# Jquery常见面试题

**jQuery库中的$()是什么？**

jQuery函数的别称；$()函数用于将任何对象包裹成jQuery对象，之后允许调用定义在jQuery对象上的多个方法。（你甚至可以将一个选择器字符串传入 $() 函数，它会返回一个包含所有匹配的 DOM 元素数组的 jQuery 对象。）

**jQuery哪些选择器？**

id选择器，class选择器，标签选择器；

**id选择器和class选择器区别？**

原生js：

1.同一个文档中id必须唯一（需要js控制的东西可以用id定义）。class可以重复给同一类元素定义；

2.css选择优先级不同id--class--标签--子选择器ul>li --后代选择器 li a--伪类选择器 a:hover

!important优先级最高，优先级相同时采用就近原则，选择最后出现的样式；继承得来的属性优先级最低。

jQuery：选择一个元素时用id选择器，选择一组元素时用class选择器；

**$(document).ready()是什么函数和window.onload有什么区别**

在（文档进入ready状态）dom完全加载例如HTML被完全解析成dom树构建完成后，jQuery允许执行代码。使用这个函数最大的优点是它适用于所有浏览器

|  |  |
| --- | --- |
| window.onload() | $(document).ready() |
| 等待dom被创建，同时包括大型文件视频在内的外部资源完全加载 | 只需要等待dom树构建，执行起来较快 |
| 只能在单一函数中使用 | 可以在网页中多次使用，浏览器按照它们在HTML页面里出现的顺序执行它们 |

**语法以及函数问题**

|  |  |
| --- | --- |
| 实现隐藏图片： | $("#imageTohide").hide(); |
| 找到HTMLselect标签的选中项 | $('[name=nameofSelectedTag]:selected') |
| each()函数 | 可以遍历一个元素的集合 |
| HTML元素添加到dom树 | append，appendto等节点操作 |
| 提取一个连接的href（attr()方法获取属性值） | $(a).each(function(){console.log($(this).attr(href));}) |
| 插入属性值 | attr(name,value) |
| 添加和移除css类 | addClass()和removeClass() |

**使用CDN加载jQuery库的优势？**

节省服务器带宽，更快的下载速度。浏览器如果已经从同一个CDN下载过相同的jQuery版本，那么它不会再去下载一次。

**detach()和remove()区别**

empty()无参，清空后的子集和文本无法进行还原；detach()和remove()可以将其保存在一个变量中，需要新添加时可以再添加。同时remove可以删除指定元素remove('a')而detach不可以。但是remove移除元素后子集的事件不会存在，detach()子集的事件还可以执行；

**哪种方式更高效：document.getElementbyId("myId") 还是 $("#myId")？**

第一种，它直接使用了JavaScript引擎

**jQuery事件处理程序里返回了false会怎样？**

通常用于阻止事件向上冒泡

**jQuery的链式调用**：

对一个方法返回的结果继续调用另一个方法，使代码简洁明了，同时只对dom进行了一轮查找，性能更加出色。jQuery的方法都是挂在原型上的。

所有对象的方法返回的都是对象本身，返回的都是同一个对象，也就是说没有返回值；

使用链式调用：节省代码还有更好的异步体验：链条越靠后位置的方法越后执行；（可以提一下promise的链式调用）

# JS阻止冒泡和取消默认事件：

防止冒泡和捕获：W3C使用e.stopPropagation();IE则是使用e.cancelBubble=true;

取消默认事件：W3C是e.preventDefault();IE则是使用e.returnValue=false

什么是默认事件：链接<a>的默认动作是跳转到指定页面，

父级元素如何组织子级元素冒泡：（和委托类似）

|  |  |  |
| --- | --- | --- |
| 原生js写法 | jQuery | |
| 添加if(e.target==this){操作父元素} | 子元素function最后添上  e.stopPropagation(); | 子元素function最后添上  return false; |

**setInterval的缺点；setTimeout实现setInterval**

|  |  |
| --- | --- |
| **setInterval(异步** | **setTimeout(异步** |
| setInterval 执行无限次直到取消定时器； | 只执行一次 |
| **问题：**出现程序不是按照我们设定的精确时间调用的问题（原因就是回调函数运行时间大于间隔时间.(只是每隔xms将fn放入任务队列不一定每隔这个时间执行 | 使用循环+setTimeout解决setInterval时间不精确的问题 |

# 有关js基本类型和引用类型，浅拷贝和深拷贝

|  |  |  |  |
| --- | --- | --- | --- |
| 访问方式 | | | |
| 基本类型 Number String Boolean Null Undefined | | 引用类型 | |
| 操作、保存在变量的实际的值（保存在栈区中） | | 引用放在栈区，实际对象放在堆区。值保存在内存中，**js不允许直接访问内存**，操作的时候操作的是对象的引用。 | |
| var a=3;  var b=a;  b=0;//a还是3 | a赋予b的时候传的是值 | var a=[1,2,3];  var b=a;  b[0]=0;//a也变成了[0,2,3] | a赋予b的时候传的是地址 |

浅拷贝：

var b=a;

当修改b的属性为对象和数组时，那么父子对象之间产生关联，修改b，a也会发生改变

深拷贝：

不希望父子对象之间产生关联；递归将父对象中所有属于对象的属性类型都遍历赋予子对象。

# JS创建对象的方法

|  |  |  |
| --- | --- | --- |
| 工厂模式factory | 解决了创建多个相似对象的问题 | 无法识别对象即无法知道一个对象的类型 |
| 构造函数模式constructor | 可以将构造函数的实例标识成某种特定的类型 | 构造函数中的每个方法都要在实例上重新创建一遍。但是这个方法最后又不是同一个函数的实例，因为在ECMAScript中，每定义一个函数就是实例化了一个对象。 |
| 原型模式（prototype） | 解决了构造函数中每个方法都要在实例上重新创建一遍的问题 | 所有实例共享原型上的属性 |
| 组合使用构造函数模式和原型模式 | 可以向构造函数传递参数，并且每个实例都有自己的一份实例属性的副本是用来定义引用类型的默认模式 | 暂无 |
| 动态原型（dynamic prototype）模式 | 动态定义原型中的方法，创建的对象可以确定它的类型 | 不能使用对象字面量重写原型，不然会切断现有实例和新原型的联系 |
| 寄生构造函数模式（parasitic） | 仅仅封装构造对象的代码返回新的构建对象，类似构造函数。（和工厂模式代码一样只是调用的时候用new调用）该模式下构造函数返回的对象与构造函数外部创建的对象没区别 | 无法确定创建的对象的类型 |
| 稳妥构建函数模式（durable constructor） | 没有公共属性，新创建对象的实例方法不引用this  不使用new操作符调用构造函数  安全性得到保障 | 只能调用其定义的方法，没有其他方法访问其数据成员 |

注：对象的constructor属性可以用来标识对象类型

# JS实现继承：

|  |  |  |
| --- | --- | --- |
| 构造函数继承 | call apply | 无法继承父类原型上的属性和方法； |
| 原型链继承 | sub.prototype=new parent(); | 引用类型的原型属性会被共享，一个实例向该引用类型里添加数据 。另一个实例的这个属性也会跟着变化。 |
| 组合继承 | 构造函数实现对实例属性的继承  原型链实现对原型属性和方法的继承 | 两次调用父类的构造函数 |
| 原型式继承 | Object.create() | 引用类型的属性被所有实例共享  原型式继承是浅拷贝，所以引用类型的数据共享在不同实例中 |
| 寄生式继承 | 思路和构造函数和工厂模式类似。创建一个封装继承过程的函数。函数内部以某种方式增强对象 | 每次创建对象都会创建一遍方法 |
| 寄生组合式继承 | 不必为了指定子类型的原型而调用父类的构造函数。  让子类型的原型对象等于父类型原型对象的副本 | 寄生式继承和组合式继承混合；规避组合式继承的调用两次构造函数的问题，但是比较繁琐 |

# cookie session localStorage sessionStorage适用场景：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 类型 | 存储位置 | 存储大小 | 特性 | 生命周期 | 适用场景 | 缺点或特点 |
| cookie | 浏览器端  以文本的方式保存字符串 | 不超过4k | 浏览器端和服务器端来回传递数据  会在浏览器请求头或者ajax请求头中发送cookie内容 | 设置过期时间戳 | （1）存储用户名密码，下次登录时可以自动登录  （2）保存上次登录的时间  （3）保存上次查看的页面，浏览次数 | 大小受限  用户可操作  安全性低  每次访问都传送cookie给服务器，浪费带宽 |
| session | 服务器  以类似于hashtable的数据结构保存支持任何类型的对象 | 无限制 | session安全性大于cookie。  sessionID存储在cookie中，若要攻破session首先要攻破cookie（sessionID是惟一的） | 浏览器关闭后就不存在 | 保存每个用户的专用信息，变量的值保存在服务器端，通过sessionID区分不同用户  （1）购物车（将某些数据放在session中，供同一用户的不同页面使用）  （2）用户登录信息（3）防止用户非法登录 | session保存东西越多越占用服务器内存，用户在线人数较多时，服务器内存压力大。  依赖cookie，cookie禁用后需要使用URL重写 |
| localStorage | 客户端  字符串类型，复杂对象可以用JSON.parse和JSON.stringify处理 | 不超过5M |  | 永久存在 | 统计页面访问次数  常用于长期登录 | 只在请求时使用数据，不参与与服务器的通信 |
| sessionStorage | 客户端  字符串类型，复杂对象可以用JSON.parse和JSON.stringify处理 | 不超过5M | 将数据保存在session对象中（注意和session机制没关系）。session就是用户浏览这个网站花费的时间，session对象可以保存这段时间内所要求的保存的任何数据 | 临时保存：关闭浏览器窗口就会销毁 | 统计当前页面元素的点击次数  敏感账号一次登录 |

注：H5提出webstorage的目的：克服cookie的限制，把数据控制在客户端不需要持续将数据发回服务器。

# 判断对象是否为空

for in 循环遍历

JSON.stringify(obj)=="{}"

Object.keys(obj).length==0

# 遍历属性

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 遍历属性 | 继承属性 | symbol属性 |
| for(var i in obj) | 可枚举 | 包含继承 | 不包含 |
| Object.keys(obj) | 可枚举 | 不包含继承 | 不包含 |

# 同源政策与跨域

非同源受到的限制：

cookie localstorage无法读取；

Dom无法获得

Ajax请求无法发送

# 闭包

**ECMA**：European Computer Manufacture's Association欧洲计算机制造商协会。ECMAScript是一个重要的标准，一个完整的JavaScript实现由核心ECMAScript文档对象模型DOM和浏览器对象模型BOM。ECMAScript可以为不同种类的宿主环境提供核心的脚本编程能力，web浏览器就是一种宿主环境。它仅仅是一个描述，定义了脚本语言的所有属性方法和对象。其他的语言可以实现ECMAScript来作为功能的基准，JavaScript就是这样。

**闭包：**

**前言：**执行环境与作用域：执行环境定义了变量和函数有权访问的其他数据，决定了他们各自的行为。全局执行环境是最外围的执行环境，在web浏览器中全局执行环境被认为是window对象。**某个执行环境中的代码全部执行完毕后，环境被销毁，保存在其中的变量和函数定义也随之销毁，全局执行环境直到应用程序退出--例如browser关闭时才被销毁。**

作用域链查找：当我们想知道某个变量的值，首先在当前层即当前执行环境查找，没有则父级查找一直到全局执行环境，仍没有返回undefined。

作用域链的前端总是当前执行代码所在环境的变量对象，下一个变量对象来自包含（外部）环境，...最后一个对象是全局执行环境的变量对象。作用域链实际上是一个指向变量对象的指针列表，它只引用但不实际包含变量对象。

块级作用域：js没有。如C或Java中定义if(true){var color="blue"}if执行完后color就会被销毁，而js不存在这种块级作用域。同样，for语句初始化变量的表达式所定义的变量，在for循环执行完毕后也依旧存在于循环外部的执行环境中。

模仿块级作用域：匿名函数

|  |  |
| --- | --- |
| (function(){  //块级作用域  })(); | var someF=function(){  //块级作用域  };  someF(); |

闭包就是可以访问另一个函数作用域中的变量的函数

闭包特点：1.读取另一个函数作用域中的变量；2.外部函数和闭包中的整个变量对象会一直保存在内存中即使函数执行完毕。

优点：变量长期储存在内存中；避免全局变量的污染；私有成员的存在

缺点：闭包会携带外部（包含它的）函数的作用域，因此比其他函数占用更多内存。长时间不清除会引起内存泄露。

解决弊端：不使用的变量及时清除；

注意：闭包只能取得包含函数中任何变量的最后一个值（详见for循环）

# JS事件循环

**JS运行机制**

**js事件循环机制分为浏览器和node事件循环机制。**（以下是浏览器）

同步任务

异步任务

同步任务在主线程上执行形成执行栈--异步任务有了结果就向任务队列添加一个事件---执行栈中同步任务执行完后读取任务队列--任务队列进入执行栈--重复上面一步

线程中：事件循环唯一，但是任务队列有多个

除了广义的同步任务和异步任务还有更精细的定义：

宏任务：整体代码，setTimeout setInterval setImmediate I/O

微任务：process,nextTick promise

js执行优先级：同步代码（promise构造函数立即执行属于同步代码）--任务队列所有的微任务（process.nextTick>promise.then）--其中的一个任务队列--所有微任务--其中一个任务队列

注意：js运行机制：在第一次事件循环中：遇到宏任务先执行宏任务将它放入任务队列，执行回调函数时从微任务里拿回调函数再从（已经是下次循环了）宏任务开始找到其中一个任务队列执行完毕再执行微任务以此类推