深拷贝与浅拷贝，

浅拷贝指单层拷贝（并不单指赋值 “=”）

深拷贝：

function deepClone(*source*, *target*) {

for (let key in source) {

if (!source.hasOwnProperty(key)) {

if (source[key] instanceof *Object*) {

if (*Array*.isArray(source[key])) {

target[key] = []

} else {

target[key] = {}

}

deepClone(source[key], target[key])

} else if (source[key] != undefined) {

target[key] = source[key]

}

}

}

}

基本数据类型比较是否相等通过值比较（基本类型的值存放在栈内存中，赋值则在栈中开辟新的空间存放该值，互不干扰）

引用数据类型是否相等通过比较地址所指向的数据（引用类型存放在堆内存中，地址存放在栈内存中，栈内存中的地址指向堆内存的数据）

Bind()方法会创建一个新函数，并不会立即执行，需要立即执行需要在末尾加()

New发生过程：创建一个新的空 对象，将该对象的原型指向构造函数的原型，修改该对象的this指针，有返回值则返回对应值没有则返回改对象

判断变量的类型可以使用instanceof

区分Array和Object可以使用Array.isArray(target)

函数的调用方法：（非箭头函数的this指向为最后的调用者，匿名函数没有函数名无法被其他变量调用，（类似与直接在全局环境中作为函数调用）所以匿名函数的this指向为window，通过函数表达式调用匿名函数）

作为函数直接调用

函数作为方法被调用

作为构造函数调用

作为函数方法在函数里被调用（在js strict环境下，函数的第一个参数将作为this的值，即使这个参数不是对象，在js non strict模式下，如果第一个参数的值是null或者undefined 它将使用全局作对象替代）

箭头函数的this指向：箭头函数的this始终指向函数定义时的this，而非执行时，箭头函数中没有this绑定，必须通过查找作用域链来决定，如果箭头函数被废箭头函数包裹，则this绑定的是最近一层非箭头函数的this，否则，this为undefined

BFC (块级格式化上下文)

通过float overflow不为visible …触发

作用：BFC元素不被浮动元素所覆盖

使得内部元素的margin不重叠

父级设置为BFC会被子float元素所撑开高度

Vue：动态改变数组或者对象数据并不会触发视图的刷新，使用$set（target，key，value）改变可以触发视图刷新

组件命名：html不识别大小写，应该使用-连接，在其他地方都使用单词首字母大写的·命名规则，v-if和v-for不能作用于同一个元素，对循环数据源进行筛选应将数据源通过计算属性筛选后在用于v-for循环

区分对象和数组

Object.prototype.toString().call(target)===”[Object Array]” || ”[Object Object]”

Object.prototype.toString().call(target).slice(8,-1)===”Array” || ”Object”

跨域：产生原因ajax浏览器同源策略

解决方法：script发起jsonp，后台在头部设置安全域名，node.js中间件代理跨域，nginx反向代理，后台服务器上设置cors

Img的title和alt属性：图片正常输出时，不显示alt信息（包括鼠标移入），不正常输出时，显示alt信息，title属性鼠标移入时无论图片是否正常输出都会出现

ES5和ES6继承的区别：

ES5组合继承，缺点，父类实例的属性会被所有子类共享需要借用构造函数重新创建实例覆盖原型中的实例，导致调用两次父类的构造函数（一次父类实例化赋值给子类原型，一次子类构造函数首行调用覆盖父类实例属性：supertype.call(this)）

ES6寄生组合继承：将父类的原型拷贝一份副本，将副本（Object.create(source)）的construct属性指向子类构造函数，将副本赋值给子类原型，子类构造函数行首调用父类构造函数（ es6写法为super() ）

浏览器渲染：js下载也会中断DOM树的更新，影响首屏时间主要因素在于css解析，是否生成renderTree，页面解析流程：构建DOM，构建CSSOM，生成renderTree，layout，panting，浏览器可以显示不完整的renderTree，也就是说可以构建一部分就先显示一部分，所以首屏时间主要受css影响，css解析快则renderTree构建快则显示的快，script

标签的async和defer，async会并行下载并执行，，[MDN](https://link.juejin.im/?target=https%3A%2F%2Fdeveloper.mozilla.org%2Fzh-CN%2Fdocs%2FWeb%2FHTML%2FElement%2Fscript)对此的描述是用来通知浏览器该脚本将在文档完成解析后，触发 DOMContentLoaded 事件前执行，css不会阻塞dom解析，link标签会阻塞script标签，遇到script（没有异步标识符）会促使页面渲染更新

子资源加载

网站通常会使用图像，CSS和JavaScript等外部资源。 这些文件需要从网络或缓存中加载。 主线程可以在解析构建DOM时逐个请求它们，但为了加快速度，“预加载扫描器”会同时运行。 如果HTML文档中存在错误!未指定文件名。或之类的元素 ，则预加载扫描程序会检查由HTML解析器生成的标记，并向网络线程发送请求

Iterate.map(callback())callback默认传递三个参数（currentValue，index，包含该元素的数组对象） parseInt(string，radix)方法 如果string大于radix会返回NaN 多位有效字符串按位进行运算（不传radix，传0或者10 为10进制）

parseInt（1，0） return 1

praseInt（2，1） return NaN

parse Int（11,9）return 10

==和===：==会进行类型转换，当左右都是undefined或者null是为true 当为string和number时，会将string转化为number，当为基本类型时，会向number转换，当存在Object时 会向基本类型转化

DOMContentLoad和Load的区别：DOMContentLoad 初始html文档被解析完成时触发，无需等待样式表，图像和子框架的完成加载（必须等待所属的script之前的样式表加载解析完成才会触发），load用于检测一个完全加载的页面

事件流：stoppropagation 既可以组织事件冒泡 也可以阻止事件捕获，stopImmediatePropagation具有stoppropagation的作用而且可以阻止该注册节点的其余注册函数，如果一个节点同时注册了捕获和冒泡事件，则触发顺序按照注册顺序

CreateDocumentFrame() 文档片段：将需要动态生成的子元素加入到文档片段 ，然后将该文本片段加入到目标父节点，文档片段会在此过程中被其所有子元素一起替换，应为文档片段处于内存中且不再dom树中，所以使用文档片段插入可以减轻回溯，

Typeof实现原理：js存储数据前1-3位机器码用来表示类型

000：对象

010：浮点数

100：字符串

110：布尔

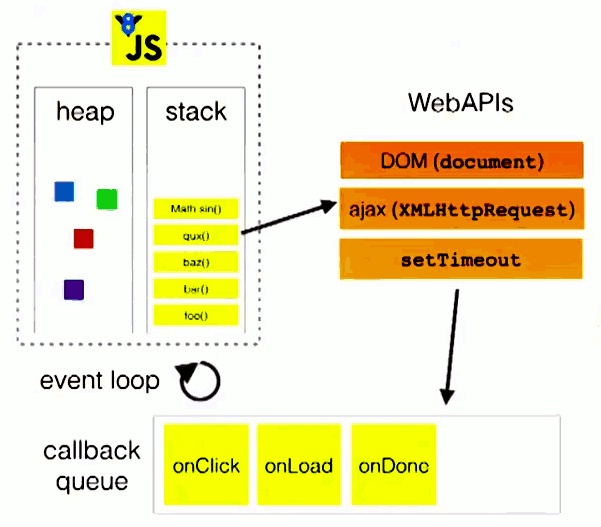
1：整数

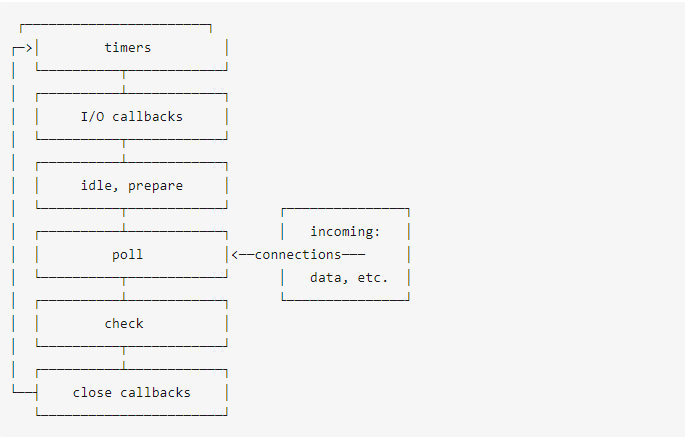
Instanceof实现原理，判断右边的prototype是否在左边的原型链上

requestAnimationFrame：requestAnimationFrame关键的就是他只是请求浏览器在下一次可以获得的机会去展示一帧画面，而不是在一个已经规划好的间隔。也就是说浏览器能够根据页面加载，元素显示，电池的状态来选择requestAnimationFrame的性能，它能够将所有的动画都放到一个浏览器重绘周期里去做，这样能保存你的CPU的循环次数，让你的设备存活时间更长

事件循环：

Browser：



Node：

解释：https://zhuanlan.zhihu.com/p/33058983

http缓存：强制缓存:查找浏览器缓存如果有则直接调用，没有(请求失败)则向服务器发起请求,包含请求参数expires（过期时间）参数cache-control

协商缓存:第一次请求时从服务器获取资源以及该资源的标识，之后请求时在请求头中携带该标识直接像服务器发请求，如果标识未过期则返回304（not modified）

http1为if-modified-since（如果资源改变则重新发送）if-unmodified-since(如果资源没变则重新发送) （存在资源修改但是并没有实质性变化的情况） http1.1则使用etag：服务器响应请求时，通过该字段告诉浏览器当前资源的唯一标识（由服务器生成）

if-none-match：再次请求包含此字段，后面的值为在缓存中获取的标识。服务器接收到次报文后发现If-None-Match则与被请求资源的唯一标识进行对比。不同，说明资源被改动过，则响应整个资源内容，返回状态码200。相同，说明资源无心修改，则响应header，浏览器直接从缓存中获取数据信息。返回状态码304.但是实际应用中由于Etag的计算是使用算法来得出的，而算法会占用服务端计算的资源，所有服务端的资源都是宝贵的，所以就很少使用Etag了。

使用原生js模拟

call：

*Function*.prototype.\_call = function(*context*) {

var context = context || window

var result

let args = []

let arg\_len = *arguments*.length

for (let i = 1; i < arg\_len; i++) {

args.push('arguments[' + i + ']')

}

context.fn = *this*

result = eval('context.fn(' + args + ')')

delete context.fn

return result

}

apply：

*Function*.prototype.\_apply = function(*context*, *arr*) {

var context = *Object*(context) || window

context.fn = *this*

var result

if (!arr) {

result = context.fn()

} else {

*console*.log(`arr: ${arr}`)

var args = []

for (var i = 0, len = arr.length; i < len; i++) {

args.push('arr[' + i + ']')

}

result = eval('context.fn(' + args + ')')

}

delete context.fn

return result

}

http部分：

http是无连接（即一次连接只能处理一条请求），无状态的（即无记忆性，如果请求需要上一次的信息则需要重新请求）

url(统一资源定位符): 由协议名，域名，端口号，路由(虚拟目录，文件名),哈希值（锚）,携带参数组成

uri(统一资源标识符):

URI一般由三部组成：  
①访问资源的命名机制  
②存放资源的主机名  
③资源自身的名称，由路径表示，着重强调于资源。

http request：1.请求行，用来说明请求类型,要访问的资源以及所使用的HTTP版本.

2.请求头

3.空行

4.请求数据主体

http response： 1.状态行（协议号，状态码，状态消息）

2.消息报头

3.空行

4.响应正文

http 状态码：

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

http请求过程：

1. 客户端连接到web服务器，与服务器接口建立一个tcp套接字(socket:tcp建立的端点由主机ip和主机端口组成)连接
2. 发送http请求：通过tcp套接字，向服务器发送关于文本的请求报文，请求报文由请求行，请求头，空行，请求数据组成
3. 服务器接收请求并返回http响应：服务器解析请求，定位请求资源，将资源副本写到TCP套接字，由客户端读取，响应报文由响应状态，响应头，空行，响应体组成
4. 释放tcp连接：如果connection为close则服务器端主动释放连接，客户端被动释放连接，如果connection为keepalive状态，则该连接会持续一段时间，在该时间内可以继续接收请求

5.客户端解析html内容：浏览器的解析过程上面有