**1、介绍js的基本数据类型。**

 Undefined、Null、Boolean、Number、String、

 ECMAScript 2015 新增:Symbol(创建后独一无二且不可变的数据类型 )

**2、介绍js有哪些内置对象？**

Object 是 JavaScript 中所有对象的父对象

数据封装类对象：Object、Array、Boolean、Number 和 String

其他对象：Function、Arguments、Math、Date、RegExp、Error

**3、说几条写JavaScript的基本规范？**

（1）不要在同一行声明多个变量。

（2）请使用 ===/!==来比较true/false或者数值

（3）使用对象字面量替代new Array这种形式

（4）不要使用全局函数。

（5）Switch语句必须带有default分支

（6）函数不应该有时候有返回值，有时候没有返回值。

（7）For循环必须使用大括号

（8）If语句必须使用大括号

（9）for-in循环中的变量应该使用var关键字明确限定作用域，从而避免作用域污染。

**4、JavaScript原型，原型链 ? 有什么特点？**

每个对象都会在其内部初始化一个属性，就是prototype(原型)，当我们访问一个对象的属性时，

如果这个对象内部不存在这个属性，那么他就会去prototype里找这个属性，这个prototype又会有自己的prototype，

于是就这样一直找下去，也就是我们平时所说的原型链的概念。

关系：instance.constructor.prototype = instance.\_\_proto\_\_

特点：

JavaScript对象是通过引用来传递的，我们创建的每个新对象实体中并没有一份属于自己的原型副本。当我们修改原型时，与之相关的对象也会继承这一改变。

 当我们需要一个属性的时，Javascript引擎会先看当前对象中是否有这个属性， 如果没有的话，

 就会查找他的Prototype对象是否有这个属性，如此递推下去，一直检索到 Object 内建对象。

   function Func(){}

   Func.prototype.name = "Sean";

   Func.prototype.getInfo = function() {

     return this.name;

    }

   var person = new Func();//现在可以参考var person = Object.create(oldObject);

   console.log(person.getInfo());//它拥有了Func的属性和方法

   //"Sean"

   console.log(Func.prototype);

// Func {name="Sean", getInfo=function()}

**5、JavaScript有几种类型的值？，你能画一下他们的内存图吗？**

栈：原始数据类型（Undefined，Null，Boolean，Number、String）

堆：引用数据类型（对象、数组和函数）

两种类型的区别是：存储位置不同；

原始数据类型直接存储在栈(stack)中的简单数据段，占据空间小、大小固定，属于被频繁使用数据，所以放入栈中存储；

引用数据类型存储在堆(heap)中的对象,占据空间大、大小不固定。如果存储在栈中，将会影响程序运行的性能；引用数据类型在栈中存储了指针，该指针指向堆中该实体的起始地址。当解释器寻找引用值时，会首先检索其在栈中的地址，取得地址后从堆中获得实体

**6、如何将字符串转化为数字，例如'12.3b'?**

\* parseFloat('12.3b');

\* 正则表达式，'12.3b'.match(/(\d)+(\.)?(\d)+/g)[0]\* 1, 但是这个不太靠谱，提供一种思路而已。

**7、如何将浮点数点左边的数每三位添加一个逗号，如12000000.11转化为『12,000,000.11』?**

function commafy(num){

   return num && num

       .toString()

       .replace(/(\d)(?=(\d{3})+\.)/g, function($1, $2){

           return $2 + ',';

       });

}

**8、如何实现数组的随机排序？**

方法一： ```javascriptvar arr = [1,2,3,4,5,6,7,8,9,10]; function randSort1(arr){ for(var i = 0,len =arr.length;i < len; i++ ){ var rand = parseInt(Math.random()\*len); var temp= arr[rand]; arr[rand] = arr[i]; arr[i] = temp; } return arr; } console.log(randSort1(arr));

```

方法二：

```javascript

       var arr = [1,2,3,4,5,6,7,8,9,10];

       function randSort2(arr){

           var mixedArray = [];

           while(arr.length > 0){

                var randomIndex =parseInt(Math.random()\*arr.length);

               mixedArray.push(arr[randomIndex]);

                arr.splice(randomIndex, 1);

           }

           return mixedArray;

       }

       console.log(randSort2(arr));

```

方法三：

```javascript

       var arr = [1,2,3,4,5,6,7,8,9,10];

       arr.sort(function(){

           return Math.random() - 0.5;

       })

       console.log(arr);

```

**9、Javascript如何实现继承？**

（1）构造继承

（2）原型继承

（3）实例继承

（4）拷贝继承

原型prototype机制或apply和call方法去实现较简单，建议使用构造函数与原型混合方式。

```javascript

       function Parent(){

           this.name = 'wang';

       }

       function Child(){

           this.age = 28;

       }

       Child.prototype = new Parent();//继承了Parent，通过原型

       var demo = new Child();

       alert(demo.age);

       alert(demo.name);//得到被继承的属性

```

**10、javascript创建对象的几种方式？**

javascript创建对象简单的说,无非就是使用内置对象或各种自定义对象，当然还可以用JSON；但写法有很多种，也能混合使用。

（1）对象字面量的方式

   person={firstname:"Mark",lastname:"Yun",age:25,eyecolor:"black"};

（2）用function来模拟无参的构造函数

   function Person(){}

   var person=new Person();//定义一个function，如果使用new"实例化",该function可以看作是一个Class

   person.name="Mark";

   person.age="25";

   person.work=function(){

   alert(person.name+" hello...");

    }

   person.work();

（3）用function来模拟参构造函数来实现（用this关键字定义构造的上下文属性）

   function Pet(name,age,hobby){

      this.name=name;//this作用域：当前对象

      this.age=age;

      this.hobby=hobby;

      this.eat=function(){

         alert("我叫"+this.name+",我喜欢"+this.hobby+",是个程序员");

      }

    }

   var maidou =new Pet("麦兜",25,"coding");//实例化、创建对象

   maidou.eat();//调用eat方法

（4）用工厂方式来创建（内置对象）

    var wcDog =new Object();

    wcDog.name="旺财";

    wcDog.age=3;

    wcDog.work=function(){

      alert("我是"+wcDog.name+",汪汪汪......");

    }

    wcDog.work();

（5）用原型方式来创建

   function Dog(){

    }

    Dog.prototype.name="旺财";

    Dog.prototype.eat=function(){

    alert(this.name+"是个吃货");

    }

    var wangcai =new Dog();

    wangcai.eat();

（6）用混合方式来创建

   function Car(name,price){

     this.name=name;

     this.price=price;

    }

    Car.prototype.sell=function(){

      alert("我是"+this.name+"，我现在卖"+this.price+"万元");

     }

   var camry =new Car("凯美瑞",27);

camry.sell();

**11、Javascript作用链域?**

全局函数无法查看局部函数的内部细节，但局部函数可以查看其上层的函数细节，直至全局细节。

当需要从局部函数查找某一属性或方法时，如果当前作用域没有找到，就会上溯到上层作用域查找，

直至全局函数，这种组织形式就是作用域链。

**12、谈谈This对象的理解。**

this总是指向函数的直接调用者（而非间接调用者）；

如果有new关键字，this指向new出来的那个对象；

在事件中，this指向触发这个事件的对象，特殊的是，IE中的attachEvent中的this总是指向全局对象Window；

**13、eval是做什么的？**

它的功能是把对应的字符串解析成JS代码并运行；

应该避免使用eval，不安全，非常耗性能（2次，一次解析成js语句，一次执行）。

由JSON字符串转换为JSON对象的时候可以用eval，var obj =eval('('+ str +')');

**14、什么是window对象? 什么是document对象?**

window对象是指浏览器打开的窗口。

document对象是Documentd对象（HTML 文档对象）的一个只读引用，window对象的一个属性。

**15、null，undefined 的区别？**

null       表示一个对象是“没有值”的值，也就是值为“空”；

undefined  表示一个变量声明了没有初始化(赋值)；

undefined不是一个有效的JSON，而null是；

undefined的类型(typeof)是undefined；

null的类型(typeof)是object；

Javascript将未赋值的变量默认值设为undefined；

Javascript从来不会将变量设为null。它是用来让程序员表明某个用var声明的变量时没有值的。

typeof undefined

   //"undefined"

   undefined :是一个表示"无"的原始值或者说表示"缺少值"，就是此处应该有一个值，但是还没有定义。当尝试读取时会返回 undefined；

    例如变量被声明了，但没有赋值时，就等于undefined

typeof null

   //"object"

   null : 是一个对象(空对象, 没有任何属性和方法)；

    例如作为函数的参数，表示该函数的参数不是对象；

注意：

    在验证null时，一定要使用　=== ，因为 == 无法分别 null 和　undefined

   null == undefined // true

    null=== undefined // false

再来一个例子：

   null

    Q：有张三这个人么？

    A：有！

    Q：张三有房子么？

    A：没有！

   undefined

    Q：有张三这个人么？

    A：有！

   Q: 张三有多少岁？

   A: 不知道（没有被告诉）

**16、写一个通用的事件侦听器函数。**

   // event(事件)工具集，来源：github.com/markyun

   markyun.Event = {

       // 页面加载完成后

       readyEvent : function(fn) {

           if (fn==null) {

                fn=document;

           }

           var oldonload = window.onload;

           if (typeof window.onload != 'function') {

                window.onload = fn;

           } else {

                window.onload = function() {

                    oldonload();

                    fn();

                };

           }

       },

       // 视能力分别使用dom0||dom2||IE方式来绑定事件

       // 参数：操作的元素,事件名称 ,事件处理程序

       addEvent : function(element, type, handler) {

           if (element.addEventListener) {

                //事件类型、需要执行的函数、是否捕捉

                element.addEventListener(type,handler, false);

           } else if (element.attachEvent) {

                element.attachEvent('on' +type, function() {

                    handler.call(element);

                });

           } else {

                element['on' + type] = handler;

           }

       },

       // 移除事件

       removeEvent : function(element, type, handler) {

           if (element.removeEventListener) {

               element.removeEventListener(type, handler, false);

           } else if (element.datachEvent) {

                element.detachEvent('on' +type, handler);

           } else {

                element['on' + type] = null;

           }

       },

       // 阻止事件 (主要是事件冒泡，因为IE不支持事件捕获)

       stopPropagation : function(ev) {

           if (ev.stopPropagation) {

                ev.stopPropagation();

           } else {

                ev.cancelBubble = true;

           }

       },

       // 取消事件的默认行为

       preventDefault : function(event) {

           if (event.preventDefault) {

                event.preventDefault();

           } else {

                event.returnValue = false;

           }

       },

       // 获取事件目标

       getTarget : function(event) {

           return event.target || event.srcElement;

       },

       // 获取event对象的引用，取到事件的所有信息，确保随时能使用event；

       getEvent : function(e) {

           var ev = e || window.event;

           if (!ev) {

                var c = this.getEvent.caller;

                while (c) {

                    ev = c.arguments[0];

                    if (ev && Event ==ev.constructor) {

                        break;

                    }

                    c = c.caller;

                }

           }

           return ev;

       }

};

**17、["1","2", "3"].map(parseInt) 答案是多少？**

parseInt() 函数能解析一个字符串，并返回一个整数，需要两个参数 (val, radix)，

其中 radix 表示要解析的数字的基数。【该值介于 2 ~ 36 之间，并且字符串中的数字不能大于radix才能正确返回数字结果值】;

但此处 map 传了 3 个 (element, index, array),我们重写parseInt函数测试一下是否符合上面的规则。

function parseInt(str, radix) {

   return str+'-'+radix;

};

var a=["1", "2","3"];

a.map(parseInt);  // ["1-0", "2-1","3-2"] 不能大于radix

因为二进制里面，没有数字3,导致出现超范围的radix赋值和不合法的进制解析，才会返回NaN

所以["1", "2", "3"].map(parseInt) 答案也就是：[1,NaN, NaN]

**18、事件是？IE与火狐的事件机制有什么区别？ 如何阻止冒泡？**

 （1）我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被 JavaScript 侦测到的行为。

 （2）事件处理机制：IE是事件冒泡、Firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件；

 （3）ev.stopPropagation();（旧ie的方法ev.cancelBubble = true;）

**19、什么是闭包（closure），为什么要用它？**

闭包是指有权访问另一个函数作用域中变量的函数，创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量,利用闭包可以突破作用链域，将函数内部的变量和方法传递到外部。

闭包的特性：

（1）函数内再嵌套函数

（2）内部函数可以引用外层的参数和变量

（3）参数和变量不会被垃圾回收机制回收

//li节点的onclick事件都能正确的弹出当前被点击的li索引

 <ul id="testUL">

   <li> index = 0</li>

   <li> index = 1</li>

   <li> index = 2</li>

   <li> index = 3</li>

</ul>

<script type="text/javascript">

   var nodes = document.getElementsByTagName("li");

   for(i = 0;i<nodes.length;i+= 1){

       nodes[i].onclick = (function(i){

                  return function() {

                     console.log(i);

                  } //不用闭包的话，值每次都是4

                })(i);

    }

</script>

执行say667()后,say667()闭包内部变量会存在,而闭包内部函数的内部变量不会存在

使得Javascript的垃圾回收机制GC不会收回say667()所占用的资源

因为say667()的内部函数的执行需要依赖say667()中的变量

这是对闭包作用的非常直白的描述

 function say667() {

   // Local variable that ends up within closure

   var num = 666;

   var sayAlert = function() {

       alert(num);

    }

   num++;

   return sayAlert;

}

 varsayAlert = say667();

 sayAlert()//执行结果应该弹出的667

**20、javascript 代码中的"usestrict";是什么意思 ? 使用它区别是什么？**

use strict是一种ECMAscript 5 添加的（严格）运行模式,这种模式使得Javascript 在更严格的条件下运行,使JS编码更加规范化的模式,消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为。

默认支持的糟糕特性都会被禁用，比如不能用with，也不能在意外的情况下给全局变量赋值;

全局变量的显示声明,函数必须声明在顶层，不允许在非函数代码块内声明函数,arguments.callee也不允许使用；

消除代码运行的一些不安全之处，保证代码运行的安全,限制函数中的arguments修改，严格模式下的eval函数的行为和非严格模式的也不相同;

提高编译器效率，增加运行速度；

为未来新版本的Javascript标准化做铺垫。