Ajax

Ajax的工作原理相当于在用户和服务器之间加了—个中间层,使用户操作与服务器响应异步化。并不是所有的用户请求都提交给服务器,像—些数据验证和数据处理等都交给Ajax引擎自己来做, 只有确定需要从服务器读取新数据时再由Ajax引擎代为向服务器提交请求。

Ajax的一个最大的特点是无需刷新页面便可向服务器传输或读写数据(又称无刷新更新页面),这一特点主要得益于XMLHTTP组件XMLHTTPRequest对象。

 Ajax其核心只有JavaScript、XMLHTTPRequest和DOM。

function ajax(url, fnSucc, fnFaild)

{

//1.创建Ajax对象

var oAjax=null;

if(window.XMLHttpRequest)

{

oAjax=new XMLHttpRequest();

}

else

{

oAjax=new ActiveXObject("Microsoft.XMLHTTP");

}

//2.连接服务器

oAjax.open('GET', url, true);

//3.发送请求

oAjax.send();

//4.接收服务器的返回

oAjax.onreadystatechange=function ()

{

if(oAjax.readyState==4) //完成

{

if(oAjax.status==200) //成功

{

fnSucc(oAjax.responseText);

}

else

{

if(fnFaild)

fnFaild(oAjax.status);

}

}

};

}

JSON

要实现从对象转换为 JSON 字符串，使用 JSON.stringify() 方法：

var json = JSON.stringify({a: 'Hello', b: 'World'}); //结果是 '{"a": "Hello", "b": "World"}'

要实现从 JSON 转换为对象，使用 JSON.parse() 方法：

var obj = JSON.parse('{"a": "Hello", "b": "World"}'); //结果是 {a: 'Hello', b: 'World'}

与 POST 相比，GET 更简单也更快，并且在大部分情况下都能用。

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
* GET的目的就如同其名字一样是用于获取信息的。它旨在显示出页面上你要阅读的信息。浏览器会缓冲GET请求的执行结果，如果同样的GET请求再次发出，浏览器就会显示缓冲的结果而不是重新运行整个请求。这一流程不同于浏览器的处理过程，但是它是有意设计成这样以使GET调用更有效率。GET调用会检索要显示在页面中的数据，数据不会在服务器上被更改，因此重新请求相同数据的时候会得到相同的结果。
* POST方法应该用于你需要更新服务器信息的地方。如某调用要更改保存在服务器上的数据，而从两个同样的POST调用返回的结果或许会完全不同，因为第二个POST调用的值与第一个的值不相同，这是由于第一个调用已经更新了其中一些值。POST调用通常会从服务器上获取响应而不是保持前一个响应的缓冲。

如果需要像 HTML 表单那样 POST 数据，请使用 setRequestHeader() 来添加 HTTP 头。

readyState  
存有 XMLHttpRequest 的状态。从 0 到 4 发生变化。

* 0: 请求未初始化
* 1: 服务器连接已建立
* 2: 请求已接收
* 3: 请求处理中
* 4: 请求已完成，且响应已就绪

路由：<http://blog.csdn.net/liuhe688/article/details/44709173>

location.hash和history.pushState

判断js中的数据类型有一下几种方法：typeof、instanceof、 constructor、 prototype、 $.type()/jquery.type()、Object.prototype.toString.call(a) == [Object Array]

typeof返回的类型都是字符串形式，另外typeof 可以判断function的类型；在判断除Object类型的对象时比较方便。alert(typeof b) ------------> number

instanceof 后面一定要是对象类型，并且大小写不能错。alert(f instanceof Function) ------------> true

通常情况下用typeof 判断就可以了，遇到预知Object类型的情况可以选用instanceof或constructor方法,实在没辙就使用$.type()方法。

数据存储

Html5支持两种的WebStorage，一种是永久性的本地存储（localStorage），另外一种是会话级别的本地存储（sessionStorage）。

Database，本地数据库

跨域：<https://segmentfault.com/a/1190000008776669>

数据双向绑定单向绑定优缺点：<https://www.cnblogs.com/Breaveleon/p/6680175.html>

tap、touch

一、click 和 tap 比较

两者都会在点击时触发，但是在手机WEB端，click会有 200~300 ms，所以请用tap代替click作为点击事件。

singleTap和doubleTap 分别代表单次点击和双次点击。

三、touch事件touch是针对触屏手机上的触摸事件。现今大多数触屏手机webkit内核提供了touch事件的监听，让开发者可以获取用户触摸屏幕时的一些信息。

其中包括：touchstart,touchmove,touchend,touchcancel 这四个事件

touchstart,touchmove,touchend事件可以类比于mousedown，mouseover  
，mouseup的触发。

touchstart ： 当手指触摸到屏幕会触发；

touchmove : 当手指在屏幕上移动时，会触发；

touchend : 当手指离开屏幕时，会触发；

而touchcancel许多人不知道它在什么时候会被触发而忽略它，其实当你的手指还没有离开屏幕时，有系统级的操作发生时就会触发touchcancel，例如alert和confirm弹框，又或者是android系统的功能弹窗。

对象继承

<http://blog.csdn.net/longyuhome/article/details/7544222>

http://blog.csdn.net/tongle\_deng/article/details/7713490

ES5最经典的继承方法是用组合继承的方式，原型链继承方法，借用函数继承属性，ES6也是基于这样的方式，但是封装了更优雅简洁的api，让Javascript越来越强大，修改了一些属性指向，规范了继承的操作，区分开了继承实现和实例构造，此外ES6继承还能实现更多的继承需求和场景。

组合继承是使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，通过在原型上定义方法来实现

行内元素嵌套块级元素：行内元素若有内容，会显示内容，没有内容则不会显示，块级元素正常。

CDN的基本原理是广泛采用各种缓存服务器，将这些缓存服务器分布到用户访问相对集中的地区或网络中，在用户访问网站时，利用全局负载技术将用户的访问指向距离最近的工作正常的缓存服务器上，由缓存服务器直接响应用户请求。

defer和async

1. <script src="script.js"></script>

没有 defer 或 async，浏览器会立即加载并执行指定的脚本，“立即”指的是在渲染该 script 标签之下的文档元素之前，也就是说不等待后续载入的文档元素，读到就加载并执行。

1. <script async src="script.js"></script>

有 async，加载和渲染后续文档元素的过程将和 script.js 的加载与执行并行进行（异步）。

1. <script defer src="myscript.js"></script>

有 defer，加载后续文档元素的过程将和 script.js 的加载并行进行（异步），但是 script.js 的执行要在所有元素解析完成之后，DOMContentLoaded 事件触发之前完成。

CSS

Canvas 与 SVG 的比较

Canvas（通过 JavaScript 来绘制 2D 图形）

依赖分辨率

不支持事件处理器

弱的文本渲染能力

能够以 .png 或 .jpg 格式保存结果图像

最适合图像密集型的游戏，其中的许多对象会被频繁重绘

SVG（使用 XML 描述 2D 图形的语言）

不依赖分辨率

支持事件处理器

最适合带有大型渲染区域的应用程序（比如谷歌地图）

复杂度高会减慢渲染速度（任何过度使用 DOM 的应用都不快）

不适合游戏应用

link和@import的区别

link属于html标签，而@import是css提供的。

页面被加载时，link会同时被加载，而@import引用的css会等到页面加载结束后加载。（无样式内容闪烁）

link是html标签，因此没有兼容性，而@import只有IE5以上才能识别。

link方式样式的权重高于@import的。

函数内部有2个特殊的对象：arguments和this

arguments：实际传入参数组成的类数组对象

this：引用的是函数执行的环境对象

内部函数在搜索这2个变量时，只会搜索到其活动对象为止，永远不能直接访问外部函数中的这2个值，只能用that=this才能得到。

Web安全：https://www.villainhr.com/page/2016/07/12/%E7%A1%AE%E4%BF%9D%E4%BD%A0%E7%BD%91%E9%A1%B5%E7%9A%84%E5%AE%89%E5%85%A8#SQL injection

http

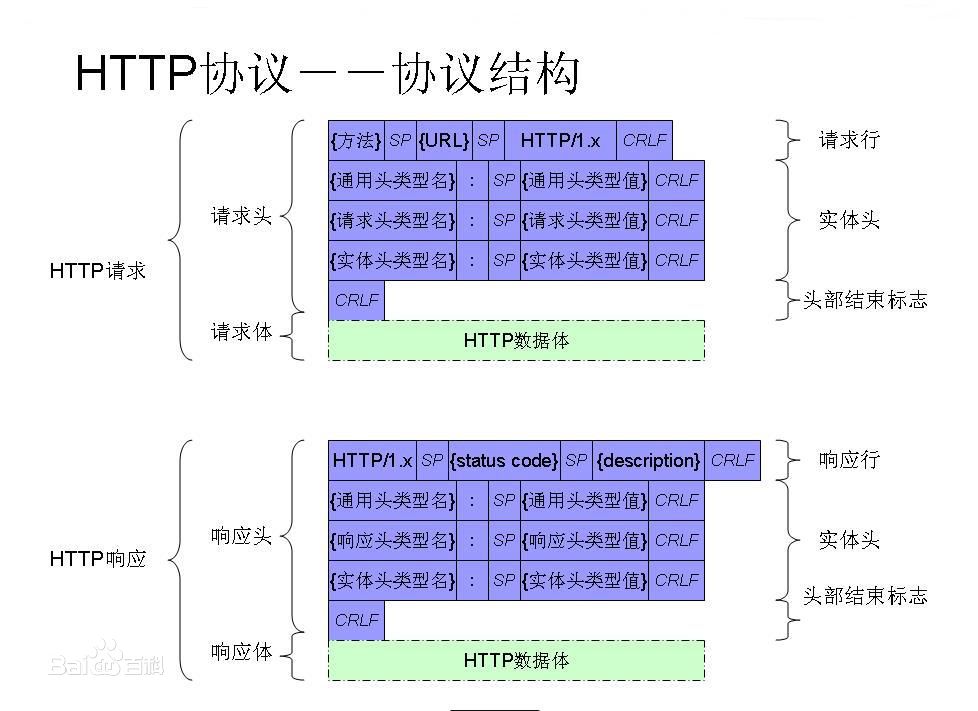
1xx:信息响应类，表示接收到请求并且继续处理

2xx:处理成功响应类，表示动作被成功接收、理解和接受

3xx:[重定向](https://baike.baidu.com/item/%E9%87%8D%E5%AE%9A%E5%90%91)响应类，为了完成指定的动作，必须接受进一步处理

4xx:客户端错误，客户请求包含语法错误或者是不能正确执行

5xx:服务端错误，服务器不能正确执行一个正确的请求



HTTP头部

Accept  就是告诉服务器端，我接受那些MIME类型

Accept-Encoding  这个看起来是接受那些压缩方式的文件

Accept-Lanague   告诉服务器能够发送哪些语言

Connection       告诉服务器支持keep-alive特性

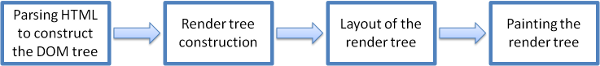
Cookie           每次请求时都会携带上Cookie以方便服务器端识别是否是同一个客户端

Host             用来标识请求服务器上的那个虚拟主机，比如Nginx里面可以定义很多个虚拟主机

                那这里就是用来标识要访问那个虚拟主机。

User-Agent       用户代理，一般情况是浏览器，也有其他类型，如：wget curl 搜索引擎的蜘蛛等

解析html以构建dom树 -> 构建render树 -> 布局render树 -> 绘制render树



js事件模型

DOM0，DOM2

DOM0：这种事件模型就是直接在dom对象上注册事件名称,而解除事件则更加简单，就是将null复制给事件函数

DOM2：在DOM2级中使用addEventListener和removeEventListener来注册和解除事件（IE8及之前版本不支持）。这种函数较之之前的方法好处是一个dom对象可以注册多个相同类型的事件，不会发生事件的覆盖，会依次的执行各个事件函数。addEventListener('事件名称','事件回调','捕获/冒泡')，首先我们要知道addEventListenr的第一个参数是事件名称，与DOM0级不同的是没有”on“，另外第三个参数代表捕获还是冒泡，true代表捕获事件，false代表冒泡事件。stopPropagation函数用来阻止冒泡。

IE没有事件捕获，event.preventDefault(),阻止默认事件

IE事件模型共有两个过程:

事件处理阶段(target phase)。事件到达目标元素, 触发目标元素的监听函数。

事件冒泡阶段(bubbling phase)。事件从目标元素冒泡到document, 依次检查经过的节点是否绑定了事件监听函数，如果有则执行。

事件绑定监听函数的方式如下:

**attachEvent**(eventType, handler)

事件移除监听函数的方式如下:

**detachEvent**(eventType, handler)

promise对象 <http://es6.ruanyifeng.com/#docs/promise>

Promise，就是一个对象，用来传递异步操作的消息。有三种状态：Pending（进行中）、Resolved（已完成，又称 Fulfilled）和 Rejected（已失败）。

有了 Promise 对象，就可以将异步操作以同步操作的流程表达出来，避免了层层嵌套的回调函数。

promise里面的then函数仅仅是注册了后续需要执行的代码，真正的执行是在resolve方法里面执行的

const promise = new Promise(function(resolve, reject) {

// ... some code

if (/\* 异步操作成功 \*/){

resolve(value);

} else {

reject(error);

}

});

Promise构造函数接受一个函数作为参数，该函数的两个参数分别是resolve和reject。它们是两个函数，由 JavaScript 引擎提供，不用自己部署。

Promise实例生成以后，可以用then方法分别指定resolved状态和rejected状态的回调函数。

promise.then(function(value) {

// success

}, function(error) {

// failure

});

下面是一个Promise对象的简单例子。

function timeout(ms) {

return new Promise((resolve, reject) => {

setTimeout(resolve, ms, 'done');

});

}

timeout(100).then((value) => {

console.log(value);

});

上面代码中，timeout方法返回一个Promise实例，表示一段时间以后才会发生的结果。过了指定的时间（ms参数）以后，Promise实例的状态变为resolved，就会触发then方法绑定的回调函数。

## [雅虎网站页面性能优化的34条黄金守则](http://www.cnblogs.com/li0803/archive/2009/09/20/1570581.html)

跨域

CORS：使用自定义的HTTP头部让浏览器和服务器进行沟通

服务器端对于CORS的支持，主要就是通过设置Access-Control-Allow-Origin来进行的。如果浏览器检测到相应的设置，就可以允许Ajax进行跨域的访问。

Access-Control-Allow-Origin:\*

图像Ping：《img》标签。常用于跟踪用户点击页面或动态广告曝光次数

JSONP：利用动态创建script，src指定一个跨域url

原理是：动态插入script标签，通过script标签引入一个js文件，这个js文件载入成功后会执行我们在url参数中指定的函数，并且会把我们需要的json数据作为参数传入。

由于同源策略的限制，XmlHttpRequest只允许请求当前源（域名、协议、端口）的资源，为了实现跨域请求，可以通过script标签实现跨域请求，然后在服务端输出JSON数据并执行回调函数，从而解决了跨域的数据请求。

优点是兼容性好，简单易用，支持浏览器与服务器双向通信。缺点是只支持GET请求。

JSONP：json+padding（内填充），顾名思义，就是把JSON填充到一个盒子里

Comet：长轮询（浏览器定时向服务器发送请求，看有没有数据更新）和流

SSE（服务器发送事件）：这个API用于创建服务器的单向连接，服务器通过这个连接可以发送任意数量的数据。

Web Sockets：在一个单独持久的连接上提供全双工、双向的通信，建立连接后HTTP协议转变为Web Socket协议。 ws:// wss://

通过修改document.domain来跨子域

将子域和主域的document.domain设为同一个主域.前提条件：这两个域名必须属于同一个基础域名!而且所用的协议，端口都要一致，否则无法利用document.domain进行跨域

主域相同的使用document.domain

使用window.name来进行跨域

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

使用HTML5中新引进的window.postMessage方法来跨域传送数据

还有flash、在服务器上设置代理页面等跨域方式。个人认为window.name的方法既不复杂，也能兼容到几乎所有浏览器，这真是极好的一种跨域方法。

惰性载入：函数执行的分支只会发生一次。（能力检测只检测一次，不用每次调用都检测）

函数绑定：bind（）方法，传入作为this值的对象。

函数柯里化：用于创建已经设置好了一个或多个参数的函数。

函数节流：常用于onresize，设置定时器延迟执行。

如果替换了prototype对象，那么，下一步必然是为新的prototype对象加上constructor属性，并将这个属性指回原来的构造函数。

　Cat.prototype = new Animal();

　Cat.prototype.constructor = Cat;

任何一个prototype对象都有一个constructor属性，指向它的构造函数。

每一个实例也有一个constructor属性，默认调用prototype对象的constructor属性。

【1】函数(Function也是函数)是new Function的结果，所以函数可以作为实例对象，其构造函数是Function()，原型对象是Function.prototype

　　【2】对象(函数也是对象)是new Object的结果，所以对象可以作为实例对象，其构造函数是Object()，原型对象是Object.prototype

　　【3】Object.prototype的原型对象是null

实例对象继承原型对象的constructor指向构造函数

实例对象的proto属性指向原型对象prototype

确定原型和实例的关系

instanceof操作符测试实例和原型链中出现过的构造参数

alert(person1 instanceof Person);//true

isPrototypeOf()方法，只要是原型链中出现的原型，都可以说是该链所派生的实例的原型

alert(Person.prototype.isPrototypeOf(person1));//true

hasOwnProperty()方法，可以检测一个属性是存在于实例中，还是原型中

alert(person1.hasOwnProperty(“name”));实例true，原型false

in操作符，会在通过对象能够访问给定属性时返回true，无论存在于实例还是原型中

alert(“name”in person1);

for-in循环返回所有能够通过对象访问的，可枚举的属性，包括实例和原型中的属性

apply和 call

每个函数都包含两个非继承而来的方法，用途都是在特定的作用域中调用函数，实际上等于设置函数体内this对象的值。第一个参数都是在其中运行函数的作用域，第二个参数apply是一个参数数组（可以是Array实例，也可以是arguments对象），call是一个个参数。

严格模式

设立”严格模式”的目的，主要有以下几个：

消除Javascript语法的一些不合理、不严谨之处，减少一些怪异行为;

消除代码运行的一些不安全之处，保证代码运行的安全；

提高编译器效率，增加运行速度；

为未来新版本的Javascript做好铺垫。

注：经过测试IE6,7,8,9均不支持严格模式。

严格模式主要有以下限制：

变量必须声明后再使用

函数的参数不能有同名属性，否则报错

不能使用with语句

不能对只读属性赋值，否则报错

不能使用前缀0表示八进制数，否则报错

不能删除不可删除的属性，否则报错

不能删除变量delete prop，会报错，只能删除属性delete global[prop]

eval不会在它的外层作用域引入变量

eval和arguments不能被重新赋值

arguments不会自动反映函数参数的变化

不能使用arguments.callee

不能使用arguments.caller

禁止this指向全局对象

不能使用fn.caller和fn.arguments获取函数调用的堆栈

增加了保留字（比如protected、static和interface）

箭头函数

箭头函数有几个使用注意点。

（1）函数体内的this对象，就是定义时所在的对象，而不是使用时所在的对象。

（2）不可以当作构造函数，也就是说，不可以使用new命令，否则会抛出一个错误。

（3）不可以使用arguments对象，该对象在函数体内不存在。如果要用，可以用 rest 参数代替。

（4）不可以使用yield命令，因此箭头函数不能用作 Generator 函数。

说说你对闭包的理解

使用闭包主要是为了设计私有的方法和变量。闭包的优点是可以避免全局变量的污染，缺点是闭包会常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。在js中，函数即闭包，只有函数才会产生作用域的概念

闭包有三个特性：

1.函数嵌套函数

2.函数内部可以引用外部的参数和变量

3.参数和变量不会被垃圾回收机制回收

闭包只能取得包含函数中任何变量的最后一个值（for循环中的i，只能取得i最后的值）

页面编码和被请求的资源编码不一致如何处理

<script src=’’ charset=’’></script>

栈和队列的区别?

栈的插入和删除操作都是在一端进行的，而队列的操作却是在两端进行的。

队列先进先出，栈先进后出。

栈只允许在表尾一端进行插入和删除，而队列只允许在表尾一端进行插入，在表头一端进行删除

栈和堆的区别？

栈区（stack）— 由编译器自动分配释放 ，存放函数的参数值，局部变量的值等。

堆区（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OS回收。

堆（数据结构）：堆可以被看成是一棵树，如：堆排序；

栈（数据结构）：一种先进后出的数据结构。

ES6的了解

新增模板字符串（为JavaScript提供了简单的字符串插值功能）、箭头函数（操作符左边为输入的参数，而右边则是进行的操作以及返回的值Inputs=>outputs。）、for-of（用来遍历数据—例如数组中的值。）arguments对象可被不定参数和默认参数完美代替。ES6将promise对象纳入规范，提供了原生的Promise对象。增加了let和const命令，用来声明变量。增加了块级作用域。let命令实际上就增加了块级作用域。ES6规定，var命令和function命令声明的全局变量，属于全局对象的属性；let命令、const命令、class命令声明的全局变量，不属于全局对象的属性。。还有就是引入module模块的概念

用过哪些设计模式？

工厂模式：

主要好处就是可以消除对象间的耦合，通过使用工程方法而不是new关键字。将所有实例化的代码集中在一个位置防止代码重复。

工厂模式解决了重复实例化的问题 ，但还有一个问题,那就是识别问题，因为根本无法搞清楚他们到底是哪个对象的实例。

function createObject(name,age,profession){//集中实例化的函数var obj = new Object();

obj.name = name;

obj.age = age;

obj.profession = profession;

obj.move = function () {

return this.name + ' at ' + this.age + ' engaged in ' + this.profession;

};

return obj;

}

var test1 = createObject('trigkit4',22,'programmer');//第一个实例var test2 = createObject('mike',25,'engineer');//第二个实例

构造函数模式

使用构造函数的方法 ，即解决了重复实例化的问题 ，又解决了对象识别的问题，该模式与工厂模式的不同之处在于：

1.构造函数方法没有显示的创建对象 (new Object());

2.直接将属性和方法赋值给 this 对象;

3.没有 renturn 语句。

使用构造函数的问题在于每个方法都要在每个实例上重新创建一遍。

原型模式

所有属性被很多实例共享，虽然在实例上添加一个同名属性，可以隐藏原型中的对应属性，但对于包含引用数据类型的属性来说，就有问题了。

构造函数模式用于定义实例属性，原型模式用于定义方法和共有的属性。

js继承方式及其优缺点

原型链继承（本质是重写原型对象，代之以一个新类型的实例）

Sub.prototype=new Super();

缺点：一是字面量重写原型会中断关系，使用引用类型的原型，并且子类型还无法给超类型传递参数。

借用构造函数（类式继承）

借用构造函数虽然解决了刚才两种问题，但没有原型，则复用无从谈起。所以我们需要原型链+借用构造函数的模式，这种模式称为组合继承

组合式继承

组合式继承是比较常用的一种继承方法，其背后的思路是 使用原型链实现对原型属性和方法的继承，而通过借用构造函数来实现对实例属性的继承。这样，既通过在原型上定义方法实现了函数复用，又保证每个实例都有它自己的属性。

XML和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。

说说TCP传输的三次握手四次挥手策略

为了准确无误地把数据送达目标处，TCP协议采用了三次握手策略。用TCP协议把数据包送出去后，TCP不会对传送 后的情况置之不理，它一定会向对方确认是否成功送达。握手过程中使用了TCP的标志：SYN和ACK。

发送端首先发送一个带SYN标志的数据包给对方。接收端收到后，回传一个带有SYN/ACK标志的数据包以示传达确认信息。

最后，发送端再回传一个带ACK标志的数据包，代表“握手”结束。

若在握手过程中某个阶段莫名中断，TCP协议会再次以相同的顺序发送相同的数据包。

断开一个TCP连接则需要“四次握手”：

第一次挥手：主动关闭方发送一个FIN，用来关闭主动方到被动关闭方的数据传送，也就是主动关闭方告诉被动关闭方：我已经不会再给你发数据了(当然，在fin包之前发送出去的数据，如果没有收到对应的ack确认报文，主动关闭方依然会重发这些数据)，但是，此时主动关闭方还可 以接受数据。

第二次挥手：被动关闭方收到FIN包后，发送一个ACK给对方，确认序号为收到序号+1（与SYN相同，一个FIN占用一个序号）。

第三次挥手：被动关闭方发送一个FIN，用来关闭被动关闭方到主动关闭方的数据传送，也就是告诉主动关闭方，我的数据也发送完了，不会再给你发数据了。

第四次挥手：主动关闭方收到FIN后，发送一个ACK给被动关闭方，确认序号为收到序号+1，至此，完成四次挥手。

说说你对作用域链的理解

作用域链的作用是保证执行环境里有权访问的变量和函数是有序的，作用域链的变量只能向上访问，变量访问到window对象即被终止，作用域链向下访问变量是不被允许的。

HTTP和HTTPS

HTTP协议通常承载于TCP协议之上，在HTTP和TCP之间添加一个安全协议层（SSL或TLS），这个时候，就成了我们常说的HTTPS。

默认HTTP的端口号为80，HTTPS的端口号为443。

为什么HTTPS安全

因为网络请求需要中间有很多的服务器路由器的转发。中间的节点都可能篡改信息，而如果使用HTTPS，密钥在你和终点站才有。https之所以比http安全，是因为他利用ssl/tls协议传输。它包含证书，卸载，流量转发，负载均衡，页面适配，浏览器适配，refer传递等。保障了传输过程的安全性

cookie

Cookie 是在 HTTP 协议下，服务器或脚本可以维护客户工作站上信息的一种方式。Cookie 是由 Web 服务器保存在用户浏览器（客户端）上的小文本文件，它可以包含有关用户的信息。无论何时用户链接到服务器，Web 站点都可以访问 Cookie 信息

1.`Cookie`数量和长度的限制。每个domain最多只能有20条cookie，每个cookie长度不能超过4KB，否则会被截掉.

2.安全性问题。如果cookie被人拦截了，那人就可以取得所有的session信息。即使加密也与事无补，因为拦截者并不需要知道cookie的意义，他只要原样转发cookie就可以达到目的了。

3.有些状态不可能保存在客户端。例如，为了防止重复提交表单，我们需要在服务器端保存一个计数器。如果我们把这个计数器保存在客户端，那么它起不到任何作用。

cookie 和session 的区别：

1、cookie数据存放在客户的浏览器上，session数据放在服务器上。

2、cookie不是很安全，别人可以分析存放在本地的COOKIE并进行COOKIE欺骗

考虑到安全应当使用session。

3、session会在一定时间内保存在服务器上。当访问增多，会比较占用你服务器的性能

考虑到减轻服务器性能方面，应当使用COOKIE。

4、单个cookie保存的数据不能超过4K，很多浏览器都限制一个站点最多保存20个cookie。

5、所以个人建议：

将登陆信息等重要信息存放为SESSION

其他信息如果需要保留，可以放在COOKIE中

内存泄漏

内存泄漏指任何对象在您不再拥有或需要它之后仍然存在。

垃圾回收器定期扫描对象，并计算引用了每个对象的其他对象的数量。如果一个对象的引用数量为 0（没有其他对象引用过该对象），或对该对象的惟一引用是循环的，那么该对象的内存即可回收。

setTimeout 的第一个参数使用字符串而非函数的话，会引发内存泄漏。

闭包、控制台日志、循环（在两个对象彼此引用且彼此保留时，就会产生一个循环）

列举IE 与其他浏览器不一样的特性？

IE支持currentStyle，FIrefox使用getComputStyle

IE 使用innerText，Firefox使用textContent

滤镜方面：IE:filter:alpha(opacity= num)；Firefox：-moz-opacity:num

事件方面：IE：attachEvent：火狐是addEventListener

鼠标位置：IE是event.clientX；火狐是event.pageX

IE使用event.srcElement；Firefox使用event.target

IE中消除list的原点仅需margin:0即可达到最终效果；FIrefox需要设置margin:0;padding:0以及list-style:none

CSS圆角：ie7以下不支持圆角

attribute和property的区别是什么？

attribute是dom元素在文档中作为html标签拥有的属性；

property就是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，但是对于自定义的属性来说，他们是不同步的

Etag

聪明的服务器开发者会把ETags和GET请求的“If-None-Match”头一起使用，这样可利用客户端（例如浏览器）的缓存。因为服务器首先产生ETag，服务器可在稍后使用它来判断页面是否已经被修改。本质上，客户端通过将该记号传回服务器要求服务器验证其（客户端）缓存。

其过程如下：

客户端请求一个页面（A）。 服务器返回页面A，并在给A加上一个ETag。 客户端展现该页面，并将页面连同ETag一起缓存。 客户再次请求页面A，并将上次请求时服务器返回的ETag一起传递给服务器。 服务器检查该ETag，并判断出该页面自上次客户端请求之后还未被修改，直接返回响应304（未修改——Not Modified）和一个空的响应体。

私有变量

包括函数的参数、局部变量和在函数内部定义的其他函数

特权方法

1.在构造函数中定义特权方法（实例变量）

function(){

//私有属性和私有函数

var privateVariable=10;

function privateFunction(){

return false;

}

//特权方法

this.publicMethod=function(){

privateVariable++;

return privateFunction();

};

}

2.在私有作用域(立即执行匿名函数)中定义私有变量和函数，同样也可以创建特权方法（静态私有变量）

(function(){

//私有属性和私有函数

var privateVariable=10;

function privateFunction(){

return false;

}

//构造函数（函数声明只能创建局部函数，出于同样原因，未使用var）

MyObject=function(){

};

//公有、特权方法

MyObject.prototype.publicMethod=function(){

privateVariable++;

return privateFunction();

};

})();

BFC（块级格式化上下文）就是一个相对独立的布局环境，它内部元素的布局不受外面布局的影响。它可以通过以下任何一种方式来创建：

float 的值不为 none

position 的值不为 static 或者 relative

display 的值为 table-cell , table-caption , inline-block , flex , 或者 inline-flex 中的其中一个

overflow 的值不为 visible

约束规则：

1、内部的Box会在垂直方向上一个接一个的放置

2、垂直方向上的距离由margin决定。（完整的说法是：属于同一个BFC的两个相邻Box的margin会发生重叠，与方向无关。）

3、每个元素的左外边距与包含块的左边界相接触（从左向右），即使浮动元素也是如此。（这说明BFC中子元素不会超出他的包含块，而position为absolute的元素可以超出他的包含块边界）

4、BFC的区域不会与float的元素区域重叠

5、计算BFC的高度时，浮动子元素也参与计算

6、BFC就是页面上的一个隔离的独立容器，容器里面的子元素不会影响到外面元素，反之亦然

分域名请求图片的原因和好处

浏览器的并发请求数目限制是针对同一域名的，超过限制数目的请求会被阻塞

浏览器并发请求有个数限制，分域名可以同时并发请求大量图片

页面的加载顺序

html顺序加载，其中js会阻塞后续dom和资源的加载，css不会阻塞dom和资源的加载但是会阻塞js的加载。

浏览器会使用prefetch对引用的资源提前下载

1.没有 defer 或 async，浏览器会立即加载并执行指定的脚本

2.有 async，加载和渲染后续文档元素的过程将和 script.js 的加载与执行并行进行(下载异步，执行同步，加载完就执行)。

3.有 defer，加载后续文档元素的过程将和 script.js 的加载并行进行（异步），但是 script.js 的执行要在所有元素解析完成之后，DOMContentLoaded 事件触发之前完成。

二叉树的遍历的三种方式

//(1)前序遍历（DLR

function preOrder(node){

if(node){

arr.push(node);

preOrder(node.firstElementChild);

preOrder(node.lastElementChild);

}

}

//(2)中序遍历（LDR）

function inOrder(node){

if(node){

inOrder(node.firstElementChild);

arr.push(node);

inOrder(node.lastElementChild);

}

}

//(3)后序遍历（LRD）

function postOrder(node){

if(node){

postOrder(node.firstElementChild);

postOrder(node.lastElementChild);

arr.push(node);

}

}