前端-JavaScript面试题

笔记本:	前端知识点		
创建时间:	2020/8/29 9:47	**更新时间:**	2020/9/10 13:22
作者:	1317468898@qq.com		
URL:	file:///D:/learning/面试题.docx		

1.基本数据类型

基本数据类型是存放在栈中的简单数据段，数据大小确定，内存空间大小可分配，可按值存放，按值访问。

Number、String、 Boolean、Null、Undefined、Symbol（es6， 表示独一无二的值，最大的用法是用来定义对象的唯一属性名**）**

2.引用数据类型

引用数据类型是存放在堆内存中的对象，变量其实是保存在栈内存中的一个指针，也就是堆内存中的引用地址，这个指针指向堆内存。

Array、Object、Function、Date、RegExp、包装类（String、Number、Boolean ）、单体内置对象（Global、Math）

判断数据类型：typeof运算符、instanceof操作符、Object.prototype.toString.apply()

https://blog.csdn.net/yCharlee/article/details/52424603?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-7.edu_weight&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-7.edu_weight

基本数据类型和引用数据类型区别:内存分配不同（自动分配/动态分配）、访问机制不同（值/内存地址）、复制变量不同（值/内存地址）

https://www.cnblogs.com/c2016c/articles/9328725.html

3.apply、call、bind 区别

call、bind、apply这三个函数的第一个参数都是this的指向对象，call和bind之后的参数以逗号分隔开，apply后面的参数以数组的形式传入。

call和apply立即执行，bind返回的是一个函数，不会立即执行，被调用后执行，新函数调用时可继续传参，拼接在bind参数的后面。

https://www.runoob.com/w3cnote/js-call-apply-bind.html

6.new操作符做了哪些事情

1.创建一个空对象；

2.将构造函数的作用域赋给新对象（this指向了新对象）；

3.执行构造函数中的代码（为这个新对象添加属性）；

12.递归

递归相当于一个有终止条件的循环。

尾递归是一种形式, 只是用这种形式表达出的概念可以被某些编译器优化. 尾递归的特殊形式决定了这种递归代码在执行过程中是可以不需要回溯的(通常的递归都是需要回溯的). 如果编译器针对尾递归形式的递归代码作了这种优化, 就可能把原本需要线性复杂度栈内存空间的执行过程用常数复杂度的空间完成.

12.继承

A对象通过继承 B 对象，就能直接拥有 B 对象的所有属性和方法。

函数的原型对象constructor默认指向函数本身，原型对象除了有原型属性外，为了实现继承，还有一个原型链指针proto，该指针指向上一层的原型对象，而上一层的原型对象的结构依然类似，这样利用proto一直指向Object的原型对象上，而Object的原型对象用Object.proto = null表示原型链的最顶端，如此变形成了javascript的原型链继承，同时也解释了为什么所有的javascript对象都具有Object的基本方法。

原型链继承（可继承父类原型上的属性，但是继承单一、无法传参，且所有新实例不会继承父类实例的属性，共享父类实例的属性，一个改了原型的属性，所有的都改）

借用构造函数继承（解决了原型链继承的缺点，且可以继承多个构造函数的属性，但是不能继承父类原型上的属性，且无法复用，且每个新实例都有父类构造函数的副本，臃肿）

组合继承（组合原型链和借用构造函数继承）(常用)（可继承父类原型上的属性，可传参，可复用，每个新实例引入的构造函数属性都是私有的，但是调用了两次父类构造函数（耗内存），子类的构造函数会代替原型上的那个父类构造函数）

原型式继承即用函数包装一个对象，然后返回这个函数的调用。（所有实例都会继承原型上的属性，无法实现复用）

寄生式继承（没有创建自定义类型，因为只是套了个盒子返回对象，这个函数顺理成章成了创建的新对象，但是没有用到原型，无法复用）

寄生组合式继承（常用）（函数的原型等于另一个实例，在函数中call或apply引入另一个构造函数，可传参，修复了组合继承的问题）

https://www.cnblogs.com/ranyonsue/p/11201730.html

class类继承　class　a　extends　A () {}

4.原型和原型链

原型：在JavaScript中，每当定义一个函数数据类型(普通函数、类)时候，都会天生自带一个`prototype`属性，这个属性指向函数的原型对象，并且这个属性是一个对象数据类型的值。

原型链：每一个对象数据类型(普通的对象、实例、prototype.....)也天生自带一个属性__proto__，这个属性指向构造函数的原型对象。原型对象中有一个属性constructor, 它指向函数对象。

链接：https://www.jianshu.com/p/ddaa5179cda6

1. prototype （每个函数都有这个属性）
2. __proto__ （每个对象都有这个属性，函数也是对象，指向构造函数的原型对象）
3. constructor （原型对象上的一个指向构造函数的属性）
   + **Function**.prototype （**Function**原型对象）-> .__proto__ ->指向 **Object**.prototype -> .__proto__ -> null

5.闭包

闭包就是有权访问另一个函数作用域中的变量的函数。

在js中变量的作用域属于函数作用域, 在函数执行完后,作用域就会被清理,内存也会随之被回收,但是由于闭包函数是建立在函数内部的子函数, 由于其可访问上级作用域,即使上级函数执行完, 作用域也不会随之销毁, 这时的子函数(也就是闭包),便拥有了访问上级作用域中变量的权限,即使上级函数执行完后作用域内的值也不会被销毁。

闭包解决了什么问题：可以读取函数内部的变量；可以让变量的值一直保存在内存中，不会在函数调用后被清除；

优点：实现共有变量； 可以做缓存； 可以实现封装，属性私有化； 模块化开发，防止污染全局变量；

缺点：内存泄漏： 由于闭包会是的函数中的变量都被保存到内存中,滥用闭包很容易造成内存消耗过大,导致网页性能问题。解决方法是在退出函数之前，将不再使用的局部变量全部删除；

变量变化问题：如果你把父函数当作对象(object)使用，把闭包当作它的公用方法(Public Method)，把内部变量当作它的私有属性(private value)，这时一定要小心，不要随便改变父函数内部变量的值。

5.垃圾清理机制

原理： 找出那些不再继续使用的变量，然后释放其占用的内存，垃圾收集器会按照固定的时间间隔，或是咱们在代码预订的收集时间，去周期性的执行这个操作，完成垃圾的清理。

标记清除： 当变量进入环境时，如在函数中var一个变量，此时将这个变量标记为进入环境，当变量离开环境的时候，则将其标记为离开环境，可以通过翻转某一个位来标记一个变量何时进入了环境。但标记不是重点，重点是标记了之后怎么来将其处理。垃圾收集器会在运行的时候给存储在内存中的所有变量都加上标记，然后，它会去掉环境中的变量以及被环境中的变量应用的标记，在此之后再把加上标记的变量都将被视为准备删除的变量。最后，垃圾收集器完成内存的清除工作，销毁那些带标记的值并收回它们所占用的内存空间。

引用计数： 跟踪记录每个值被引用的次数，当这个值的引用次数变成0的时候，说明没有办法再访问这个这个值，就将其占用的内存空间收回来，下次再运行垃圾收集器的时候，就会释放哪些引用次数为0的值所占用的内存了。

16.内存泄漏

内存泄漏指不再用到的内存没有及时释放。

引起内存泄漏原因：

1. setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。
2. 闭包
3. 控制台日志
4. 循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）
5. 意外的全局变量
6. 不清除定时器

避免策略：

1. 减少不必要的全局变量，或者生命周期较长的对象，及时对无用的数据进行垃圾回收(**即赋值为null**)；
2. 注意程序逻辑，避免“死循环”之类的 ；
3. 避免创建过多的对象 原则：**不用了的东西要记得及时归还；**
4. 减少层级过多的引用；

17.作用域和作用域链

作用域分为全局作用域和函数作用域；

全局作用域：在程序的任何地方都能被访问，window对象的内置属性都拥有全局作用域；

函数作用域：在固定的代码片段才能被访问；

作用域链：一般情况下，变量取值到创建这个变量的函数的作用域中取值，但是如果在当前作用域中没有查到值，就会向上级作用域去查，直到查到全局作用

域，这个查找过程形成的链条就叫作用域链。

9.事件绑定

行内绑定（DOM上绑定）： <input type='button' id="int" onclick='display()' /> ； this指向全局window对象； 缺点：不利于后期维护；

动态绑定（JavaScript中代码绑定）： document.getElementById('int').onclick = function() {}；this指向正在操作的dom对象

事件监听绑定： document.getElemengById('int').addEventListener('click', function(){});

11.事件监听

常用event.addEventListener()进行事件监听，但是当页面中存在大量需要绑定事件的元素是，这种方式可能会带来性能影响。此时，我们可以用事件委托的方式来进行事件的监听。

11.JavaScript事件的三个阶段

事件捕获阶段：事件源依次从defaultView（可以理解为整个页面document），一直传播到具体的目标（点击目标target），从面到点。

事件目标阶段：就是点击的那个点，目标节点。

事件冒泡阶段：事件源依次从target传播到defaultView，范围越来越大，像冒泡一样，气泡越来越大，从点到面。

https://www.jianshu.com/p/925fbeb538b9

10.事件委托

利用事件冒泡，只需要在父元素上绑定一个事件处理程序，就可以管理一类子元素的所有事件。

优点：1.管理的函数变少了。不需要为每个元素都添加监听函数。对于同一个父节点下面类似的子元素，可以通过委托给父元素的监听函数来处理事件。 2.可以方便地动态添加和修改元素，不需要因为元素的改动而修改事件绑定。

3.JavaScript和DOM节点之间的关联变少了，这样也就减少了因循环引用而带来的内存泄漏发生的概率。

```
//事件委托，添加的子元素也有事件
oUl.onmouseover = function(ev){
    var ev = ev || window.event;
    var target = ev.target || ev.srcElement;
    if(target.nodeName.toLowerCase() == 'li'){
        target.style.background = "red";
    }

};
```

12.事件循环

同步任务和异步任务分别进入不同的执行环境。同步的进入主线程，异步的进入任务队列。主线程的任务执行完毕，回去任务队列读取对应的任务，推入主线程执行。这个过程的不断重复就是时间循环。

12.深浅拷贝

深拷贝和浅拷贝是只针对Object和Array这样的引用数据类型的。

浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存。但深拷贝会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会改到原对象。

object.assign() 浅拷贝

json.parse(json.stringfy()) 深拷贝 会出错（NaN、Infinity、Infinity转为null、Date()转为字符串非日期对象、函数转为undefined）

如何实现深拷贝：jQuery：$.extend([deep], target, object1 [, objectN])

原生JS：递归

扩展运算符（...）

17.常用的ES6特性

let、const
promise
函数参数的默认值
箭头函数
模板字符串
class类继承
export和import 导出导入模块
展开运算符 ...
对象键值对重名的简写
对象方法的简写
Object.assign()实现浅复制
解构赋值（简化提取数组或对象中的值）
set数组去重

13.数组去重

set(ES6)

```
[...new  Set(arr)]    /
function  unique(arr) {
   return Array.form(new  Set(arr))
}
```

for 嵌套 for， splice去重（ES5）

```
function  unique(arr) {
   for  (var i = 0;  i < arr.length;  i++)  {
      for  (var j = i+1;  j < arr.length;  j++)  {
         if (arr[i] === arr[j]) {
            arr.splice(j, 1);
            j--;
         }
      }
   }
}
```

indexOf / includes
sort
filter
递归
https://segmentfault.com/a/1190000016418021

14.数组排序

list.sort((a, b) => a - b) 从小到大
list.sort((a, b) => b - a) 从大到小

15.数组降维

concat

```
function reduce(arr) {
    var reduced = [];
    for（var i = 0;  i < arr.length;  i++）{
      reduced = reduced.concat(arr[i])
    }
    return reduced
}
```

prototype

```
Array.prototype.concat.apply([], arr)
```

join

```
arr.join(",").split(",")
```

17.数字

parseInt （去掉小数部分）
parseFloat（获取浮点数）
Math.ceil（向上取整）正的方向前进
Math.floor（向下取整）负的方向前进
Math.round（四舍五入取整）

10.==和=== 区别

==是值判断，会调用隐式类型转换，不判断类型；
===是值判断加类型判断，判断两个值是否严格相等；

11.typeof和instanceof 区别

typeof用于判断数据类型，返回值为6个字符串，string、boolean、number、function、object、undefined;
instanceof用来判断对象，根据对象的原型链依次向下查询；

7.宏任务和微任务

宏任务：script（全局任务），setTimeout， setInterval， setImmediate， I/O， UI rendering
微任务：process.nextTick（优先级高） （node.js中进程相关的对象），Promise， Object.observer， MutationObserver

8.防抖、节流

debounce 防抖（指触发事件后在规定时间内回调函数只能执行一次）

```
function debounce (fn,  wait) {
    let  timer = null;
    return  function( ) {
        if(timer) {
            clearTimeout (timer)
        }
         const args = arguments;
        timer = setTimeout(() => {
            fn.apply(this,  args)
        },  wait)
    }

}
```

throttle 节流（当持续触发事件时，在规定时间段内只能调用一次回调函数）

```
function throttle(fn,  wait=50) {
    let  prev = new  Date();
    return  function( )  {
        const args = arguments;
        const  now = new  Date();
        if(now - prev  >  wait)  {
            fn.apply(this,  args)
            prev = new  Date();
        }
    }
}
```