**[我给出的一份前端面试题](http://www.cnblogs.com/hzg1981/p/5718157.html)**

因为公司招人的关系，需要我出一份Web前端开发的面试题，现在招聘工作结束了，把题目记录在这里，供大家参考。

第一部分 考察项目经历

**这一部分的套路就是给出一个topic，让面试者给出较为详细的说明，进而问一些更为深入的问题，达到考察的目的。**

1 自我介绍。

2 介绍一个印象最深的项目，使用何种技术，解决何种问题？

3 迄今为止遇到的最大的一个技术挑战？是如何解决的？

**第二部分 定点考察知识点**

0 **计算机基础**

**这一部分主要考察对计算机基础知识的掌握。根据我面试的经验，如果你所在的是一家逼格不是特别高的创业公司的话，那么来面你这个web前端职位的人，大部分会是非计算机专业转过来的，所以这部分他们大概率会掌握的不是很好。。所以没有必要考察太深。**

**1 简述线程与进程的区别？**

进程由线程组成，线程是CPU调度的基本单元。

**2请从插入删除与查询的效率角度，对链表与数组进行比较。**

链表插入删除的效率高，为常量时间复杂度O(1)，但查询效率低，为线性时间复杂度O(n)；

数组查询效率高，为常量时间复杂度O(1)，但插入删除效率低，为线性时间复杂度O(n)

**3 简述栈和队列的区别？**

栈是先入后出，队列是先入先出。

**HTML**

***基础***

**div, span, a, strong, form, em, label, p, h1, input, ol, ul ,select, textarea, img, table**

上述中哪些是inline tags（默认）?  span, a, strong, em, label, input, select, textarea, img

上述中哪些是block tags（默认）?  div, form, p, h1, ol, ul , table

**块元素与内联元素的区别？**

块元素一般都从新行开始，它可以容纳内联元素和其他块元素,常见块元素是段落标签'P". 如果没有css的作用，块元素会顺序以每次另起一行的方式一直往下排。而有了css以后，我们可以改变这种html的默认布局模式，把块元素摆放到你想要 的位置上去。而不是每次都愚蠢的另起一行。

内联元素(inline element)一般都是基于语义级(semantic)的基本元素。内联元素只能容纳文本或者其他内联元素，常见内联元素 “a”。

inline-block的元素特点：将对象呈递为内联对象，但是对象的内容作为块对象呈递。旁边的内联对象会被呈递在同一行内，允许空格。(准确地说，应用此特性的元素呈现为内联对象，周围元素保持在同一行，但可以设置宽度和高度地块元素的属性)

**<b> 和<strong>的区别？**

strong是web标准中xhtml的标签，strong的意思是“强调”；b是html的，b的意思是bold（粗体）。为什么用strong代替b？其实这个问题不妨改问：xhtml和html有什么不同，为什么要用xhtml代替html？

简单地说：web标准主张xhtml不涉及具体的表现形式，**“强调”可以用加粗来强调，也可以用其它方式来强调，比如下划线，比如字体加大，比如红色，等等，可以通过css来改变strong的具体表现，这就是为什么b要改为strong**

strong代表其中内容文字的意义。b代表其中文字的样式是粗体。 在html规范中二者的区别基本上看不出来。在xhtml中由于强调“样式与内容的分离”所以代表样式的b被掏汰了。取而代之的是其它标签。注意，不是strong代替了b。strong代表强调，你可以自定义任何样式来代表强调。只是strong的默认样式与b相同而已。

Html的发展历史是：Html4 xhtml html5。

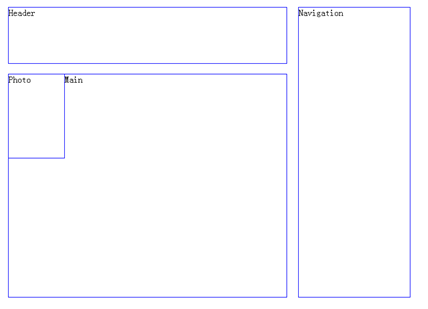
**H5的新特性有哪些？**

用于绘画的 canvas 元素 Websocket, <video> <audio>  <header> <footer> 正则表达式

 HTML5添加了许多新新的语法特征，其中包括**<video>、<audio>和<canvas>元素**，同时集成了[SVG](http://www.bing.com/knows/SVG_%E5%8F%AF%E7%BC%A9%E6%94%BE%E7%9F%A2%E9%87%8F%E5%9B%BE%E5%BD%A2)内容。

***写html代码题***

1 按照下面的样子布局：



2  如何使用原生的create/add/remove/move/copy/search DOM中的元素？（写出代码）

[复制代码](javascript:void(0);)

1 var para=document.createElement("p"); // 创建

2 var element=document.getElementById("div1"); // 查找

3 getElementByTagName

4 getElementByName

5 element.appendChild(para); //追加 add

6 parent.removeChild(child); //删除

[复制代码](javascript:void(0);)

如何选取特定的HTML元素：

1 element = **document.getElementById(id)**;

Document 对象：每个载入浏览器的 HTML 文档都会成为 Document 对象。Document 对象使我们可以从脚本中对 HTML 页面中的所有元素进行访问。

提示：**Document 对象**是 **Window 对象**的一部分，可通过 **window.document** 属性对其进行访问。

HTML DOM 定义了多种查找元素的方法，除了 getElementById() 之外，还有 **getElementsByName()** 和 **getElementsByTagName()和Document.getElementByClassName();。**不过，如果您需要查找文档中的一个特定的元素，最有效的方法是 getElementById()。

如需替换 HTML DOM 中的元素，请使用 **replaceChild()** 方法。

**cloneNode(deepBoolean)**  
复制并返回当前节点的复制节点，复制节点是一个孤立节点，它复制了原节点的属性，在把这个新节点加入到document前，根据需要修改ID属性确保其ID的唯一。

Move：

1 var divs = document.getElementsByTagName("div"); // order: first, second, third

2 divs[0].parentNode.appendChild(divs[0]); // order: second, third, first

3 divs[1].parentNode.insertBefore(divs[0], divs[1]); // order: third, second, third

appendChild() 方法：可向节点的子节点列表的末尾添加新的子节点。语法：appendChild(newchild)

insertBefore() 方法：可在已有的子节点前插入一个新的子节点。语法 ：insertBefore(newchild,refchild)

相同之处：插入子节点

**什么是DOM，DOM使用了什么数据结构？**

HTML DOM 定义了访问和操作***获取、修改、添加或删除 HTML 元素***的标准方法。DOM 将 HTML 文档表达为树结构。

可通过 JavaScript （以及其他编程语言）对 HTML DOM 进行访问。

**CSS**

**1 如何理解CSS中的盒子模型？ Margin和Padding的区别是什么？**

边框border 内边距 外边距。

Margin是外边距，padding是内边距。

<http://www.w3school.com.cn/css/css_boxmodel.asp>

**2 如何验证(validate)一个 HTML文件和CSS文件?**

一般sublime这样的编辑器，都会对html和css文件进行语法验证。

<http://validator.w3.org/>

<http://jigsaw.w3.org/css-validator/>

**3 解释在这个CSS 选择器(selector)中发生了什么:**

[**role**=**navigation**] > **ul a**:**not**([**href**^=**mailto**]) {  
}

**定义了role属性，并且值为navigation的任何元素**，其子元素**列表**下的**除邮箱链接之外的所有链接元素。**

在 CSS 中，选择器是一种模式，用于选择需要添加样式的元素。上述css 选择器主要使用了下面的三种模式：

|  |  |  |
| --- | --- | --- |
| [[*attribute*=*value*]](http://www.w3school.com.cn/cssref/selector_attribute_value.asp) | [target=\_blank] | **选择 target="\_blank" 的所有元素** |

|  |  |  |
| --- | --- | --- |
| [*element*>*element*](http://www.w3school.com.cn/cssref/selector_element_gt.asp) | div>p | 选择父元素为 <div> 元素的所有 <p> 元素。 |

|  |  |  |
| --- | --- | --- |
| [:not(*selector*)](http://www.w3school.com.cn/cssref/selector_not.asp) | :not(p) | 选择非 <p> 元素的每个元素。 |

**什么是LESS? <LESS>(选做)**

**Less 扩充了 CSS 语言**，增加了诸如变量、混合（mixin）、运算、函数等。 Less 既可以运行在服务器端（Node.js 和 Rhino 平台）也可以运行在客户端（浏览器）。Sass、LESS和Stylus是CSS预处理器。他是CSS上的一种抽象层。他们是一种特殊的语法/语言编译成CSS。

**JavaScript**

**基础**

**如何理解JS是单线程这一说法？为什么不会卡住？**  
Javascript除了一个主线程外,还配有一个代码队列,这个队列用以存放定时器、HTTP请求、事件响应的回调。

**什么是浏览器事件模型？请描述js的事件冒泡和捕获(event bubble and capturing), 如何停止冒泡(bubble)?**

浏览器事件模型与js事件模型是同一个概念。

假设你在一个元素中又嵌套了另一个元素

-----------------------------------

| element1                        |

|   -------------------------     |

|   |element2               |     |

|   -------------------------     |

|                                 |

-----------------------------------

：并且两者都有一个onClick事件处理函数(event handler)。如果用户单击元素2，则元素1和元素2的单击事件都会被触发。但是哪一个事件先被触发？哪一个事件处理函数会被首先执行？换句话说，事件的发生顺序到底如何？

**两种模型**

　　不出所料，在那些“不堪回首”（[浏览器](http://blog.jobbole.com/12749/)大战）的日子里，Netscape和微软有两种截然不同的处理方法：

* Netscape主张元素1的事件首先发生，这种事件发生顺序被称为*捕获型*
* 微软则保持元素2具有优先权，这种事件顺序被称为*冒泡型*

W3c明智的在这场争斗中选择了一个择中的方案。任何发生在[w3c事件模型](http://www.w3.org/TR/2000/REC-DOM-Level-2-Events-20001113/)中的事件，首是进入捕获阶段，直到达到目标元素，再进入冒泡阶段

                 | |  / \

-----------------| |--| |-----------------

| element1       | |  | |                |

|   -------------| |--| |-----------     |

|   |element2    \ /  | |          |     |

|   --------------------------------     |

|        W3C event model                 |

------------------------------------------

　　为一个web开发者，你可以选择是在捕获阶段还是冒泡阶段绑定事件处理函数，这是通过**addEventListener()方法实现的，如果这个函数的最后一个参数是true，则在捕获阶段绑定函数，反之false，在冒泡阶段绑定函数**。

(如何选择在哪个阶段绑定事件处理函数？)

**如何取消冒泡？**

在微软的模型中，你必须设置事件的cancelBubble的属性为true：

1 window.event.cancelBubble = true

在w3c模型中你必须调用事件的stopPropagation()方法：

1 e.stopPropagation()

这会阻止所有冒泡向外传播。

**在js中我们为什么需要event delegation？简要描述一下事件委托？**

如今的JavaScript技术界里最火热的一项技术应该是‘事件委托(event delegation)’了。使用事件委托技术能让你避免对特定的每个节点添加事件监听器；相反，事件监听器是被添加到它们的父元素上。事件监听器会分析从子元素冒泡上来的事件，找到是哪个子元素的事件。 也就是说把监听子元素上的事件监听函数放在它的父元素上来。

**Ajax**

**Ajax的全称？原理？优点？**

全称：Asynchronous  js and xml

原理：Ajax的原理简单来说通过XmlHttpRequest对象来向服务器发异步请求，从服务器获得数据，然后用javascript来操作DOM而更新页面。

把服务器端看成一个数据接口(**只负责吐数据**)，它返回的是一个纯文本流，当然，这个文本流可以是XML格式，可以是Html，可以是Javascript代码，也可以只是一个字符串。这时候，XMLHttpRequest向服务器端请求这个页面，服务器端将文本的结果写入页面，这和普通的web开发流程是一样的，不同的是，客户端在异步获取这个结果后，不是直接显示在页面，而是先由javascript来处理，然后再显示在页面。

优点： 局部刷新，避免重新刷新整个页面。

**什么是XMLHTTPRequest对象？**

XMLHttpRequest是ajax的核心机制，是JavaScript的一个内置对象。它是在IE5中首先引入的，是一种支持异步请求的技术。简单的说，也就是javascript可以及时向服务器提出请求和处理响应，而不阻塞用户。达到无刷新的效果。

1 if (window.XMLHttpRequest)

2 {// code for all new browsers

3 xmlhttp=new XMLHttpRequest();

4 }

**Js读代码题**

多项选择题：下列哪一项会返回‘true’?

1. null == undefined
2. Boolean( “true” )
3. Boolean( 0 )
4. isNaN( 10 )
5. isNaN( “10” )
6. NaN === NaN

AB.

A **扩展问题：null == undefined，如果是null === undefined，那是真是假？ Null与undefined的区别是什么？**

undefined 表示一个变量声明但未赋值:

1 var TestVar;

2 alert(TestVar); //shows undefined

3 alert(typeof TestVar); //shows undefined

undefined is a type itself (undefined) while null 是一个对象。

总所周知：null == undefined

但是：null !== undefined

null这是一个对象，但是为空。因为是对象，所以 typeof null  返回 'object' 。

同样，当我们定义一个变量但未赋予其初始值，例如：

　　　　var aValue;

这时，JavaScript在所谓的预编译时会将其初始值设置为对window.undefined属性的引用，

Undefined Null 区别？

typeof 返回的是字符串，有六种可能："number"、"string"、"boolean"、"object"、"**function**"、"undefined"

ECMAScript 有 **5 种原始类型（primitive type）**，即 Number 、 String、Boolean、Undefined、Null。

Number 对象是原始数值的包装对象。

**创建 Number 对象的语法：**

1 var myNum=new Number(value);

2 var myNum=Number(value);

1 <script type=text/javascript>

2 var a = 123;

3 alert(typeof(a));

4 </script>

typeof 一个函数的返回值是function，如果把这个function赋给一个var，则typeof的返回值是object。

**==和===的区别是什么？**

javaScript has two sets of equality operators: === and !==, and their evil twins == and !=. The good ones work the way you would expect.

如果两个操作数有相同的类型，并具有相同的值 then === produces true and !== produces false.

The evil twins do the right thing when the operands are of the same type, but if they are of different types, they attempt to coerce the values. the rules by which they do that are complicated and unmemorable.

==和!=在两个操作数类型相同的时候，会根据值是否相等来判断。 如果类型不相同，那就邪恶了。。

C: 对于不等于1的整数，为false；

DEF：A NaN按照定义永远不等于它自己。. 在任何语言中都是这样。. 在js中NaN是为一个一个不等于它自身的东东。

之所以Nan不等于它本身，是为了实现bool IsNaN(number) 函数:

1 function isNaN(x)

2 {

3 return x != x;

4 }

**3 读代码，并写出代码的执行结果：**

[复制代码](javascript:void(0);)

1 var t = true;

2 window.setTimeout(function (){

3 t = false;

4 },1000);

5 while (t){}

6 alert('end');

[复制代码](javascript:void(0);)

Output： 没有输出，死循环。

**分析：**

确实是死循环导致setTimeout不执行，也导致alert不执行。  
js是单线程是对的，所以会先执行while(t){}再alert，但这个循环体是死循环，所以永远不会执行alert。  
至于说为什么不执行setTimeout，是因为js的工作机制是：**当线程中没有执行任何同步代码的前提下才会执行异步代码，setTimeout是异步代码**，所以setTimeout只能等js空闲才会执行，但死循环是永远不会空闲的，所以setTimeout也永远不会执行。

深入理解js单线程：

JavaScript引擎是单线程运行的,浏览器无论在什么时候都只且只有一个线程在运行JavaScript程序。

一、浏览器的内核是多线程的，它们在内核制控下相互配合以保持同步，一个浏览器至少实现三个常驻线程：javascript引擎线程，GUI渲染线程，浏览器事件触发线程。

1. javascript引擎是基于事件驱动单线程执行的，JS引擎一直等待着任务队列中任务的到来，然后加以处理，浏览器无论什么时候都只有一个JS线程在运行JS程序。

2. GUI渲染线程负责渲染浏览器界面，当界面需要重绘（Repaint）或由于某种操作引发回流(reflow)时,该线程就会执行。但需要注意 GUI渲染线程与JS引擎是互斥的，当JS引擎执行时GUI线程会被挂起，GUI更新会被保存在一个队列中等到JS引擎空闲时立即被执行。

3. 事件触发线程，当一个事件被触发时该线程会把事件添加到待处理队列的队尾，等待JS引擎的处理。这些事件可来自JavaScript引擎当前执行的代码块如setTimeOut、也可来自浏览器内核的其他线程如鼠标点击、AJAX异步请求等，但由于JS的单线程关系所有这些事件都得排队等待JS引擎处理。（当线程中没有执行任何同步代码的前提下才会执行异步代码）

       什么是递归？ (递归是一个通用的概念。不仅限于js)

       什么是闭包？

　　　在js中，如果你在一个function中使用了另外一个function关键字，你就创建了一个闭包。

[复制代码](javascript:void(0);)

1 function say667() {

2 // Local variable that ends up within closure

3 var num = 42;

4 var say = function() { console.log(num); }

5 num++;

6 return say;

7 }

8 var sayNumber = say667();

9 sayNumber(); // logs 43

[复制代码](javascript:void(0);)

 闭包(上述代码中的匿名函数)中，保存的是outter函数的local变量的一个引用。

**Output：43**

**分析： 闭包。**1.函数嵌套函数2.函数内部可以访问到外部的变量或者对象3.避免了垃圾回收。

js在处理对象时，永远使用引用reference。 比如说，你使用foo调用了一个对象，那么返回的闭包将引用那个原始的对象。

[复制代码](javascript:void(0);)

1 function foo(x) {

2 var tmp = 3;

3

4 return function (y) {

5 console.log(x + y + tmp);

6 x.memb = x.memb ? x.memb + 1 : 1;

7 console.log(x.memb);

8 }

9 }

10

11 var age = new Number(2);

12 var bar = foo(age); // bar is now a closure referencing age.

13 bar(10);

[复制代码](javascript:void(0);)

As expected, each call to bar(10) will increment x.memb. What might not be expected, is that xis simply referring to the same object as the age variable! After a couple of calls to bar, age.memb will be 2! This referencing is **the basis for memory leaks with HTML objects.**

http://stackoverflow.com/questions/111102/how-do-javascript-closures-work

[复制代码](javascript:void(0);)

1 <script>

2 function sayAlice() {

3 var say = function() { alert(alice); }

4 var alice = 'Hello Alice';

5 return say;

6 }

7 sayAlice()();

8 </script>

[复制代码](javascript:void(0);)

**Output：**Hello Alice

分析：如果把**sayAlice()()**;改为**sayAlice()会怎么样？**

[复制代码](javascript:void(0);)

1 <script>

2 function foo(x) {

3 var tmp = 3;

4 return function (y) {

5 alert(x + y + (++tmp));

6 }

7 }

8 var bar = foo(2);

9 bar(10);

10 </script>

[复制代码](javascript:void(0);)

**Output：16**

**分析：闭包。**

[复制代码](javascript:void(0);)

1 <script>

2 var a = 10;

3 function test() {

4 alert(a);

5 alert(b);

6 }

7 var b = 6;

8 test();

9 </script>

[复制代码](javascript:void(0);)

**Output：**10 6

分析：跟viriable hoisting 什么区别？这也是一个闭包吗？

 When a JavaScript function is invoked, a new execution context is created. Together with the function arguments and the parent object, this execution context also receives all the variables declared outside of it (in the above example, both 'a' and 'b').

上述代码，如果使用c语言实现，会输出什么结果？？

上述代码，如果使用Java实现，

[复制代码](javascript:void(0);)

1 <script>

2 function foo(x) {

3 var tmp = 3;

4 return function (y) {

5 alert(x + y + tmp);

6 x.memb = x.memb ? x.memb + 1 : 1;

7 alert(x.memb);

8 }

9 }

10 var age = new Number(2);

11 var bar = foo(age); // bar is now a closure referencing age.

12 bar(10);

13 </script>

[复制代码](javascript:void(0);)

 Output： 15 1

1 <script>

2 var foo = {};

3 foo.bar = 'hello';

4 alert(foo.length);

5 </script>

  Output：undefined

[复制代码](javascript:void(0);)

1 <script>

2 function hi(){

3 　　var a;

4 　　alert(a);

5 }

6 hi();

7 </script>

[复制代码](javascript:void(0);)

 Output：undefined

问题： 与不使用hi 函数什么区别？ 也形成了闭包，但是a不是在Outter space定义的。

[复制代码](javascript:void(0);)

1 function addTen(num) {

2 num += 10;

3 return num;

4 }

5

6 var count = 20;

7 var result = addTen(count);

8 alert(count);

9 alert(result);

[复制代码](javascript:void(0);)

 output : 20 30 考察js的参数传递

js与java类似，本质上都是值传递，但是引用是地址，值传递时拷贝了引用值的副本，但两个引用指向的还是同一个内存地址。

[复制代码](javascript:void(0);)

1 function setName(obj) {

2 obj.name = "Nicholas";

3 obj = new Object();

4 obj.name = "Greg";

5 }

6 var person = new Object();

7 setName(person);

8 alert(person.name)

[复制代码](javascript:void(0);)

 output：Nicholas

分析： **主要考察引用传递。**

JavaScript 是面向对象的语言，但 JavaScript 不使用类。

在 JavaScript 中，不会创建类，也不会通过类来创建对象（就像在其他面向对象的语言中那样）。

1 var city = "Rome"+function() {

2 console.log(city);

3 var city = "Paris";

4 }();

**output：** undefined

**分析：**为什么？

**(variable hoisting)**在Js中，变量被移动到脚本的顶部，然后执行。但是这种说法并不准确。在下面的代码中:

1 console.log(a);

2 var a = 'Hello World!';

会输出 undefined, 并不是 'Hello World', 所以上述代码等效于下面的代码：

1 var a;

2 console.log(a);

3 a = 'Hello World!';

而不是下面的代码：

1 var a = 'Hello World!';

2 console.log(a);

实际上，js并没有移动任何代码，你需要理解js的执行上下文(context):  context分为两个阶段：创建阶段和执行阶段。 在创建阶段，为这些变量和函数创建内存空间，人们往往把这个阶段与variable hoisting混淆起来。 对于变量的赋值，是在执行上下文的执行阶段进行的。当你对变量a赋值“Hello World”的时候，js引擎只有在执行阶段才知道a的值，在创建阶段，js只是把一个undefined的占位符放在那里，所以所有的变量都是被初始化为undefined的。 所以建议永远把变量声明和函数放在你的代码的顶部。

[复制代码](javascript:void(0);)

1 var name = "The Window";

2 var object = {

3 name: "My Object",

4 getNameFunc: function () {

5 return function () {

6 return this.name;

7 };

8 }

9 };

10 alert(object.getNameFunc()());

[复制代码](javascript:void(0);)

**output:** The Window

**分析：** 闭包？

[复制代码](javascript:void(0);)

1 var name = "The Window";

2 var object = {

3 name: "My Object",

4 getNameFunc: function () {

5 var that = this;

6 return function () {

7 return that.name;

8 };

9 }

10 };

11 alert(object.getNameFunc()());

[复制代码](javascript:void(0);)

**output:**My Object

**分析：**this指的是调用函数的那个对象

当点击button#2的时候，console中会打印什么？

[复制代码](javascript:void(0);)

1 var nodes = document.getElementsByTagName('button');

2 // assume there is totally 5 nodes

3 for (var i = 0; i < nodes.length; i++) {

4 nodes[i].addEventListener('click', function() {

5 console.log('You clicked element #' + i);

6 });

7 }

[复制代码](javascript:void(0);)

**Output:** you clicked element #5

**分析：** 闭包？

1 element.addEventListener('click', function() { /\* do stuff here\*/ }, false);

最后一个参数决定这个linstener如何响应bubbling事件。

<input id="File1" type="file" name="up1"/>  
<input id="File2" type="file" name="up1" />  
<input id="File3" type="file" name="up2" />

**选做：**

[复制代码](javascript:void(0);)

1 function Mammal(name) {

2 this.name = name;

3 this.offspring = [];

4 }

5 Mammal.prototype.haveABaby = function () {

6 var newBaby = new Mammal("Baby " + this.name);

7 this.offspring.push(newBaby);

8 return newBaby;

9 }

10

11 Mammal.prototype.toString = function () {

12 return '[Mammal "' + this.name + '"]';

13 }; // 到目前为止，这 是一个Mammal对象的实现。

14

15

16

17 // 将Employee的原型指向Person的一个实例

18

19 // 因为Person的实例可以调用Person原型中的方法, 所以Employee的实例也可以调用Person原型中的所有属性。

20 Cat.prototype = new Mammal();

21

22 //修正constructor

23 Cat.prototype.constructor = Cat;

24 function Cat(name) {

25 this.name = name;

26 }

27

28 Cat.prototype.toString = function () {

29 return '[Cat "' + this.name + '"]';

30 } // 到目前为止，这是Mammal的一个子类Cat。

31

32

33 var someAnimal = new Mammal('Mr. Biggles');

34 var myPet = new Cat('Felix');

35 alert(someAnimal);

36 alert(myPet);

37

38 myPet.haveABaby();

39 alert(myPet.offspring.length);

40 alert(myPet.offspring[0]);

[复制代码](javascript:void(0);)

**Output:**

[Mammal “Mr. Biggles”]

[Cat, “Felix”]

1

**[Mammal “Baby Felix”]**

**进一步问题：**什么是prototype？什么是原型链?

[复制代码](javascript:void(0);)

1 console.log("New york");

2 setTimeout(function () {

3 console.log("Vienne");

4 }, 1000);

5 setTimeout(function () {

6 console.log("London");

7 });

8 console.log("Ottawa");

[复制代码](javascript:void(0);)

 Output:  New York， Ottawa，Landon，Vienne。

**分析：setTimeout异步函数。**

**js写代码题**

1 尝试实现下面的需求：

(1)      测试array numbers中是不是每一个元素都比2大

E.G.

1 var numbers = [1, 2, 3, 4, 5, 4, 3, 2, 1];

2 var everyResult = numbers.every(function (item, index, array) {

3 return (item > 2);

4 });

5 alert(everyResult); //false

**分析：对数组中的每个元素都执行一次指定的函数（callback），直到此函数返回 false，如果发现这个元素，every 将返回 false，如果回调函数对每个元素执行后都返回 true ，every 将返回 true。**

(2)      得到新的array，其中的元素是numbers中所有比2大的元素。

**var *numbers***= [1, 2, 3, 4, 5, 4, 3, 2, 1];

[复制代码](javascript:void(0);)

1 <script>

2 var numbers = [1, 2, 3, 4, 5, 4, 3, 2, 1];

3 var numbersLargerThan2 = [];

4 for(i in numbers){ // i has some problems here.

5 if(numbers[i]>2) {

6 numbersLargerThan2.push(numbers[i]);

7 }

8 }

9 for(i in numbersLargerThan2) {

10 alert(numbersLargerThan2[i]);

11 }

12 // push pop

13 </script>

[复制代码](javascript:void(0);)

 (3)      一个新的array,其中的每一个元素都比numbers中的元素大1

**var *numbers***= [1, 2, 3, 4, 5, 4, 3, 2, 1];

[复制代码](javascript:void(0);)

1 //一个新的array,其中的每一个元素都比numbers中的元素大1

2 <script>

3 var numbers = [1, 2, 3, 4, 5, 4, 3, 2, 1];

4 var numbersAfterAdd1 = [];

5 for(i in numbers) {

6 numbersAfterAdd1.push(numbers[i]+1);

7 }

8

9 for(i in numbersAfterAdd1) {

10 alert(numbersAfterAdd1[i]);

11 }

12 </script>

[复制代码](javascript:void(0);)

**2列出js中创建对象的方法，并写出一个js中的继承关系的实现。**

**应该知道4种常见创建对象的方法：**

1 用{}  ， **对象字面量。**

 先创建再添加：

[复制代码](javascript:void(0);)

1 //创建一个简单对象字面量

2 var person = {};

3 // 加入属性和方法

4 person.name = 'ifcode';

5 person.setName = function(theName) {

6 person.name = theName;

7 }

[复制代码](javascript:void(0);)

2  JS good parts中推荐这种写法:

[复制代码](javascript:void(0);)

1 var person = {

2 **name: 'ifcode',**

3 setName: function(theName) {

4 this.name = theName;

5 }

6 }

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 var clock={

2 **hour:12,**

3 minute:10,

4 second:10,

5 showTime:function(){

6 alert(this.hour+":"+this.minute+":"+this.second);

7 }

8 }

9 clock.showTime();//调用

10 //var m = new myObj(); //不支持

[复制代码](javascript:void(0);)

**上述两种方式只存在于一个实例的对象，也就是单例模式。**

**下面是可以创建多个实例的方式：**

3 构造函数一般都符合factory pattern，根据默认的规则，**构造函数应当首字母大写**：

[复制代码](javascript:void(0);)

1 Person = function(defaultName) {

2 this.name = defaultName;

3 this.setName = function(theName) {

4 this.name = theName;

5 }

6 }

7 person = new Person('ifcode'); // new是调用构造函数

[复制代码](javascript:void(0);)

利用构造函数就可以方便地创建多个对象实例了。

4 利用 prototype的构造函数：

[复制代码](javascript:void(0);)

1 Person = function(defaultName) {

2 this.name = defaultName;

3 }

4

5

6 Person.prototype.setName = function(theName) {

7 this.name = theName;

8 }

[复制代码](javascript:void(0);)

**所有创建在prototype上得属性和方法，都将被所有对象实例分享。**

**一般在创建单例的时候，用2 ，在创建多个对象的时候，使用4.；**

----------------------------------

**2.创建Object实例**

[复制代码](javascript:void(0);)

1 var clock = new Object();

2 clock.hour=12;

3 clock.minute=10;

4 clock.showHour=function(){alert(clock.hour);};

5

6 clock.showHour();//调用

[复制代码](javascript:void(0);)

由此可见 属性是可以动态添加，修改的

**3用 function 关键字模拟 class**

**上述两种。。。**

<http://www.cnblogs.com/lucas/archive/2009/03/17/1411656.html>

<http://www.jianshu.com/p/f9a1203e33d0>

**js中继承关系的实现：**

JavaScript中要实现继承，其实就是实现三层含义：  
1、子类的实例可以共享父类的方法；  
2、子类可以覆盖父类的方法或者扩展新的方法；  
3、子类和父类都是子类实例的“类型”。

什么是prototype？ 原型

什么是原型链?

JavaScript 是**基于原型的语言**。当我们调用一个对象的属性时，如果对象没有该属性，JavaScript 解释器就会从对象的原型对象上去找该属性，如果原型上也没有该属性，那就去找原型的原型。这种属性查找的方式被称为原型链（prototype chain）。

**ES6 并没有改变 JavaScript 基于原型的本质，只是在此之上提供了一些语法糖。class 就是其中之一。其他的还有 extends，super 和 static 。它们大多数都可以转换成等价的 ES5 语法。**

       javascript是面向对象的，怎么体现javascript的继承关系？使用prototype来实现。

       ES6中启用了class关键字，如何使用该关键字实现继承关系？

[复制代码](javascript:void(0);)

1 // ES6 中启用了关键字 class

2 **class Person** {

3 **constructor**(name, gender, age) {

4 this.name = name;

5 this.gender = gender;

6 this.age = age;

7 }

8 tellAge() {

9 console.log(this.age);

10 }

11 }

12

13 var puya = new Person('PuYart', 'male', '21');

14 puya.tellAge();

[复制代码](javascript:void(0);)

CLASS继承

[复制代码](javascript:void(0);)

1 /\*

2 \*class：ES6及ES6以上才有该方法。

3 \*class的出现将原型继承简化了很多，class的目的就是让定义类更简单。

4 \*extends来继承对象，中间的原型之类的就可以免去，就可以继承扩展class

5 \*/

6 //用class创建对象

7 **class Leader**{

8 **constructor**(name){//constructor构造函数

9 this.name=name;

10 }

11 **hello(){//定义在原型上的函数**

12 alert('Hello, '+this.name+'!');

13 }

14 }

15 var liyi= new Leader('liyi');

16 liyi.name;//输出'liyi'

17 liyi.hello();//输出'Hello, liyi!'

18 //用extends继承扩展

19 **class extendLeader extends Leader{**

20 **constructor(name,grade,skill)**{//若是不扩展Leader的构造函数，就可以将constructor这一步省去

21 super(name);

22 this.grade=grade;

23 　　　　 this.skill=skill;

24 　　}

25 　　run(){

26 　　　　console.log(this.name+'职位：'+this.grade+' 技能：'+this.skill);

27 　　}

28 }

29 var liyi=new extendLeader('liyi','研发经理','精通各种技术');

30 liyi.name;//'liyi'

31 liyi.grade;//'研发经理'

32 liyi.skill;//'精通各种技术'

33 liyi.hello;//'Hello, liyi!'

34 liyi.run();//'liyi职位：研发经理 技能：精通各种技术'

[复制代码](javascript:void(0);)

**Two awesome javascript inheritance libraries that come to mind are**[**klass**](https://github.com/ded/klass)**and**[**selfish.js**](https://github.com/Gozala/selfish)**(I've used both, they're amazing.)**

<http://www.cnblogs.com/sanshi/archive/2009/07/08/1519036.html>

**知不知道ES6 中的class和extends关键字，如何使用上述关键字实现一个继承关系。**

[复制代码](javascript:void(0);)

1 function Person(name) {

2 this.name = name;

3 }

4

5 Person.prototype = {

6 getName: function() {

7 return this.name;

8 }

9 }

10

11 function Employee(name, employeeID) {

12 this.name = name;

13 this.employeeID = employeeID;

14 }

15

16 // 这是不好的。

17 Employee.prototype = new Person();

18 Employee.prototype.getEmployeeID = function() {

19 return this.employeeID;

20 };

21

22 var zhang = new Employee("ZhangSan", "1234");

23 console.log(zhang.getName()); // "ZhangSan"

[复制代码](javascript:void(0);)

[复制代码](javascript:void(0);)

1 function Employee(name, employeeID) {

2 this.name = name;

3 this.employeeID = employeeID;

4 }

5 // 创建Employee时实例化Person是不合适的

6 Employee.prototype = new Person();

7 Employee.prototype.constructor = Employee;

8 Employee.prototype.getEmployeeID = function() {\

9 return this.employeeID;

10 };

11

12 var zhang = new Employee("ZhangSan", "1234");

13 console.log(zhang.constructor === Employee); // true

14 console.log(zhang.constructor === Object); // false

[复制代码](javascript:void(0);)

提供一个原型方法（比如init）来初始化数据。

[复制代码](javascript:void(0);)

1 // 空的构造函数

2 function Person() {

3 }

4

5 Person.prototype = {

6 init: function(name) {

7 this.name = name;

8 },

9

10 getName: function() {

11 return this.name;

12 }

13 }

14

15 // 空的构造函数

16 function Employee() {

17 }

18

19 // 创建类的阶段不会初始化父类的数据，因为Person是一个空的构造函数

20 Employee.prototype = new Person();

21 Employee.prototype.constructor = Employee;

22 Employee.prototype.init = function(name, employeeID) {

23 this.name = name;

24 this.employeeID = employeeID;

25 };

26

27 Employee.prototype.getEmployeeID = function() {

28 return this.employeeID;

29 };

[复制代码](javascript:void(0);)

这种方式下，必须在实例化一个对象后手工调用init函数，如下：

1 var zhang = new Employee();

2 zhang.init("ZhangSan", "1234");

3 console.log(zhang.getName()); // "ZhangSan"

必须达到两个效果，构造类时不要调用init函数和实例化对象时自动调用init函数。看来我们需要在调用空的构造函数时有一个状态标示。

可以用 call 来实现继承:

[复制代码](javascript:void(0);)

1 function Class1() {

2 this.showTxt = function(txt) {

3 alert(txt);

4 }

5 }

6 function Class2() {

7 Class1.call(this);

8 }

9 var c2 = new Class2();

10 c2.showTxt("cc");

[复制代码](javascript:void(0);)

这样 Class2 就继承Class1了，Class1.call(this) 的 意思就是使用 Class1 对象代替this对象，那么 Class2 中不就有Class1 的所有属性和方法了吗，c2 对象就能够直接调用Class1 的方法以及属性了，执行结果就是：alert（“cc”）;

**Framework**

1 如何使用JQuery选取一个ID为myDivId的元素？如何选取一组class为myCssClass的元素？写出代码。

1 $( "#myDivId" );

2 $( ".myDivId" );

**3 简述你使用过的前端框架，并介绍其优缺点？**

JQuery/Query Mobile

Dojo/[Dojo Mobile](http://dojotoolkit.org/features/mobile)

zepto.js是一个专为mobile WebKit浏览器(如：Safari和Chrome), 语法借鉴并兼容JQuery。

Node.js

angularJS

bootstrap 是一套html css和js的框架。

React.js（React）是 Facebook 推出的一个用来构建用户界面的 JavaScript 库.

Slider 滑动条控件。（不算一个框架）

(选做)**如果熟悉angularJS, 请选择(1), 如果熟悉React, 请选择(2)**:

**(1)**描述angular中的数据绑定(data-binding)，描述angular依赖注入(dependency injection)

在angular中，model和view组件之间的Data Binding是可以自动同步数据的。angular实现Data Binding的方法可以让你确信在你的应用中model是single-source-of-truth，view仅仅是model的投影。当model改变时，view跟着改变，反之亦然。

angular的模板系统则不同，template是被浏览器去编译的，编译这步会产生一个live的view。对view进行的任何更改会立即反映到model中，对model进行的更改也会立即反映到view中。model是应用程序的single-source-of-truth，极大地简化了开发人员的编程模型，你仅仅把view当成model的瞬间投影即可。

Angular JS框架与Spring类似，都实现了DI。

**(2)      virtual DOM的优势是什么？解释FLUX中的数据流**

来人们使用了 MVC、MVP 的架构模式，希望能从代码组织方式来降低维护这种复杂应用程序的难度。但是 MVC 架构没办法减少你所维护的状态，也没有降低状态更新你需要对页面的更新操作（前端来说就是DOM操作），你需要操作的DOM还是需要操作，只是换了个地方。

相对于 DOM 对象，原生的 JavaScript 对象处理起来更快，而且更简单。DOM 树上的结构、属性信息我们都可以很容易地用 JavaScript 对象表示出来。

既然状态改变了要操作相应的DOM元素，为什么不做一个东西可以让视图和状态进行绑定，状态变更了视图自动变更，就不用手动更新页面了。这就是后来人们想出了 MVVM 模式。

React 标榜自己是 MVC 里面 V 的部分，那么 Flux 就相当于添加 M 和 C 的部分。

Flux 是 Facebook 使用的一套前端应用的架构模式。

一个 Flux 应用主要包含四个部分：

the dispatcher

处理动作分发，维护 Store 之间的依赖关系

the stores

数据和逻辑部分

the views

React 组件，这一层可以看作 controller-views，作为视图同时响应用户交互

the actions

提供给 dispatcher 传递数据给 store

Flux 的核心“单向数据流“怎么运作的：

Action -> Dispatcher -> Store -> View

更多时候 View 会通过用户交互触发 Action，所以一个简单完整的数据流类似这样：

(选做)**什么是npm, 你可以如何使用它？**

npm是Node.js 的包管理工具，用来安装各种 Node.js 的扩展。

**test/build等杂项**

**请描述Git中merge和rebase的区别(选做)**

git merge和git rebase从最终效果来看没有任何区别，都是将不同分支的代码融合在一起，但是生成的代码树就稍微有些不同。rebase操作不会生成新的节点，是将两个分支融合成一个线性的提交。而merge操作生成的代码树会显得比较乱。

**什么是mock? (选做)**

mock测试就是在测试过程中，对于某些不容易构造或者不容易获取的对象，用一个虚拟的对象来创建以便测试的测试方法。

**延伸问题：什么是冒烟测试（smoke test）?**

冒烟测试源自硬件行业，对一个硬件或者硬件组件改动后，直接给设备加电，看看设备会不会冒烟，没冒烟，就表示待测组件是通过了测试。    在软件开发过程中，一直有高内聚，低耦合这样的说法，各个功能模块之间的耦合还是存在的，因此一个功能的改动，还是会影响到其他功能模块。    因此在开发人员修复了先前测试中发现的bug后，想知道这个bug的修复是否会影响到其他功能模块，需要做的就是冒烟测试。

**列出你知道的前端的build/test automation/unit test/的工具/框架，以及你知道的其他工具，并给出简要描述。(选做)**

静态代码检查工具JSLint。 JSHint跟JSLint非常像，都是Javascript代码验证工具

前端的测试框架很多，像QUnit、jasmine、mocha、jest、intern等

QUnit诞生之初是为了jquery的单元测试

Bower 包管理 包依赖

Yeoman 快速构建

**Build工具：**

**Grunt**能做什么：  
1、合并，压缩JS，CSS。  
2、单元测试(Qunit)。  
3、验证(JSHint)。  
4、初始化一个Project。

Build工具：

Gulp + Webpack

**服务器端相关**

**HTTP**

**这一部分主要考察对网络协议，主要是HTTP协议的理解。**

**HTTP请求共有几种类型？给出你常用的类型，并结合你给出的类型描述HTTP请求的幂等性。**

8 种类型： GET POST PUT DELETE OPTION HEAD等等。

**列出常见的HTTP请求的返回状态(Status Code)，并简述其含义？**

**如何解决HTTP的无状态性所带来的问题？(选做)**

**如何理解HTTP是无连接的？ (选做)**

TCP是有连接的，而HTTP是无连接的。无连接：服务器处理完客户的请求，并收到客户的应答后，即断开连接。

什么是长连接？

HTTP响应码 1 2 3 4 5 的各种含义

什么是RESTful风格？

常见HTTP报头的含义：

Via

Keep-alive

是否熟悉抓包工具，平时都使用什么抓包工具？ 比如Fiddler？

什么是Proxy？ 什么是反向代理？

**杂项**

简述什么是MVC ？(选做)

什么是websocket？什么是REST？什么样的需求(使用场景)会需要websocket和REST？(选做)

**其他问题：**

期望薪资；

如何看待加班，婚育情况，住址，何时到岗；

是否有问题需要提问？

**备选题**

**计算机基础**

快排的主要思想？时间复杂度？ 如何避免最差的时间复杂度？(选做)

6二叉树有哪些遍历算法？ 如何从递归算法构造非递归算法？使用什么辅助数据结构？(选做)

4 什么是平衡二叉树？常见的平衡二叉树有哪些？(选做)

**HTML**

如何选取特定的HTML元素？

**Js**

5 javascript的方法可以分为哪几类？

**类方法(静态方法)，对象方法(实例方法)，原型方法**。

javascript的方法可以分为三类：

a 类方法

b 对象方法

c 原型方法

例子：

[复制代码](javascript:void(0);)

1 function People(name)

2 {

3 this.name=name;

4 //对象方法

5 this.Introduce=function(){

6 alert("My name is "+this.name);

7 }

8 }

9 //类方法

10 People.Run=function(){

11 alert("I can run");

12 }

13 //原型方法

14 People.prototype.IntroduceChinese=function(){

15 alert("我的名字是"+this.name);

16 }

17

18

19

20 //测试

21

22 var p1=new People("Windking");

23

24 p1.Introduce();

25

26 People.Run();

27

28 p1.IntroduceChinese();

[复制代码](javascript:void(0);)

 1 什么是document对象？如何通过Window对象访问document对象？

9 是否熟悉抓包工具，平时都使用什么抓包工具？

8 列出常见的HTTP报头的字段，并简述其含义？

 Keep-alive什么是长连接？

10 什么是Proxy？ 什么是反向代理？

[复制代码](javascript:void(0);)

1 <script>

2 var foo = [];

3 foo.push(1);

4 foo.push(2);

5 alert(foo.length);

6 </script>

[复制代码](javascript:void(0);)

Output：2

               什么是平衡二叉树？常见的平衡二叉树有哪些？()AVL 红黑。

**算法：**

快排的思想，时间复杂度？ 如何避免最差的时间复杂度？

        二叉树的遍历算法？ 如何从递归算法构造非递归算法，使用什么辅助数据结构？

5.  js中的3种弹出式消息提醒（警告窗口，确认窗口，信息输入窗口）的命令是什么？  
alert  
confirm  
prompt