# JavaScript基础面试题

## 1. 介绍JavaScript的基本数据类型

Number、String 、Boolean 、Null、Undefined
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
新类型：Symbol

## 2. 说说写JavaScript的基本规范?

1) 不要在同一行声明多个变量
2) 使用 ===或!==来比较true/false或者数值
3) switch必须带有default分支
4) 函数应该有返回值
5) for if else 必须使用大括号
6) 语句结束加分号
7) 命名要有意义，使用驼峰命名法

## 3. jQuery使用建议

1) 尽量减少对dom元素的访问和操作
2) 尽量避免给dom元素绑定多个相同类型的事件处理函数，可以将多个相同类型事件处理函数合并到一个处理函数，通过数据状态来处理分支
3) 尽量避免使用toggle事件

## 4. Ajax使用

全称： Asynchronous Javascript And XML
所谓异步，就是向服务器发送请求的时候，我们不必等待结果，而是可以同时做其他的事情，等到有了结果它自己会根据设定进行后续操作，与此同时，页面是不会发生整页刷新的，提高了用户体验。
创建Ajax的过程：
1) 创建XMLHttpRequest对象（异步调用对象）

```
var xhr = new XMLHttpRequest();
```

2) 创建新的Http请求（方法、URL、是否异步）

```
xhr.open('get','example.php',false);
```

3) 设置响应HTTP请求状态变化的函数。

onreadystatechange事件中readyState属性等于4。响应的HTTP状态为200(OK)或者304(Not Modified)。

4) 发送http请求

```
xhr.send(data);
```

5) 获取异步调用返回的数据

注意：

1) 页面初次加载时，尽量在web服务器一次性输出所有相关的数据，只在页面加载完成之后，用户进行操作时采用ajax进行交互。

2) 同步ajax在IE上会产生页面假死的问题。所以建议采用异步ajax。

3) 尽量减少ajax请求次数

4) ajax安全问题，对于敏感数据在服务器端处理，避免在客户端处理过滤。对于关键业务逻辑代码也必须放在服务器端处理。

## 5. JavaScript有几种类型的值？你能画一下他们的内存图吗？

基本数据类型存储在栈中，引用数据类型（对象）存储在堆中，指针放在栈中。
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. 栈和堆的区别?

栈（stack）：由编译器自动分配释放，存放函数的参数值，局部变量等；
堆（heap）：一般由程序员分配释放，若程序员不释放，程序结束时可能由操作系统释放。

## 7. Javascript实现继承的几种方式

1）借用构造函数法（又叫经典继承）

```
function SuperType(name) {
    this.name = name;
    this.sayName = function() {
        window.alert(this.name);
    };
}

function SubType(name, age) {
    SuperType.call(this, name); //在这里借用了父类的构造函数
    this.age = age;
}
```

2）对象冒充

```
function SuperType(name) {
```

```
    this.name = name;

   this.sayName = function() {
        window.alert(this.name);
    };
}

function SubType(name, age) {
    this.supertype = SuperType; //在这里使用了对象冒充
    this.supertype(name);

    this.age = age;
}
```

3）组合继承（最常用）

```
function SuperType(name) {
    this.name = name;
}

SuperType.prototype = {

    sayName : function() {
        window.alert(this.name);
    }
};

function SubType(name, age) {

    SuperType.call(this, name); //在这里继承属性
    this.age = age;
}

SubType.prototype = new SuperType(); //这里继承方法
```

组合继承的方法是对应着我们用'组合使用构造函数和原型方法'定义父类的一种继承方法。同样的，我们的属性和方法是分开继承的。

## 8. Javascript创建对象的几种方式?

JavaScript定义类的4种方法

**1）工厂方法**

```
function creatPerson(name, age) {
    var obj = new Object();

    obj.name = name;
    obj.age = age;

    obj.sayName = function() {
        window.alert(this.name);
    };
    return obj;
}
```

#### 2）构造函数方法

```
function Person(name, age) {

    this.name = name;
    this.age = age;

    this.sayName = function() {
        window.alert(this.name);
    };
}
```

#### 3）原型方法

```
function Person() {

}

Person.prototype = {
    constructor : Person,
    name : "Ning",
    age : "23",
    sayName : function() {
        window.alert(this.name);
    }
};
```

可以看到这种方法有缺陷，类里属性的值都是在原型里给定的。

#### 4）组合使用构造函数和原型方法（使用最广）

```
function Person(name, age) {
    this.name = name;
    this.age = age;
}

Person.prototype = {
    constructor : Person,
    sayName : function() {
        window.alert(this.name);
    }
};
```

将构造函数方法和原型方法结合使用是目前最常用的定义类的方法。这种方法的好处是实现了属性定义和方法定义的分离。比如我可以创建两个对象 person1 和 person2 ，它们分别传入各自的 name 值和 age 值，但 sayName() 方法可以同时使用原型里定义的。

## 9. Javascript作用链域

作用域链的原理和原型链很类似，如果这个变量在自己的作用域中没有，那么它会寻找父级的，直到最顶层。

注意：JS没有块级作用域，若要形成块级作用域，可通过（function（） {} ） （）；立即执行的形式实现。

## 10. 谈谈this的理解

1) this总是指向函数的直接调用者（而非间接调用者）

2) 如果有new关键字，this指向new出来的那个对象

3) 在事件中，this指向目标元素，特殊的是IE的attachEvent中的this总是指向全局对象window。

## 11. 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）。

## 12. 什么是window对象? 什么是document对象?

window对象代表浏览器中打开的一个窗口。document对象代表整个html文档。实际上，document对象是window对象的一个属性。

## 13. null，undefined的区别?

null表示一个对象被定义了，但存放了空指针，转换为数值时为0。

undefined表示声明的变量未初始化，转换为数值时为NAN。

typeof(null) -- object;

typeof(undefined) -- undefined

## 14. ["1", "2", "3"].map(parseInt) 答案是多少?

[1,NaN,NaN]

解析：

Array.prototype.map()

array.map(callback[, thisArg])

callback函数的执行规则

参数：自动传入三个参数

currentValue（当前被传递的元素）；

index（当前被传递的元素的索引）；

array（调用map方法的数组）

parseInt方法接收两个参数

第三个参数["1", "2", "3"]将被忽略。parseInt方法将会通过以下方式被调用

parseInt("1", 0)

parseInt("2", 1)

parseInt("3", 2)

parseInt的第二个参数radix为0时，ECMAScript5将string作为十进制数字的字符串解析；

parseInt的第二个参数radix为1时，解析结果为NaN；

parseInt的第二个参数radix在2—36之间时，如果string参数的第一个字符（除空白以外），不属于radix指定进制下的字符，解析结果为NaN。

parseInt("3", 2)执行时，由于"3"不属于二进制字符，解析结果为NaN。

## 15. 关于事件，IE与火狐的事件机制有什么区别？ 如何阻止冒泡?

IE为事件冒泡，Firefox同时支持事件捕获和事件冒泡。但并非所有浏览器都支持事件捕获。jQuery中使用 event.stopPropagation() 方法可阻止冒泡；(旧IE的方法 ev.cancelBubble = true; )

## 16. 什么是闭包（closure），为什么要用它?

闭包指的是一个函数可以访问另一个函数作用域中变量。常见的构造方法，是在一个函数内部定义另外一个函数。内部函数可以引用外层的变量；外层变量不会被垃圾回收机制回收。

注意，闭包的原理是作用域链，所以闭包访问的上级作用域中的变量是个对象，其值为其运算结束后的最后一个值。

优点：避免全局变量污染。缺点：容易造成内存泄漏。

例子：

```
function makeFunc() {
    var name = "Mozilla";
    function displayName() {
        console.log(name);
    }
    return displayName;
}

var myFunc = makeFunc();
myFunc(); //输出Mozilla
```

myFunc 变成一个 闭包。闭包是一种特殊的对象。它由两部分构成：函数，以及创建该函数的环境。环境由闭包创建时在作用域中的任何局部变量组成。在我们的例子中，myFunc 是一个闭包，由 displayName 函数和闭包创建时存在的 "Mozilla" 字符串形成。

## 17. javascript 代码中的"use strict";是什么意思？使用它区别是什么?

除了正常模式运行外，ECMAscript添加了第二种运行模式："严格模式"。

作用：

1) 消除js不合理，不严谨地方，减少怪异行为

2) 消除代码运行的不安全之处，

3) 提高编译器的效率，增加运行速度

4) 为未来的js新版本做铺垫。

## 18. 如何判断一个对象是否属于某个类?

使用instanceof 即if(a instanceof Person){alert('yes');}

## 19. new操作符具体干了什么呢?

1) 创建一个空对象，并且 this 变量引用该对象，同时还继承了该函数的原型。

2) 属性和方法被加入到 this 引用的对象中。

3) 新创建的对象由 this 所引用，并且最后隐式的返回 this 。

## 20. Javascript中，执行时对象查找时，永远不会去查找原型的函数?

Object.hasOwnProperty(proName)：是用来判断一个对象是否有你给出名称的属性。不过需要注意的是，此方法无法检查该对象的原型链中是否具有该属性，该属性必须是对象本身的一个成员。

## 21. 对JSON的了解?

全称：JavaScript Object Notation
JSON中对象通过“{}”来标识，一个“{}”代表一个对象，如{“AreaId”:”123”}，对象的值是键值对的形式（key：value）。JSON是JS的一个严格的子集，一种轻量级的数据交换格式，类似于xml。数据格式简单，易于读写，占用带宽小。

两个函数：

JSON.parse(str)

解析JSON字符串 把JSON字符串变成JavaScript值或对象

JSON.stringify(obj)

将一个JavaScript值(对象或者数组)转换为一个 JSON字符串

eval('(' + json + ')')

用eval方法注意加括号 而且这种方式更容易被攻击

## 22. JS延迟加载的方式有哪些?

JS的延迟加载有助与提高页面的加载速度。

defer和async、动态创建DOM方式（用得最多）、按需异步载入JS

defer：延迟脚本。立即下载，但延迟执行（延迟到整个页面都解析完毕后再运行），按照脚本出现的先后顺序执行。

async：异步脚本。下载完立即执行，但不保证按照脚本出现的先后顺序执行。

## 23. 同步和异步的区别?

同步的概念在操作系统中：不同进程协同完成某项工作而先后次序调整（通过阻塞、唤醒等方式），同步强调的是顺序性，谁先谁后。异步不存在顺序性。

同步：浏览器访问服务器，用户看到页面刷新，重新发请求，等请求完，页面刷新，新内容出现，用户看到新内容之后进行下一步操作。

异步：浏览器访问服务器请求，用户正常操作，浏览器在后端进行请求。等请求完，页面不刷新，新内容也会出现，用户看到新内容。

## 24. 什么是跨域问题，如何解决跨域问题?

### 1）什么是跨域?

要明白什么是跨域之前，首先要明白什么是同源策略?

同源策略就是用来限制从一个源加载的文档或脚本与来自另一个源的资源进行交互。那怎样判断是否是同源呢?

如果协议，端口（如果指定了）和主机对于两个页面是相同的，则两个页面具有相同的源，也就是同源。也就是说，要同时满足以下3个条件，才能叫同源：

1. 协议相同
2. 端口相同

3. 主机相同

举个例子就一目了然了：

我们来看下面的页面是否与 http://store.company.com/dir/index.html 是同源的?

1. http://store.company.com/dir/index2.html 同源
2. http://store.company.com/dir2/index3.html 同源 虽然在不同文件夹下
3. https://store.company.com/secure.html 不同源 不同的协议(https)
4. http://store.company.com:81/dir/index.html 不同源 不同的端口(81)
5. http://news.company.com/dir/other.html 不同源 不同的主机(news)

所以当面对跨域问题的时候，有什么解决方案呢?

**2）跨域的几种解决方案**

**（1）document.domain方法**

我们来看一个具体场景：有一个页面 http://www.example.com/a.html ，它里面有一个 iframe ，这个 iframe 的源是 http://example.com/b.html ，很显然它们是不同源的，所以我们无法在父页面中操控子页面的内容。

解决方案如下：

```
<!-- b.html -->
<script>
document.domain = 'example.com';
</script>
<!-- a.html -->
<script>
document.domain = 'example.com';
var iframe = document.getElementById('iframe').contentWindow.document;

//后面就可以操作iframe里的内容了...

</script>
```

所以我们只要将两个页面的document.domain设置成一致就可以了，要注意的是，document.domain的设置是有限制的，我们只能把document.domain设置成自身或更高一级的父域。

但是，这种方法只能解决主域相同的跨域问题。

**（2）window.name方法**

window对象有个name属性，该属性有个特征：即在一个窗口(window)的生命周期内,窗口载入的所有的页面都是共享一个window.name的，每个页面对window.name都有读写的权限，window.name是持久存在一个窗口载入过的所有页面中的，并不会因新页面的载入而进行重置。

我们来看一个具体场景，在一个页面 example.com/a.html 中，我们想获取data.com/data.html中的数据，以下是解决方案：

```
<!-- data.html -->
<script>
window.name = 'data'; //这是就是我们需要通信的数据
```

```
</script>
<!-- a.html -->
<html>
<head>
<script>
    function getData () {
        var iframe = document.getElementById('iframe');
        iframe.src = 'example.com/b.html'; // 这里让iframe与父页面同源

        iframe.onload = function () {
            var data = iframe.contentWindow.name; //在这里我们得到了跨域页面中传来的数
据
        };
    }
</script>
</head>
<body>
</body>
</html>
```

**（3）JSONP方法**

JONSP(JSON with Padding)是JSON的一种使用模式。基本原理如下：

```
<!-- a.html -->
<script>
    function dealData (data) {
        console.log(data);
    }
</script>

<script src='http://example.com/data.php?callback=dealData'></script>
<?php
    $callback = $_GET['callback'];
    $data = 'data';
    echo $callback.'('.json_encode($data).')';
?>
```

这时候在 a.html 中我们得到了一条js的执行语句 dealData('data')，从而达到了跨域的目的。

所以JSONP的原理其实就是利用引入 script 不限制源的特点，把处理函数名作为参数传入，然后返回执行语句，仔细阅读以上代码就可以明白里面的意思了。

如果在jQuery中用JSONP的话就更加简单了：

```
<script>
$.getJSON(''http://example.com/data.php?callback=?', function (data) {
console.log(data);
});
</script>
```

注意jQuery会自动生成一个全局函数来替换 callback=? 中的问号，之后获取到数据后又会自动销毁，实际上就是起一个临时代理函数的作用。 $.getJSON 方法会自动判断是否跨域，不跨域的话，就调用普通的ajax方法；跨域的话，则会以异步加载js文件的形式来调用JSONP的回调函数。

## 25. 页面编码和被请求的资源编码如果不一致如何处理?

若请求的资源编码，如外引js文件编码与页面编码不同。可根据外引资源编码方式定义为 charset="utf-8"或"gbk"。

比如：http://www.yyy.com/a.html 中嵌入了一个http://www.xxx.com/test.js

a.html 的编码是gbk或gb2312的。 而引入的js编码为utf-8的 ，那就需要在引入的时候

```
<script src="http://www.xxx.com/test.js&amp;quot; charset="utf-8"></script>
```

## 26. 模块化开发怎么做?

模块化开发指的是在解决某一个复杂问题或者一系列问题时，依照一种分类的思维把问题进行系统性的分解。模块化是一种将复杂系统分解为代码结构更合理，可维护性更高的可管理的模块方式。对于软件行业：系统被分解为一组高内聚，低耦合的模块。

（1）定义封装的模块

（2）定义新模块对其他模块的依赖

（3）可对其他模块的引入支持。在JavaScript中出现了一些非传统模块开发方式的规范。 CommonJS的模块规范，AMD（Asynchronous Module Definition），CMD（Common Module Definition）等。AMD是异步模块定义，所有的模块将被异步加载，模块加载不影响后边语句运行。

## 27. AMD（Modules/Asynchronous-Definition）、CMD（Common Module Definition）规范区别?

AMD 是 RequireJS 在推广过程中对模块定义的规范化产出。CMD 是 SeaJS 在推广过程中对模块定义的规范化产出。

区别：

1) 对于依赖的模块，AMD 是提前执行，CMD 是延迟执行。不过 RequireJS 从 2.0 开始，也改成可以延迟执行（根据写法不同，处理方式不同）。

2) CMD 推崇依赖就近，AMD 推崇依赖前置。

3) AMD 的 API 默认是一个当多个用，CMD 的 API 严格区分，推崇职责单一。

```
// CMD
    define(function(require, exports, module) {
        var a = require('./a')
        a.doSomething()
        // 此处略去 100 行
        var b = require('./b') // 依赖可以就近书写
        b.doSomething()
})

// AMD 默认推荐
    define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好
        a.doSomething();
        // 此处略去 100 行`
```

```
        b.doSomething();
})
```

## 28. requireJS的核心原理是什么？（如何动态加载的？如何避免多次加载的？如何缓存的？）

核心是js的加载模块，通过正则匹配模块以及模块的依赖关系，保证文件加载的先后顺序，根据文件的路径对加载过的文件做了缓存。

## 29. call和apply

call（）方法和apply（）方法的作用相同，动态改变某个类的某个方法的运行环境。他们的区别在于接收参数的方式不同。在使用call（）方法时，传递给函数的参数必须逐个列举出来。使用apply（）时，传递给函数的是参数数组。

## 30. documen.write和 innerHTML的区别

document.write()只能重绘整个页面

```
setTimeout(function(){
        document.write('<p>5 secs later</p>');
    }, 5000);
或
window.onload = function() { document.write("HI");
```

innerHTML可以重绘页面的一部分

## 31. 回流与重绘

当渲染树中的一部分(或全部 )因为元素的规模尺寸，布局，隐藏等改变而需要重新构建。这就称为回流(reflow)。每个页面至少需要一次回流，就是在页面第一次加载的时候。在回流的时候，浏览器会使渲染树中受到影响的部分失效，并重新构造这部分渲染树。完成回流后，浏览器会重新绘制受影响的部分到屏幕中，该过程成为重绘

## 32. DOM操作

### （1）创建新节点

```
createDocumentFragment() //创建一个DOM片段
createElement() //创建一个具体的元素
createTextNode() //创建一个文本节点
```

### （2）添加、移除、替换、插入

```
appendChild()
removeChild()
replaceChild()
insertBefore() //在已有的子节点前插入一个新的子节点
```

**(3) 查找**

```
getElementsByTagName() //通过标签名称
getElementsByName() //通过元素的Name属性的值(IE容错能力较强，会得到一个数组，其中包括id等于name值的)
getElementById() //通过元素Id，唯一性
```

## 33. 数组对象有哪些原生方法，列举一下

pop、push、shift、unshift、splice、reverse、sort、concat、join、slice、toString、indexOf、lastIndexOf、reduce、reduceRight
forEach、map、filter、every、some

## 34. 那些操作会造成内存泄漏

全局变量、闭包、DOM清空或删除时，事件未清除、子元素存在引用

## 35. 什么是Cookie 隔离？（或者：请求资源的时候不要带cookie怎么做）

通过使用多个非主要域名来请求静态文件，如果静态文件都放在主域名下，那静态文件请求的时候带有的cookie的数据提交给server是非常浪费的，还不如隔离开。因为cookie有域的限制，因此不能跨域提交请求，故使用非主要域名的时候，请求头中就不会带有cookie数据，这样可以降低请求头的大小，降低请求时间，从而达到降低整体请求延时的目的。同时这种方式不会将cookie传入server，也减少了server对cookie的处理分析环节，提高了server的http请求的解析速度。

## 36. 响应事件

onclick鼠标点击某个对象；

onfocus获取焦点；

onblur失去焦点；

onmousedown鼠标被按下

## 37. flash和js通过什么类如何交互?

Flash提供了ExternalInterface接口与JavaScript通信，ExternalInterface有两个方法，call和addCallback，call的作用是让Flash调用js里的方法，addCallback是用来注册flash函数让js调用。

## 38. Flash与Ajax各自的优缺点?

Flash：适合处理多媒体、矢量图形、访问机器。但对css、处理文本不足，不容易被搜索。
Ajax：对css、文本支持很好，但对多媒体、矢量图形、访问机器不足。

## 39. 有效的javascript变量定义规则

第一个字符必须是一个字母、下划线（_）或一个美元符号（$）；

其他字符可以是字母、下划线、美元符号或数字。

## 40. XML与JSON的区别?

1) 数据体积方面。JSON相对于XML来讲，数据的体积小，传递的速度更快些。

2) 数据交互方面。JSON与JavaScript的交互更加方便，更容易解析处理，更好的数据交互。

3) 数据描述方面。JSON对数据的描述性比XML较差。

4) 传输速度方面。JSON的速度要远远快于XML。

## 41. HTML与XML的区别?

（1）XML用来传输和存储数据，HTML用来显示数据；

（2）XML使用的标签不用预先定义

（3）XML标签必须成对出现

（4）XML对大小写敏感

（5）XML中空格不会被删减

（6）XML中所有特殊符号必须用编码表示

（7）XML中的图片必须有文字说明

## 42. 渐进增强与优雅降级

渐进增强：针对低版本浏览器进行构建页面，保证最基本的功能，然后再针对高级浏览器进行效果、交互等改进，达到更好的用户体验。

优雅降级：一开始就构建完整的功能，然后再针对低版本浏览器进行兼容。

## 43. Web Worker和Web Socket?

web socket：在一个单独的持久连接上提供全双工、双向的通信。使用自定义的协议（ws://、wss://），同源策略对web socket不适用。

web worker：运行在后台的JavaScript，不影响页面的性能。

创建worker：var worker = new Worker(url);

向worker发送数据：worker.postMessage(data);

接收worker返回的数据：worker.onmessage

终止一个worker的执行：worker.terminate();

## 44. JS垃圾回收机制?

**1) 标记清除：**

这个算法把“对象是否不再需要”简化定义为“对象是否可以获得”。

这个算法假定设置一个叫做根（root）的对象（在Javascript里，根是全局对象）。定期的，垃圾回收器将从根开始，找所有从根开始引用的对象，然后找这些对象引用的对象。从根开始，垃圾回收器将找到所有可以获得的对象和所有不能获得的对象。

**2) 引用计数：**
这是最简单的垃圾收集算法。此算法把“对象是否不再需要”简化定义为“对象有没有其他对象引用到它”。如果没有引用指向该对象（零引用），对象将被垃圾回收机制回收。

该算法有个限制：无法处理循环引用。两个对象被创建，并互相引用，形成了一个循环。它们被调用之后不会离开函数作用域，所以它们已经没有用了，可以被回收了。然而，引用计数算法考虑到它们互相都有至少一次引用，所以它们不会被回收。

## 45. web应用从服务器主动推送data到客户端的方式?

JavaScript数据推送：commet（基于http长连接的服务器推送技术）。
基于web socket的推送：SSE（server-send Event）

## 46. 如何删除一个cookie?

1） 将cookie的失效时间设置为过去的时间（expires）

```
document.cookie = ‘user=’+ encodeURIComponent(‘name’) + ';
expires=’+ new Date(0);
```

2） 将系统时间设置为当前时间往前一点时间

```
var data = new Date();
date.setDate(date.getDate()-1);
```

## 47. attribute与property的区别?

attribute是dom元素在文档中作为html标签拥有的属性

property是dom元素在js中作为对象拥有的属性。

所以，对于html的标准属性来说，attribute和property是同步的，是会自动更新的。但对于自定义属性，他们不同步。

## 48. Ajax请求的页面历史记录状态问题?

（1）通过location.hash记录状态，让浏览器记录Ajax请求时页面状态的变化。

（2）通过HTML5的history.pushstate，来实现浏览器地址栏的无刷新改变。