面试题

(1) 利用正则将字符串**"my-little-brother"**变成驼峰**"myLittleBrother"**

**var *str*** = **'my-little-brother'  
var *rst***=***str***.replace(/\-(.)/g,($0,$1)=>$1.toUpperCase())

(2) 尽可能多的列举几个在web中实现长连接的技术方案名词

websocket

(3) html属性中的src和href的区别是什么

参考地址：https://www.cnblogs.com/lujun1949/p/6500654.html?utm\_source=itdadao&utm\_medium=referral

|  |
| --- |
| src:  1)引用一个资源，用来替代自己本身的内容（img、script、iframe，没有src，script会不在脚本代码，img会显示X，iframe会显示空白页）；  2)浏览器解析其，会暂停其他资源的下载和处理。 |
| href:   1. 表示超链接，建立一个通道，让当前标签链接到目标地址上（a标签的href不会被当做一个资源替换自身;<link href=”head.css” rel=”stylesheet”/>   ,link标签本身并不包含实际的元素意义来做内容，它需要rel明确的表示 被链接的文档是做什么的）   1. 浏览器解析不会停止对当前文档的处理 |

(4) DOM中对应创建、移除、追加、复制、查找节点的方法是什么

创建 createElement----创建元素；createTextNode----创建文本节点；createDocumentFragment----创建DOM片段

移除 removeChild

追加 appendChild

复制 cloneNode

查找节点 querySelector querySelectorAll getElementById getElementsByTagName getElementBy

**var *main*** = **document**.getElementById(**'main'**)  
***main***.cloneNode()

(5)实现一个高宽为屏幕宽度一半（视口可以拖动缩放），并且始终居中的正方形

<**style**>  
 **body** {  
 display: **flex**;  
 justify-content: **center**;  
 align-items: **center**;  
 height: 100**vh**;  
 }  
 .**main** {  
 width: 50%;  
 height: 50**vh**;  
 background-color: **#333**;  
 }  
 </**style**>  
<**div** id=**"main"** class=**"main"**></**div**>

(6)浏览器中缓存分为哪几种，具体是如何工作的？

|  |  |
| --- | --- |
| cookies | （1）客户端在浏览器的地址栏中键入Web服务器的URL，浏览器发送读取网页的请求。（2）服务器接收到请求后，产生一个Set-Cookie报头，放在HTTP报文中一起回传客户端，发起一次会话。（3）客户端收到应答后，若要继续该次会话，则将Set-Cook-ie中的内容取出，形成一个Cookie.txt文件储存在客户端计算机里。（4）当客户端再次向服务器发出请求时，浏览器先在电脑里寻找对应该网站的Cookie.txt文件。如果找到，则根据此Cookie.txt产生Cookie报头，放在HTTP请求报文中发给服务器。（5）服务器接收到包含Cookie报头的请求，检索其Cookie中与用户有关的信息，生成一个客户端所请示的页面应答传递给客户端。 浏览器的每一次网页请求，都可以传递已存在的Cookie文件，例如，浏览器的打开或刷新网 |
| sessionStorage | 当前窗口|标签页有效；不同浏览器间不共享 |
| localStorage | 持续有效，除非浏览器上清除它；不同浏览器间不共享 |
| IndexedDB | 非关系型数据库，异步处理，存储量比前者大 |
| webSQL | 被H5放弃 |

(7)ES6 module require/exports/module.exports的区别

|  |  |
| --- | --- |
| require | 按照一定的规则寻找传入的参数对应的js文件，返回值传递给调用者 |
| exports | module内部使用，返回给调用者的是module.exports，它是辅助工具 |
| module.exports | 外部模块引用module.exports=exports |

(8)使用new操作符时，它在对象操作的过程中具体做了什么

var a=new Foo()

1)const o = new Object();//创建了一个新的空对象o

2) o.\_\_proto\_\_ = Foo.prototype;//让这个o对象的` \_\_proto\_\_`指向函数的原型`prototype`

3)Foo.call(o);//this指向o对象

4)a = o;//将o对象赋给a对象

(9)如下运行后的结果是什么？

***process***.nextTick(()=>{**console**.info(**"nextTick"**)})  
Promise.resolve().then(()=>{**console**.info(**"promise"**)}).then(()=>{**console**.info(**"promise2"**)})  
**setImmediate**(()=>{**console**.info(**"setImmediate"**)})  
***console***.info(**"end"**)

运行结果： end nextTick promise promise2 setImmediate

|  |  |
| --- | --- |
| setTimeout | 采用类似IO观察者 |
| setImmediate | check观察者 |
| process | Idle观察者 |
| 优先级：idle观察者>>IO观察者>>check观察者 | |
| 原始代码  A();B();C(); |  |
| A();  process.nextTick(B);  C(); |  |
| A();  setImmediate(B);//或者setTimeout(B,0);  C(); |  |
| process.nextTick()，效率最高，消费资源小，但会阻塞CPU的后续调用；  setTimeout()，精确度不高，可能有延迟执行的情况发生，且因为动用了红黑树，所以消耗资源大；  setImmediate()，消耗的资源小，也不会造成阻塞，但效率也是最低的。 | |

(10)给定一组远程URL，如何利用JS相关特性实现并发请求，结果按顺序输出？

Promise.all()

**var *urls*** = [ **"http://result.eolinker.com/getAccidentProportion"**,  
 **"http://result.eolinker.com/getMajorAccident"**,  
 **"http://result.eolinker.com/getAccidentDeath"**];  
**var *i*** = 0;  
**function** *next*() {  
 **if** (***i*** >= ***urls***.**length**) **return**;  
 Promise.resolve(**jQuery**.get(***urls***[***i***])).then(**function** (data) {  
 *next*();  
 });  
 ***i***++;  
}  
*next*()

(11)模拟实现loadash中\_.get()函数，实现如下传入参数取值效果

**const *obj*** = {**selector**: {**to**: {**toutiao**: **"FE coder"**}}, **target**: [1, 2, {**name**: **"byted"**}], **type**: {**test**: **'test'**}}

方案一：  
 **function** *getSpecArr*(obj,...***path***){  
 **var *regex*** = /[\\\s\[\].]+/g  
 **var *arr*** = []  
 **for** (**let *i*** = 0; ***i*** < ***path***.**length**; ***i***++) {  
 **let *tmp*** = ***path***[***i***].split(***regex***)  
 **let *preObj*** = ***obj* for** (**let *j*** = 0; ***j*** < ***tmp***.**length**; ***j***++) {  
 **if** (***tmp***[***j***] !== **''** && ***preObj***) {  
 ***preObj*** = ***preObj***[***tmp***[***j***]]  
 }  
 }  
 ***arr***.push(***preObj***)  
 }  
 **return** arr  
 }  
 *getSpecArr*(***obj***, **"selector to toutiao"**, **"target[0]"**, **"target[2].name"**, **"type.test.test.test"**)

结果：*["FE coder",1,"byted",undefined]*

方案二：

**var *rp*** = /^[\\\s\[\].]+|[\\\s\[\].]+$/

***path***.map((***item***) => {  
 **let preObj** = ***obj* item** .replace(***rp***,**''**).split(***regex***).map((*it*) => {  
 **if**(it !== **''** && **preObj**){  
 **preObj** = **preObj**[*it*]  
 }})  
 ***arr***.push(**preObj**)  
 })  
 **return** arr  
}