationObserver`,`MutaionObserver`	将所有任务依次执行，即清空任务队列

- 执行的时候分为：主线程 / 微任务队列 / 宏任务队列



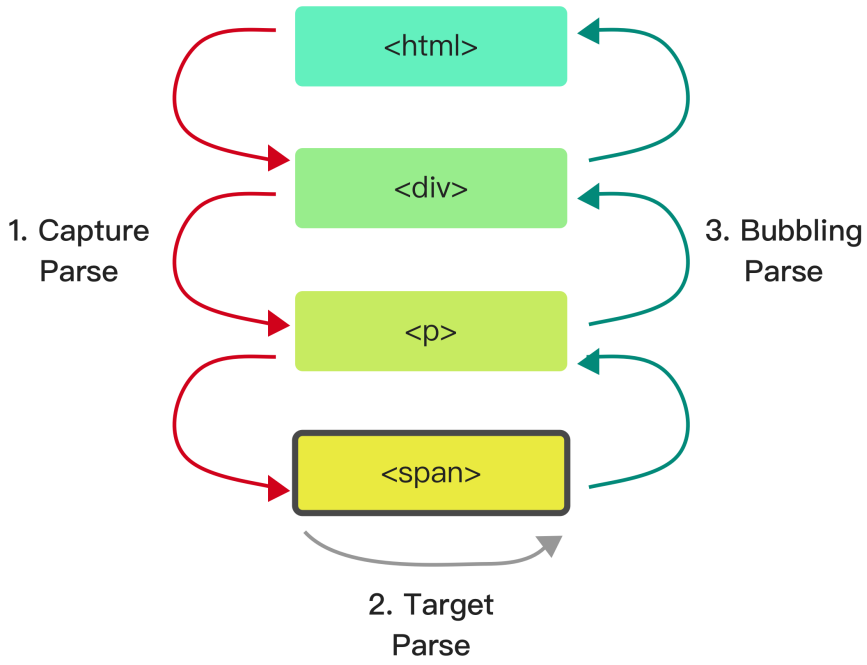


https://www.youtube.com/watch?v=cCOL7MC4Pl0&list=PLz8bg4TaC8OilP8IGXkN3A2m_uzT59I_c&index=5&t=421s

浏览器事件模型

> Bubbling and capturing

- 三个阶段：捕获阶段、目标阶段、冒泡阶段
- 事件传递模型



@BY DeCode

- 事件传播的阻止
- 事件委托
  - 基本原理：事件代理即是利用事件冒泡的机制把里层所需要响应的事件绑定到外层
  - 优点：
    - 减少内存消耗，提高性能
    - 动态绑定事件

抖动

跨域

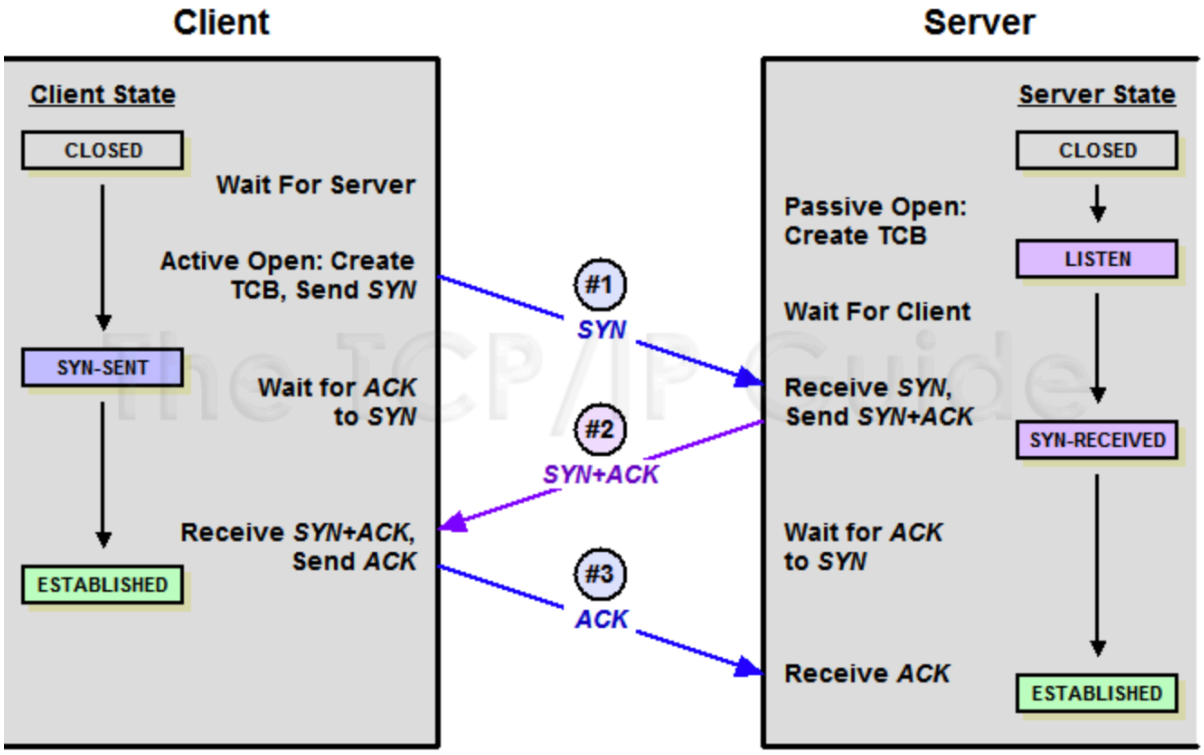
- 通源策略：端口port、域名host、协议protocol都相同
- 定义：协议、域名、端口有一个不同就是跨域
- 跨域方法：
  - JSONP：利用`<script>`标签没有跨域限制
  - CORS：前后端配合，后端开启Access-Control-Allow-Origin，表示那些域名可以访问资源
  - document.domain：利用javascript更改域名（二级域名相同就可以实现跨域）

    - postMessage：一般用于同一页面的不同iframe之间数据传输
- 比较:

跨域方法	优点	缺点
JSONP	兼容性不错	只限于Get方法
CORS		需要前后端配后
document.domain	方法简单	只适合于二级域名相同
postMessage		

TCP

- TCP的三次握手：(Three Way HandShake)



- 为什么是3次，而不是2次?
    - 三次只是理论上满足不可靠信道传输可靠消息的最小次数
    - 举一个失败传输的例子。假设A Client 和 B Server，A发出第一次握手，B返回第二次握手，但是由于某种原因，导致传输失败。A认为连接失败，所以CLOSED，但是B认为连接成功，开始传输数据，但是没有收到接受回复，所以启动重传机制，不断的重传，浪费资源。

HTTP协议

- HTTP
    - 构成: 开始行、首部行、实体部分
    - 开始行: GET PATH VERSION
    - 首部行:

        - Catch-Control
        - Date
        - Connection
        - Host
        - ...
    - 实体部分:实际传输的内容
- TLS
    - Transport Layer Security(传输层安全性协议)
    - 前身 SSL
- HTTPS
    - S => Security
    - HTTPS = HTTP + TLS(SSL)
- POST 和 GET 的区别
    - Get是无副作用的, 幂等的(无数次调用结果相同)
    - Post则相反
- 常见状态码

状态码	类别	原因短语
1xx	信息类	接受的请求正在处理
2xx	成功类	请求正常处理完毕
3xx	重定向	需要进行附加操作已完成的请求
4xx	客户端错误	服务器无法处理请求(资源没找到)
5xx	服务端错误	服务器请求出错(资源可能有,但是无法返回)

回车后发生了什么?

> https://github.com/skyline75489/what-happens-when-zh_CN

1. DNS域名解析
2. 建立TCP链接（3次握手）
3. 发送HTTP请求(TLS握手)
4. 服务器处理请求
5. 返回响应结果
6. 关闭TCP链接（4次握手）
7. 浏览器解析HTML
8. 浏览器渲染

常见的攻击行为

- XSS
    - 攻击方式:攻击者将攻击代码注入到网页中

    - 防御:
        1. 转义字符
        2. 白名单过滤
        3. CSP: 本质上也是设置白名单, HTTP Header, `<mate>`
- CSRF
    - 原理就是攻击者构造出一个后端请求地址，诱导用户点击或者通过某些途径自动发起请求
    - 防御:
        1. Samesite
        2. Referer
        3. Token
- 点击劫持
    - 是一种视觉欺骗的方式
    - 防御:
        1. X-FRAME-OPTIONS(HTTP响应头)
        2. JS防御
- 中间人攻击
    - 攻击人同时和客户端/服务端建立起连接,从而获取信息
    - 防御:
        1. HTTPS

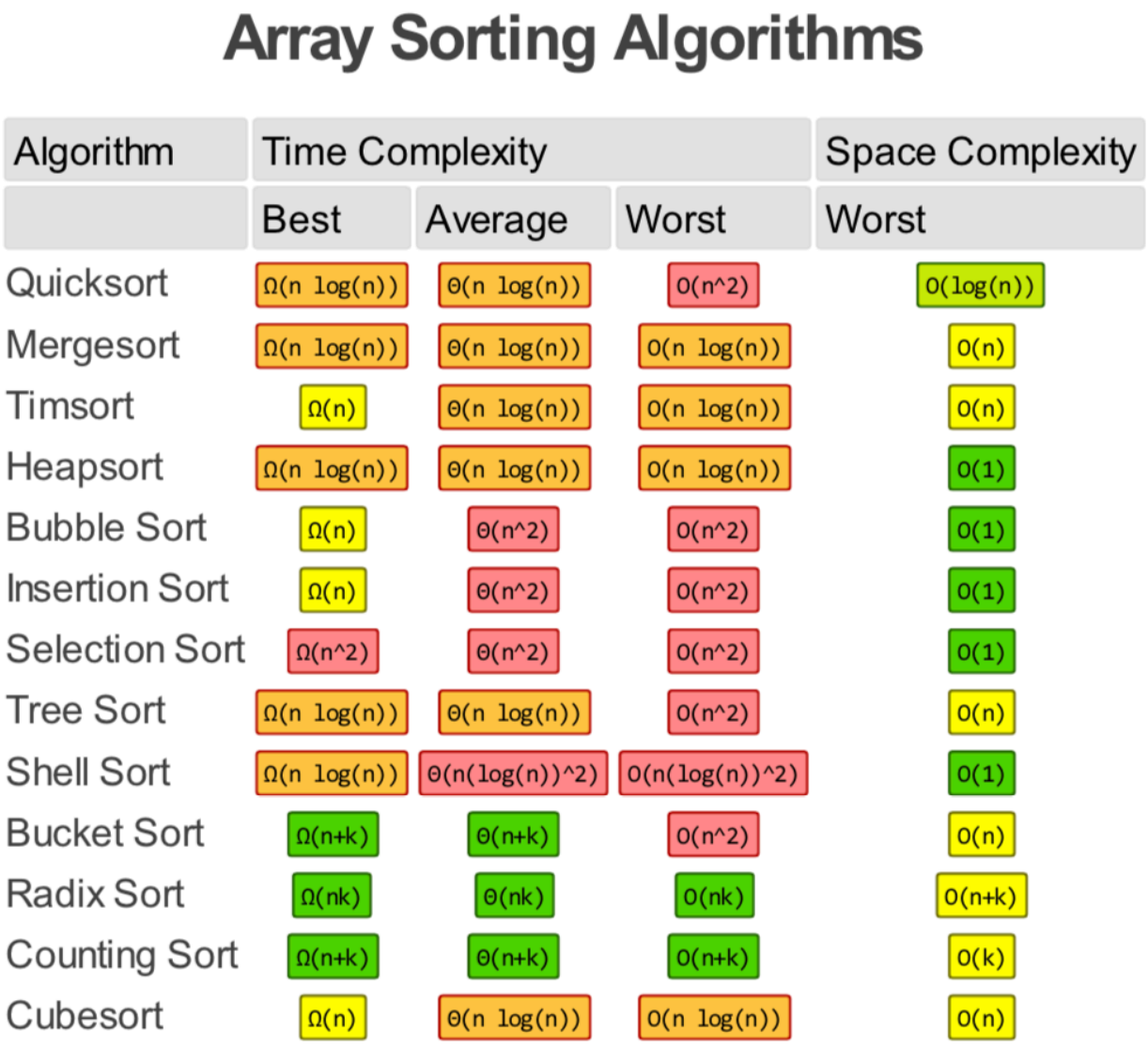
浏览器的缓存机制

- 强制缓存
    - Expires
    - Cache-Control
- 协商缓存
    - Last-Modified和If-Modified-Since
    - Etag和If-None-Match
    - Etag > Last-Modified 精度

前端基本算法

排序

所有排序算法的性能:



常见排序算法性能比较：

Sort Algorithm	Stability	Time Complexity(best/average/worst)
Selection Sort	N	O(n^2) / O(n^2) / **O(n^2)**
Insert Sort	Y	O(n) / O(n^2) / **O(n^2)**
Bubble Sort	Y	O(n) / O(n^2) / **O(n^2)**
Quicksort	N	O(nlogn) / O(nlogn) / **O(n^2)**
Mergesort	Y	O(nlogn) / O(nlogn) / **O(nlogn)**
Heapsort	N	O(nlogn) / O(nlogn) / **O(nlogn)**

Heapsort：(堆排序)

```
#define LeftChild( i ) ( 2 * ( i ) + 1 )

void
PercDown( ElementType A[ ], int i, int N )
{
    int Child;
    ElementType Tmp;

/* 1*/  for( Tmp = A[ i ]; LeftChild( i ) < N; i = Child )
        {
/* 2*/      Child = LeftChild( i );
/* 3*/      if( Child != N - 1 && A[ Child + 1 ] > A[ Child ] )
/* 4*/          Child++;
/* 5*/      if( Tmp < A[ Child ] )
/* 6*/          A[ i ] = A[ Child ];
            else
/* 7*/          break;
        }
/* 8*/  A[ i ] = Tmp;
}

void
Heapsort( ElementType A[ ], int N )
{
    int i;

/* 1*/  for( i = N / 2; i >= 0; i-- ) /* BuildHeap */
/* 2*/      PercDown( A, i, N );
/* 3*/  for( i = N - 1; i > 0; i-- )
        {
/* 4*/      Swap( &A[ 0 ], &A[ i ] ); /* DeleteMax */
/* 5*/      PercDown( A, 0, i );
        }
}
```

链表

队列

堆栈

树

二叉树

深度优先DFS

- 先序遍历
- 中序遍历
- 后序遍历

广度优先BFS

AVL树

（未完成）

关于性能

- 网络
  - DNS预解析
  - 缓存
    - 强制
    - 协商
  - HTTP/2.0
    - 多路复用
    - header压缩
  - 预加载
  - 预渲染
- 渲染
  - 懒执行（提升首屏速度）
  - 不相关的计算放在后面
  - 懒加载
  - 资源延迟请求
- 文件
  - 图片的压缩，格式的选取
  - 压缩文件大小
  - CDN
  - 其他文件优化
    - css放在 head 中
    - js放在 body 底部
    - web worker 开启新的线程

Vue.js

（未完成）

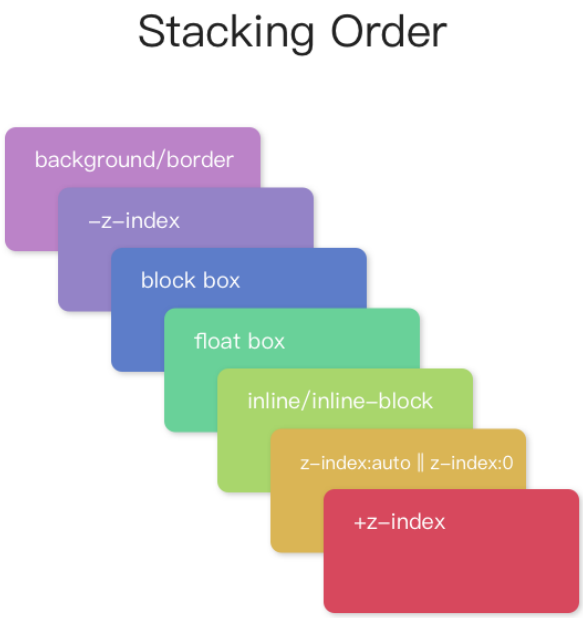
Webpack

> KieSun webpack-demo
>
> Webpack 4 Tutorial: from 0 Conf to Production Mode

CSS部分

> CSS问题

1. CSS 选择器的优先级是如何计算的?
    - 优先级不同：是否有内联样式、ID选择器数量、类选择器 | 伪类选择器 | 属性选择器数量、标签选择器 | 伪选择器数量
    - 优先级相同：那么最后出现的（在文件底部的）样式优先级更高，因此会被采纳
2. 请阐述z-index属性，并说明如何形成层叠上下文（stacking context）。
    - `z-index`只能影响`position`值不是`static`的元素
    - 没有定义`z-index`的值时，元素按照它们出现在 DOM 中的顺序堆叠（层级越低，出现位置越靠上）。非静态定位的元素（及其子元素）将始终覆盖静态定位（static）的元素，而不管 HTML 层次结构如何。
    - 层叠上下文是包含一组图层的元素。 在一组层叠上下文中，其子元素的`z-index`值是相对于该父元素而不是 document root 设置的。
    - 在同一层叠上下文中，元素按照下图的规则在Z轴上排列：



    - 在不同的层叠上下文中，先找到共同的祖先层叠上下文，然后比较共同层叠上下文下这个两个元素所在的局部层叠上下文的层叠水平。
    - 少数 CSS 属性会触发一个新的层叠上下文，例如`opacity`小于 1，`filter`不是`none`，`transform`不是`none`
3. 请阐述块格式化上下文（Block Formatting Context）及其工作原理。

    // 恶心
4. 编写高效的 CSS 应该注意什么?
    - 首先，浏览器从最右边的选择器，即关键选择器（key selector），向左依次匹配。根据关键选择器，浏览器从 DOM 中筛选出元素，然后向上遍历被选元素的父元素，判断是否匹配。选择器匹配语句链越短，浏览器的匹配速度越快。避免使用标签和通用选择器作为关键选择器，因为它们会匹配大量的元素，浏览器必须要进行大量的工作，去判断这些元素的父元素们是否匹配。
    - BEM

    - 搞清楚哪些 CSS 属性会触发重新布局（reflow）、重绘（repaint）和合成（compositing）。
5. 使用 CSS 预处理的优缺点分别是什么?
    - 优点：（1）易于维护（2）样式共享（3）可嵌套（4）变量
    - 缺点：（1）可能会生成多层级选择器（过于精确的选择器）
6. 盒模型
    - `box-sizing`
    - `width、height`是由什么确定的
    - 考虑浮动`float`
7. inline和inline-block有什么区别?

`block`	`inline-block`	`inline`	
大小	填充其父容器的宽度。	取决于内容。	取决于内容。
定位	从新的一行开始，并且不允许旁边有 HTML 元素（除非是`float`）	与其他内容一起流动，并允许旁边有其他元素。	与其他内容一起流动，并允许旁边有其他元素。
能否设置`width`和`height`	能	能	不能。 设置会被忽略。
可以使用`vertical-align`对齐	不可以	可以	可以
边距（margin）和填充（padding）	各个方向都存在	各个方向都存在	只有水平方向存在。垂直方向会被忽略。 尽管`border`和`padding`在`content`周围，但垂直方向上的空间取决于'line-height'
浮动（float）	-	-	就像一个`block`元素，可以设置垂直边距和填充。

8. 请解释在编写网站时，响应式与移动优先的区别。
    - 响应式设计的适应性原则：网站应该凭借一份代码，在各种设备上都有良好的显示和使用效果。响应式网站通过使用媒体查询，自适应栅格和响应式图片，基于多种因素进行变化，创造出优良的用户体验。就像一个球通过膨胀和收缩，来适应不同大小的篮圈。
    - 自适应设计：更像是渐进式增强的现代解释。与响应式设计单一地去适配不同，自适应设计通过检测设备和其他特征，从早已定义好的一系列视窗大小和其他特性中，选出最恰当的功能和布局。
9. 其他
    - CSS选择器的匹配规则：从右到左，从根部遍历DOMTree看是否匹选择器

其他参考

> InterviewMap
>
> 30SECONDS OF INTERVIEW