数组去重

```
function uniq(array){
    var temp = []; //一个新的临时数组
    for(var i = 0; i < array.length; i++){
        if(temp.indexOf(array[i]) == -1){
            temp.push(array[i]);
        }
    }
    return temp;
}

var aa = [1,2,2,4,9,6,7,5,2,3,5,6,5];
console.log(aa)
console.log(uniq(aa))
```

1、谈一谈JavaScript作用域链

- 当执行一段JavaScript代码（全局代码或函数）时，JavaScript引擎会创建一个作用域又称为执行上下文（Execution Context）,在页面加载后会首先创建一个全局的作用域，然后每执行一个函数，会建立一个对应的作用域，从而形成一条作用域链。每个作用域都有一条对应的作用域链，链头是全局作用域，链尾是当前函数作用域。
- 作用域链的作用是用于解析标识符，当函数被创建时（不是执行），会将this,arguments,命名参数和该函数中所有局部变量添加到该当前作用域中，当JavaScript需要查找变量X的时候（这个过程称为变量解析），它首先会从作用域链中的链尾也就是当前作用域进行查找是否有X属性，如果没有找到就顺着作用域链继续查找，直到查找到链头，也就是全局作用域链，仍未找到该变量的话，就认为这段代码的作用域上不存在x变量，并抛出引用错误（ReferenceError）的异常。

2、如何理解JavaScript原型链

- JavaScript中每个对象都有一个prototype属性，我们称之为原型，而原型的值也是一个对象，因此它也有自己的原型，这样就串联起来了一条原型链，原型链的链头是object，它的prototype.__proto__，值为null。
- 原型链的作用是用于对象继承，函数A的原型属性（prototype property）是一个对象，当这个函数被用作函数来创建实例时，该函数的原型属性被作为原型赋值给所有对象实例，比如我们新建一个数组，数组的方法便从数组的原型上继承而来。
- 当访问对象的一个属性时，首先查找对象本身，找到则返回；未找到则继续查找原型对象的属性（如果还找不到实际上还会沿着原型链向上查找，直至到根）。只要没有被覆盖的话，对象原型的属性就能在所有的实例中找到，若整个原型链未找到则返回undefined。

2(1)、JavaScript原型，原形链？有什么特点?

- 原型对象也是普通的对象，是对象一个自带隐式__proto__属性，原型也有可能有自己的原型，如果一个原型对象的原型为null的话，我们就称为原型链。
- 原形链是由一些用来继承和共享属性的对象组成的（有限的）对象链。

如何查找构造函数和原型中的属性?

- 构造函数.prototype 查看构造函数的原型属性
- 实例对象 .__proto__查看实例对象的构造函数的原型
- 实例对象.__proto__.constructor查看实例对象的构造函数

3、JavaScript如何实现继承?

- 构造继承

- 原型继承
- 实例继承
- 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

```
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
```

3（1）JavaScript如何实现继承?

1.借用构造函数。也叫伪造对象或经典继承。

- 思路：在子类构造函数的内部调用超类型构造函数。可以通过使用apply()和call()方法在新创建的对象上执行构造函数。
- 缺点：方法都在构造函数中定义，函数的复用就无从谈起。在超类型的原型中定义的方法，对子类而言也是不可见的，结果所有的类型都只能使用构造函数模式。

```
function SuperType() {
this.colors = ["red","blue","green"];
}
function SubType() {
SuperType.call(this);//继承了SuperType
}
var instance1 = new SubType();
instance1.colors.push("black");
console.log(instance1.colors);//"red","blue","green","black"
var instance2 = new SubType();
console.log(instance2.colors);//"red","blue","green"
```

2.原型链继承

- 思路：借助原型可以基于已有的对象创建对象，同时还不必因此创建自定义类型
- 在object()函数内部，先创建一个临时的构造函数，然后将传入的对象作为这个构造函数的原型，最后返回了这个临时类型的一个新实例==原型链继承的思想可用以下函数来说明

```
function object(o) {
    function F(){}
    F.prototype = o;
    return new F();
}
```

例子

```
var person = {
    name:"EvanChen",
    friends:["Shelby","Court","Van"];
```

```
};
var anotherPerson = object(person);
anotherPerson.name = "Greg";
anotherPerson.friends.push("Rob");
var yetAnotherPerson = object(person);
yetAnotherPerson.name = "Linda";
yetAnotherPerson.friends.push("Barbie");
console.log(person.friends);//"Shelby","Court","Van","Rob","Barbie"
```

3.组合继承

- 指的是将原型链和借用构造函数的技术组合在一起，从而发挥二者之长。
- 思路：使用原型链实现对原型属性和方法的继承，通过借用构造函数来实现实例属性的继承
- 优点：即通过在原型上定义发法实现了函数复用，又能保证每一个实例都有它自己的数组。
- 组合继承避免了原型链和借用构造函数的缺陷。融合了他们的优点。成为JavaScript中常用的继承模式

例子

```
function SuperType(name) {
this.name = name;
this.colors = ["red","blue","green"];
}
SuperType.prototype.sayName = function() {
console.log(this.name);
}
function SubType(name, age) {
SuperType.call(this,name);//继承属性
this.age = age;
}
//继承方法
SubType.prototype = new SuperType();
Subtype.prototype.constructor = Subtype;
Subtype.prototype.sayAge = function() {
console.log(this.age);
}
var instance1 = new SubType("EvanChen",18);
instance1.colors.push("black");
consol.log(instance1.colors);//"red","blue","green","black"
instance1.sayName();//"EvanChen"
instance1.sayAge();//18
var instance2 = new SubType("EvanChen666",20);
console.log(instance2.colors);//"red","blue","green"
instance2.sayName();//"EvanChen666"
instance2.sayAge();//20
```

4.寄生式继承

- 思路：创建一个仅用于封装继承过程的函数，该函数在内部以某种方式来增强对象，最后再像真的是它做了所有的工作一样返回对象

例子

```
function createAnother(original) {
var clone = object(original);  //通过调用函数创建一个新对象
clone.sayHi = function () {  //以某种方式来增强这个对象
alert("hi");
};
```

```
return clone;  //返回这个对象
}
var person = {
name:"EvanChen",
friends:["Shelby","Court","Van"];
};
var anotherPerson = createAnother(person);
anotherPerson.sayHi();///"hi"
```

5.寄生组合式继承。

- 思路：通过借用构造函数来继承属性，通过原型链的混成形式来继承方法
- 本质上，就是寄生式继承来继承超类型的原型，然后再将结果指定给子类型的原型
  基本模型如下所示

```
function inheritProperty(subType, superType) {
var prototype = object(superType.prototype);//创建对象
prototype.constructor = subType;//增强对象
subType.prototype = prototype;//指定对象
}
```

例子

```
function SuperType(name){
this.name = name;
this.colors = ["red","blue","green"];
}
SuperType.prototype.sayName = function (){
alert(this.name);
};
function SubType(name,age){
SuperType.call(this,name);
this.age = age;
}
inheritProperty(SubType,SuperType);
SubType.prototype.sayAge = function() {
alert(this.age);
}
```

- 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

```
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
```

4、JavaScript的基本数据类型

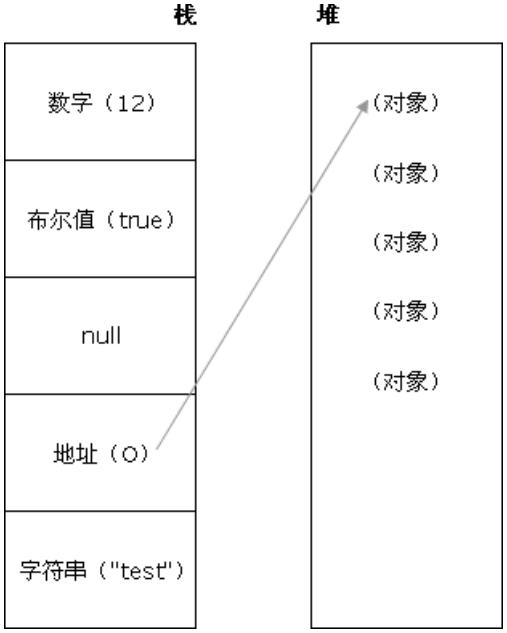
- Object number function boolean undefined

4(1)、js有哪些内置对象

- 数据封装类对象：Object、Array、Boolean、Number、String
- 其他对象：FUnction、Arguments、Math、Date、RegExp、Error

JavaScript有几种类型的值？你能画一下他们的内存图?

- 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



5、例举3种强制类型转换和2种隐式类型转换?

- 强制（parseInt,parseFloat,number）隐式（== ===）

6、split()join()的区别

- 前者是切割成数组的形式，后者是将数组转换成字符串

7、数组方法pop()push()unshift()shift()

- pop()尾部删除 push()尾部添加
- shift()头部删除 unshift()头部添加

8、IE和DOM事件流的区别

- 执行顺序不一样，
- 参数不一样、
- 事件加不加on、
- this指向问题

9、ajax请求的时候get和post方式的区别

- 一个在url后面 一个放在虚拟载体里面
- 有大小限制
- 安全问题
- 应用不同：一个是论坛等只需要请求的，一个是类似需改 密码的

10、IE的标准下有哪些兼容性的写法

```
Var ev = ev || window.event
document.documentElement.clientWidth || document.body.clientWidth
Var target = ev.srcElement||ev.target
```

11、ajax请求时，如何解释json数据

- 使用eval parse，鉴于安全性考虑，使用parse更靠谱

12、事件委托是什么

- 让利用事件冒泡的原理，让自己的所触发的事件，让他的父元素代替执行！

12（1）请说说事件委托机制？这样做有什么好处？

- 事件委托，就是某个事件本来该自己干的，但是自己不干，交给别人来干，就叫事件委托。打个比方：一个button对象，本来自己需要监控自身的点击事件，但是自己不来监控这个点击事件，让自己的父节点来监控自己的点击事件。

  好处

- A，提高性能：例如，当有很多li同时需要注册时间的时候，如果使用传统方法来注册事件的话，需要给每一个li注册事件。然而如果使用委托事件的话，就只需要将事件委托给该一个元素即可。这样就能提高性能
- B，新添加的元素还会有之前的事件

13、闭包是什么，有什么特性，对页面有什么影响？简要介绍你理解的闭包

- 闭包就是能够读取其他函数内部变量的函数
- "官方"的解释是：所谓"闭包"，指的是一个拥有许多变量和绑定了这些变量的环境的表达式（通常是一个函数），因而这些变量也是该表达式的一部分。
  通俗的讲：就是函数a的内部函数b，被函数a外部的一个变量引用的时候，就创建了一个闭包。
- ①.封闭性：外界无法访问闭包内部的数据，如果在闭包内声明变量，外界是无法访问的，除非闭包主动向外界提供访问接口；
- ②.持久性：一般的函数，调用完毕之后，系统自动注销函数，而对于闭包来说，在外部函数被调用之后，闭包结构依然保存在
- 系统中，闭包中的数据依然存在，从而实现对数据的持久使用。

优点：

- ① 减少全局变量。
- ② 减少传递函数的参数量
- ③ 封装；

缺点：

- 使用闭包会占有内存资源，过多的使用闭包会导致内存溢出等.

14、添加 插入 替换 移除 到某个接点的方法

- obj.appendChild()

- obj.insertBefore()
- obj.replaceChild()
- obj.removeChild()

14（1）、DOM怎样添加、移动、复制、创建和查找节点

```
// 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
// 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
// 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
```

15、"=="和"==="的不同

- 前者会自动转换类型，后者不会

15（1）请说出==和===的区别?

- ==判断内容是否相等不比较类型

  ```
  console.log(1=="1");//true
  ```

- ===判断内容相等且类型也相等

  ```
  console.log(1==="1"); //false
  ```

16、编写一个b继承a的方法

```
function A(name){
    this.name = name;
    this.sayHello = function(){alert(this.name+” say Hello!”);};

}
function B(name,id){
    this.temp = A;
    this.temp(name);         //相当于new A();
    delete this.temp;
     this.id = id;
    this.checkId = function(ID){alert(this.id==ID)};
}
```

17、如何阻止事件冒泡和默认事件

```
function stopBubble(e)
{
    if (e && e.stopPropagation)
        e.stopPropagation()
    else
        window.event.cancelBubble=true
```

```
}
return false
```

17（1）什么是事件冒泡/捕获

事件冒泡：子元素事件的触发会影响父元素事件

- 开关事件冒泡：
- A，开启事件冒泡：element.addEventListener(eventName,handler,false);
- B，关闭事件冒泡：假设传统方式事件的返回值为e，就可以通过e.stopPropagation()来关闭事件冒泡；

事件捕获：父元素的事件会影响子元素的事件；

- 开启事件捕获：element.addEventListener(eventName,hadler,true)

18、下面程序执行后弹出什么样的结果?

```
function fn() {
    this.a = 0;
    this.b = function() {
        alert(this.a)
    }
}
fn.prototype = {
    b: function() {
        this.a = 20;
        alert(this.a);
    },
    c: function() {
        this.a = 30;
        alert(this.a);
    }
}
var myfn = new fn();
myfn.b();
myfn.c();
```

19、谈谈this对象的理解

- this是js的一个关键字，随着函数使用场合不同，this的值会发生变化。
- 但是有一个总原则，那就是this指的是调用函数的那个对象。
- this一般情况下：是全局对象Global。作为方法调用，那么this就是指这个对象

19(1)、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attachEcent中this总是指向全局对象Window；

this对象的理解

- this是一个关键字，它代表函数运行时，自动生成一个内部对象，只能在函数内部使用
- 1.作为纯粹的函数调用this指向全局对象
- 2.作为对象的方法调用this指向调用对象
- 3.作为构造函数被调用this指向新的对象（new会改变this的指向）
- 4.apply调用this指向apply方法的第一个参数
- this总是指向函数的直接调用者（而并非间接调用者）；
- 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；

20、下面程序的结果

```
function fun(n,o) {
  console.log(o)
  return {
    fun:function(m){
      return fun(m,n);
    }
  };
}
var a = fun(0);  a.fun(1);  a.fun(2);  a.fun(3);
var b = fun(0).fun(1).fun(2).fun(3);
var c = fun(0).fun(1);  c.fun(2);  c.fun(3);

//答案:

//a: undefined,0,0,0
//b: undefined,0,1,2
//c: undefined,0,1,1
```

21、下面程序的输出结果

```
var name = 'World!';
(function () {
    if (typeof name === 'undefined') {
        var name = 'Jack';
        console.log('Goodbye ' + name);
    } else {
        console.log('Hello ' + name);
    }
})();
```

22、了解Node？Node的使用场景都有哪些?

- 高并发，聊天，实时消息推送

23、介绍下最常用的一款框架

- jQuery，rn,angular等

24、对于前端自动化构建工具有了解吗？简单介绍一下

- Gulp,Grunt等

25、说一下什么是JavaScript的同源策略?

- 一段脚本只能读取来自同一来源的窗口和文档的属性，这里的同一源指的是主机名，协议和端口号的组合

26、eval是指做什么的?

- 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，val obj=eval('('+str+')')

26（1）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，非常好性能（2次，一次解析成js语句，一次执行）；
- 由JSON字符串转换为JSON对象的时间可以用eval,var obj=eval('('+str+')');

27.请列举字符串操作的方法?

- chartCodeAt方法返回一个整数，代表指定位置字符的Unicode编码；
- chartAt方法返回指定索引位置处的字符。如果超出有效范围的索引值返回空字符串；
- slice方法返回字符串的片段
- substring方法返回位于String对象中指定位置的子字符串。
- substr方法返回一个从指定位置开始的指定长度的子字符串。
- indexOf方法返回String对象内第一次出现子字符串位置。如果没有找到子字符串。则返回-1；
- lastIndexOf方法返回String对象中字符串最后出现的位置。如果没有匹配到子字符串，则返回-1；
- search方法返回与正则表达式查找内容匹配的第一个字符串的位置；
- concat方法返回字符串值，该值包含了两个或多个提供的字符串的连接；
- split将一个字符串分割为子字符串，然后将结果作为字符串数组返回；

28、null和undefined的区别?

null是表示"无"的对象，转为数值时为0；undefined是表示"无"的原始值，转为数值时为NaN。

Q1

- 1、变量被声明了，但没有赋值，就等于undefined。
- 调用函数时
- 2，应该提供的参数没有提供该参数等于undefined。
- 3、对象没有赋值的属性，该属性的值为undefined.
- 4、函数没有返回值时，默认返回undefined。

  Q2

- 1、作为函数的参数，表示该函数的参数不是对象
- 2、作为对象原型链的终点。

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

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）
```

29、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所引用的对象中。
- 3、新创建的对象由this所引用，并且最后隐式的返回this。

30、JSON的了解?

- JSON(JavaScript Object Notation)是一种轻量级的数据交换格式。它是基于JavaScript的一个子集。数据格式简单，易于读写，占用带宽小。
- 格式：采用键值对，例如：{age:12,name:
'back'}

30（1）JSON的了解?

- json（JavaScript Object Notation）是一种轻量级的数据交换格式。
- 它是基于JavaScript的一个子集。数据格式简单，易于读写，占用带宽小

```
如：{"age":"12",""}
```

- json字符串转换为json对象

```
var obj=eval('('+str+')');
var obj=str.parseJSON();
var obj=JSON.parse(str);
```

- JSON对象转换为JSON字符串

```
var last=obj.toJSONString();
var last=JSON.stringify(obj);
```

31、call()和play()的区别和作用?

- 1、apply()函数有两个参数：第一个参数是上下文，第二个参数是参数组成的数组。如果上下文是null，则使用全局对象代替。
- 如：function.apply(thi是，[1,2,3]);
- 2、call()的第一个参数是上下文，后续是实例传入的参数序列。
- 如：function.call(this,1,2,3);

32、JS数组去重

以下是展示三种方法

```
Array.prototype.unique1 = function () {
  var n = []; //一个新的临时数组
  for (var i = 0; i < this.length; i++) //遍历当前数组
  {
    //如果当前数组的第i已经保存进了临时数组，那么跳过，
    //否则把当前项push到临时数组里面
    if (n.indexOf(this[i]) == -1) n.push(this[i]);
  }
  return n;
}

Array.prototype.unique2 = function()
{
    var n = {},r=[]; //n为hash表，r为临时数组
    for(var i = 0; i < this.length; i++) //遍历当前数组
    {
        if (!n[this[i]]) //如果hash表中没有当前项
        {
            n[this[i]] = true; //存入hash表
            r.push(this[i]); //把当前数组的当前项push到临时数组里面
        }
    }
    return r;
}

Array.prototype.unique3 = function()
{
    var n = [this[0]]; //结果数组
    for(var i = 1; i < this.length; i++) //从第二项开始遍历
    {
        //如果当前数组的第i项在当前数组中第一次出现的位置不是i，
        //那么表示第i项是重复的，忽略掉。否则存入结果数组
        if (this.indexOf(this[i]) == i) n.push(this[i]);
    }
    return n;
}
```

33、js操作获取和设置cookie

```
//创建cookie
function setCookie(name, value, expires, path, domain, secure) {
    var cookieText = encodeURIComponent(name) + '=' + encodeURIComponent(value);
    if (expires instanceof Date) {
        cookieText += '; expires=' + expires;
    }
    if (path) {
        cookieText += '; expires=' + expires;
```

```
    }
    if (domain) {
        cookieText += '; domain=' + domain;
    }
    if (secure) {
        cookieText += '; secure';
    }
    document.cookie = cookieText;
}

//获取cookie
function getCookie(name) {
    var cookieName = encodeURIComponent(name) + '=';
    var cookieStart = document.cookie.indexOf(cookieName);
    var cookieValue = null;
    if (cookieStart > -1) {
        var cookieEnd = document.cookie.indexOf(';', cookieStart);
        if (cookieEnd == -1) {
            cookieEnd = document.cookie.length;
        }
        cookieValue = decodeURIComponent(document.cookie.substring(cookieStart +
cookieName.length, cookieEnd));
    }
    return cookieValue;
}

//删除cookie
function unsetCookie(name) {
    document.cookie = name + "= ; expires=" + new Date(0);
}
```

34、GET和POST的区别，何时使用POST?

- GET：一般用于信息获取，使用URL传递参数，对发送信息的数量也有限制，一般在2000个字符
- POST：一般用于修改服务器上的资源，对发送的信息没有限制
- GET方式需要使用Request.QueryString来获取变量的值，而POST方式通过R额quest.Form来获取变量的值，也就是说GET是通过地址栏传值，而Post是通过提交表单来传值。
- 然而，在以下情况中，请使用POST请求：
- 无法使用缓存文件（更新服务器上的文件或数据库）
- 向服务器发送大量数据（POST没有数据量限制）
- 发送包含未知字符的用户输入时，POST比GET更稳定也更可靠

35、Falsh、Ajax各自的优缺点，在使用中如何取舍?

- Falsh适合处理多媒体，矢量图形、访问机器；对css、处理文本上不足，不容易被搜索。
- Ajax对css、文本支持很好、支持搜索；多媒体、矢量图形、机器访问不足。
- 共同点：与服务器的无刷新传递消息、用户离线和在线状态，操作DOM

36、ajax过程

- 1、创建XMLHttpRequest对象，也就是创建一个异步调用对象。
- 2、创建一个新的HTTP请求，并指想向该HTTP请求的方法、URL及验证信息。
- 3、设置响应HTTP请求状态变化的函数。
- 4、发送HTTP请求。
- 5、获取异步调用返回的数据。
- 6、使用JavaScript和DOM实现局部刷新。

js延迟加载的方式有哪些?

- defer和async，动态创建DOM方式（用得最多），按需异步载入js

如何解决跨越问题?

- jsonp,iframe,window.name,window.postMessage

new操作符具体干了什么呢?

- 1.创建了一个空对象，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。

```
var obj={};
obj.__proto__=Base.prototype;
Base.call(obj);
```

面向对象和类的区别?

- 简单的说类是对象的模板。
- 在js中没有类，所以在js中所谓的类就是构造函数，对象就是有构造函数创建出来的实例对象。面向对象就是使用面向对象的方式处理问题，面向对象是向过程进行封装。
- 对象的概念，面向对象编程的程序实际就是多个对象的集合，我们可以把所有的事物都抽象成对象，在程序设计中可以看作：对象=属性+方法。属性就是对象的数据，而方法就是对象的行为
- 类的概念，类是对象的模板，而对象是类的实例化。举个例子，汽车设计图可以看作是类，而具体的汽车就是对象。再比如有一个类是表示人，然后可以通过人这个模板来实例化出张三，李四...

面向对象有三大特性

- 抽象性，需要通过核心数据和特定环境才能描述对象的具体意义
- 封装性，封装就是讲数据和功能组合到一起，在js中对象就是键值对的集合，对象将属性和方法封装起来，方法将过程封装起来
- 继承性，将别人的属性的方法成为自己的，传统继承基于模板（类），js中继承基于构造函数

在js的计时器运行原理是怎样的，为什么可以触发计时器效果？计时器是多线程吗?

- 1.JavaScript引擎只有一个线程，强迫异步事件排队等待被执行。
- 2.setTimeout和setInterval本质上不同的地方是他们如何执行异步代码的。
- 3.如果一个定时器正在执行的时候被阻塞了，那么它将会被推迟到下一个可能的执行点，这既是使得延迟时间有可能会超过声明定时器时设置的值。
- Interval如果有足够的时间来执行（大于指定的延迟），那么它将会无延迟的一个紧接着一个执行。
- 原理
- 计时器通过设定一定的时间段（毫秒）来异步的执行一段代码。因为JavaScript是一个单线程，计时器提供了一种绕过这种语言限制来执行代码的能力。
- 总结：
- 计时器是单线程的，需要等待上一个执行完，如果上一个没有执行完，下一个需要延迟执行，直到上一个执行完