### 蚂蚁金服一面

面试时间：2019年3月19日下午2点

时长：52分钟。

面试官人超好，给提了很多建议，怪我自己太菜了，准备不够充分。

### 自我介绍

### 基础问题

### CSS

### 1. 讲一下float,清除浮动

float 属性定义元素在哪个方向浮动。以往这个属性总应用于图像，使文本围绕在图像周围，不过在 CSS 中，任何元素都可以浮动。浮动元素会生成一个块级框，而不论它本身是何种元素。 ### 2. 移动端适配问题，讲讲rem + px像素（Pixel）。相对长度单位。像素px是相对于显示器屏幕分辨率而言的。 + em是相对长度单位。相对于当前对象内文本的字体尺寸。 + 使用rem为元素设定字体大小时，仍然是相对大小，但相对的只是**HTML根元素**。这个单位可谓集相对大小和绝对大小的优点于一身，通过它既可以做到只修改根元素就成比例地调整所有字体大小，又可以避免字体大小逐层复合的连锁反应。 ### 3. 讲讲盒模型 + 盒模型分为标准模型和IE模型两种 + 盒模型的宽高只是内容（content）的宽高，而在IE模型中盒模型的宽高是内容(content)+填充(padding)+边框(border)的总宽高。

/\* 标准模型 \*/  
box-sizing:content-box;  
 /\*IE模型\*/  
box-sizing:border-box;

### 4. 边距重叠解决方案BFC (块级格式化上下文)

* 一个HTML元素要创建BFC，则满足下列的任意一个或多个条件即可：

1. float的值不是none。
2. position的值不是static或者relative。
3. display的值是inline-block、table-cell、flex、table-caption或者inline-flex
4. overflow的值不是visible

* 用处：

1. 利用BFC避免外边距折叠
2. BFC包含浮动。浮动元素是会脱离文档流，如果一个没有高度或者height是auto的容器的子元素是浮动元素，则该容器的高度是不会被撑开的。我们通常会利用伪元素(:after或者:before)来解决这个问题。**BFC能包含浮动，也能解决容器高度不会被撑开的问题**。
3. 使用BFC避免文字环绕。
4. 在多列布局中使用BFC。

参考： [什么是BFC](https://www.cnblogs.com/libin-1/p/7098468.html)

### 5. CSS垂直居中

参考： [纯CSS实现垂直居中的几种方法](https://www.cnblogs.com/hutuzhu/p/4450850.html)

### 6.宽高比4:3自适应

* **垂直方向的padding**: 在css中，padding-top或padding-bottom的百分比值是根据容器的width来计算的。
* .wrap{  
   position: relative;  
   height: 0; //容器的height设置为0  
   width: 100%;  
   padding-top: 75%; //100%\*3/4  
  }  
  .wrap > \*{  
   position: absolute;//容器的内容的所有元素absolute,然子元素内容都将被padding挤出容器  
   left: 0;  
   top: 0;  
   width: 100%;  
   height: 100%;  
  }
* **padding & calc()**: 跟第一种方法原理相同
* padding-top: calc(100%\*9/16);
* **padding & 伪元素**
* **视窗单位**： 浏览器100vw表示浏览器的视窗宽度
* width:100vw;  
  height:calc(100vw\*3/4)

参考： [CSS实现长宽比的几种方案](https://www.w3cplus.com/css/aspect-ratio.html)

### 7.让一个图片无限旋转

<img class="circle" src="001.jpg" width="400" height="400"/>  
  
 //infinite 表示动画无限次播放 linear表示动画从头到尾的速度是相同的  
 .circle{  
 animation: myRotation 5s linear infinite;  
 }  
@keyframes myRotation {  
 from {transform: rotate(0deg);}  
 to {transform: rotate(360deg);}  
}

### Javascript

### 1. 讲一讲this指针

### 2. 讲一讲原型链

### 3. setTimeout的this，setTimeout()时间设置为0有什么作用

* 超时调用的代码都是在全局作用域中执行的，因此函数中this的值在非严格模式下指向 **window对象**，在严格模式下是 **undefined**。
* 因此，用处就在于我们可以 **改变任务的执行顺序**,因为浏览器会在执行完当前任务队列中的任务，再执行setTimeout队列中积累的的任务。

### 4.讲一讲js里面的异步操作有哪些？

1. 回调函数。 ajax典型的异步操作，利用XMLHttpRequest，回调函数获取服务器的数据传给前端。
2. 事件监听。 当监听事件发生时，先执行回调函数，再对监听事件进行改写
3. 观察者模式，也叫订阅发布模式。 多个观察者可以订阅同一个主题，主题对象改变时，主题对象就会通知这个观察者。
4. promise.
5. es7语法糖async/await
6. co库的generator函数

参考： [JS进阶 | 分析JS中的异步操作](https://www.cnblogs.com/dirkhe/p/7384743.html)

### 5.讲一下let、var、const的区别，谈谈如何冻结变量

* **var** 没有块级作用域，支持变量提升。
* **let** 有块级作用域，不支持变量提升。不允许重复声明，暂存性死区。
* **const** 有块级作用域，不支持变量提升，不允许重复声明，暂存性死区。声明一个变量一旦声明就不能改变，改变报错。const保证的变量的**内存地址**不得改动。如果想要将对象冻结的话，使用Object.freeze()方法
* const foo=Object.freeze({});  
  foo.prop=123;  
  console.log(foo.prop);//混杂模式undefined,不起作用

### 6.数组去重

[...new Set(Array)]

### 7.正则表达式（）

### 8.函数节流和函数去抖有什么区别，怎么实现

* **函数去抖 debounce**： 当调用函数n秒后，才会执行该动作，若在这n秒内又调用该函数则将取消前一次并重新计算执行时间。
* var debounce = function(delay, cb) {  
  var timer;  
  return function() {  
   if (timer) clearTimeout(timer);  
   timer = setTimeout(function() {  
   cb();  
   }, delay);  
  }  
  }
* **函数节流 throttle**： 函数节流的基本思想是函数预先设定一个执行周期，当调用动作的时刻大于等于执行周期则执行该动作，然后进入下一个新周期
* var throttle = function(delay, cb) {  
  var startTime = Date.now();  
  return function() {  
   var currTime = Date.now();  
   if (currTime - startTime > delay) {  
   cb();  
   startTime = currTime;  
   }  
  }  
  }

### 9.js事件循环机制

* 程序开始执行之后，主程序则开始执行**同步任务**，碰到**异步任务**就把它放到任务队列中,等到同步任务全部执行完毕之后，js引擎便去查看任务队列有没有可以执行的异步任务，将异步任务转为同步任务，开始执行，执行完同步任务之后继续查看任务队列，这个过程是一直 **循环**的，因此这个过程就是所谓的**事件循环**，其中任务队列也被称为事件队列。通过一个任务队列，单线程的js实现了异步任务的执行，给人感觉js好像是多线程的。

### 10.事件捕获和事件冒泡

* IE的事件流叫做**事件冒泡**。即事件开始时由最具体的元素接收，然后逐级向上传播到较为不具体的节点window对象。
* **事件捕获**的思想是不太具体的节点应该更早的接收到事件，而在最具体的节点应该最后接收到事件。
* “DOM2级事件”规定事件流包括三个阶段，事件捕获阶段、处于目标阶段和事件冒泡阶段。首先发生的事件捕获，为截获事件提供了机会。然后是实际的目标接收了事件。最后一个阶段是冒泡阶段，可以在这个阶段对事件做出响应。
* 事件流是处于**目标阶段**，事件处理程序被调用的顺序是注册的顺序
* 事件有两个属性**e.target**和**e.currentTarget**，target是真正发生事件的DOM元素，而currentTarget是当前事件发生在哪个DOM元素上。目标阶段也就是 target == currentTarget的时候。
* 在支持addEventListener()的浏览器中，可以调用事件对象的 **stopPropagation()** 方法以阻止事件的继续传播。如果在同一对象上定义了其他处理程序，剩下的处理程序将依旧被调用，但调用 **stopPropagation()** 之后任何其他对象上的事件处理程序将不会被调用。不仅可以阻止事件在冒泡阶段的传播，还能阻止事件在捕获阶段的传播。IE9之前的IE不支持stopPropagation()方法，而是设置事件对象cancelBubble属性为true来实现阻止事件进一步传播。
* **e.preventDefault()** 可以阻止事件的默认行为发生，默认行为是指：点击a标签就转跳到其他页面、拖拽一个图片到浏览器会自动打开、点击表单的提交按钮会提交表单。

参考：[事件冒泡、事件捕获和事件委托](https://www.cnblogs.com/Chen-XiaoJun/p/6210987.html)

### 10.事件代理(事件委托)及其好处，如何知道是哪个节点的事件

* 事件委托就是利用 **事件冒泡** ，只指定一个事件处理程序，就可以管理某一类型的所有事件。
* Event对象提供了一个属性叫target，可以返回事件的目标节点，简称事件源。
* window.onload = function(){  
  　　var oUl = document.getElementById("ul1");  
  　　oUl.onclick = function(ev){  
  　　　　var ev = ev || window.event;  
  　　　　var target = ev.target || ev.srcElement;  
  　　　　if(target.nodeName.toLowerCase() == 'li'){  
  　 　　　　　　 alert(123);  
  　　　　　　　 alert(target.innerHTML);  
  　　　　}  
  　　}  
  }

### 11.排序方法，有哪些讲讲原理，js的sort（）用到哪些排序算法？

* V8 ：数组长度小于等于 22 的用插入排序，其它的用快速排序

参考： [十大经典的排序算法](https://www.cnblogs.com/onepixel/articles/7674659.html)

### 12.懂链表吗，懂二叉树吗，懂堆吗？

### 13.vue如何实现双向绑定

### 14.如何拦截变量属性

* 使用proxy。**new Proxy()** 表示生成一个 **Proxy** 实例，**target** 参数表示所要拦截的目标对象，**handler** 参数也是一个对象，用来定制拦截行为。
* var proxy = new Proxy({}, {  
   get: function(target, property) {  
  return 35;  
   }  
  });  
  let obj = Object.create(proxy);  
  obj.time // 35

### 15.箭头函数和普通函数的区别是什么？

* 普通函数this：

1. this总是代表它的直接调用者。
2. 在默认情况下，没找到直接调用者，this指的是window。
3. 在严格模式下，没有直接调用者的函数中的this是undefined。
4. 使用call,apply,bind绑定，this指的是绑定的对象。

* 箭头函数this：

1. 在使用=>定义函数的时候，this的指向是 **定义时所在的对象**，而不是使用时所在的对象；
2. **不能够用作构造函数**，这就是说，不能够使用new命令，否则就会抛出一个错误；
3. 不能够使用 **arguments** 对象；
4. 不能使用 **yield** 命令；

### 16.null和undefined的区别

* 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。
* 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。

### 17.讲讲es6的promise函数

### 18.讲讲es6的generator函数

### 19.函数柯里化理解

* 只传递给函数一部分参数来调用它，让它返回一个函数去处理剩下的参数。
* 用途：1.延迟计算；2.参数复用；3.动态生成函数
* function curry (fn, currArgs) {  
  return function() {  
   let args = [].slice.call(arguments);  
   // 首次调用时，若未提供最后一个参数currArgs，则不用进行args的拼接  
   if (currArgs !== undefined) {  
   args = args.concat(currArgs);  
   }  
   // 递归调用  
   if (args.length < fn.length) {  
   return curry(fn, args);  
   }  
   // 递归出口  
   return fn.apply(null, args);  
  }  
  }

### 20.写个js继承的例子

//寄生组合继承  
function Animal (name) {  
 // 属性  
 this.name = name || 'Animal';  
 // 实例方法  
 this.sleep = function(){  
 console.log(this.name + '正在睡觉！');  
 }  
}  
// 原型方法  
Animal.prototype.eat = function(food) {  
 console.log(this.name + '正在吃：' + food);  
};  
function Cat(name){  
 Animal.call(this);  
 this.name = name || 'Tom';  
}  
(function(){  
 // 创建一个没有实例方法的类  
 var Super = function(){};  
 Super.prototype = Animal.prototype;  
 //将实例作为子类的原型  
 Cat.prototype = new Super();  
})();  
Cat.prototype.constructor = Cat;

参考： [JS继承实现方式](https://www.cnblogs.com/humin/p/4556820.html)

### 网络相关

### 1. http 1.0 1.1 2.0的区别

* 1.0和1.1的区别

1. 缓存处理：HTTP1.1则引入了更多的缓存控制策略
2. 带宽优化：HTTP1.1则在请求头引入了range头域，它允许只请求资源的某个部分，即返回码是206。
3. 错误通知的管理：在HTTP1.1中新增了24个错误状态响应码
4. Host头处理：HTTP1.1的请求消息和响应消息都应支持Host头域，且请求消息中如果没有Host头域会报告一个错误
5. 长连接：HTTP 1.1支持长连接和请求的流水线处理，在一个TCP连接上可以传送多个HTTP请求和响应。

* 1.x和2.0的区别

1. **新的二进制格式：** HTTP1.x的解析是基于文本。2.0的协议解析决定采用二进制格式，实现方便且健壮。
2. **多路复用：** 即连接共享，即每一个request都是用作连接共享机制的。一个request对应一个id，这样一个连接上可以有多个request，每个连接的request可以随机的混杂在一起，接收方可以根据request的 id将request再归属到各自不同的服务端请求里面。
3. **header压缩：** HTTP2.0使用encoder来减少需要传输的header大小，HTTP2.0可以维护一个字典，差量更新HTTP头部，大大降低因头部传输产生的流量。
4. **服务端推送：** 服务端推送能把客户端所需要的资源伴随着index.html一起发送到客户端，省去了客户端重复请求的步骤。

参考： [HTTP1.0、HTTP1.1 和 HTTP2.0 的区别](https://mp.weixin.qq.com/s/GICbiyJpINrHZ41u_4zT-A)

### 2.post请求常见的content-type

1. **application/x-www-form-urlencoded**: 最常见的 POST 提交数据的方式了。浏览器的原生 form 表单。
2. **multipart/form-data**: 使用表单上传文件时，必须让 form 的 enctyped 等于这个值。一般用来上传文件。
3. **application/json**：告诉服务端消息主体是序列化后的 JSON 字符串。上传复杂的结构化数据。
4. **text/xml**：使用 HTTP 作为传输协议，XML 作为编码方式的远程调用规范。

### 3.强制缓存和协商缓存

* **强制缓存**： 浏览器在请求某一资源时，会先获取该资源缓存的header信息，判断是否命中强缓存（cache-control和expires信息），若命中直接从缓存中获取资源信息，包括缓存header信息；本次请求根本就不会与服务器进行通信；
* **协商缓存**： 如果没有命中强缓存，浏览器会发送请求到服务器，请求会携带第一次请求返回的有关缓存的header字段信息（Last-Modified/If-Modified-Since和Etag/If-None-Match），由服务器根据请求中的相关header信息来比对结果是否协商缓存命中；若命中，则服务器返回新的响应header信息更新缓存中的对应header信息，但是并不返回资源内容，它会告知浏览器可以直接从缓存获取；否则返回最新的资源内容

参考： [http协商缓存VS强缓存](https://www.cnblogs.com/wonyun/p/5524617.html)

### 4.跨域有哪些方法

1. **JSONP**
2. **通过修改document.domain来跨子域。** 只能把document.domain设置成自身或更高一级的父域。
3. **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，并不会因新页面的载入而进行重置。
4. **使用HTML5中新引进的window.postMessage方法来跨域传送数据** 使用它来向其它的window对象发送消息，无论这个window对象是属于同源或不同源，目前IE8+、FireFox、Chrome、Opera等浏览器都已经支持window.postMessage方法。
5. **CORS**
6. **服务器代理**
7. **flash**

### 5.写一个jsonp的实现

* 利用了 **script** 标签没有跨域限制这一“漏洞”来达到与第三方通讯的目的。简单地说，该协议就是，允许用户传递一个callback参数给服务端，然后服务端返回数据时会将这个callback参数作为函数名包裹json数据，这样客户端就可以随意定制自己的函数自动处理返回的数据了。
* var flightHandler = data=>{  
   console.log(data);  
  }  
  var url = "http://flightQuery.com/jsonp/flightResult.aspx?code=CA1998&callback=flightHandler";  
  var script = document.createElement('script');  
  script.setAttribute('src', url);  
  document.getElementsByTagName('head')[0].appendChild(script);