**2017年面试题目总结：**

1. 大部分公司（创业公司）都趋向于招一个牛逼的前端而不是三四个平庸的前端

2. 性能优化、ES6必问

3. 招聘要求上清一色的要求有一门后端语言的经验，但似乎面试的时候并没有一个公司问过我后端语言方面的东西

4. 招聘要求写的和面试相关性并不是很高

5. 招人的要求越来越高，不要轻易离职，特别是裸辞

以下是我整理一些我觉得具有代表性的题目，刚好**30题**，吐血献上！

**0.谈谈对前端安全的理解，有什么，怎么防范**

前端安全问题主要有XSS、CSRF攻击

**XSS：跨站脚本攻击，指恶意攻击者往web页面插入恶意的HTML代码，当用户浏览该页时，嵌入其中web里面的HTML代码会被执行，从而达到恶意用户的特殊目的。**

**XSS的防御措施：**

1. 过滤转义输入输出

2. 避免使用eval、new Function等执行字符串的方法，除非确定字符串和用户输入无关

3. 使用cookie的httpOnly属性，加上了这个属性的cookie字段，js是无法进行读写的

4. 使用innerHTML、document.write的时候，如果数据是用户输入的，那么需要对象关键字符进行过滤与转义

**CSRF：跨站请求伪造**  
 其实就是网站中的一些提交行为，被黑客利用，在你访问黑客的网站的时候进行操作，会被操作到其他网站上

**CSRF防御措施：**

1. 检测http referer是否是同域名

2. 避免登录的session长时间存储在客户端中

3. 关键请求使用验证码或者token机制

其他的一些攻击方法还有HTTP劫持、界面操作劫持

**1.使用箭头函数需要注意的地方**

当要求动态上下文的时候，你就不能使用箭头函数，比如：定义方法，用构造器创建对象，处理时间时用 this 获取目标。

**2.webpack.load的原理**

loaders是你用在app源码上的转换元件。他们是用node.js运行的，把源文件作为参数，返回新的资源的函数。

**3.ES6 let、const**

**let**  
let是更完美的var

1. let声明的变量拥有块级作用域,let声明仍然保留了提升的特性，但不会盲目提升。

2. let声明的全局变量不是全局对象的属性。不可以通过window.变量名的方式访问

3. 形如for (let x…)的循环在每次迭代时都为x创建新的绑定

4. let声明的变量直到控制流到达该变量被定义的代码行时才会被装载，所以在到达之前使用该变量会触发错误。

**const**定义常量值，不可以重新赋值，但是如果值是一个对象，可以改变对象里的属性值

const OBJ = {"a":1, "b":2}; OBJ.a = 3; OBJ = {};// 重新赋值，报错！ console.log(OBJ.a); // 3

**4.CSS3 box-sizing的作用**

设置CSS盒模型为标准模型或IE模型。标准模型的宽度只包括content，二IE模型包括border和padding

box-sizing属性可以为三个值之一：

1. content-box，默认值，border和padding不计算入width之内

2. padding-box，padding计算入width内

3. border-box，border和padding计算入width之内

**5.说说HTML5中有趣的标签（新标签及语义化）**

如果代码写的语义化，有利于SEO。搜索引擎就会很容易的读懂该网页要表达的意思。例如文本模块要有大标题，合理利用h1-h6，列表形式的代码使用ul或dl，重要的文字使用strong等等。总之就是要充分利用各种HTML标签完成他们本职的工作

**6.git命令，如何批量删除分支**

git branch |grep 'branchName' |xargs git branch -D,从分支列表中匹配到指定分支，然后一个一个(分成小块)传递给删除分支的命令，最后进行删除。(参考[这里](http://link.zhihu.com/?target=http%3A//www.jianshu.com/p/3602f666799a)http://www.jianshu.com/p/3602f666799a)

**7.创建对象的三种方法**

**第一种方式，字面量**

var o1 = {name: "o1"}

var o2 = new Object({name: "o2"})

**第二种方式，通过构造函数**

var M = function(name){ this.name = name }

var o3 = new M("o3")

**第三种方式，Object.create**

var p = {name: "p"}

var o4 = Object.create(p)

新创建的对o4的原型就是p，同时o4也拥有了属性name

**8.JS实现继承的几种方式**

**借用构造函数实现继承**

function Parent1(){

this.name = "parent1"

}

function Child1(){

Parent1.call(this); this.type = "child1";

}

缺点：Child1无法继承Parent1的原型对象，并没有真正的实现继承（部分继承）

**借用原型链实现继承**

function Parent2(){

this.name = "parent2";

this.play = [1,2,3];

}

function Child2(){

this.type = "child2";

}

Child2.prototype = new Parent2();

缺点：原型对象的属性是共享的

**组合式继承**

function Parent3(){

this.name = "parent3";

this.play = [1,2,3];

}

function Child3(){

Parent3.call(this);

this.type = "child3";

}

Child3.prototype = Object.create(Parent3.prototype);

Child3.prototype.constructor = Child3;

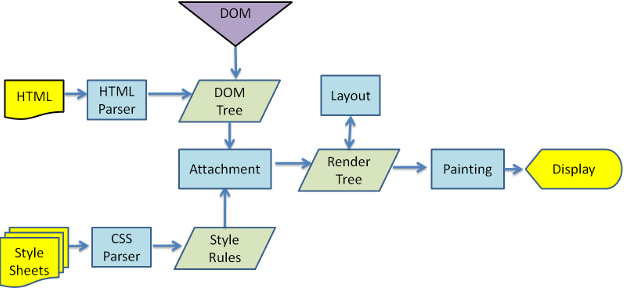
**9.当new Foo()时发生了什么**

1.创建了一个新对象  
2.将this指向这个新对象  
3.执行构造函数里面的代码  
4.返回新对象（this）  
参考《JS高程》6.6.2

**10.你做过哪些性能优化**

雪碧图，移动端响应式图片，静态资源CDN，减少Dom操作（事件代理、fragment），压缩JS和CSS、HTML等，DNS预解析

**11.浏览器渲染原理**

首先来看一张图：  


1. HTML被解析成DOM Tree，CSS被解析成CSS Rule Tree

2. 把DOM Tree和CSS Rule Tree经过整合生成Render Tree（布局阶段）

3. 元素按照算出来的规则，把元素放到它该出现的位置，通过显卡画到屏幕上

更多详情看[这里](http://link.zhihu.com/?target=http%3A//www.cnblogs.com/slly/p/6640761.html)http://www.cnblogs.com/slly/p/6640761.html

**12.前端路由的原理**

**什么是路由？**简单的说，路由是根据不同的 url 地址展示不同的内容或页面

**使用场景？**前端路由更多用在单页应用上, 也就是SPA, 因为单页应用, 基本上都是前后端分离的, 后端自然也就不会给前端提供路由。

前端的路由和后端的路由在实现技术上不一样，但是原理都是一样的。在 HTML5 的 history API 出现之前，前端的路由都是通过 hash 来实现的，hash 能兼容低版本的浏览器。

**两种实现前端路由的方式 HTML5 History两个新增的API：**history.pushState 和 history.replaceState，两个 API 都会操作浏览器的历史记录，而不会引起页面的刷新。

Hash就是url 中看到 # ,我们需要一个根据监听哈希变化触发的事件( hashchange) 事件。我们用 window.location 处理哈希的改变时不会重新渲染页面，而是当作新页面加到历史记录中，这样我们跳转页面就可以在 hashchange 事件中注册 ajax 从而改变页面内容。

**优点**  
 从性能和用户体验的层面来比较的话，后端路由每次访问一个新页面的时候都要向服务器发送请求，然后服务器再响应请求，这个过程肯定会有延迟。而前端路由在访问一个新页面的时候仅仅是变换了一下路径而已，没有了网络延迟，对于用户体验来说会有相当大的提升。

更多内容请看[这里](http://link.zhihu.com/?target=http%3A//www.cnblogs.com/yuqing6/p/6731980.html)http://www.cnblogs.com/yuqing6/p/6731980.html

**缺点**使用浏览器的前进，后退键的时候会重新发送请求，没有合理地利用缓存。

**13.Restful API是什么**

1. Restful的意思就是表现层状态转化。

2. "表现层"其实指的是"资源"（Resources）的"表现层"，把"资源"具体呈现出来的形式，叫做它的"表现层"（Representation）。

3. 所谓"资源"，就是网络上的一个实体，或者说是网络上的一个具体信息。它可以是一段文本、一张图片、一首歌曲、一种服务，总之就是一个具体的实在，每一个URI代表一种资源。

4. 如果客户端想要操作服务器，必须通过某种手段，让服务器端发生"状态转化"（State Transfer）。而这种转化是建立在表现层之上的，所以就是"表现层状态转化"。

5. Restful就是客户端和服务器之间，传递这种资源的某种表现层

6. 客户端通过四个HTTP动词，对服务器端资源进行操作，实现"表现层状态转化"

Restful API就是符合Restful架构的API设计。

**Restful API一些具体实践：**

1. 应该尽量将API部署在专用域名之下。如果确定API很简单，不会有进一步扩展，可以考虑放在主域名下。

2. 应该将API的版本号放入URL。

3. 对于资源的具体操作类型，由HTTP动词表示

4. 如果记录数量很多，服务器不可能都将它们返回给用户。API应该提供参数，过滤返回结果

5. 如果状态码是4xx，就应该向用户返回出错信息。一般来说，返回的信息中将error作为键名  
.....

**14.script标签的defer、async的区别**

**defer**是在HTML解析完之后才会执行，如果是多个，按照加载的顺序依次执行

**async**是在加载完成后立即执行，如果是多个，执行顺序和加载顺序无关

**15.同源与跨域**

**什么是同源策略？**  
限制从一个源加载的文档或脚本如何与来自另一个源的资源进行交互。  
一个源指的是主机名、协议和端口号的组合，必须相同

**跨域通信的几种方式**

• JSONP

• Hash

• postMessage

• WebSocket

• CORS

**JSONP原理**基本原理：利用script标签的异步加载特性实现  
给服务端传一个回调函数，服务器返回一个传递过去的回调函数名称的JS代码

更多请查看：[《前后端通信类知识》](http://link.zhihu.com/?target=http%3A//34585f3f.wiz03.com/share/s/0Qm5Y_0RRQtc2F-3Zy2piy1K1bpkC_0tckUz2WUwOp2JCve5)

http://34585f3f.wiz03.com/share/s/0Qm5Y\_0RRQtc2F-3Zy2piy1K1bpkC\_0tckUz2WUwOp2JCve5

**16.原型与闭包相关问题**

**原型是什么**  
原型就是一个普通的对象，每个对象都有一个原型（Object除外），原型能存储我们的方法，构造函数创建出来的实例对象能够引用原型中的方法。

**查看原型**以前一般使用对象的\_\_proto\_\_属性，ES6推出后，推荐用Object.getPrototypeOf()方法来获取对象的原型

**闭包是什么？**  
1.专业说法：当一个内部函数被其外部函数之外的变量引用时，就形成了一个闭包。

2.还可以这么理解：  
闭包就是一个具有封闭功能与包裹功能的结构，是为了实现具有私有访问空间的函数的，函数可以构成闭包，因为函数内部定义的数据函数外部无法访问，即函数具有封闭性；函数可以封装代码即具有包裹性，所以函数可以构成闭包。  
3.创建闭包的最常见的方式就是在一个函数内创建另一个函数，通过另一个函数访问这个函数的局部变量

**闭包的特性**  
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

**闭包有什么用，使用场景**  
当我们需要在模块中定义一些变量，并希望这些变量一直保存在内存中但又不会“污染”全局的变量时，就可以用闭包来定义这个模块。

**闭包的缺点**闭包的缺点就是常驻内存，会增大内存使用量，使用不当很容易造成内存泄露。

**函数套函数就是闭包吗？**不是！，当一个内部函数被其外部函数之外的变量引用时，才会形成了一个闭包。

更多内容请看[这里](http://link.zhihu.com/?target=http%3A//mp.weixin.qq.com/s%3F__biz%3DMzAxODE2MjM1MA%3D%3D%26mid%3D2651551344%26idx%3D1%26sn%3Df6493bc7d3305139fae7c9059d52e846%26chksm%3D8025a1b1b75228a711eb8089ebd19802fff51eb1b16ee00e3fa70401ffde664fc3e0173bb8f3%26mpshare%3D1%26scene%3D1%26srcid%3D1014PQeGkxJBchXg2fpatR4W%23rd)

https://mp.weixin.qq.com/s?\_\_biz=MzAxODE2MjM1MA==&mid=2651551344&idx=1&sn=f6493bc7d3305139fae7c9059d52e846&chksm=8025a1b1b75228a711eb8089ebd19802fff51eb1b16ee00e3fa70401ffde664fc3e0173bb8f3&mpshare=1&scene=1&srcid=1014PQeGkxJBchXg2fpatR4W#rd

**17.如何进行错误监控**

**前端错误的分类**

• 即时运行错误（代码错误）

• 资源加载错误

**错误的捕获方式**

即时运行错误的捕获方式：

• try...catch

• window.onerror

资源加载错误：

• object.onerror（如img,script）

• performance.getEntries()

• Error事件捕获

延伸：跨域的js运行错误可以捕获吗，错误提示什么，应该怎么处理？  
可以。

Script error  
1.在script标签增加crossorigin属性  
2.设置js资源响应头Access-Control-Allow-Orgin:\*

**上报错误的基本原理**  
采用Ajax通信方式上报  
利用Image对象上报

**18.DOM事件类**

**DOM事件的级别**

• DOM0，element.onclick = function(){}

• DOM2，element.addEventListener('click', function(){}, false);

**DOM事件模型是什么：**指的是冒泡和捕获

**DOM事件流是什么：**捕获阶段 -> 目标阶段 -> 冒泡阶段

**描述DOM事件捕获的具体流程**  
window --> document --> documentElement(html标签) --> body --> .... --> 目标对象

**Event对象常见应用**

• event.preventDefault()，阻止默认行为

• event.stopPropagation()，阻止事件冒泡

• event.stopImmediatePropagation()，阻止剩余的事件处理函数执行并且防止事件冒泡到DOM树上，这个方法不接受任何参数。

• event.currentTarget，返回绑定事件的元素

• event.target，返回触发事件的元素

**如何自定义事件**

Event，不能传递参数

var eve = new Event('自定义事件名');

ev.addEventListener('自定义事件名',

function(){ console.log('自定义事件') });

ev.dispatchEvent(eve);

CustomEvent，还可以指定参数

**19.本地起了一个http server，为什么只能在同一个WIFI(局域网)上访问？**

你没有公网IP当然就不能被外网访问了。常见的WIFI情况下，一般的ip会是~192.168.0.x·这样的，只是对局域网(同WIFI下)可见，但是外网是访问不了的。（[segmentfault上的答案](http://link.zhihu.com/?target=https%3A//segmentfault.com/q/1010000010863426)）

https://segmentfault.com/q/1010000010863426

**20.回流和重绘**

参考[《如何写出高性能DOM？》](http://link.zhihu.com/?target=http%3A//34585f3f.wiz03.com/share/s/0Qm5Y_0RRQtc2F-3Zy2piy1K0E4QKp0IAQvZ2PEFvB08u3fM)

http://34585f3f.wiz03.com/share/s/0Qm5Y\_0RRQtc2F-3Zy2piy1K0E4QKp0IAQvZ2PEFvB08u3fM

**21.数组去重的方法**

参考：[《JavaScript数组去重》](http://link.zhihu.com/?target=https%3A//mp.weixin.qq.com/s%3F__biz%3DMzAxODE2MjM1MA%3D%3D%26mid%3D2651550928%26idx%3D1%26sn%3D0912e56c7ccbb68cf36562e723e29da0%26scene%3D1%26srcid%3D0612ekFt6xkwpwwFhCDSPKnM%23rd)

https://mp.weixin.qq.com/s?\_\_biz=MzAxODE2MjM1MA==&mid=2651550928&idx=1&sn=0912e56c7ccbb68cf36562e723e29da0&scene=1&srcid=0612ekFt6xkwpwwFhCDSPKnM#rd

**22.深拷贝与浅拷贝**

**是什么？**  
浅拷贝只复制指向某个对象的指针，而不复制对象本身，新旧对象还是共享同一块内存。但深拷贝会另外创造一个一模一样的对象，新对象跟原对象不共享内存，修改新对象不会改到原对象。

**实现浅拷贝**

var obj1 = { a: 10, b: 20, c: 30 };

var obj2 = obj1;

obj2.b = 100;

console.log(obj1); // { a: 10, b: 100, c: 30 } <-- b 被改到了

console.log(obj2); // { a: 10, b: 100, c: 30 }

**实现深拷贝**

var obj1 = { a: 10, b: 20, c: 30 };

var obj2 = { a: obj1.a, b: obj1.b, c: obj1.c };

obj2.b = 100; console.log(obj1); // { a: 10, b: 20, c: 30 } <-- b 沒被改到

console.log(obj2); // { a: 10, b: 100, c: 30 }

**深拷贝实现方式**

• 手动复制方式，如上面的代码，缺点就是

• Object.assign，ES6 的新函数，可以帮助我们达成跟上面一样的功能。var obj1 = { a: 10, b: 20, c: 30 }; var obj2 = Object.assign({}, obj1);  
obj2.b = 100;  
console.log(obj1); // { a: 10, b: 20, c: 30 } <-- 沒被改到  
console.log(obj2); // { a: 10, b: 100, c: 30 }

• 转成 JSON 再转回来  
用JSON.stringify把对象转成字符串，再用JSON.parse把字符串转成新的对象。  
缺点：只有可以转成JSON格式的对象才可以这样用，像function没办法转成JSON。

• jquery，有提供一个$.extend可以用来做 Deep Copy。

1. lodash，也有提供\_.cloneDeep用来做 Deep Copy。

2. 递归实现深拷贝function clone( o ) { var temp = {}; for( var k in o ) { if( typeof o[ k ] == 'object' ){  
temp[ k ] = clone( o[ k ] ); } else {  
temp[ k ] = o[ k ]; } } return temp; }

参考文章：关于 JS 中的浅拷贝和深拷贝,进击JavaScript之（四）玩转递归与数列

**23.如何快速合并雪碧图**

Gulp：gulp-css-spriter  
webpack：optimize-css-assets-webpack-plugin  
Go！Png  
在线工具

**24.代码优化基本方法**

**减少HTTP请求**  
HTML优化：

• 使用语义化标签

• 减少iframe：iframe是SEO的大忌，iframe有好处也有弊端

• 避免重定向

**CSS优化：**

• 布局代码写前面

• 删除空样式

• 不滥用浮动，字体，需要加载的网络字体根据网站需求再添加

• 选择器性能优化

• 避免使用表达式，避免用id写样式

**js优化：**

• 压缩

• 减少重复代码

**图片优化：**

• 使用WebP

• 图片合并，CSS sprite技术

**减少DOM操作**

• 缓存已经访问过的元素

• "离线"更新节点, 再将它们添加到树中

• 避免使用 JavaScript 输出页面布局--应该是 CSS 的事儿

**使用JSON格式来进行数据交换 使用CDN加速 使用HTTP缓存：**添加 Expires 或 Cache-Control 信息头

**使用DNS预解析**Chrome内置了DNS Prefetching技术, Firefox 3.5 也引入了这一特性，由于Chrome和Firefox 3.5本身对DNS预解析做了相应优化设置，所以设置DNS预解析的不良影响之一就是可能会降低Google Chrome浏览器及火狐Firefox 3.5浏览器的用户体验。

**预解析的实现**：

1. 用meta信息来告知浏览器, 当前页面要做DNS预解析:<meta http-equiv="x-dns-prefetch-control" content="on" />

2. 在页面header中使用link标签来强制对DNS预解析: <link rel="dns-prefetch" href="[http://bdimg.share.baidu.com](http://link.zhihu.com/?target=http%3A//bdimg.share.baidu.com)" />

**25.HTTPS的握手过程**

1. 浏览器将自己支持的一套加密规则发送给服务器。

2. 服务器从中选出一组加密算法与HASH算法，并将自己的身份信息以证书的形式发回给浏览器。证书里面包含了网站地址，加密公钥，以及证书的颁发机构等信息。

3. 浏览器获得网站证书之后浏览器要做以下工作：

• 验证证书的合法

• 如果证书受信任，或者是用户接受了不受信的证书，浏览器会生成一串随机数的密码，并用证书中提供的公钥加密。

• 使用约定好的HASH算法计算握手消息，并使用生成的随机数对消息进行加密，最后将之前生成的所有信息发送给服务器

1. 网站接收浏览器发来的数据之后要做以下的操作：

• 使用自己的私钥将信息解密取出密码，使用密码解密浏览器发来的握手消息，并验证HASH是否与浏览器发来的一致。

• 使用密码加密一段握手消息，发送给浏览器。

1. 浏览器解密并计算握手消息的HASH，如果与服务端发来的HASH一致，此时握手过程结束，之后所有的通信数据将由之前浏览器生成的随机密码并利用对称加密算法进行加密。

参考文章：[《HTTPS 工作原理和 TCP 握手机制》](http://link.zhihu.com/?target=http%3A//blog.jobbole.com/105633/)

http://blog.jobbole.com/105633/

**26.BFC相关问题**

**BFC**(Block formatting context)直译为"块级格式化上下文"。它是一个独立的渲染区域，只有 Block-level box 参 与， 它规定了内部的 Block-level Box 如何布局，并且与这个区域外部毫不相干。

**BFC的渲染规则**

• BFC这个元素的垂直方向的边距会发生重叠

• BFC的区域不会与浮动元素的box重叠（清除浮动原理）

• BFC在页面上是一个独立的容器，外面的元素不会影响它里面的元素，反过来它里面的元素也不会影响外面的元素

• 计算BFC的高度的时候，浮动元素也会参与计算

**如何创建BFC？**

• overflow属性不为visible

• float属性不为none

• position属性为absolute或fixed

• display属性为inline-block、table-cell、table-caption、flex、inline-flex

**BFC的使用场景**  
他的很常用的一个应用场景就是解决边距重叠的问题.

**27.响应式图片**

1.JS或者服务端硬编码，resize事件，判断屏幕大小加载不同的图片  
2.img srcset 方法  
3.picture标签 -> source  
4.svg  
5.第三方库polyfill

**28.判断一个变量是否是数组**

var a = [];

// 1.基于instanceof a instanceof Array;

// 2.基于constructor a.constructor === Array;

// 3.基于Object.prototype.isPrototypeOf Array.prototype.isPrototypeOf(a);

// 4.基于getPrototypeOf Object.getPrototypeOf(a) === Array.prototype;

// 5.基于Object.prototype.toString Object.prototype.toString.apply(a) === '[object Array]';

// 6.Array.isArray Array.isArray([]); // true

以上，除了Object.prototype.toString外，其它方法都不能正确判断变量的类型。

**29.UTF-8和Unicode的区别**

UTF-8就是在互联网上使用最广的一种unicode的实现方式。

Unicode的出现是为了统一地区性文字编码方案，为解决unicode如何在网络上传输的问题，于是面向传输的众多 UTF（UCS Transfer Format）标准出现了，顾名思义，UTF-8就是每次8个位传输数据，而UTF-16就是每次16个位。

ASCII --> 地区性编码（GBK） --> Unicode --> UTF-8