本文旨在加深对前端知识点的理解，资料来源于网络，由本人(博客：http://segmentfault.com/u/trigkit4) 收集整理。

一些开放性题目

1.自我介绍：除了基本个人信息以外，面试官更想听的是你与众不同的地方和你的优势。

2.项目介绍

3.如何看待前端开发？

4.平时是如何学习前端开发的？

5.未来三年的规划是怎样的？

position的值， relative和absolute分别是相对于谁进行定位的？

- `absolute` :生成绝对定位的元素， 相对于最近一级的 定位不是 static 的父元素来进行定位。
- `fixed` （老IE不支持）生成绝对定位的元素，相对于浏览器窗口进行定位。
- `relative` 生成相对定位的元素，相对于其在普通流中的位置进行定位。
- `static` 默认值。没有定位，元素出现在正常的流中

如何解决跨域问题

JSONP：

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

```
<script>

    function createJs(sUrl){

        var oScript = document.createElement('script');
        oScript.type = 'text/javascript';
        oScript.src = sUrl;
        document.getElementsByTagName('head')[0].appendChild(oScript);
    }

    createJs('jsonp.js');

    box({
       'name': 'test'
    });

    function box(json){
        alert(json.name);
    }
</script>
```

CORS

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。

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。

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。
```

谈谈你会webpack的看法

WebPack 是一个模块打包工具，你可以使用 WebPack 管理你的模块依赖，并编绎输出模块们所需的静态文件。它能够很好地管理、打包Web开发中所用到的 HTML、Javascript、CSS 以及各种静态文件（图片、字体等），让开发过程更加高效。对于不同类型的资源， webpack 有对应的模块加载器。 webpack 模块打包器会分析模块间的依赖关系，最后 生成了优化且合并后的静态资源。

webpack 是加强版的 Browserify 。

webpack 的两大特色：

```
1.code splitting（可以自动完成）

2.loader 可以处理各种类型的静态文件，并且支持串联操作
```

webpack的优势：

- require.js 的所有功能它都有。
- 编绎过程更快，因为 require.js 会去处理不需要的文件

webpack 是以 commonJS 的形式来书写脚本滴，但对 AMD/CMD 的支持也很全面，方便旧项目进行代码迁移。

webpack 具有 requireJs 和 browserify 的功能，但仍有很多自己的新特性：

```
1. 对 CommonJS 、 AMD 、ES6的语法做了兼容

2. 对js、css、图片等资源文件都支持打包

3. 串联式模块加载器以及插件机制，让其具有更好的灵活性和扩展性，例如提供对CoffeeScript、ES6的支持

4. 有独立的配置文件webpack.config.js

5. 可以将代码切割成不同的chunk，实现按需加载，降低了初始化时间

6. 支持 SourceUrls 和 SourceMaps，易于调试

7. 具有强大的Plugin接口，大多是内部插件，使用起来比较灵活

8.webpack 使用异步 IO 并具有多级缓存。这使得 webpack 很快且在增量编译上更加快
```

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。

TCP和UDP的区别

TCP （Transmission Control Protocol，传输控制协议）是基于连接的协议，也就是说，在正式收发数据前，必须和对方建立可靠的连接。一个 TCP 连接必须要经过三次“对话”才能建立起来

UDP （User Data Protocol，用户数据报协议）是与TCP相对应的协议。它是面向非连接的协议，它不与对方建立连接，而是直接就把数据包发送过去！ UDP适用于一次只传送少量数据、对可靠性要求不高的应用环境。

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。

onmousemove和onmouseover的区别：

时间上：onmousemove事件触发后，再触发onmouseover事件。

按钮上：不区分鼠标按钮。

动作上：onmouseover只在刚进入区域时触发，onmousemove除了刚进入区域触发外，在区域内移动鼠标，也会触发

创建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,并指定该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.

渐进增强和优雅降级

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、交互等改进和追加功能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。

常见web安全及防护原理

sql注入原理

就是通过把SQL命令插入到Web表单递交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的SQL命令。

总的来说有以下几点：

1.永远不要信任用户的输入，要对用户的输入进行校验，可以通过正则表达式，或限制长度，对单引号和双"-"进行转换等。

2.永远不要使用动态拼装SQL，可以使用参数化的SQL或者直接使用存储过程进行数据查询存取。

3.永远不要使用管理员权限的数据库连接，为每个应用使用单独的权限有限的数据库连接。

4.不要把机密信息明文存放，请加密或者hash掉密码和敏感的信息。

XSS原理及防范

Xss(cross-site scripting)攻击指的是攻击者往Web页面里插入恶意 html 标签或者 javascript 代码。比如：攻击者在论坛中放一个

看似安全的链接，骗取用户点击后，窃取 cookie 中的用户私密信息；或者攻击者在论坛中加一个恶意表单，

当用户提交表单的时候，却把信息传送到攻击者的服务器中，而不是用户原本以为的信任站点。

XSS防范方法

首先代码里对用户输入的地方和变量都需要仔细检查长度和对 "<",">",";","'" 等字符做过滤；其次任何内容写到页面之前都必须加以 encode ，避免不小心把 html tag 弄出来。这一个层面做好，至少可以堵住超过一半的 XSS 攻击。

首先，避免直接在 cookie 中泄露用户隐私，例如email、密码等等。

其次，通过使 cookie 和系统 ip 绑定来降低 cookie 泄露后的危险。这样攻击者得到的 cookie 没有实际价值，不可能拿来重放。

尽量采用 POST 而非 GET 提交表单

XSS与CSRF有什么区别吗？

XSS 是获取信息，不需要提前知道其他用户页面的代码和数据包。 CSRF 是代替用户完成指定的动作，需要知道其他用户页面的代码和数据包。

要完成一次 CSRF 攻击，受害者必须依次完成两个步骤：

> 登录受信任网站A，并在本地生成Cookie。
>
> 在不登出A的情况下，访问危险网站B。

CSRF的防御

- 服务端的 CSRF 方式方法很多样，但总的思想都是一致的，就是在客户端页面增加伪随机数。
- 通过验证码的方法

Web Worker 和webSocket

worker主线程:

> 1.通过 worker = new Worker(url) 加载一个JS文件来创建一个worker，同时返回一个worker实例。
>
> 2.通过worker.postMessage(data) 方法来向worker发送数据。
>
> 3.绑定worker.onmessage方法来接收worker发送过来的数据。
>
> 4.可以使用 worker.terminate() 来终止一个worker的执行。

WebSocket 是 Web 应用程序的传输协议，它提供了双向的，按序到达的数据流。他是一个 HTML5 协议， WebSocket 的连接是持久的，他通过在客户端和服务器之间保持双工连接，服务器的更新可以被及时推送给客户端，而不需要客户端以一定时间间隔去轮询。

HTTP和HTTPS

HTTP 协议通常承载于TCP协议之上，有时也承载于 TLS 或 SSL 协议层之上，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

对前端模块化的认识

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。

AMD 是提前执行， CMD 是延迟执行。

AMD 推荐的风格通过返回一个对象做为模块对象， CommonJS 的风格通过对 module.exports 或 exports 的属性赋值来达到暴露模块对象的目的。

CMD模块方式

```
define(function(require, exports, module) {

  // 模块代码

});
```

Javascript垃圾回收方法

标记清除（mark and sweep）

这是JavaScript最常见的垃圾回收方式，当变量进入执行环境的时候，比如函数中声明一个变量，垃圾回收器将其标记为“进入环境”，当变量离开环境的时候（函数执行结束）将其标记为“离开环境”。

垃圾回收器会在运行的时候给存储在内存中的所有变量加上标记，然后去掉环境中的变量以及被环境中变量所引用的变量（闭包），在这些完成之后仍存在标记的就是要删除的变量了

引用计数(reference counting)

在低版本IE中经常会出现内存泄露，很多时候就是因为其采用引用计数方式进行垃圾回收。引用计数的策略是跟踪记录每个值被使用的次数，当声明了一个 变量并将一个引用类型赋值给该变量的时候这个值的引用次数就加1，如果该变量的值变成了另外一个，则这个值得引用次数减1，当这个值的引用次数变为0的时 候，说明没有变量在使用，这个值没法被访问了，因此可以将其占用的空间回收，这样垃圾回收器会在运行的时候清理掉引用次数为0的值占用的空间。

在IE中虽然 JavaScript 对象通过标记清除的方式进行垃圾回收，但BOM与DOM对象却是通过引用计数回收垃圾的， 也就是说只要涉及 BOM 及DOM就会出现循环引用问题。

你觉得前端工程的价值体现在哪

```
为简化用户使用提供技术支持（交互部分）

为多个浏览器兼容性提供支持

为提高用户浏览速度（浏览器性能）提供支持

为跨平台或者其他基于webkit或其他渲染引擎的应用提供支持

为展示数据提供支持（数据接口）
```

谈谈性能优化问题

代码层面：避免使用css表达式，避免使用高级选择器，通配选择器。

缓存利用：缓存Ajax，使用CDN，使用外部js和css文件以便缓存，添加Expires头，服务端配置Etag，减少DNS查找等

请求数量：合并样式和脚本，使用css图片精灵，初始首屏之外的图片资源按需加载，静态资源延迟加载。

请求带宽：压缩文件，开启GZIP，

代码层面的优化

- 用 `hash-table` 来优化查找
- 少用全局变量
- 用 `innerHTML` 代替 `DOM` 操作，减少 `DOM` 操作次数，优化 `javascript` 性能
- 用 `setTimeout` 来避免页面失去响应
- 缓存DOM节点查找的结果
- 避免使用CSS Expression
- 避免全局查询
- 避免使用width(width会创建自己的作用域，会增加作用域链长度)
- 多个变量声明合并

移动端性能优化

1.尽量使用css3动画，开启硬件加速。适当使用 `touch` 事件代替 `click` 事件。避免使用 `css3` 渐变阴影效果。

什么是Etag？

当发送一个服务器请求时，浏览器首先会进行缓存过期判断。浏览器根据缓存过期时间判断缓存文件是否过期。

情景一：若没有过期，则不向服务器发送请求，直接使用缓存中的结果，此时我们在浏览器控制台中可以看到 `200 OK` (from cache) ，此时的情况就是完全使用缓存，浏览器和服务器没有任何交互的。

情景二：若已过期，则向服务器发送请求，此时请求中会带上①中设置的文件修改时间，和 `Etag`

然后，进行资源更新判断。服务器根据浏览器传过来的文件修改时间，判断自浏览器上一次请求之后，文件是不是没有被修改过；根据 `Etag` ，判断文件内容自上一次请求之后，有没有发生变化

情形一：若两种判断的结论都是文件没有被修改过，则服务器就不给浏览器发 `index.html` 的内容了，直接告诉它，文件没有被修改过，你用你那边的缓存吧—— `304 Not Modified` ，此时浏览器就会从本地缓存中获取 `index.html` 的内容。此时的情况叫协议缓存，浏览器和服务器之间有一次请求交互。

情形二：若修改时间和文件内容判断有任意一个没有通过，则服务器会受理此次请求，之后的操作同①

① 只有get请求会被缓存，post请求不会

Expires和Cache-Control

`Expires` 要求客户端和服务端的时钟严格同步。 `HTTP1.1` 引入 `Cache-Control` 来克服Expires头的限制。如果max-age和Expires同时出现，则max-age有更高的优先级。

```
    Cache-Control: no-cache, private, max-age=0

    ETag: abcde

    Expires: Thu, 15 Apr 2014 20:00:00 GMT

    Pragma: private

    Last-Modified: $now // RFC1123 format
```

ETag应用:

Etag 由服务器端生成，客户端通过 If-Match 或者说 If-None-Match 这个条件判断请求来验证资源是否修改。常见的是使用 If-None-Match 。请求一个文件的流程可能如下：

====第一次请求===

```
1.客户端发起 HTTP GET 请求一个文件；

2.服务器处理请求，返回文件内容和一堆Header，当然包括Etag(例如"2e681a-6-5d044840")(假设服务器支持Etag生成和已经开启了Etag).状态码200
```

====第二次请求===

```
客户端发起 HTTP GET 请求一个文件，注意这个时候客户端同时发送一个If-None-Match头，这个头的内容就是第一次请求时服务器返回的Etag：2e681a-6-5d0448402.服务器判断发送过来的Etag和计算出来的Etag匹配，因此If-None-Match为False，不返回200，返回304，客户端继续使用本地缓存；流程很简单，问题是，如果服务器又设置了Cache-Control:max-age和Expires呢，怎么办
```

答案是同时使用，也就是说在完全匹配 If-Modified-Since 和 If-None-Match 即检查完修改时间和 Etag 之后，

服务器才能返回304.(不要陷入到底使用谁的问题怪圈)

为什么使用Etag请求头?

Etag 主要为了解决 Last-Modified 无法解决的一些问题。

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。
```

快速 排序的思想并实现一个快排？

"快速排序"的思想很简单，整个排序过程只需要三步：

（1）在数据集之中，找一个基准点

（2）建立两个数组，分别存储左边和右边的数组

（3）利用递归进行下次比较

```
  <script type="text/javascript">
```

```
        function quickSort(arr){
            if(arr.length<=1){
                return arr;//如果数组只有一个数，就直接返回；
            }

            var num = Math.floor(arr.length/2);//找到中间数的索引值，如果是浮点数，则向下取整

            var numValue = arr.splice(num,1);//找到中间数的值
            var left = [];
            var right = [];

            for(var i=0;i<arr.length;i++){
                if(arr[i]<numValue){
                    left.push(arr[i]);//基准点的左边的数传到左边数组
                }
                else{
                   right.push(arr[i]);//基准点的右边的数传到右边数组
                }
            }

            return quickSort(left).concat([numValue],quickSort(right));//递归不断重复比较
        }

        alert(quickSort([32,45,37,16,2,87]));//弹出“2,16,32,37,45,87”

    </script>
```

你觉得jQuery或zepto源码有哪些写的好的地方

`jquery` 源码封装在一个匿名函数的自执行环境中，有助于防止变量的全局污染，然后通过传入window对象参数，可以使window对象作为局部变量使用，好处是当 `jquery` 中访问window对象的时候，就不用将作用域链退回到顶层作用域了，从而可以更快的访问 `window` 对象。同样，传入 `undefined` 参数，可以缩短查找undefined时的作用域链。

```
    (function( window, undefined ) {

        //用一个函数域包起来，就是所谓的沙箱

        //在这里边var定义的变量，属于这个函数域内的局部变量，避免污染全局

        //把当前沙箱需要的外部变量通过函数参数引入进来

        //只要保证参数对内提供的接口的一致性，你还可以随意替换传进来的这个参数



        window.jQuery = window.$ = jQuery;

    })( window );
```

jquery将一些原型属性和方法封装在了 `jquery.prototype` 中，为了缩短名称，又赋值给了 `jquery.fn` ，这是很形象的写法。

有一些数组或对象的方法经常能使用到，应将它们保存为局部变量以提高访问速度。

将全局对象window作为参数传入，则可以使之在匿名函数内部作为局部变量访问，提供访问速度。

`jquery` 实现的链式调用可以节约代码，所返回的都是同一个对象，可以提高代码效率。

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

js继承方式及其优缺点

原型链继承的缺点

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。

具体请看：JavaScript继承方式详解

关于Http 2.0 你知道多少？

HTTP/2 引入了“服务端推（server push）”的概念，它允许服务端在客户端需要数据之前就主动地将数据发送到客户端缓存中，从而提高性能。

HTTP/2 提供更多的加密支持

HTTP/2 使用多路技术，允许多个消息在一个连接上同时交差。

它增加了头压缩（header compression），因此即使非常小的请求，其请求和响应的 header 都只会占用很小比例的带宽。

defer并行加载js文件，会按照页面上script标签的顺序执行 async并行加载js文件，下载完成立即执行，不会按照页面上script标签的顺序执行

谈谈浮动和清除浮动

浮动的框可以向左或向右移动，直到他的外边缘碰到包含框或另一个浮动框的边框为止。由于浮动框不在文档的普通流中，所以文档的普通流的块框表现得就像浮动框不存在一样。浮动的块框会漂浮在文档普通流的块框上。

如何评价AngularJS和BackboneJS

backbone 具有依赖性，依赖 underscore.js 。 Backbone + Underscore + jQuery(or Zepto) 就比一个 AngularJS 多出了2 次HTTP请求.

Backbone 的 Model 没有与UI视图数据绑定，而是需要在View中自行操作DOM来更新或读取UI数据。 AngularJS 与此相反，Model直接与UI视图绑定， Model 与UI视图的关系，通过 directive 封装， AngularJS 内置的通用 directive ，就能实现大部分操作了，也就是说，基本不必关心 Model 与UI视图的关系，直接操作Model就行了，UI视图自动更新。

AngularJS 的 directive ，你输入特定数据，他就能输出相应UI视图。是一个比较完善的前端MVW框架，包含模板，数据双向绑定，路由，模块化，服务，依赖注入等所有功能，模板功能强大丰富，并且是声明式的，自带了丰富的 Angular 指令。

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。
```

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。

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

具体请看：详解js闭包

请你谈谈Cookie的弊端

cookie 虽然在持久保存客户端数据提供了方便，分担了服务器存储的负担，但还是有很多局限性的。

第一：每个特定的域名下最多生成20个 cookie

```
1.IE6或更低版本最多20个cookie

2.IE7和之后的版本最后可以有50个cookie。

3.Firefox最多50个cookie

4.chrome和Safari没有做硬性限制
```

IE 和 Opera 会清理近期最少使用的 cookie ， Firefox 会随机清理 cookie 。

cookie 的最大大约为 4096 字节，为了兼容性，一般不能超过 4095 字节。

IE 提供了一种存储可以持久化用户数据，叫做 userdata ，从 IE5.0 就开始支持。每个数据最多128K，每个域名下最多1M。这个持久化数据放在缓存中，如果缓存没有清理，那么会一直存在。

优点：极高的扩展性和可用性

```
1.通过良好的编程，控制保存在cookie中的session对象的大小。

2.通过加密和安全传输技术（SSL），减少cookie被破解的可能性。

3.只在cookie中存放不敏感数据，即使被盗也不会有重大损失。

4.控制cookie的生命期，使之不会永远有效。偷盗者很可能拿到一个过期的cookie。
```

缺点：

```
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。
```

浏览器本地存储

在较高版本的浏览器中， js 提供了 sessionStorage 和 globalStorage 。在 HTML5 中提供了 localStorage 来取代 globalStorage 。

html5 中的 Web Storage 包括了两种存储方式： sessionStorage 和 localStorage 。

sessionStorage 用于本地存储一个会话（session）中的数据，这些数据只有在同一个会话中的页面才能访问并且当会话结束后数据也随之销毁。因此 sessionStorage 不是一种持久化的本地存储，仅仅是会话级别的存储。

而 localStorage 用于持久化的本地存储，除非主动删除数据，否则数据是永远不会过期的。

web storage和cookie的区别

Web Storage 的概念和 cookie 相似，区别是它是为了更大容量存储设计的。 Cookie 的大小是受限的，并且每次你请求一个新的页面的时候 Cookie 都会被发送过去，这样无形中浪费了带宽，另外 cookie 还需要指定作用域，不可以跨域调用。

除此之外， Web Storage 拥有 setItem,getItem,removeItem,clear 等方法，不像 cookie 需要前端开发者自己封装 setCookie，getCookie 。

但是 cookie 也是不可以或缺的： cookie 的作用是与服务器进行交互，作为 HTTP 规范的一部分而存在 ，而 Web Storage 仅仅是为了在本地"存储"数据而生

浏览器的支持除了 IE 7 及以下不支持外，其他标准浏览器都完全支持(ie及FF需在web服务器里运行)，值得一提的是IE总是办好事，例如IE7、IE6中的 userData 其实就是 javascript 本地存储的解决方案。通过简单的代码封装可以统一到所有的浏览器都支持 web storage 。

localStorage 和 sessionStorage 都具有相同的操作方法，例如 setItem、getItem 和 removeItem 等

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

CSS 相关问题

C

display:none 和 visibility:hidden 的区别？

display:none 隐藏对应的元素，在文档布局中不再给它分配空间，它各边的元素会合拢，就当他从来不存在。

visibility:hidden 隐藏对应的元素，但是在文档布局中仍保留原来的空间。

C

CSS中 link 和 @import 的区别是？

```
(1) link属于HTML标签，而@import是CSS提供的;

(2) 页面被加载的时，link会同时被加载，而@import引用的CSS会等到页面被加载完再加载;

(3) import只在IE5以上才能识别，而link是HTML标签，无兼容问题;

(4) link方式的样式的权重 高于@import的权重.
```

C

position:absolute 和 float 属性的异同

- 共同点：对内联元素设置 `float` 和 `absolute` 属性，可以让元素脱离文档流，并且可以设置其宽高。
- 不同点： `float` 仍会占据位置， `position` 会覆盖文档流中的其他元素。

C

介绍一下box-sizing属性？

`box-sizing` 属性主要用来控制元素的盒模型的解析模式。默认值是 `content-box` 。

- `content-box` ：让元素维持W3C的标准盒模型。元素的宽度/高度由 `border + padding + content` 的宽度/高度决定，设置 `width/height` 属性指的是 `content` 部分的宽/高
- `border-box` ：让元素维持IE传统盒模型（IE6以下版本和IE6~7的怪异模式）。设置 `width/height` 属性指的是 `border + padding + content`

标准浏览器下，按照W3C规范对盒模型解析，一旦修改了元素的边框或内距，就会影响元素的盒子尺寸，就不得不重新计算元素的盒子尺寸，从而影响整个页面的布局。

C

CSS 选择符有哪些？哪些属性可以继承？优先级算法如何计算？ CSS3新增伪类有那些？

```
1.id选择器（ # myid）

2.类选择器（.myclassname）

3.标签选择器（div, h1, p）

4.相邻选择器（h1 + p）

5.子选择器（ul > li）

6.后代选择器（li a）

7.通配符选择器（ * ）

8.属性选择器（a[rel = "external"]）

9.伪类选择器（a: hover, li:nth-child）
```

优先级为:

!important > id > class > tag

important 比 内联优先级高,但内联比 id 要高

C

CSS3新增伪类举例:

```
p:first-of-type 选择属于其父元素的首个 <p> 元素的每个 <p> 元素。

p:last-of-type  选择属于其父元素的最后 <p> 元素的每个 <p> 元素。

p:only-of-type  选择属于其父元素唯一的 <p> 元素的每个 <p> 元素。

p:only-child    选择属于其父元素的唯一子元素的每个 <p> 元素。

p:nth-child(2)  选择属于其父元素的第二个子元素的每个 <p> 元素。

:enabled  :disabled 控制表单控件的禁用状态。

:checked        单选框或复选框被选中。
```

C

CSS3有哪些新特性?

```
CSS3实现圆角（border-radius），阴影（box-shadow），

对文字加特效（text-shadow、），线性渐变（gradient），旋转（transform）

transform:rotate(9deg) scale(0.85,0.90) translate(0px,-30px) skew(-9deg,0deg);//旋转,缩放,定位,倾斜

增加了更多的CSS选择器  多背景 rgba  透明度 opacity 1显示 0 隐藏

在CSS3中唯一引入的伪元素是::selection.  只能向 ::selection 选择器应用少量 CSS 属性：color、background、cursor 以及 outline。

媒体查询，多栏布局  使用 @media 查询，你可以针对不同的媒体类型定义不同的样式。@media 可以针对不同的屏幕尺寸设置不同的样式，特别是如果你需要设置设计响应式的页面，@media 是非常有用的

border-image
```

CSS3中新增了一种盒模型计算方式： box-sizing 。盒模型默认的值是 content-box , 新增的值是 padding-box 和 border-box ，几种盒模型计算元素宽高的区别如下：

content-box（默认）

布局所占宽度Width：

```
Width = width + padding-left + padding-right + border-left + border-right
```

布局所占高度Height:

```
Height = height + padding-top + padding-bottom + border-top + border-bottom

padding-box
```

布局所占宽度Width：

```
Width = width(包含padding-left + padding-right) + border-top + border-bottom
```

布局所占高度Height:

```
Height = height(包含padding-top + padding-bottom) + border-top + border-bottom

border-box
```

布局所占宽度Width：

```
Width = width(包含padding-left + padding-right + border-left + border-right)
```

布局所占高度Height:

```
Height = height(包含padding-top + padding-bottom + border-top + border-bottom)
```

对BFC规范的理解？

BFC，块级格式化上下文，一个创建了新的BFC的盒子是独立布局的，盒子里面的子元素的样式不会影响到外面的元素。在同一个BFC中的两个毗邻的块级盒在垂直方向（和布局方向有关系）的margin会发生折叠。

（W3C CSS 2.1 规范中的一个概念，它决定了元素如何对其内容进行布局，以及与其他元素的关系和相互作用。）

html部分

说说你对语义化的理解？

1，去掉或者丢失样式的时候能够让页面呈现出清晰的结构

2，有利于SEO：和搜索引擎建立良好沟通，有助于爬虫抓取更多的有效信息：爬虫依赖于标签来确定上下文和各个关键字的权重；

3，方便其他设备解析（如屏幕阅读器、盲人阅读器、移动设备）以意义的方式来渲染网页；

4，便于团队开发和维护，语义化更具可读性，是下一步吧网页的重要动向，遵循W3C标准的团队都遵循这个标准，可以减少差异化。

Doctype作用? 严格模式与混杂模式如何区分？它们有何意义?

1）、 <!DOCTYPE> 声明位于文档中的最前面，处于 <html> 标签之前。告知浏览器以何种模式来渲染文档。

2）、严格模式的排版和 JS 运作模式是 以该浏览器支持的最高标准运行。

3）、在混杂模式中，页面以宽松的向后兼容的方式显示。模拟老式浏览器的行为以防止站点无法工作。

4）、 DOCTYPE 不存在或格式不正确会导致文档以混杂模式呈现。

你知道多少种 Doctype 文档类型？

该标签可声明三种 DTD 类型，分别表示严格版本、过渡版本以及基于框架的 HTML 文档。

HTML 4.01 规定了三种文档类型：Strict、Transitional 以及 Frameset。

XHTML 1.0 规定了三种 XML 文档类型：Strict、Transitional 以及 Frameset。

Standards （标准）模式（也就是严格呈现模式）用于呈现遵循最新标准的网页，而 Quirks （包容）模式（也就是松散呈现模式或者兼容模式）用于呈现为传统浏览器而设计的网页。

HTML与XHTML——二者有什么区别

区别：

1.所有的标记都必须要有一个相应的结束标记

2.所有标签的元素和属性的名字都必须使用小写

3.所有的XML标记都必须合理嵌套

4.所有的属性必须用引号""括起来

5.把所有<和&特殊符号用编码表示

6.给所有属性赋一个值

7.不要在注释内容中使“--”

8.图片必须有说明文字

常见兼容性问题？

png24位的图片在iE6浏览器上出现背景，解决方案是做成PNG8.也可以引用一段脚本处理.

浏览器默认的margin和padding不同。解决方案是加一个全局的*{margin:0;padding:0;}来统一。

IE6双边距bug:块属性标签float后，又有横行的margin情况下，在ie6显示margin比设置的大。

浮动ie产生的双倍距离（IE6双边距问题：在IE6下，如果对元素设置了浮动，同时又设置了margin-left或margin-right，margin值会加倍。）

```
#box{ float:left; width:10px; margin:0 0 0 100px;}
```

这种情况之下IE会产生20px的距离，解决方案是在float的标签样式控制中加入 _display:inline;将其转化为行内属性。(_这个符号只有ie6会识别)

渐进识别的方式，从总体中逐渐排除局部。

首先，巧妙的使用“\9”这一标记，将IE游览器从所有情况中分离出来。

接着，再次使用“+”将IE8和IE7、IE6分离开来，这样IE8已经独立识别。

```
css

    .bb{
```

```
 background-color:#f1ee18;/*所有识别*/

.background-color:#00deff\9; /*IE6、7、8识别*/

+background-color:#a200ff;/*IE6、7识别*/

_background-color:#1e0bd1;/*IE6识别*/

}
```

怪异模式问题：漏写DTD声明，Firefox仍然会按照标准模式来解析网页，但在IE中会触发怪异模式。为避免怪异模式给我们带来不必要的麻烦，最好养成书写DTD声明的好习惯。现在可以使用[html5](http://www.w3.org/TR/html5/single-page.html)推荐的写法：`<doctype html>`

C

上下margin重合问题

ie和ff都存在，相邻的两个div的margin-left和margin-right不会重合，但是margin-top和margin-bottom却会发生重合。

解决方法，养成良好的代码编写习惯，同时采用margin-top或者同时采用margin-bottom。

解释下浮动和它的工作原理？清除浮动的技巧

浮动元素脱离文档流，不占据空间。浮动元素碰到包含它的边框或者浮动元素的边框停留。

1.使用空标签清除浮动。

这种方法是在所有浮动标签后面添加一个空标签 定义css clear:both. 弊端就是增加了无意义标签。

2.使用overflow。

给包含浮动元素的父标签添加css属性 overflow:auto; zoom:1; zoom:1用于兼容IE6。

3.使用after伪对象清除浮动。

该方法只适用于非IE浏览器。具体写法可参照以下示例。使用中需注意以下几点。一、该方法中必须为需要清除浮动元素的伪对象中设置 height:0，否则该元素会比实际高出若干像素；

浮动元素引起的问题和解决办法？

浮动元素引起的问题：

（1）父元素的高度无法被撑开，影响与父元素同级的元素

（2）与浮动元素同级的非浮动元素（内联元素）会跟随其后

（3）若非第一个元素浮动，则该元素之前的元素也需要浮动，否则会影响页面显示的结构

C

解决方法：

使用 CSS 中的 clear:both ;属性来清除元素的浮动可解决2、3问题，对于问题1，添加如下样式，给父元素添加 clearfix 样式：

```
.clearfix:after{content: ".";display: block;height: 0;clear: both;visibility: hidden;}

.clearfix{display: inline-block;} /* for IE/Mac */
```

清除浮动的几种方法：

```
1，额外标签法，<div style="clear:both;"></div>（缺点：不过这个办法会增加额外的标签使HTML结构看起来不够简洁。）

2，使用after伪类

#parent:after{

    content:".";

    height:0;

    visibility:hidden;

    display:block;

    clear:both;

    }


3,浮动外部元素

4,设置overflow为hidden或者auto
```

DOM操作——怎样添加、移除、移动、复制、创建和查找节点。

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() //并没有insertAfter()
```

C
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
```

html5有哪些新特性、移除了那些元素？如何处理HTML5新标签的浏览器兼容问题？如何区分 HTML 和 HTML5？

```
HTML5 现在已经不是 SGML 的子集，主要是关于图像，位置，存储，多任务等功能的增加。

拖拽释放(Drag and drop) API

语义化更好的内容标签（header,nav,footer,aside,article,section）

音频、视频API(audio,video)

画布(Canvas) API

地理(Geolocation) API

本地离线存储 localStorage 长期存储数据，浏览器关闭后数据不丢失；

sessionStorage 的数据在浏览器关闭后自动删除


表单控件，calendar、date、time、email、url、search

新的技术webworker, websocket, Geolocation
```

C
移除的元素

```
纯表现的元素：basefont，big，center，font, s，strike，tt，u；

对可用性产生负面影响的元素：frame，frameset，noframes；
```

C
支持HTML5新标签：

IE8/IE7/IE6支持通过document.createElement方法产生的标签，

可以利用这一特性让这些浏览器支持HTML5新标签，

当然最好的方式是直接使用成熟的框架、使用最多的是html5shim框架

```
<!--[if lt IE 9]>
<script> src="http://html5shim.googlecode.com/svn/trunk/html5.js"</script>
<![endif]-->
```

如何区分： DOCTYPE声明\新增的结构元素\功能元素

如何实现浏览器内多个标签页之间的通信?

调用localstorge、cookies等本地存储方式

什么是 FOUC（无样式内容闪烁）？你如何来避免 FOUC？

FOUC - Flash Of Unstyled Content 文档样式闪烁

```
<style type="text/css" media="all">@import "../fouc.css";</style>
```

而引用CSS文件的@import就是造成这个问题的罪魁祸首。IE会先加载整个HTML文档的DOM，然后再去导入外部的CSS文件，因此，在页面DOM加载完成到CSS导入完成中间会有一段时间页面上的内容是没有样式的，这段时间的长短跟网速，电脑速度都有关系。

解决方法简单的出奇，只要在<head>之间加入一个<link>或者<script>元素就可以了。

null和undefined的区别？

`null` 是一个表示"无"的对象，转为数值时为0； `undefined` 是一个表示"无"的原始值，转为数值时为 `NaN` 。

当声明的变量还未被初始化时，变量的默认值为 `undefined` 。

`null` 用来表示尚未存在的对象，常用来表示函数企图返回一个不存在的对象。

`undefined` 表示"缺少值"，就是此处应该有一个值，但是还没有定义。典型用法是：

（1）变量被声明了，但没有赋值时，就等于undefined。

（2） 调用函数时，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。

`null` 表示"没有对象"，即该处不应该有值。典型用法是：

（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。

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 。

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

js延迟加载的方式有哪些？

defer和async、动态创建DOM方式（创建script，插入到DOM中，加载完毕后callBack）、按需异步载入js

call() 和 apply() 的区别和作用？

作用：动态改变某个类的某个方法的运行环境（执行上下文）。

区别参见：JavaScript学习总结（四）function函数部分

哪些操作会造成内存泄漏？

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）

详见：详解js变量、作用域及内存

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

WEB应用从服务器主动推送Data到客户端有那些方式？

Javascript数据推送

- `Commet` ：基于HTTP长连接的服务器推送技术
- 基于 `WebSocket` 的推送方案
- `SSE` （Server-Send Event）：服务器推送数据新方式

对前端界面工程师这个职位是怎么样理解的？它的前景会怎么样？

前端是最贴近用户的程序员，比后端、数据库、产品经理、运营、安全都近。

1、实现界面交互

2、提升用户体验

3、有了Node.js，前端可以实现服务端的一些事情

前端是最贴近用户的程序员，前端的能力就是能让产品从 90分进化到 100 分，甚至更好，

参与项目，快速高质量完成实现效果图，精确到1px；

与团队成员，UI设计，产品经理的沟通；

做好的页面结构，页面重构和用户体验；

处理hack，兼容、写出优美的代码格式；

针对服务器的优化、拥抱最新前端技术。

一个页面从输入 URL 到页面加载显示完成，这个过程中都发生了什么？

分为4个步骤：

（1），当发送一个URL请求时，不管这个URL是Web页面的URL还是Web页面上每个资源的URL，浏览器都会开启一个线程来处理这个请求，同时在远程DNS服务器上启动一个DNS查询。这能使浏览器获得请求对应的IP地址。

（2）， 浏览器与远程`Web`服务器通过`TCP`三次握手协商来建立一个`TCP/IP`连接。该握手包括一个同步报文，一个同步-应答报文和一个应答报文，这三个报文在 浏览器和服务器之间传递。该握手首先由客户端尝试建立起通信，而后服务器应答并接受客户端的请求，最后由客户端发出该请求已经被接受的报文。

（3），一旦`TCP/IP`连接建立，浏览器会通过该连接向远程服务器发送`HTTP`的`GET`请求。远程服务器找到资源并使用HTTP响应返回该资源，值为200的HTTP响应状态表示一个正确的响应。

（4），此时，`Web`服务器提供资源服务，客户端开始下载资源。

请求返回后，便进入了我们关注的前端模块

简单来说，浏览器会解析`HTML`生成`DOM Tree`，其次会根据CSS生成CSS Rule Tree，而`javascript`又可以根据`DOM API`操作`DOM`

详情：从输入 URL 到浏览器接收的过程中发生了什么事情？

javascript对象的几种创建方式

1，工厂模式

2，构造函数模式

3，原型模式

4，混合构造函数和原型模式

5，动态原型模式

6，寄生构造函数模式

7，稳妥构造函数模式

javascript继承的6种方法

1，原型链继承

2，借用构造函数继承

3，组合继承(原型+借用构造)

4，原型式继承

5，寄生式继承

6，寄生组合式继承

详情：JavaScript继承方式详解

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,并指定该`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.

```
var xmlHttp = new XMLHttpRequest();

xmlHttp.open('GET','demo.php','true');

xmlHttp.send()

xmlHttp.onreadystatechange = function(){
```

```
        if(xmlHttp.readyState === 4 & xmlHttp.status === 200){

        }

    }
```

详情：[JavaScript学习总结（七）Ajax和Http状态字]

异步加载和延迟加载

```
1.异步加载的方案： 动态插入script标签

2.通过ajax去获取js代码，然后通过eval执行

3.script标签上添加defer或者async属性

4.创建并插入iframe，让它异步执行js

5.延迟加载：有些 js 代码并不是页面初始化的时候就立刻需要的，而稍后的某些情况才需要的。
```

ie各版本和chrome可以并行下载多少个资源

```
IE6 两个并发，iE7升级之后的6个并发，之后版本也是6个
Firefox，chrome也是6个
```

Flash、 Ajax 各自的优缺点，在使用中如何取舍？

- Flash 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

请解释一下 JavaScript 的同源策略。

概念:同源策略是客户端脚本（尤其是 Javascript ）的重要的安全度量标准。它最早出自 Netscape Navigator2.0 ，其目的是防止某个文档或脚本从多个不同源装载。

这里的同源策略指的是：协议，域名，端口相同，同源策略是一种安全协议。

指一段脚本只能读取来自同一来源的窗口和文档的属性。

为什么要有同源限制？

我们举例说明：比如一个黑客程序，他利用 Iframe 把真正的银行登录页面嵌到他的页面上，当你使用真实的用户名，密码登录时，他的页面就可以通过 Javascript 读取到你的表单中 input 中的内容，这样用户名，密码就轻松到手了。

缺点：

现在网站的 JS 都会进行压缩，一些文件用了严格模式，而另一些没有。这时这些本来是严格模式的文件，被 merge 后，这个串就到了文件的中间，不仅没有指示严格模式，反而在压缩后浪费了字节。

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，对所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，对所发送的信息没有限制。
```

GET方式需要使用Request.QueryString来取得变量的值，而POST方式通过Request.Form来获取变量的值，

也就是说Get是通过地址栏来传值，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

事件、IE与火狐的事件机制有什么区别？ 如何阻止冒泡？

1．我们在网页中的某个操作（有的操作对应多个事件）。例如：当我们点击一个按钮就会产生一个事件。是可以被JavaScript 侦测到的行为。

2．事件处理机制：IE是事件冒泡、firefox同时支持两种事件模型，也就是：捕获型事件和冒泡型事件。；

3．`ev.stopPropagation()`;注意旧ie的方法 `ev.cancelBubble = true`;

ajax的缺点和在IE下的问题？

详情请见：JavaScript学习总结（七）Ajax和Http状态字

C ajax的缺点

1、ajax不支持浏览器back按钮。

2、安全问题 AJAX暴露了与服务器交互的细节。

3、对搜索引擎的支持比较弱。

4、破坏了程序的异常机制。

5、不容易调试。

C IE缓存问题

在IE浏览器下，如果请求的方法是 GET ，并且请求的 URL 不变，那么这个请求的结果就会被缓存。解决这个问题的办法可以通过实时改变请求的 URL ，只要URL改变，就不会被缓存，可以通过在URL末尾添加上随机的时间戳参数('t'= + new Date().getTime())

或者：

```
open('GET','demo.php?rand=+Math.random()',true);//
```

Ajax请求的页面历史记录状态问题

可以通过锚点来记录状态，`location.hash`。让浏览器记录Ajax请求时页面状态的变化。

还可以通过`HTML5`的`history.pushState`，来实现浏览器地址栏的无刷新改变

js对象的深度克隆

```
function clone(Obj) {
      var buf;
      if (Obj instanceof Array) {
          buf = [];  //创建一个空的数组
          var i = Obj.length;
          while (i--) {
              buf[i] = clone(Obj[i]);
          }
          return buf;
      }else if (Obj instanceof Object){
          buf = {};  //创建一个空对象
          for (var k in Obj) {  //为这个对象添加新的属性
              buf[k] = clone(Obj[k]);
          }
          return buf;
      }else{
          return Obj;
      }
  }
```

网站重构的理解？

网站重构：在不改变外部行为的前提下，简化结构、添加可读性，而在网站前端保持一致的行为。也就是说是在不改变UI的情况下，对网站进行优化，在扩展的同时保持一致的UI。

对于传统的网站来说重构通常是：

表格(table)布局改为DIV+CSS

使网站前端兼容于现代浏览器(针对于不合规范的CSS、如对IE6有效的)

对于移动平台的优化

针对于SEO进行优化

深层次的网站重构应该考虑的方面

减少代码间的耦合

让代码保持弹性

严格按规范编写代码

设计可扩展的API

代替旧有的框架、语言(如VB)

增强用户体验

通常来说对于速度的优化也包含在重构中

压缩JS、CSS、image等前端资源(通常是由服务器来解决)

程序的性能优化(如数据读写)

采用CDN来加速资源加载

对于JS DOM的优化

HTTP服务器的文件缓存

js数组去重

以下是数组去重的三种方法：

```
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;
}
```

HTTP状态码

100 Continue 继续，一般在发送post请求时，已发送了http header之后服务端将返回此信息，表示确认，之后发送具体参数信息

200 OK 正常返回信息

201 Created 请求成功并且服务器创建了新的资源

202 Accepted 服务器已接受请求，但尚未处理

301 Moved Permanently 请求的网页已永久移动到新位置。

302 Found 临时性重定向。

303 See Other 临时性重定向，且总是使用 GET 请求新的 URI。

```
304  Not Modified  自从上次请求后，请求的网页未修改过。

400 Bad Request  服务器无法理解请求的格式，客户端不应当尝试再次使用相同的内容发起请求。

401 Unauthorized  请求未授权。

403 Forbidden  禁止访问。

404 Not Found  找不到如何与 URI 相匹配的资源。

500 Internal Server Error  最常见的服务器端错误。

503 Service Unavailable 服务器端暂时无法处理请求（可能是过载或维护）。
```

说说你对Promise的理解

依照 `Promise/A+` 的定义， `Promise` 有四种状态：

```
pending: 初始状态, 非 fulfilled 或 rejected.

fulfilled: 成功的操作.

rejected: 失败的操作.

settled: Promise已被fulfilled或rejected，且不是pending
```

另外， `fulfilled` 与 `rejected` 一起合称 `settled`。

`Promise` 对象用来进行延迟(deferred) 和异步(asynchronous) 计算。

Promise 的构造函数

构造一个 `Promise` ，最基本的用法如下：

```
var promise = new Promise(function(resolve, reject) {
    if (...) {  // succeed
        resolve(result);
    } else {   // fails
        reject(Error(errMessage));
    }
});
```

`Promise` 实例拥有 `then` 方法（具有 `then` 方法的对象，通常被称为 `thenable` ）。它的使用方法如下：

```
promise.then(onFulfilled, onRejected)
```

接收两个函数作为参数，一个在 `fulfilled` 的时候被调用，一个在 `rejected` 的时候被调用，接收参数就是 `future`，`onFulfilled` 对应 `resolve`，`onRejected` 对应 `reject`。

说说你对前端架构师的理解

负责前端团队的管理及与其他团队的协调工作，提升团队成员能力和整体效率； 带领团队完成研发工具及平台前端部分的设计、研发和维护； 带领团队进行前端领域前沿技术研究及新技术调研，保证团队的技术领先 负责前端开发规范制定、功能模块化设计、公共组件搭建等工作，并组织培训。

实现一个函数clone，可以对JavaScript中的5种主要的数据类型（包括Number、String、Object、Array、Boolean）进行值复制

```
Object.prototype.clone = function(){
        var o = this.constructor === Array ? [] : {};
        for(var e in this){
                o[e] = typeof this[e] === "object" ? this[e].clone() : this[e];
        }
        return o;
}
```

说说严格模式的限制

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）

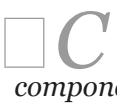
设立"严格模式"的目的，主要有以下几个：

- 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。

什么是组件？

所谓组件，即封装起来的具有独立功能的UI部件。



component , widget , module , plugin ...etc

如何删除一个cookie

1.将时间设为当前时间往前一点。

```
var date = new Date();

date.setDate(date.getDate() - 1);//真正的删除
```

setDate() 方法用于设置一个月的某一天。

2.expires的设置

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

， 和<b> ， <i>标签

```
    <strong> 标签和 <em> 标签一样，用于强调文本，但它强调的程度更强一些。

    em 是 斜体强调标签，更强烈强调，表示内容的强调点。相当于html元素中的 <i>...</i>;

    < b > < i >是视觉要素，分别表示无意义的加粗，无意义的斜体。

    em 和 strong 是表达要素(phrase elements)。
```

说说你对AMD和Commonjs的理解

CommonJS 是服务器端模块的规范，Node.js采用了这个规范。 CommonJS 规范加载模块是同步的，也就是说，只有加载完成，才能执行后面的操作。AMD规范则是非同步加载模块，允许指定回调函数。

AMD 推荐的风格通过返回一个对象做为模块对象， CommonJS 的风格通过对 module.exports 或 exports 的属性赋值来达到暴露模块对象的目的。

document.write()的用法

document.write() 方法可以用在两个方面：页面载入过程中用实时脚本创建页面内容，以及用延时脚本创建本窗口或新窗口的内容。

document.write 只能重绘整个页面。 innerHTML 可以重绘页面的一部分

编写一个方法 求一个字符串的字节长度

假设：一个英文字符占用一个字节，一个中文字符占用两个字节

```
 function GetBytes(str){
        var len = str.length;
        var bytes = len;
        for(var i=0; i<len; i++){
            if (str.charCodeAt(i) > 255) bytes++;
        }
        return bytes;
    }
alert(GetBytes("你好,as"));
```

git fetch和git pull的区别

```
git pull：相当于是从远程获取最新版本并merge到本地
git fetch：相当于是从远程获取最新版本到本地，不会自动merge
```

如何居中一个浮动元素

父元素和子元素同时左浮动，然后父元素相对左移动50%，再然后子元素相对右移动50%，或者子元素相对左移动-50%也就可以了。

```
<style type="text/css">
    .p{
        position:relative;
        left:50%;
        float:left;
    }
    .c{
        position:relative;
```

```
        float:left;
        right:50%;
    }
</style>

<div class="p">
    <h1 class="c">Test Float Element Center</h1>
</div>
```

css实现水平垂直居中

```
<style type="text/css">
    .align-center{
        /*
    负边距+定位：水平垂直居中（Negative Margin）

    使用绝对定位将content的定点定位到body的中心，然后使用负margin（content宽高的一半），
    将content的中心拉回到body的中心，已到达水平垂直居中的效果。
        */
        position:absolute;
        left:50%;
        top:50%;
        width:400px;
        height:400px;
        margin-top:-200px;
        margin-left:-200px;
        border:1px dashed #333;
    }
```

css实现三栏布局，中间自适应

方法一：自身浮动法。左栏左浮动，右栏右浮动。

```
.left , .right{
            height: 300px;
            width: 200px;
        }
        .right{
            float: right;
            background-color: red;
        }
        .left{
            float: left;
            background-color: #080808;
        }
        .middle{
            height: 300px;
            margin: 0 200px;//没有这行，当宽度缩小到一定程度的时候，中间的内容可能换行
            background-color: blue;
        }
```

C

方法二：margin负值法

```
<style>
        body{
            margin: 0;
            padding: 0;
        }
        .left , .right{
            height: 300px;
            width: 200px;
            float: left;
        }
```

```
            .right{
                margin-left: -200px;
                background-color: red;
            }
            .left{
                margin-left: -100%;
                background-color: #080808;
            }
            .middle{
                height: 300px;
                width: 100%;
                float: left;
                background-color: blue;
            }
        </style>

    <!--放第一行-->
    <div class="middle">middle</div>
    <div class="left">left</div>
    <div class="right">right</div>
```

C

方法三：绝对定位法。左右两栏采用绝对定位，分别固定于页面的左右两侧，中间的主体栏用左右margin值撑开距离。

```
<style>
    body{
        margin: 0;
        padding: 0;
    }
    .left , .right{
        top: 0;
        height: 300px;
```

```
        width: 200px;
        position: absolute;
    }
    .right{
        right: 0;
        background-color: red;
    }
    .left{
        left: 0;
        background-color: #080808;
    }
    .middle{
        margin: 0 200px;
        height: 300px;
        background-color: blue;
    }
</style>

<div class="left">left</div>
<!--这种方法没有严格限定中间这栏放置何处-->
<div class="middle">middle</div>
<div class="right">right</div>
```

js常用设计模式的实现思路，单例，工厂，代理，装饰，观察者模式等

参考答案：

```
    1) 单例：  任意对象都是单例，无须特别处理

    var obj = {name: 'michaelqin', age: 30};

    2) 工厂：就是同样形式参数返回不同的实例
    function Person() { this.name = 'Person1'; }
    function Animal() { this.name = 'Animal1'; }

    function Factory() {}
    Factory.prototype.getInstance = function(className) {
        return eval('new ' + className + '()');
    }

    var factory = new Factory();
    var obj1 = factory.getInstance('Person');
```

```
var obj2 = factory.getInstance('Animal');
console.log(obj1.name); // Person1
console.log(obj2.name); // Animal1

3) 代理: 就是新建个类调用老类的接口,包一下
function Person() { }
Person.prototype.sayName = function() { console.log('michaelqin'); }
Person.prototype.sayAge = function() { console.log(30); }

function PersonProxy() {
    this.person = new Person();
    var that = this;
    this.callMethod = function(functionName) {
        console.log('before proxy:', functionName);
        that.person[functionName](); // 代理
        console.log('after proxy:', functionName);
    }
}

var pp = new PersonProxy();
pp.callMethod('sayName'); // 代理调用Person的方法sayName()
pp.callMethod('sayAge'); // 代理调用Person的方法sayAge()

4) 观察者: 就是事件模式，比如按钮的onclick这样的应用.
function Publisher() {
    this.listeners = [];
}
Publisher.prototype = {
    'addListener': function(listener) {
        this.listeners.push(listener);
    },

    'removeListener': function(listener) {
        delete this.listeners[listener];
    },

    'notify': function(obj) {
        for(var i = 0; i < this.listeners.length; i++) {
            var listener = this.listeners[i];
            if (typeof listener !== 'undefined') {
                listener.process(obj);
            }
        }
    }
}; // 发布者

function Subscriber() {

}
Subscriber.prototype = {
    'process': function(obj) {
        console.log(obj);
    }
};  // 订阅者

var publisher = new Publisher();
publisher.addListener(new Subscriber());
publisher.addListener(new Subscriber());
publisher.notify({name: 'michaelqin', ageo: 30}); // 发布一个对象到所有订阅者
publisher.notify('2 subscribers will both perform process'); // 发布一个字符串到所有订阅者
```

说说你对MVC和MVVM的理解


```
View 传送指令到 Controller

Controller 完成业务逻辑后，要求 Model 改变状态

Model 将新的数据发送到 View，用户得到反馈
```

所有通信都是单向的。

Angular 它采用双向绑定（data-binding）：View 的变动，自动反映在 ViewModel ，反之亦然。

```
组成部分Model、View、ViewModel


View：UI界面


ViewModel：它是View的抽象，负责View与Model之间信息转换，将View的Command传送到Model；


Model：数据访问层
```

请解释什么是事件代理

事件代理（Event Delegation），又称之为事件委托。是 JavaScript 中常用绑定事件的常用技巧。顾名思义，“事件代理”即是把原本需要绑定的事件委托给父元素，让父元素担当事件监听的职务。事件代理的原理是 DOM 元素的事件冒泡。使用事件代理的好处是可以提高性能。

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，

说说网络分层里七层模型是哪七层

应用层：应用层、表示层、会话层（从上往下）（ HTTP、FTP、SMTP、DNS ）

传输层（ TCP 和 UDP ）

网络层（ IP ）

物理和数据链路层（以太网）


```
    物理层：通过媒介传输比特,确定机械及电气规范（比特Bit）
```

```
数据链路层：将比特组装成帧和点到点的传递（帧Frame）
网络层：负责数据包从源到宿的传递和网际互连（包PacketT）
传输层：提供端到端的可靠报文传递和错误恢复（段Segment）
会话层：建立、管理和终止会话（会话协议数据单元SPDU）
表示层：对数据进行翻译、加密和压缩（表示协议数据单元PPDU）
应用层：允许访问OSI环境的手段（应用协议数据单元APDU）
```

ICMP协议 ： 因特网控制报文协议。它是TCP/IP协议族的一个子协议，用于在IP主机、路由器之间传递控制消息。 TFTP协议 ： 是TCP/IP协议族中的一个用来在客户机与服务器之间进行简单文件传输的协议，提供不复杂、开销不大的文件传输服务。 HTTP协议 ： 超文本传输协议，是一个属于应用层的面向对象的协议，由于其简捷、快速的方式，适用于分布式超媒体信息系统。 DHCP协议 ： 动态主机配置协议，是一种让系统得以连接到网络上，并获取所需要的配置参数手段。

说说mongoDB和MySQL的区别

MySQL 是传统的关系型数据库， MongoDB 则是非关系型数据库

mongodb 以 BSON 结构（二进制）进行存储，对海量数据存储有着很明显的优势。

讲讲304缓存的原理

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。

304是HTTP状态码，服务器用来标识这个文件没修改，不返回内容，浏览器在接收到个状态码后，会使用浏览器已缓存的文件

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。

什么样的前端代码是好的

高复用低耦合，这样文件小，好维护，而且好扩展。

Edit By MaHua