1-3 几个面试题







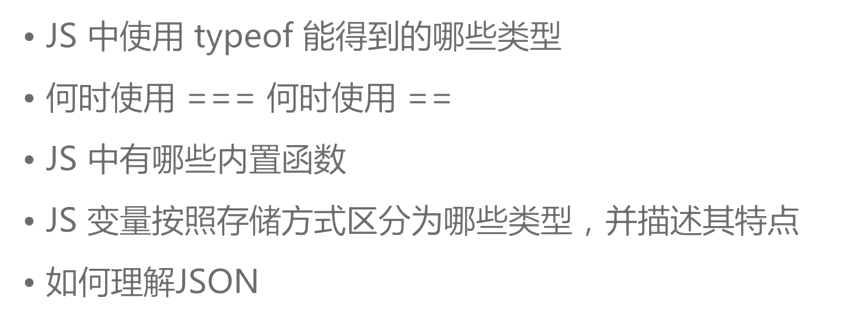




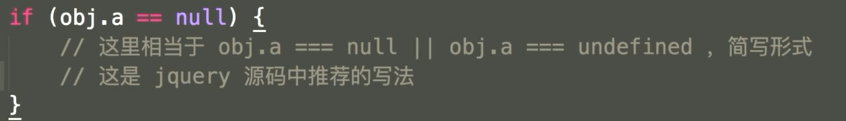


2-1 变量类型和计算

一、题目：



2.何时使用==何时使用===？

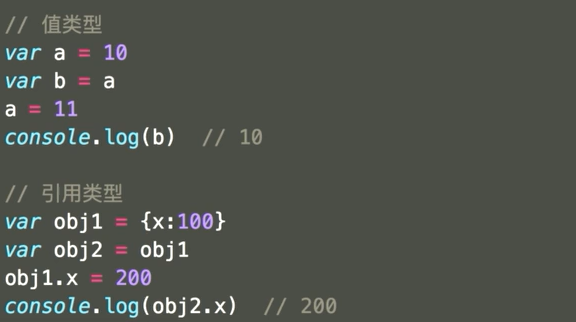
答：

除了判断对象中某个属性是否存在使用==，其余均使用===。

3.Js中有哪些内置函数？

答：

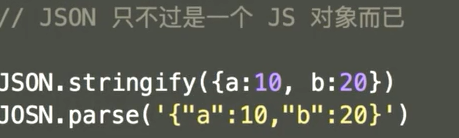
4.js按照存储方式有哪些类型？

答：

值类型的存储方式为栈，引用类型为堆。引用类型为节省空间。

5.如何理解json？

答：



Json还是一种数据格式。

二、知识点：

1.变量类型：

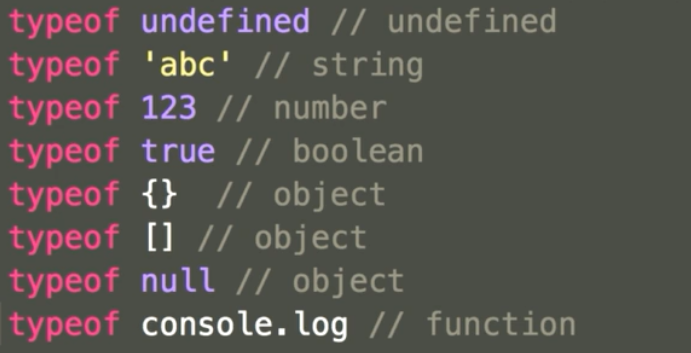
①值类型vs引用类型：

引用类型包括对象、数组和函数，浅拷贝，指针。

②Typeof运算符详解

结果有6种类型，undefined、string、number、boolean、object、function，详情见下图。

只能区分值类型，引用类型不能区分，引用类型中只有function函数能被区分，其余都是object。



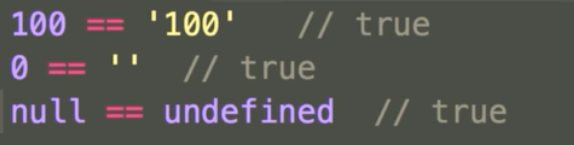
2.变量计算—强制类型转换

①字符串拼接：

Number类型 + string类型，结果为string类型

Number类型 - string类型，结果为number类型

②==运算符：

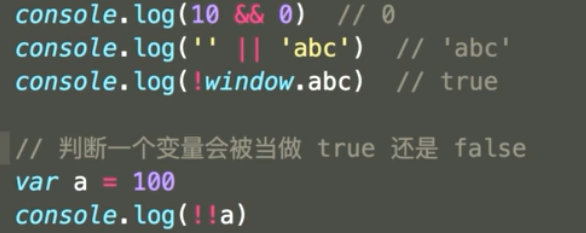


==会试图将两边变为相等，0、null、undefined、空字符等都会转换为Boolean型的false。

③if语句

If语句会先进行boolean型转换，非零为true，非空字符为真等。

④逻辑运算



&&遇假返回，全真返回最后一个数，||遇真返回，全假返回最后一个，返回结果均为当前数。

Window.abc为undefined，!转换之后为true。

!!可以判断这个值转换为Boolean型的值。

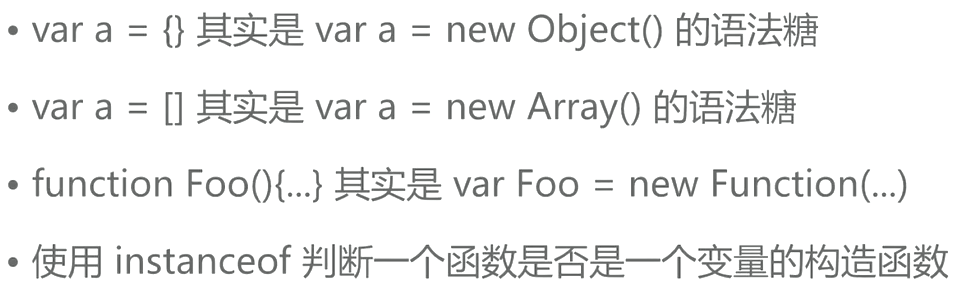
2-5 原型和原型链-构造函数

1.构造函数：



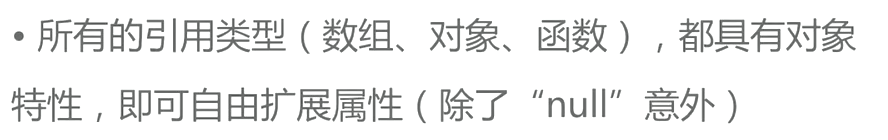
构造函数一般大写字母开头，this指向f。return this 是默认有的，不管写不写都有。

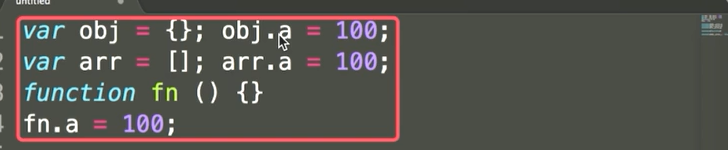
2.构造函数扩展：



所有的引用类型都有构造函数。推荐前面的表达方式。

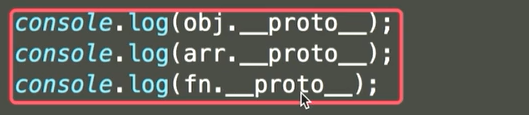
3.原型规则和实例：

①



②

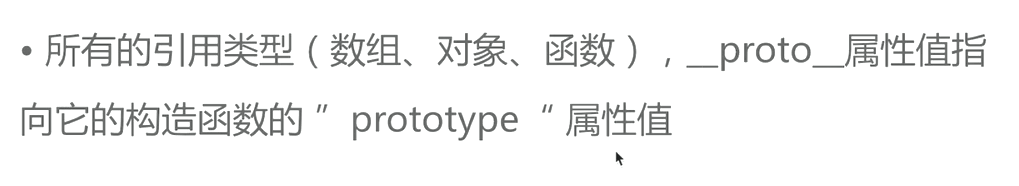




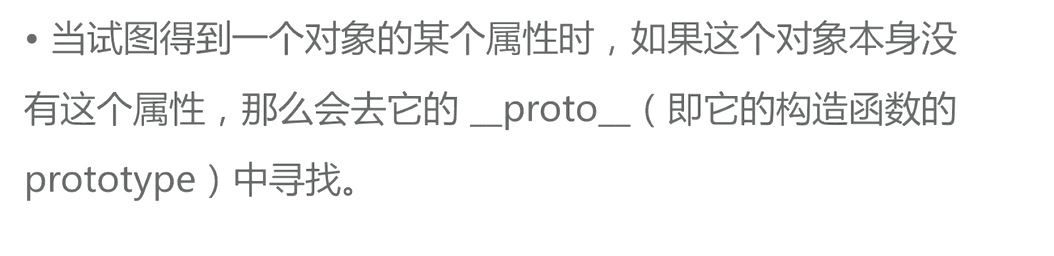
③



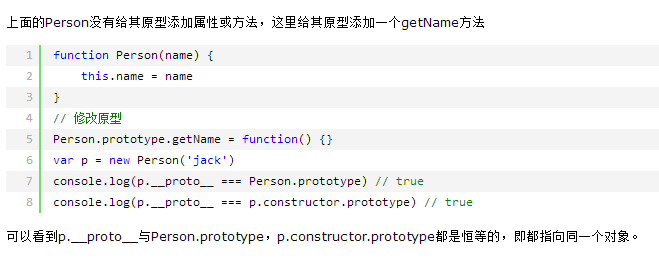
④

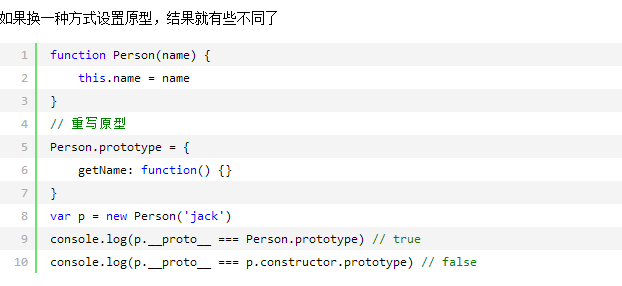


 ⑤



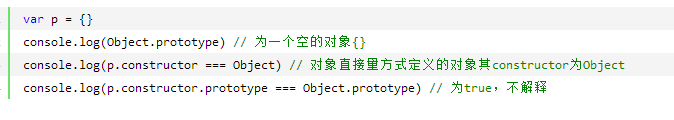
补充：



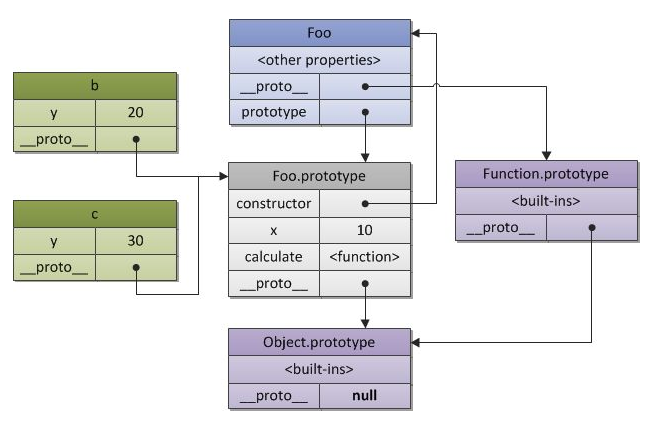


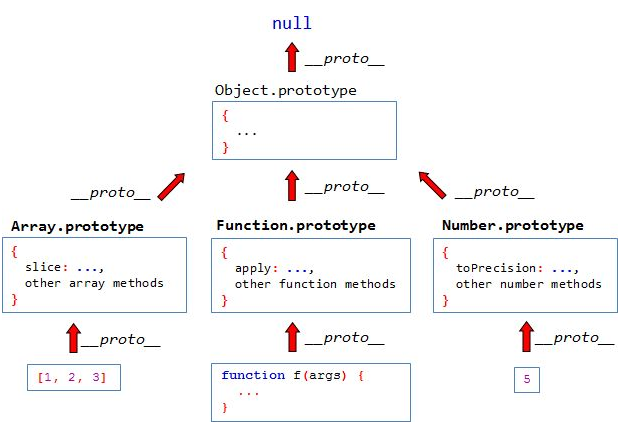
这里直接重写了Person.prototype（注意：上一个示例是修改原型）。输出结果可以看出p.\_\_proto\_\_仍然指向的是Person.prototype，而不是p.constructor.prototype。

这也很好理解，给Person.prototype赋值的是一个对象直接量{getName: function(){}}，使用对象直接量方式定义的对象其构造器（constructor）指向的是根构造器Object，Object.prototype是一个空对象{}，{}自然与{getName: function(){}}不等。如下



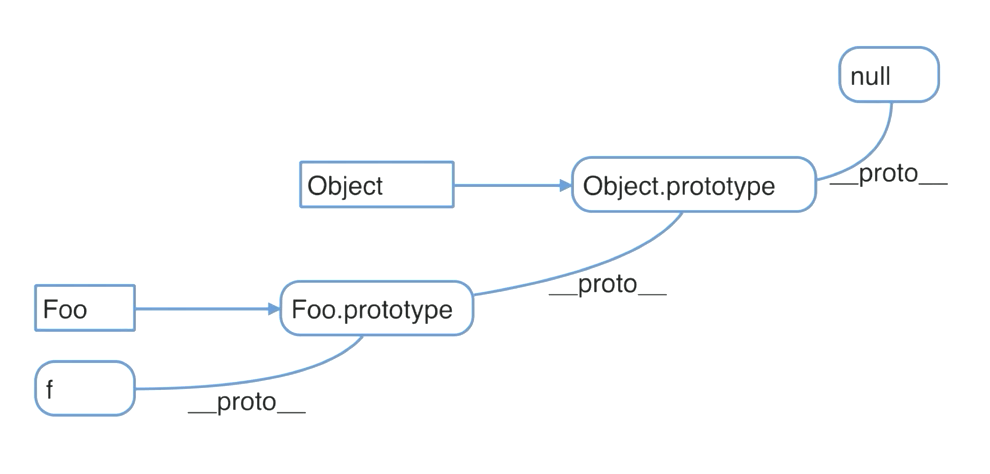
自我补充：prototype属性指向一个对象，所以Function.prototype.\_\_proto\_\_ === Object.prototype。constructor存在于每个function的prototype属性中，constructor易变，重写prototype会改变constructor。Object.prototype.\_\_proto\_\_指向null，特例。







4.原型链：



5.instanceof：

用于判断引用类型属于哪个构造函数的方法。



解题：

①

用instanceof来判断 arr instanceof Array，或者是使用Object.prototype.toString.call(),e.g.

Object.prototype.toString.call([])    输出   [object Array]

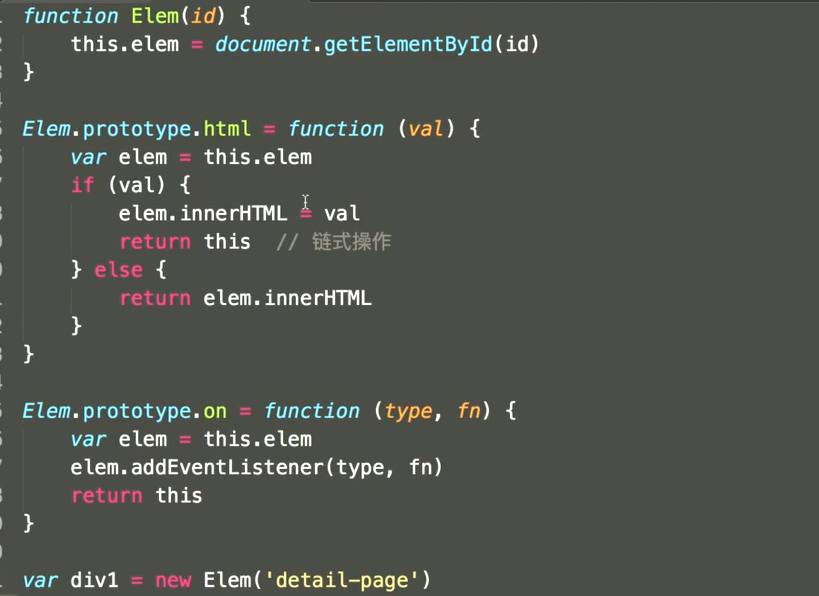
②



基础：



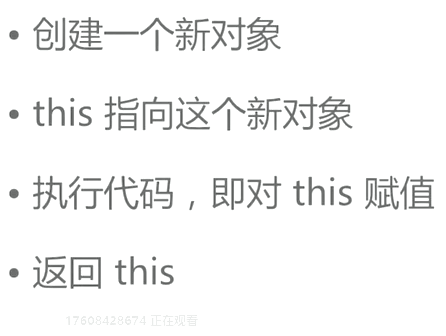
进阶：（封装jq的html方法和on方法）



③



答：

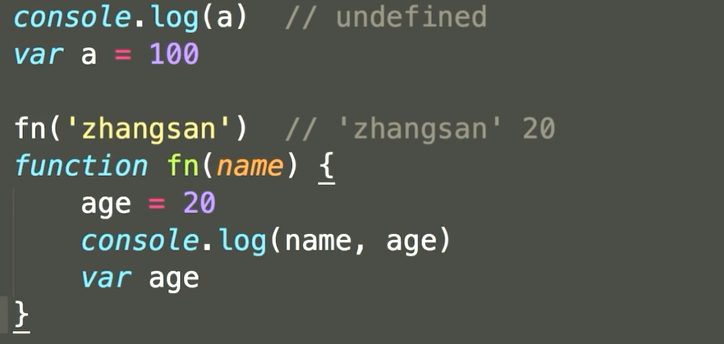


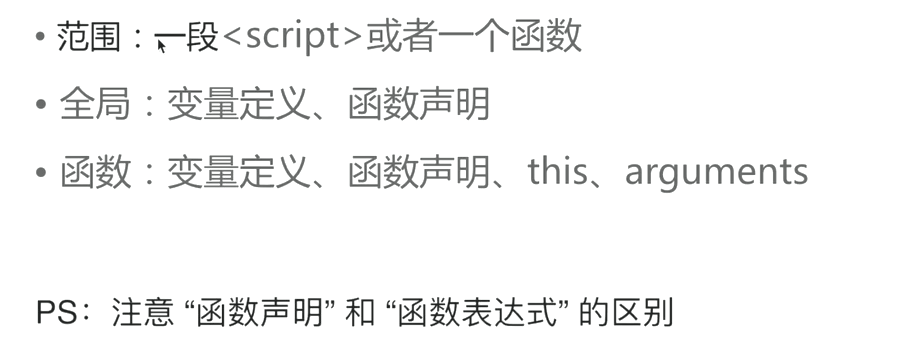
④



第三章：作用域与闭包

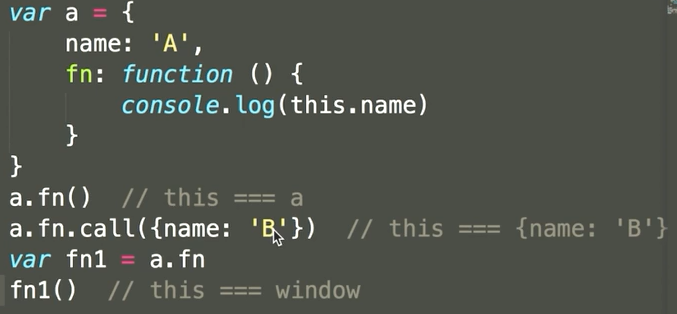
3-1 执行上下文：





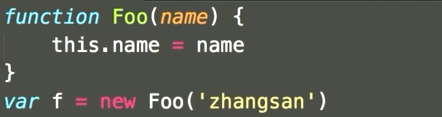
3-2 this





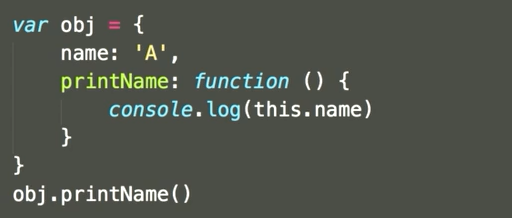
①作为构造函数执行：

this指向新创建的对象，参考new一个对象的步骤。



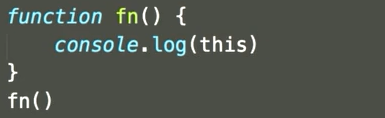
this指向f；new改变了this的指向。

②作为对象属性执行：



this指向obj。

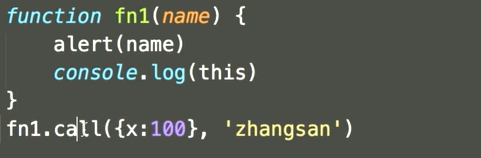
③作为普通函数执行：



this指向window。

④call apply bind

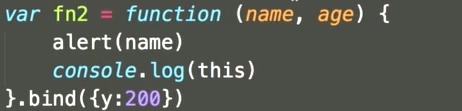
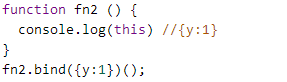
Call：



this指向{x:100}，{x:100}这个对象借用了fn1。

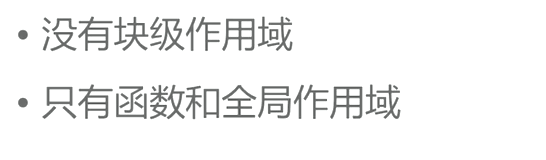
Apply和call一样，但apply传参数是以数组形式。

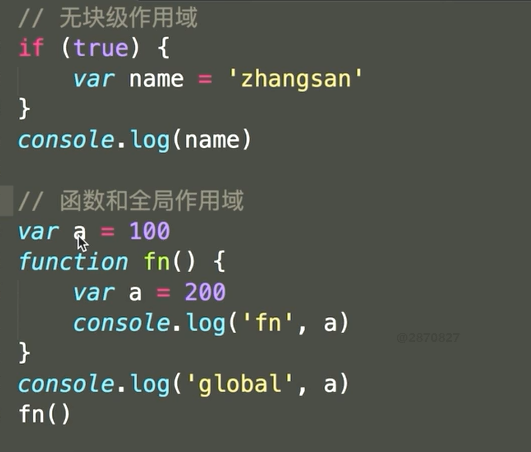
Bind：



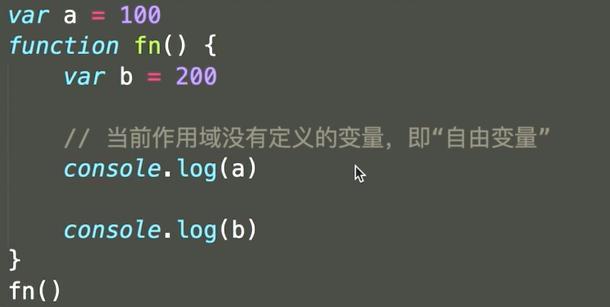
this指向{y:200},bind不会马上执行，需要加上();

3-3 作用域





3-4 作用域链



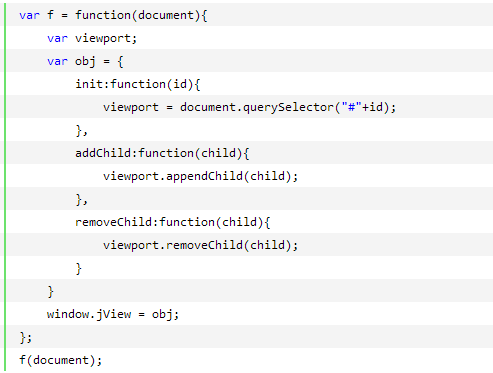


F1的父级作用域为全局，F2的父级作用域为F1，定义时决定其作用域。

3-5 闭包







obj 是在函数 f 中定义的一个对象，这个对象中定义了一系列方法， 执行window.jView = obj 就是在 window 全局对象定义了一个变量 jView，并将这个变量指向 obj 对象，即全局变量 jView 引用了 obj . 而 obj 对象中的函数又引用了函数 f 中的变量 viewport ,因此函数 f 中的 viewport 不会被 GC 回收，viewport 会一直保存到内存中，所以这种写法满足了闭包的条件。

解题：

①说一下对变量提升的理解：



变量定义会被提前，函数声明也会被提前。

②说明this几种不同的使用场景

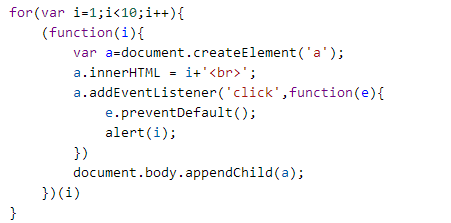
1.作为构造函数执行

2.作为普通函数执行

3.call apply bind

4.作为对象属性执行

③创建10个<a>标签，点击的时候弹出对于的序号



把i转到另一函数的作用域即可。

④如何理解作用域

1.自由变量：没有在此作用域定义的变量。

2.作用域链

3.闭包

⑤实际开发中的闭包应用

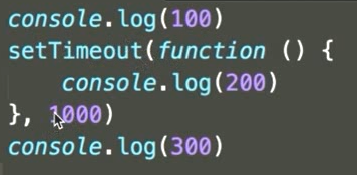
1.封装变量，收敛权限



在isFirstLoad外面无法修改\_list数组。

四。异步和单线程

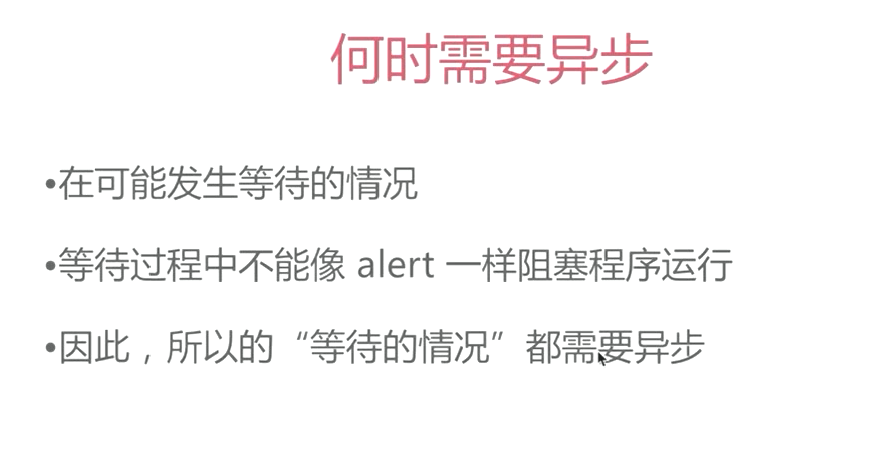
①什么是异步？



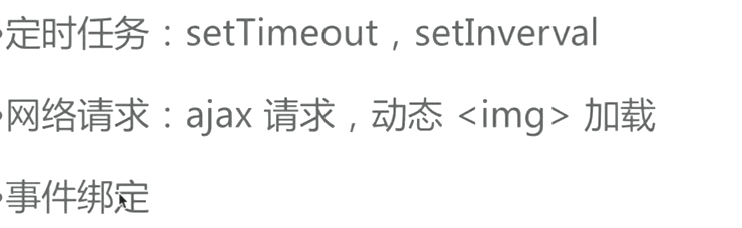
输出100,300，200。异步没有阻塞程序运行。



对比同比，alert阻塞了程序的运行



②前端使用异步的场景：



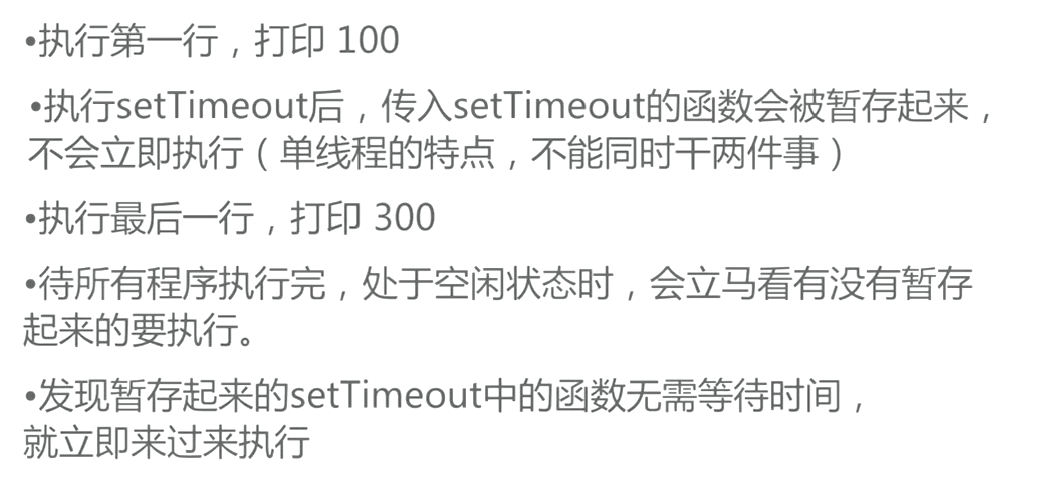
③异步和单线程

Js是单线程的，js遇到需要异步的场景时，先将异步中的函数放一边，执行下面的代码，下面代码执行结束后，回头执行异步中的函数。若是有多个异步代码，例如：

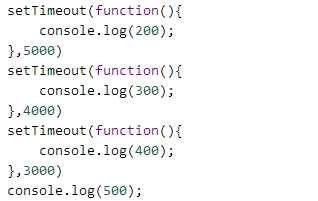


代码结束后，执行第一个setTimeout,发现要等待1s，则先执行后面的setTimeout，输出顺序为，100,400，300，200。

单线程：每次只能做一件事。



Eg.



例如：这个代码输出，500,400,300,200，最后用时约5s。

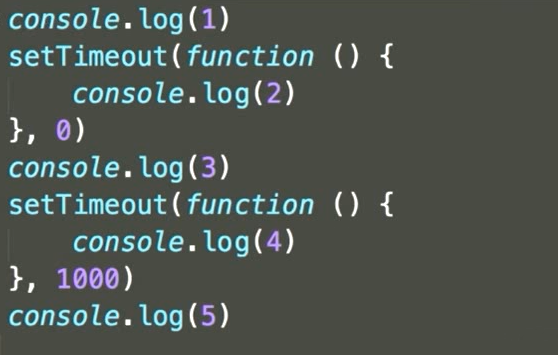
解答：

1.同步和异步有什么区别？分别举个例子。

答：同步会阻塞代码执行，异步不会。例子：alert同步，setTimeout异步。

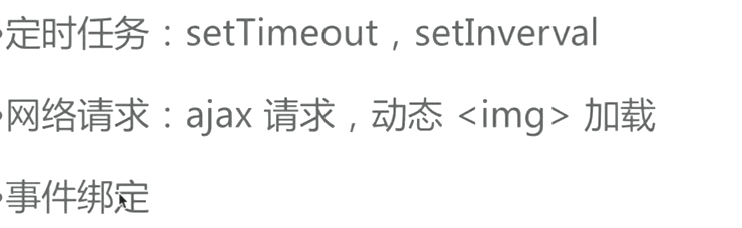
2.setTimeout的笔试题

①



输出 1,3,5,2,4

3.前端使用异步的场景

答：

都需要等待。

五.补充

1.知识点：

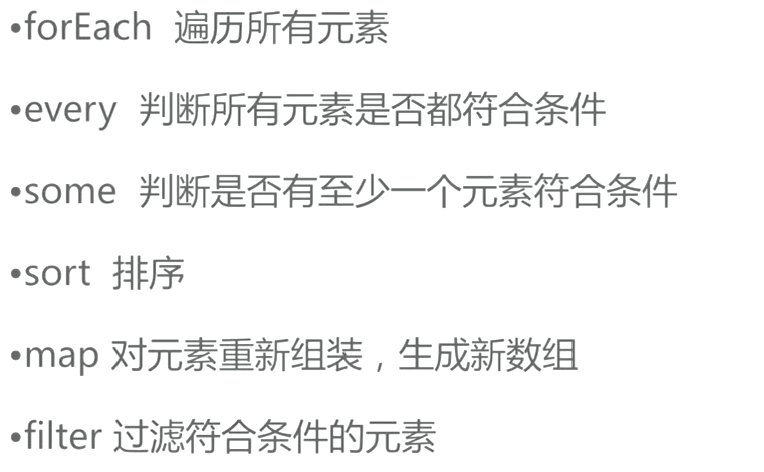
①日期：



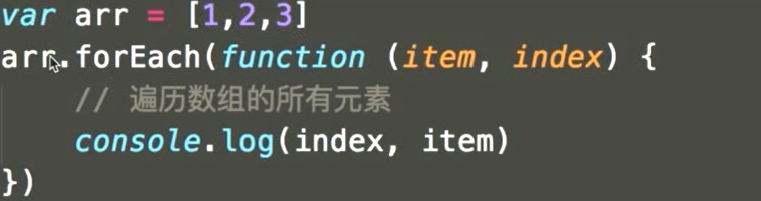
②Math:

获取随机数Math.random(),常用于清除缓存

③数组API：

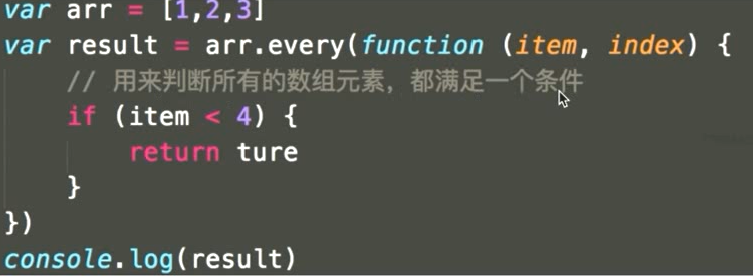


1.forEach

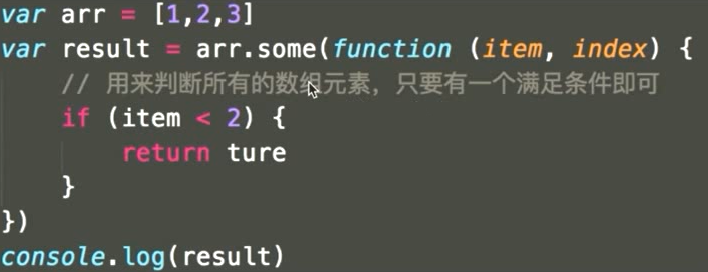


Item是数组每个元素的值，index是对应的索引值。

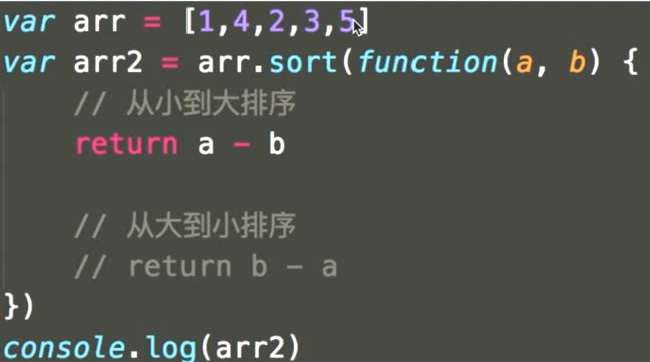
2.every



3.some

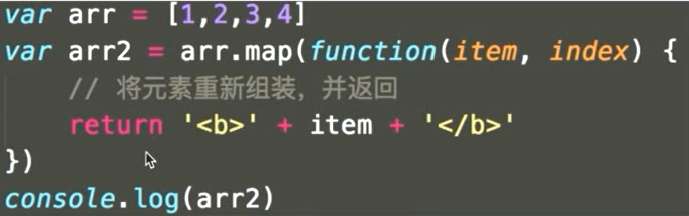


4.sort



a-b为升序，b-a为降序，会改变原数组。

5.map



重新整合数组，不会改变元素组，返回一个新数组。

6.filter



过滤指定条件的元素，符号条件的return true，返回新数组，不改变原数组。

④对象API：

1.for in

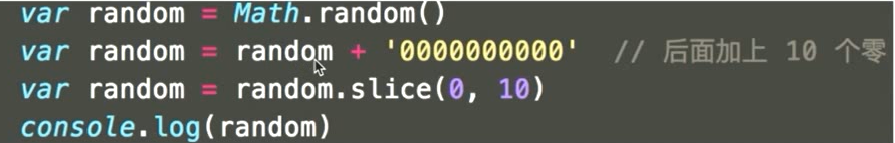


解答：

1.获取2018-03-11格式的日期：



2.获取随机数，长度一致



3.写个能遍历对象和数组得到forEach函数

