前端知识点

js基础

1、执行上下文/作用域链/闭包

（1）简单来说，执行上下文是js代码运行时所在的环境的抽象，主要定义变量或函数等数据，以及他们的操作。分为全局和局部（函数）执行上下文。每个执行上下文都有一个与之关联的变量对象，而环境中定义的所有变量和函数都保存在这个对象中。

而当代码在一个环境中执行时，会创建变量对象的一个作用域链，而这个作用域链能保证变量和函数的有序访问。

（2）闭包：是指有权访问另一个函数作用域中的变量的函数。

两点：内部函数，具有外部变量的引用。

应用场景：定时器，函数防抖



2、this/call/apply/bind

（1）this：函数内部的属性，this引用的是函数执行时的执行上下文对象

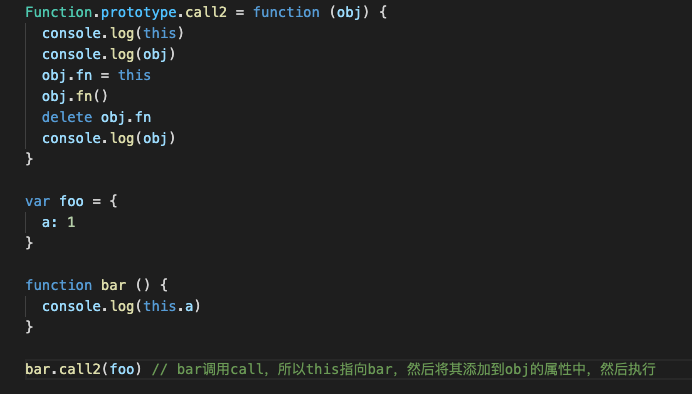
（2）改变this的指向：call()、apply()、bind()和new一个对象

（3）call和apply的区别：call和apply都是将this重新指向第一个参数，不同的是，apply只能接收两个参数，第二参数是一个数组，而call接收多个参数。严格模式下，this值就是call和apply的第一个参数，无论是null还是undefined。非严格模式下，如果this被指定为null或者undefined，那么会自动重新指向window。

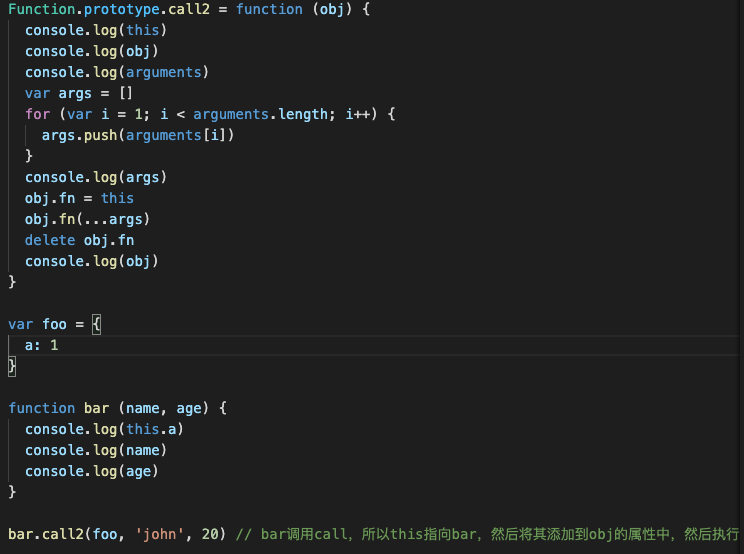
（4）call和apply的模拟实现：

想法：当我们调用fn,call(0bj)时，实际上可以把fn变成obj的一个属性，这样再执行这个函数，执行完删除这个属性，就完成了目标。

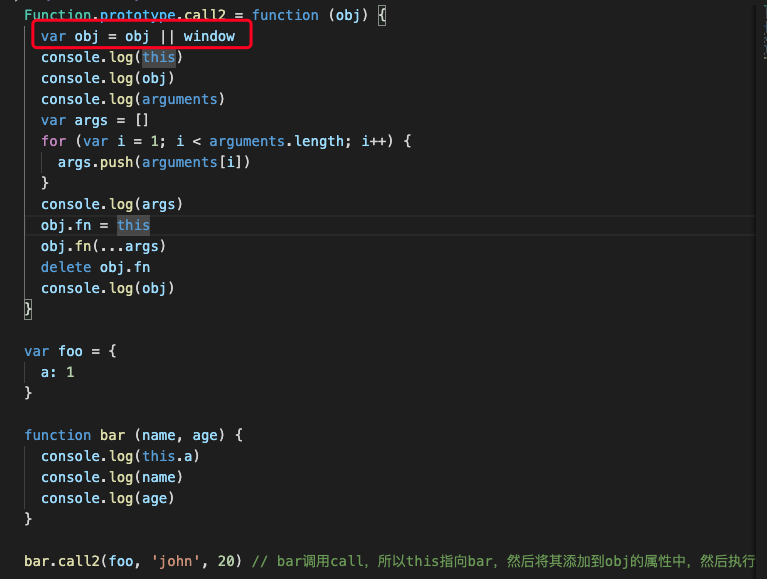
那么其简易实现如下：



接着，call2里需要可以指定参数，进一步改进，也就是将参数都封装到一个数组里，然后解构就行了



虽然解决了参数问题，但是还有当传入的第一个参数为null或者undefined时，this指向window。这个实现就相对简单些。只需要将obj重新赋值下即可



这样就完成了一个简易的call2函数，apply的实现与之类似。

（5）bind函数：也是改变this的指向，指向第一个参数，和call和apply不同的是，bind绑定不会立即执行函数。如果要执行，那么就需要加一个括号。并且bind只生效一次，永久绑定到第一个对象上。

3、原型/继承

（1）原型和原型链

简单来说，我们没创建一个函数，它都有一个prototype属性，这个属性就是原型，通常用来保存公共属性和方法。其原型里包含一个constructor指针，其指向函数本身。而函数的实例也包含一个\_\_proto\_\_属性，指向函数的原型。

类似java中，所有类都是Object的子类，js中，所有函数对象中的原型中都有一个\_\_proto\_\_属性指向Object，也就是说，函数的原型是Object实例。这其中就形成了一个原型链。

（2）如何利用原型实现继承

继承就是让子类拥有父类的所有属性和方法。那么要用原型实现继承，只需要将父类的所有属性和方法都绑定到子类的原型上，那么子类就获得了父类的所有属性和方法，前面说过，函数的实例对象包含一个指向父类原型的\_\_proto\_\_指针，而实例对象包含构造的所有属性，那么我们只需要将父类的实例对象赋值给子类的原型，那么子类的原型就包含了父类实例对象上的属性，也就是包含了\_\_proto\_\_和父类的属性。从而实现了继承。

（3）实现继承的几种方式：

1、原型链，也就是上面红色部分所说的

问题：当父类中的属性是一个引用类型时，在子类中修改属性，会导致所有其他子类的属性也跟着改变。 同时，这种方式创建子类的实例时，不能项父类型的构造函数中传递参数。

2、借用构造函数

也就是在子类型的构造函数中使用call(this))或者apply(this)方法。

SuperType.call(this)，就把父类中的this指向了之类，那么父类中的this.color等属性就属于子类型了，也就实现了继承。这种方式可以向父类构造函数中传参。

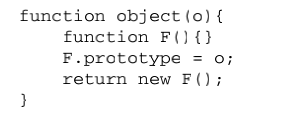
问题：也很明显，所有属性和方法都要写在父类型的构造函数上，不能在原型中，否则引用不到。

3、组合继承

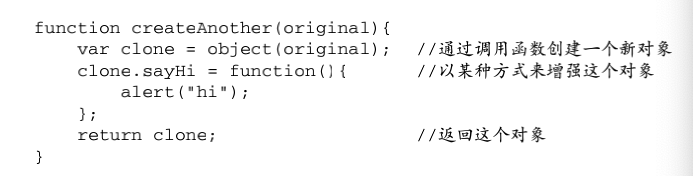
也就是将原型链继承和构造函数组合起来使用，也就是既在子类构造函数中，通过父类型构造函数call(this)，也将父类型的实例赋值给子类型的原型。

4、原型式继承

声明一个函数，传递一个对象，在函数内部新声明一个函数，令这个的原型等于传递过来的对象，最后返回这个函数的一个实例，返回的实例也就有了这个对象的所有属性和方法。



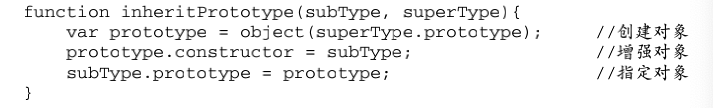
5、寄生式继承。



6、寄生组合式继承

组合式继承方式需要调用两次构造函数，分别是父类构造函数call(this)时和new 一个父类型实例时。

寄生组合式继承的想法是，将第二次调用的构造函数替换为寄生式的方式。不去调用构造函数，而是通过object(superType.prototype)赋值一份父类型的原型，并赋值给子类型的原型。



个人觉得还不如直接调用构造函数，虽然重复了一次实例属性的赋值。

4、Promise

（1）Promise是什么

Promise是实现异步编程的一种新的解决方案，相比于旧的解决方案有两点优点：

1、不必在每次执行异步操作时，事先指定回调函数。

2、可以解决回调地狱问题。

Promise有几个方法，原型的上的方法有.then和.catch Promise的构造函数方法有resolve, reject,all,race等

（2）Promise的手写实现：

<https://github.com/WangCheng0928/promise>

（3）宏任务和微任务

这就涉及到事件循环了！

宏任务：I/O操作，setTimeout,setInterval,ajax,dom事件回调

微任务：MutationObserver(观察dom节点的变化)，Promise.

由于js是单线程的，所以一般非阻塞的同步任务会放在主线程中执行，而异步任务就会由事件管理模块管理，放在回调队列中，等待同步任务执行完后，再去执行异步任务。那么对于宏任务和微任务，它们会分别放到宏队列和微队列中。那么就有这么一个规定，在准备执行宏任务之前，必须先把微任务一个个执行完。

由上可以总结出：js执行代码时，首先在主线程中执行同步代码，对于异步任务，如果是宏任务会将其放到宏队列中，如果是微任务，就将其放到微队列中，待同步任务执行完后，在执行每一个宏任务执行之前，首先得把微队列里的微任务一个一个执行完。然后再去执行宏任务。如果宏任务里又有微任务，那么微任务再次会放到微队列里。那么会在执行下一个宏任务之前，首先把微队列里任务执行完，再去执行宏任务。

（4）async await

async和await是目前解决异步编程的终极解决方案

使用async修饰的函数最终会返回一个Promise对象，然后可以用.then方法得到里面的结果，或者使用await 可以得到异步操作的结果。其原理是将generator和自动执行器封装在一个函数里。

5、深浅拷贝

（1）、基本概念

深浅拷贝都是对于js中的引用类型而言的，浅拷贝就指示复制对象的引用（地址），如果拷贝后对象发生变化，原对象也会发生变化。只有深拷贝才是真正的对对象的拷贝。

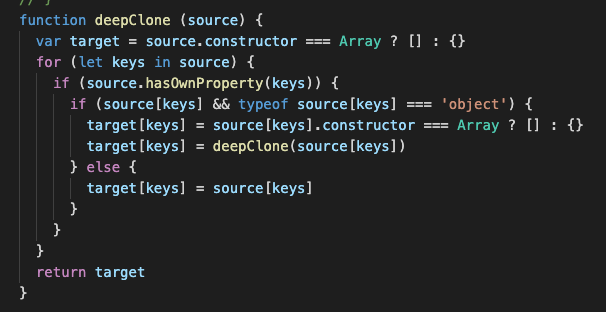
浅拷贝：object.assign()方法， = 赋值

深拷贝：1、利用json对象中的parse和stringfy

2、利用递归实现每一层都重新创建对象并复制。

其中，利用json.parse(json.stringfy())只能适用一些简单的对象，而对于对象中包含undefined、function和symbol就会导致这些属性丢失。

深拷贝的递归实现：

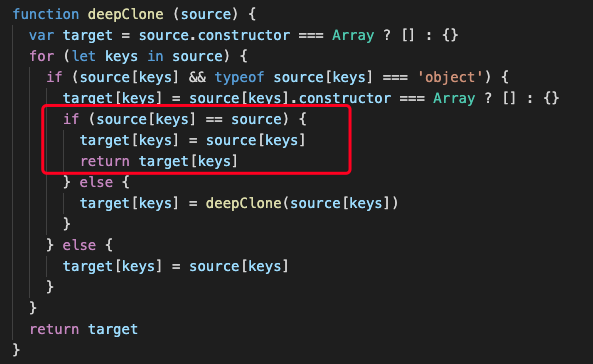


数组的concat，slice，object的assign方法都是只对对象的第一层进行深拷贝。包括...展开符也是一样

（2）循环引用问题

当要拷贝的对象的一个属性是对象本身时，例如obj.z=obj时，这时候就会出现循环引用问题。导致报错。

解决办法：只需要在赋值的时候判断下source[keys]=source即可



6、事件机制/Event Loop

（1）如何实现一个事件的发布订阅



最简易的实现方式，运用到队列，首先设置一个监听器，表示要监听的事件名称，以及当监听到后，怎么处理。然后发布事件的时候只需要，传事件名和参数，就能自动触发事件的执行。

（2）事件循环/Event Loop

在前面已经将过了，不过这里需要注意浏览器的执行机制。

当我们在浏览器中输入网址后，会发生哪些事情？

1、DNS查询：查询顺序：浏览器dns缓存、本地dns缓存、本地hosts缓存、路由器dns、isp的dns服务器、根服务器。查询到后就可以直接访问ip地址了。

2、建立连接：tcp/ip协议，三次握手和四次挥手

3、发送请求

4、服务器响应数据

5、浏览器接受到html页面：这里才真正到了浏览器该做事的时候了。

此时的具体步骤是：1、解析html文本，构建dom树，2、构建呈现树RenderTree，这一步主要是将css样式附着在DOM节点上。3、布局、也就是节点应该呈现在什么地方

4、绘制，将呈现树绘制出来，显示在屏幕上。

需要注意的是html文本自上而下解析，遇到任何样式和脚本都会阻塞。

1、css加载不会阻塞html文件解析，但会阻塞dom的渲染

2、css加载会阻塞后面js语句的执行

3、js会阻塞html的解析和渲染

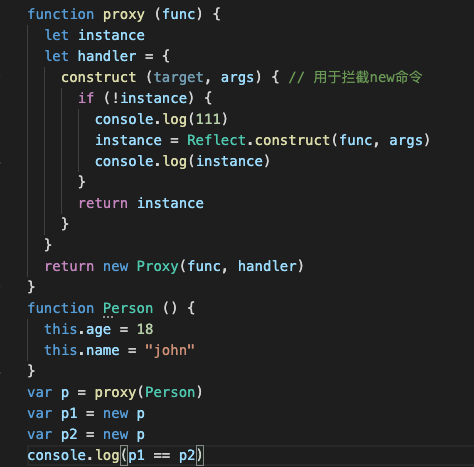
回到event loop中，渲染更新会在event loop中的任务宏任务和微任务完成后执行，但并不是每轮event loop都会更新渲染，这取决于是否修改了dom和浏览器是否觉得有必要在此时立即更新状态呈现给用户。如果在一帧的事件内（16.7ms）修改了多处dom，浏览器可能将变动积攒起来，只进行一次绘制。如果希望在每轮event loop都即时呈现，可以使用requestAnimationFrame.

7、web worker

web worker 是html5标准的一部分，包含一套api，它允许js程序运行在主线程之外的另一个线程中。通过new worker(xxx.js)创建一个实例对象，通过worker.postMessage(发送数据)，主线程就可以通过woker.onMessage来接受到数据。

附：单例模式：

proxy，相当于一个拦截层，在执行函数前



css基础

1、position

2、行内元素/块状元素

3、flex布局

（1）介绍下flex布局

（2）flex:1值得是什么

（3）flex-shrink 和 flex-basis属性