变量类型和计算

*<!--根据题目来扩充知识点 举例几个题目-->  
<!--js中使用typeof能得到哪些类型；如何使用===和==；js中有哪些内置函数；js变量按照存储方式区分哪些类型，并描述其特点；如何理解json-->  
  
<!--变量类型-->  
<!--变量类型：分为值类型和引用类型-->*<**script**>  
 *//按存储方式分为值类型和引用类型；值类型是分块存储在内存中，值类型的值不会相互干涉，引用类型是好几个变量共用一个内存块节省内存空间  
// 值类型 每个变量都会存储各自的值不会相互影响；值类型有undefined string number boolean* **var *a*** = 100;  
 **var *b*** = ***a*** ;  
 ***a*** = 200;  
 **console**.log(***b***)*//100  
  
// 引用类型有对象，数组，函数  
// 特点可以扩展属性* **var *c***={**age**:20};  
 **var *d***=***c***;  
 ***d***.**age**=21;  
 **console**.log(***c***.**age**)*//21*</**script**>  
  
*<!--typeof详解 例子-->  
<!--typeof 只能区分值类型的详细类型；不能区分引用类型 但是能区分是否是函数-->*<**script**>  
 **typeof undefined** *//undefined* **typeof 'abc'***//string* **typeof** 123 *//number* **typeof true** *//boolean* **typeof** {}*//object* **typeof** []*//object* **typeof null***//object* **typeof console**.log*//function*</**script**>  
  
  
  
  
*<!--变量计算 --强制类型转换-->  
<!--会发生类型转换的情况：字符串拼接；==运算符；if语句；逻辑运算-->  
<!--字符串拼接例子-->  
<!--默认会转换成false的有：0；NaN；'';null；undefined；false-->*<**script**>  
 **var *a1*** = 100 + 10;*//110* **var *b1*** = 100 +**'10'**;*//10010*</**script**>  
*<!--==运算符例子-->*<**script**>  
 100 == **'100'** *//true* 0 == **''**;*//true* **null** == **undefined** *//true*</**script**>  
  
  
*<!--使用typeof能得到哪些类型；如何使用===和==；js中有哪些内置函数；js变量按照存储方式区分哪些类型，并描述其特点；如何理解json-->  
  
<!--何时使用==，===是没有强制类型转换的-->*<**script**>  
 **if**(***obj***.f == **null**){  
 *//这里相当于obj.f === null || obj.f === undefined  
 //这是jquery 源码中推荐的写法* }  
</**script**>  
  
  
*<!--js中有哪些内置函数 在不考虑浏览器环境的前提下-->  
<!--object;Array;Boolean;Number;String;Function;Date;RegExp;Error-->  
  
  
<!--如何理解json：json是js中一个对象；同时也是一种数据格式-->*<**script**>  
 **JSON**.stringify({**a**:10,**b**:10});*//把对象变成字符串* **JSON**.parse(**'{"a":10,"b":20}'**)*//把字符串变成对象*</**script**>

原型和原型链

*<!--题目：如何准确判断一个变量是数组类型；写一个原型链继承的例子；描述new一个对象的过程；zepto或其他框架如何使用原型链-->  
  
<!--知识点-->  
<!--构造函数-->*<**script**>  
 **function** *Foo*(name,age){  
 **this**.**name** = name;  
 **this**.**age** = age;  
 **this**.**class** = **'class-1'** *//return this //默认有* }  
 **var *f*** = **new** *Foo*(**'jizongquan'**,20)*//new是一个构造函数形成实例的过程  
 //var f1 = new Foo('lisi',22)//创建多个对象  
 // new 一个构造函数返回一个对象的过程，当你new的时候（传参数或者不传参数），new这个函数执行的时候里面的this会变成一个空对象  
 //然后在构造函数里面this点属性的赋值，默认把this给return回来；然后赋值给你new的那个函数，然后他就具备构造函数里面的属性了*</**script**>  
  
*<!--构造函数扩展 所有引用类型都有构造函数-->*<**script**>  
 **var *a*** = {}*//其实是var a=new Object()* **var *b*** = []*//其实是 var b = new Array()* **function** *Foo1*(){...}*//其实是var Foo1 = new Function(...)  
  
 //使用instanceof 判断一个函数是否是一个变量的构造函数  
 //instanceof 用来判断引用类型属于哪个构造函数的方法  
 // 例子 判断一个变量是否为数组:就是 变量 instanceof Array  
 //他的判断逻辑以下面c为例；c的\_proto\_一层一层网上，能否对应找到Foo2.prototype;然后在判断c instanceof Object*</**script**>  
  
*<!--原型规则和示例-->  
<!--1、所有的引用类型（数组、对象、函数），都具有对象特性，即可自由扩展属性（除了null以外）-->  
<!--2、所有的引用类型（数组、对象、函数），都有一个\_proto\_属性，属性值是一个普通的对象-->  
<!--3、所有的函数，都有一个prototype属性，属性值也是一个普通的对象-->  
<!--4、所有的引用类型（数组、对象、函数），\_proto\_属性值指向它的构造函数的“prototype”属性值-->  
<!--5、当试图得到一个对象的某个属性时，如果这个对象本事没有这个属性，那么会去它的\_proto\_（即它的构造函数的prototype）中寻找-->*<**script**>  
*// 对应第一条* **var *obj*** = {};***obj***.**a** = 100;  
 **var *arr*** = [];***arr***.**a** = 100;  
 **function** *fn* (){}  
 *fn*.**a**=100;  
*// 对应第二条* **console**.log(***obj***.\_proto\_);  
 **console**.log(***arr***.\_proto\_);  
 **console**.log(*fn*.\_proto\_);  
*// 对应第三条* **console**.log(*fn*.**prototype**)  
*// 对应第四条* **console**.log(***obj***.\_proto\_ === Object.**prototype**)  
*// 对应第五条  
 //构造函数* **function** *Foo2*(name,age){  
 **this**.**name** = name  
 }  
 *Foo2*.**prototype**.alertName = **function**(){  
 alert(**this**.**name**)  
 };  
 *//创建实例* **var *c*** = **new** *Foo2*(**"lisi"**)  
 ***c***.printName = **function**(){  
 **console**.log(**this**.**name**)  
 }  
 *//测试* ***c***.printName();*//lisi* ***c***.alertName();*//lisi* ***c***.toString();*//会去f.\_proto\_.\_proto\_中查找  
 //以c为例 c.\_proto\_=>Foo2.prototype;Foo2.prototype.\_proto\_=>Object.prototype=>Object.prototype\_proto\_如果还没有就是null；避免死循环  
 // 判断自己本身的属性* **var *item***;  
 **for**(***item* in *c***){  
 *//高级浏览器已经在for in中屏蔽了来自原型的属性  
 //但是还是 加上保证建创性* **if**(***c***.hasOwnProperty(***item***)){  
 **console**.log(***item***);  
 }  
 }  
</**script**>  
  
  
*<!--解题：如何准确判断一个变量是数组类型；写一个原型链继承的例子；描述new一个对象的过程；-->*<**script**>  
 *//如何准确判断一个变量是数组* **var *arr*** = [];  
 ***arr* instanceof** Array *//true* **typeof *arr*** *//object;无法判断  
  
 //描述new一个对象的过程  
 // 创建一个对象，this指向这个新对象，执行代码的时候，即对this赋值，返回this  
  
 // 写一个原型链继承的例子：写一个封装DOM查询的例子* **function** *Elem*(id){  
 **this**.**elem** = **document**.getElementById(id);  
 };  
 *Elem*.**prototype**.html = **function**(val){  
 **var** elem = **this**.**elem**;  
 **if**(val){  
 elem.**innerHTML** = val;  
 **return this** *//返回整个对象；链式操作* }**else**{  
 **return** elem.**innerHTML***//返回当前的html* }  
 };  
 *Elem*.**prototype**.on = **function**(type,fn){  
 **var** elem = **this**.**elem**;  
 elem.addEventListener(type,fn);  
 **return this** *//加return可以链式操作* };  
 **var *div1*** = **new** *Elem*(**'div1'**);  
*// console.log(div1.html())* ***div1***.html(**'<p>jizongquan</p>'**).on(**"click"**,**function**(){  
 alert(**"jizongquan"**)  
 }).html(**'<p>javascript</p>'**)  
</**script**>

作用域和闭包

*<!--题目：说一下对变量提升的理解；说明this几种不同的使用场景；创建10个<a>标签，点击的时候弹出来对应的序号；如何理解作用域；事件开发中闭包的应用;-->  
  
<!--执行上下文-->  
<!--范围：一段<script>或者一个函数，然后先去把变量声明、函数声明、this、arguments拿出来提到前面；-->*<**script**>  
 **console**.log(***a***)*//undefined* **var *a*** = 100;  
  
 *fn*(**'lisi'**);*//lisi,20* **function** *fn*(name){  
 age=20;  
 **console**.log(name,age);  
 **var** age  
 }  
</**script**>  
  
  
*<!--this要在执行时才能确认值，定义时无法确认-->  
<!--使用场景：1、作为构造函数执行；2、作为对象属性执行；3、作为普通函数执行；4、call，apply，bind-->*<**script**>  
 *//例子* **var *b*** = {  
 **name**: **"B"**,  
 fn:**function**(){  
 **console**.log(**this**.**name**)  
 }  
 };  
 ***b***.fn();*//this ===b* ***b***.fn.call({**name**:**"C"**});*//this ==={name:"C"}* **var *fn1*** = ***b***.fn;  
 ***fn1***();*//this === window  
 //下面例子对应上面的场景  
 //1、作为构造函数* **function** *Foo*(name){  
 **this**.**name** = name*//this就是Foo* };  
 **var *f*** = **new** *Foo*(**'lisi'**);  
 *//2、作为对象属性执行* **var *obj*** = {  
 **name**:**"A"**,  
 printName:**function**(){  
 **console**.log(**this**.**name**)  
 }  
 };  
 ***obj***.printName()*//this 就是obj  
  
 //3、作为普通函数执行* **function** *fn2*(){  
 **console**.log(**this**) *//this就是window* };  
 *//4、call，apply，bind* **function** *fn3*(name){  
 alert(name);  
 **console**.log(**this**)*//这样this就是x:100第一个参数* };  
 *fn3*.call({**x**:100},**'lisi'**)  
</**script**>  
  
  
*<!--作用域-->  
<!--如何理解作用域：自由变量，作用域链，即自由变量的查找，闭包的两个场景-->  
<!--js没有块级作用域-->*<**script**>  
 *//无块级作用域* **if**(**true**){  
 **var *name*** = **'jizongquan'** }  
 **console**.log(***name***)*//jizongquan*</**script**>  
  
*<!--函数和全局作用域-->*<**script**>  
 *//函数和全局作用域* **var *a*** = 100;  
 **function** *jq*(){  
 **var** a = 200;  
 **console**.log(**'jq'**,a)*//200* }  
 **console**.log(**'global'**,***a***)*//100  
 jq*()  
</**script**>  
  
*<!--作用域链-->  
<!--作用域链就是一个自由变量一直不断的往父级作用域找；形成一个链式结构-->*<**script**>  
 **var *b*** = 100;  
 **function** *jq1*(){  
 **var** c = 200  
 *//单枪作用域没有定义的变量，即“自由变量”* **console**.log(***b***);*//100* **console**.log(c);*//200* }  
 *jq1*()  
</**script**>  
  
  
*<!--闭包-->  
<!--闭包的使用场景：1、函数作为返回值；2、函数作为参数传递-->*<**script**>  
 *//1、函数作为返回值* **function** *F1*(){  
 **var** d = 100;  
 *//返回一个函数（函数作为返回值）* **return function** (){  
 **console**.log(d);*//100* }  
 }  
 *//f1得到一个函数* **var *f1*** = *F1*();  
 **var *d*** = 200;  
 ***f1***();*//100  
  
  
 //2、函数作为参数传递* **function** *F1*(){  
 **var** d = 100;  
 **return function** (){  
 **console**.log(d);  
 }  
 }  
 **var *f1*** = *F1*();  
 **function** *F2*(fn){  
 **var** d =200  
 fn()  
 }  
 *F2*(***f1***)*//100*</**script**>  
  
*<!--创建10个<a>标签，点击的时候弹出来对应的序号-->*<**script**>  
 **var *i***;  
 **for**(***i***=0;***i***<10;***i***++){  
 (**function**(i){  
 **var** c= **document**.createElement(**'a'**);  
 c.**innerHTML** = i + **'<br>'**;  
 c.addEventListener(**"click"**,**function**(e){  
 e.preventDefault();  
 alert(i)  
 })  
 **document**.**body**.appendChild(c)  
 })(***i***)  
 }  
</**script**>  
  
*<!--实际开发中闭包的应用-->  
<!--主要用于封装变量，收敛权限-->*<**script**>  
 **function** *isFirstLoad*(){  
 **var** \_list = [];  
 **return function** (id){  
 **if**(\_list.indexOf(id) >= 0){  
 **return false** }**else**{  
 \_list.push(id)  
 **return true** }  
 }  
 }  
 *//使用* **var *firstLoad*** = *isFirstLoad*();  
 ***firstLoad***(10)*//true* ***firstLoad***(10)*//false* ***firstLoad***(20)*//true*</**script**>

事件

*<!--题目：编写 一个通用的事件监听函数；描述事件冒泡流程；对于一个无限下拉加载图片的页面，如何给每个图片绑定事件-->  
  
<!--通用事件绑定-->*<**script**>  
 **var *btn*** = **document**.getElementById(**'btn1'**);  
 ***btn***.addEventListener(**'click'**,**function**(event){  
 **console**.log(**'clicked'**)  
 });  
  
 *//一个通用函数* **function** *bindEvent*(elem,type,fn){  
 elem.addEventListener(type,fn);  
 }  
 **var *a*** =**document**.getElementById(**'link1'**);  
 *bindEvent*(***a***,**'click'**,**function**(e){  
 e.preventDefault();  
 alert(**'clicked'**)  
 })  
</**script**>  
  
*<!--ie低版本的兼容 使用attachEvent绑定事件；和W3C标准不一样-->  
  
<!--事件冒泡例子-->*<**script**>  
 **var *p1*** = **document**.getElementById(**'p1'**);  
 **var *body*** = **document**.**body**;  
 *bindEvent*(***p1***,**'click'**,**function**(e){  
 e.stopPropagation()  
 alert(**'激活'**)  
 });  
 *bindEvent*(***body***,**'click'**,**function**(e){  
 alert(**'激活'**)  
 })  
</**script**>  
  
*<!--事件冒泡的应用=》代理-->*<**script**>  
 *//这个示例是指如果点击的是a元素就会弹出a元素里面的内容* **var *div2*** = **document**.getElementById(**'div2'**);  
 ***div2***.addEventListener(**'click'**,**function**(e){  
 **var** target = e.**target**;  
 **if**(target.**nodeName** ===**'A'**){  
 alert(target.**innerHTML**);  
 }  
 })  
</**script**>  
  
*<!--相对完美的通用绑定事件的函数示例-->*<**script**>  
 **function** *bindEvent*(elem,type,selector,fn){  
 **if**(fn==**null**){  
 fn = selector;  
 selector = **null** }  
 elem.addEventListener(type,**function**(e){  
 **var** target;  
 **if**(selector){  
 target= e.**target**;  
 **if**(target.matches(selector)){  
 fn.call(target,e)  
 }  
 }**else**{  
 fn(e)  
 }  
 })  
 };  
 *//具体应用  
 //使用代理* **var *div3*** = **document**.getElementById(**'div3'**);  
 *bindEvent*(***div3***,**'click'**,**'a'**,**function**(e){  
 **console**.log(**this**.**innerHTML**)  
 })  
 *//不使用代理* **var *a*** = **document**.getElementById(**'a1'**);  
 *bindEvent*(***div3***,**'click'**,**function**(e){  
 **console**.log(***a***.**innerHTML**)  
 })  
 *//代理的好处：减少浏览器内存占用*</**script**>

DOM和BOM

*<!--js-web-API:js基础知识是遵循ECMA262标准；js-WEB-API是遵循W3C标准-->  
  
<!--DOM就是Document Object Model 文档对象模型-->  
<!--题目：DOM是哪种基本的数据结构；DOM操作的常用API有哪些；DOM节点的attr和property有何区别-->  
  
<!--DOM本质就是将html进行了结构化，结构化成了一个浏览器可识别js可识别的模型-->  
  
<!--DOM节点操作-->  
<!--1、获取DOM节点-->*<**script**>  
 *//原生js获取元素和类的方式* **var *div1*** = **document**.getElementById(**'div1'**)*//获取元素* **var *divList*** = **document**.getElementsByName(**'div'**)*//获取集合* **console**.log(***divList***.**length**);  
 **console**.log(***divList***[0]);  
  
 **var *containerList*** = **document**.getElementsByClassName(**'.container'**)*//获取集合* **var *pList*** = **document**.querySelectorAll(**'p'**)*//获取集合*</**script**>  
  
*<!--2、DOM结构操作-->*<**script**>  
 **var *div2*** =**document**.getElementById(**'div2'**)  
 *//添加新节点* **var *p1*** = **document**.createElement(**'p'**);  
 ***p1***.**innerHTML** = **'this is p1'**;  
 ***div2***.appendChild(***p1***)*//添加新节点  
  
 //移动已有节点 假设已有p2* **var *p2***=**document**.getElementById(**'p2'**)  
 ***div2***.appendChild(***p2***)  
  
 *//获取父元素和子元素* **var *div3*** = **document**.getElementById(**'div3'**);  
 **var *parent*** = ***div3***.parentElement;  
  
 **var *child*** = ***div3***.**childNodes**;*//获取子元素集合* ***div3***.removeChild(***child***[0]);  
  
</**script**>  
*<!--DOM是哪种基本的数据结构：树结构-->  
<!--DOM操作的常用API有哪些：获取DOM节点，以及节点的property和Attribute;获取父节点，获取子节点；新增节点，删除节点-->  
<!--DOM节点的Attribute和property有何区别：-->  
<!--property只是一个js对象的属性的修改-->  
<!--Attribute是对html标签属性的修改-->  
  
  
<!--BOM：Browser Object Model 浏览器对象模型-->  
<!--题目：如何检测浏览器的类型；拆解url各部分-->  
  
<!--navigator：浏览器；基本用法如下；检测浏览器类型-->*<**script**>  
 **var *ua*** = **navigator**.**userAgent**;  
 **var *isChrome*** = ***ua***.indexOf(**'Chrome'**);  
 **console**.log(***isChrome***)  
</**script**>  
*<!--screen：屏幕-->*<**script**>  
 **console**.log(**screen**.**width**)  
 **console**.log(**screen**.**height**)  
</**script**>  
*<!--loaction：地址-->*<**script**>  
 **console**.log(**location**.**href**)*//打印整个url* **console**.log(**location**.**protocol**)*//协议http:或者https* **console**.log(**location**.**host**)*//域名* **console**.log(**location**.**pathname**)*//域名后面的路径比如/learn* **console**.log(**location**.search)*//shi 地址里面？后面的参数* **console**.log(**location**.**hash**)*//#后面的*</**script**>  
*<!--history“前进后退-->*<**script**>  
 **history**.back()*//返回* **history**.forward()*//前进*</**script**>

异步和单线程

*<!--题目：同步和异步的区别是什么，分别举一个同步和异步的例子；一个关于setTimeout的笔试题；前端使用异步的场景有哪些；-->  
<!--什么是异步-->  
<!--同步会阻塞代码执行；异步不会阻塞代码和程序执行-->  
<!--何时需要异步：在可能发生等待的情况；-->  
<!--前端使用异步的场景：1、定时任务;setTimeout,setInverval；2、网络请求：ajax请求，动态<img>加载；3、事件绑定-->*<**script**>  
 **console**.log(100);  
 setTimeout(**function**(){  
 **console**.log(200)  
 },1000)  
 **console**.log(300)*//先输出100然后300后200*</**script**>  
  
*<!--单线程：一次只能干一个事-->  
<!--以上面例子为例：执行第一行，打印100；执行setTimeout后，传入setTimeout的函数会被暂存起来，不会立即执行（单线程的特点不能同时做两件事）；  
 然后执行最后一行答应300；待所有程序执行完，处于空闲状态时，会立马看有没有暂存起来的要执行；发型暂存起来的setTimeout中的函数无需等待时间  
 ，就立即过来执行-->  
  
<!--解答：-->  
<!--同步和异步的区别：同步会阻塞代码执行，异步不会；-->  
  
<!--一个关于setTimeout的笔试题-->*<**script**>  
 **console**.log(1);  
 setTimeout(**function**(){  
 **console**.log(2)  
 },0);  
 **console**.log(3);  
 setTimeout(**function**(){  
 **console**.log(4)  
 },1000);  
 **console**.log(5);  
 *//1,3,5,2,4*</**script**>

ajax

*<!--题目：手动编写一个ajax，不依赖第三方库；跨域的几种实现方式；-->  
  
<!--XMLHttpRequest;简单写一个ajax实例-->*<**script**>  
 **var *shr*** = **new** XMLHttpRequest();  
 xhr.open(**"GET"**,**"/api"**,**false**);*//false这里面试指使用异步* xhr.onreadystatechange = **function**(){  
 *//这里的函数异步执行* **if**(xhr.**readyState** == 4){  
 **if**(xhr.**status** == 200){  
 alert(xhr.**responseText**)*//打出服务端返回的内容* }  
 }  
 }  
 xhr.send(**null**);  
 *//ie低版本使用ActiveXObject*</**script**>  
*<!--readState 主要是4其他可以不用管-->  
<!--0（未初始化）还没有调用send()方法-->  
<!--1（载入）已调用send()方法，正在发送请求-->  
<!--2（载入完成）send()方法执行完成，已经接收到全部响应内容-->  
<!--3（交互）正在解析响应内容-->  
<!--4（完成）响应内容解析完成，可以在客户端调用了-->  
  
<!--status 状态码-->  
<!--2xx 表示成功处理请求 如200-->  
  
<!--3xx 需要重定向，浏览器直接跳转-->  
  
<!--4xx 客户端请求错误，如404-->  
  
<!--5xx 服务器错误；504连接超时-->  
  
<!--跨域-->  
<!--什么事跨域：浏览器有同源策略，就是不允许ajax访问其他域接口（http默认端口80；https默认端口443）-->  
<!--跨域条件（就是不同源）：协议、域名、端口、有一个不同就是跨域；-->  
  
<!--可以跨域的三个标签：1、<img src=xxx>;2、<link href=xxxx> 3、<script src=xxx>-->  
<!--三个标签的场景-->  
<!--<img>用于打点统计，统计网站可能是其他域-->  
<!--<link><script>可以使用CDN，CDN的也是其他域-->  
<!--<script>可以用于JSONP-->  
  
<!--跨域注意事项：1、所有的跨域请求都必须经过信息提供方允许；2、如果未经允许即可获取，那是浏览器同源策略出现漏洞-->  
  
<!--JSONP实现原理-->  
<!--就是请求的地址如果是跨域的，那么就让这个借口给你返回一个callBack，然后请求方在接收这个callBack-->  
<!--我们可以先定义一个函数，然后我们请求外域的一个api，api他返回的是一个js的片段，js片段正好执行这个函数，然后把参数传过来，这就是jsonp-->  
  
  
<!--还有就是 还可以让服务器段设置http header；但是需要服务器段来做-->*<**script**>  
*// response.setHeader("Access-Control-Allow-Origin",'这里面是允许跨域的域名')*</**script**>

存储

*<!--请描述cookie，sessionStorage和localStorage的区别-->  
<!--cookie-->  
<!--本身用于客户端和服务端通信；他有本地存储的功能；使用document.cookie=...获取和修改-->  
<!--cookie存储的缺点：存储量太小，只有4kb；所有http请求都带着，会影响获取资源的效率-->  
  
<!--locationStorage和sessionStorage-->  
<!--HTML5专门为存储而设计，最大容量5M；用法：存储locationStorage.setItem(key,value),获取locationStorage.getItem(key)-->  
<!--http请求中不会带着-->  
<!--sessionStorage 如果浏览器关掉就会清理-->  
<!--locationStorage如果不手动清楚就不会清楚-->*

页面加载+性能优化+安全性

*<!--页面加载-->  
<!--题目：从输入url到得到html的详细过程；window.onload和DOMContentLoaded的区别；-->  
  
<!--加载资源的形式：1、输入url（或跳转页面）加载html；2、加载html中的静态资源-->  
  
<!--加载一个资源的过程（从输入url到得到html的详细过程）：-->  
<!--1、浏览器根据DNS服务器得到域名的IP地址；-->  
<!--2、向这个IP的机器发送http请求；-->  
<!--3、服务器收到、处理并返回http请求；-->  
<!--4、浏览器得到返回内容-->  
  
  
  
<!--浏览器渲染页面的过程：-->  
<!--1、根据HTML结构生成DOM Tree；（注：DOMTree是没有样式的）-->  
<!--2、根据css生成CSSOM（cssom就是css object model）；-->  
<!--3、将DOM和CSSOM整合形成RenderTree（也就是渲染树；这里渲染树是有样式的）-->  
<!--4、根据RenderTree开始渲染和展示-->  
<!--5、遇到<script>时，会执行并阻塞渲染-->  
  
<!--window.onload-->*<**script**>  
 **window**.addEventListener(**'load'**,**function**(){  
 *//页面的全部资源加载完才好执行，包括图片、视频等* })  
</**script**>  
  
*<!--DOMContentLoaded-->*<**script**>  
 **document**.addEventListener(**'DOMContentLoaded'**,**function**(){  
 *//DOM 渲染完即可执行，此时图片、视频还可能没有加载完* })  
</**script**>  
  
*<!--性能优化（综合性问题）：原则就是多使用内存、缓存或者其他方法；减少cpu计算减少网络请求-->  
<!--加载资源优化-->  
<!--1、静态资源的压缩合并（可以通过构建工具合并如webpack等这里简单说出即可）-->  
<!--2、静态资源缓存-->  
<!--3、使用cdn让资源加载更快-->  
<!--4、使用ssr后端渲染；数据直接输出到html中（可以不说这点。ssr就是服务端渲染的意思）-->  
  
<!--渲染优化-->  
<!--1、css放前面，js放后面-->  
<!--2、懒加载（图片懒加载、下拉加载更多）-->  
<!--懒加载例子-->  
<!--原理是一开始放一个很小的图很快加载出来，将真正的图片地址放在一个data属性里面，通过js来获取这个图片用的时候付给img的src上即可-->*<**img** src=**"xxx"** id=**"img1"** data-realsrc=**"abc.png"** alt=**""**>  
<**script**>  
 **var *img1*** = **document**.getElementById(**'img1'**);  
 ***img1***.**src**= ***img1***.getAttribute(**'data-realsrc'**)  
</**script**>  
*<!--3、减少DOM查询，对DOM查询做缓存-->  
<!--例子-->*<**script**>  
 *//未缓存DOM查询;这里执行每一遍的时候 都要对p进行一遍查询* **var *i***;  
 **for**(***i***=0;***i***<**document**.getElementsByTagName(**'p'**).**length**;***i***++){  
 *//....* }  
  
 *//缓存了DOM查询；这里查询一遍即可* **var *pList*** = **document**.getElementsByTagName(**'p'**);  
 **var *i***;  
 **for**(***i***=0;***i***<***pList***.**length**;***i***++){  
 *//...* }  
</**script**>  
*<!--4、减少DOM操作，多个操作尽量合并在一起执行-->  
<!--5、事件节流（就是快速操作的时候不执行事件，什么时候停下来在执行一次）-->  
<!--6、尽早执行操作（如DOMContentLoaded）-->  
  
  
<!--安全性-->  
<!--XSS跨站请求攻击-->  
<!--一般是出现在评论或者写文章提交里面会出现,通过在评论和写文章里面插入一段<script>,  
一般用来获取cookie进而得到一些用户信息-->  
<!--前端可以通过替换关键字如将<换为&lt;>换为&gt;-->  
  
<!--XSRF跨站请求伪造-->  
<!--问题主要就是没有验证；解决方案就是 增加验证流程，密码，短信等-->*

js基础其他知识

*<!--题目：获取2017-06-10格式的日期；获取随机数，要求是长度一致的字符串格式；写一个能遍历对象和数组的通用forEach函数-->  
  
<!--日期-->*<**script**>  
 Date.now();*//获取当前时间毫秒数* **var *dt*** = **new** Date();  
 ***dt***.getTime();*//获取毫秒数* ***dt***.getFullYear();*//年* ***dt***.getMonth();*//月（0-11）* ***dt***.getDate();*//日（0-31）* ***dt***.getHours();*//小时（0-23）* ***dt***.getMinutes();*//分钟（0-59）* ***dt***.getSeconds();*//秒（0-59）*</**script**>  
  
*<!--数组-->  
<!--forEach 遍历所有元素-->*<**script**>  
 **var *arr*** = [1,2,3];  
 ***arr***.forEach(**function**(item,index){  
 *//遍历数组的所有元素* **console**.log(index,item)  
 })  
</**script**>  
*<!--every 判断所有元素是否都符合条件-->*<**script**>  
 **var *arr1*** = [1,2,3];  
 **var *result*** = ***arr1***.every(**function**(item,index){  
 *//用来判断所有的数组元素，都满足一个条件* **if**(item<4){  
 **return true** }  
 })  
 **console**.log(***result***);  
</**script**>  
*<!--some 判断是否有至少一个元素符合条件-->*<**script**>  
 **var *arr2*** = [1,2,3];  
 **var *result1*** = ***arr2***.some(**function**(item,index){  
 *//用来判断所有的数组元素，只要有一个满足条件即可* **if**(item<2){  
 **return true** }  
 })  
 **console**.log(***result1***);  
</**script**>  
*<!--sort 排序-->*<**script**>  
 **var *arr3*** = [1,4,2,3,5];  
 **var *arr4*** = ***arr3***.sort(**function**(a,b){  
 *//从小到大* **return** a-b  
  
 *//从大到小  
 //return b-a* })  
 **console**.log(***arr4***);  
</**script**>  
*<!--map 对元素重新组装，生成新数组-->*<**script**>  
 **var *arr5*** = [1,2,3,4];  
 **var *arr6*** = ***arr5***.map(**function**(item,index){  
 *//将元素重新组装，并返回* **return '<b>'**+ item +**'</b>'** })  
 **console**.log(***arr6***);  
</**script**>  
*<!--filter 过滤符合条件的元素-->*<**script**>  
 **var *arr7*** = [1,2,3,4];  
 **var *arr8*** = ***arr7***.filter(**function**(item,index){  
 *//通过某一个条件过滤数组* **if**(item>=2){  
 **return true** }  
 })  
 **console**.log(***arr8***);  
</**script**>  
  
  
*<!--对象api就是一个for in-->*<**script**>  
 **var *obj*** = {  
 **x**:100,  
 **y**:200,  
 **z**:300  
 };  
 **var *key***;  
 **for**(***key* in *obj***){  
 *//注意这里的hasOwnProperty，* **if**(***obj***.hasOwnProperty(***key***)){  
 **console**.log(***key***,***obj***[***key***])  
 }  
 }  
</**script**>  
  
*<!--获取2017-06-1格式的日期-->*<**script**>  
 **function** *formatDate*(dt){  
 **if**(!dt){  
 dt = **new** Date()  
 }  
 **var** year = dt.getFullYear();  
 **var** month = dt.getMonth() + 1;  
 **var** date = dt.getDate();  
 **if**(month<10){  
 *//强制类型转换* month = **'0'** + month  
 }  
 **if**(date < 10){  
 *//强制类型转换* date = **'0'** + date  
 }  
 *//强制类型转换* **return** year + **'-'** +month +**'-'**+date;  
 }  
 **var *dt*** = **new** Date();  
 **var *formatDate*** = *formatDate*(***dt***);  
 **console**.log(***formatDate***)  
</**script**>  
*<!--获取随机数，要求长度一致的字符串格式-->*<**script**>  
 **var *random*** = **Math**.random();  
 **var *random*** = ***random*** + **'0000000000'**;  
 **var *random*** = ***random***.slice(0,10)  
 **console**.log(***random***)  
</**script**>  
  
*<!--写一个能遍历对象和数组的通用forEach函数-->*<**script**>  
 **function** *forEach*(obj,fn){  
 **var** key;  
 **if**(obj **instanceof** Array){  
 obj.forEach(**function**(item,index){  
 fn(index,item)  
 })  
 }**else**{  
 **for**(key **in** obj){  
 **if**(obj.hasOwnProperty(key)){  
 fn(key,obj[key])  
 }  
 }  
 }  
 }  
 *// 下面是调用* **var *arr*** = [1,2,3];  
 *forEach*(***arr***,**function**(index,item){  
 **console**.log(index,item)  
 });  
 **var *obj*** = {**x**:10,**y**:20};  
 *forEach*(***obj***,**function**(key,val){  
 **console**.log(key,val)  
 })  
</**script**>