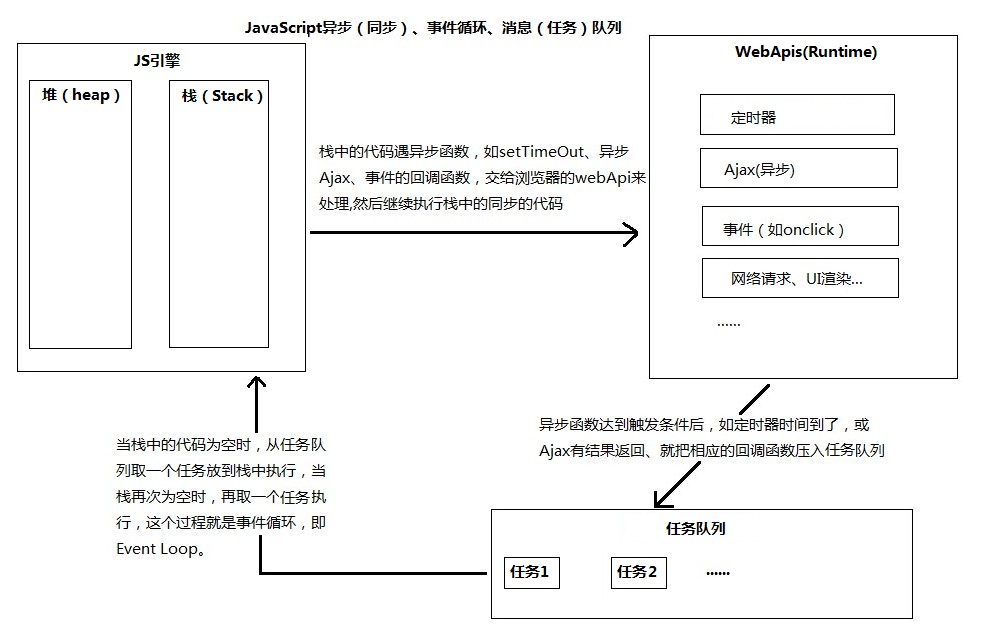
# 你真懂JavaScript运行机制吗？

# 写在前面

说起javascript（以下简称js）这门语言，相信大家已经非常熟悉了，不管是前端开发还是后端开发几乎无时无刻都要跟它打交道。虽说开发者每天几乎都要操作js，但是你真的确定你掌握了js的运行机制吗！下面我们就来聊聊这话题。

# JavaScript运行机制图解



上图我们可以分为两部分：浏览器中的JS引擎和运行环境Runtime，那它们的区别是什么？

* JS引擎：编译并执行代码的地方。
* 如上图中可以看出JS引擎分为两大核心部分：栈和堆
* 栈（Stack）:js代码的执行都要压到此栈中执行。
* 堆：存放对象、数组的地方，js垃圾回收就是检查这里。
* Runtime：浏览器的运行环境，它提供了一些对外接口供JS调用，以及跟外界打交道。

# JavaScript引擎是单线程

JS引擎是单线程的，也就是说在一个时间段内，事情只能一件一件的按先后顺序去做，第一件事没做完就不能第二件事。那么在js引擎中负责解释和执行js代码的线程只有一个，我们可以称之为**主线程**。

当然浏览器的运行环境Runtime还提供一些其他的线程，如定时器线程、ajax线程、事件线程、网络请求和UI渲染的线程，为了和js主线程分开，我们这里都统称它们为工作线程。

由于浏览器是多线程的，所以工作线程和js主线程都可以执行任务，线程间互不干扰。

# JavaScript同步（异步）任务

在JavaScript任务可以分为两种：

* 同步任务：在主线程上排队执行的任务，只有前一个任务执行完毕，才能执行后一个任务，若前一个任务耗费很长时间，则后面的任务会一直处于等待状态，即阻塞状态。
* 异步任务：在栈执行代码的过程中，如遇到异步函数，如setTimeout、异步Ajax、事件处理程序，会将这些异步代码交给浏览器的工作线程来处理，我们把这些任务称之为异步任务。异步任务是不进入主线程，而是进入任务队列（queue task）。
  + 什么异步函数？
  + 异步函数通常是由**发起函数**和**回调函数**构成的。如：
  + A（callback）  
    - 函数A就是发起函数
    - callback就是回调函数
  + 它们都是在主线程调用的，其中发起函数用来发起异步过程，回调函数用来处理结果。
  + 如：setTimeout(callback,1000)
  + setTimeout就是发起函数、callback就是回调函数。
  + 如：`异步的Ajax`
* var xhr = new new XMLHttpRequest();  
   xhr.onreadystatechange = callback; //callback为回调函数  
   xhr.open('get',url,true);  
   xhr.send(null); // send为发起函数
* 可以看出发起函数和回调函数也可以是分离的。
* 既然同步任务是在主线程中执行的，那么异步任务何时执行？
* 答：是这样的，一旦栈中同步任务执行完毕后，系统就会通过事件循环机制读取任务队列中的任务一个个移到栈中去执行。

# 事件循环

当主线程中的任务执行完毕后，会从任务队列中获取任务一个个的放在栈中执行去执行，这个过程是循环不断的，所以整个的这种运行机制又称为事件循环。

# 栈

在js中，代码最终都是在栈中执行的，栈结构的特点是：**先进后出，后进先出**。

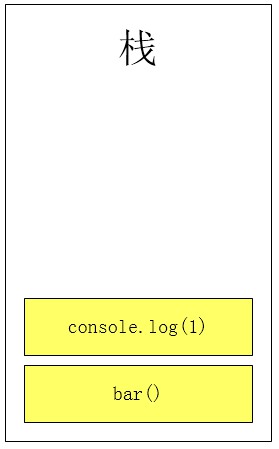
我们来看下面代码的运行结果：

function bar(){  
 console.log(1);  
 foo();  
}  
  
function foo(){  
 par();  
 console.log(3);  
}  
  
function par(){  
 setTimeout(function(){  
 console.log(2);  
 },0);  
}  
  
bar();

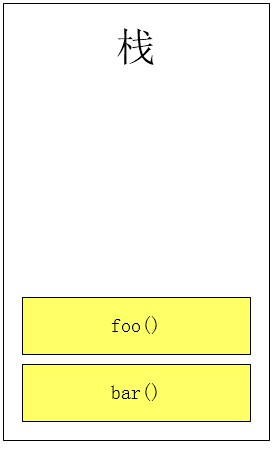
运行的最终结果是：132。 为什么结果不是123呢？

下我们来分析下代码运行时入栈和出栈的过程。

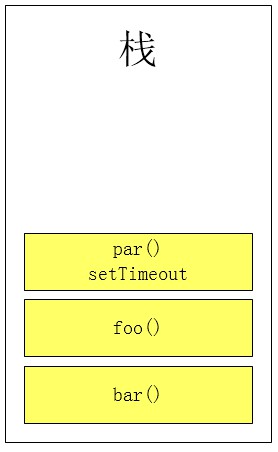
首先当调用函数bar()时，此函数就会先入栈，其内部的console.log(1)也会随之入栈执行。



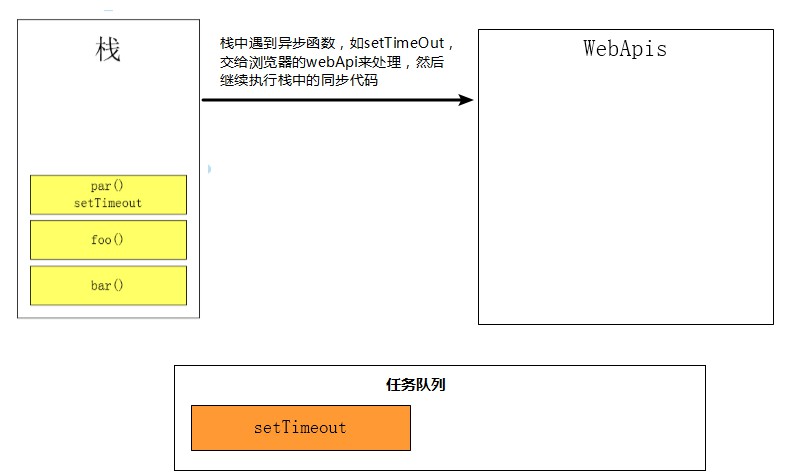
执行完console.log(1)后，就要出栈，于是控制台先打印出结果**1**，只剩下bar()在栈中。接着再执行函数bar内部的函数foo，于是函数foo也开心的入栈了。



执行函数foo的内部代码，调用函数par()，于是函数par()也要跟着入栈。



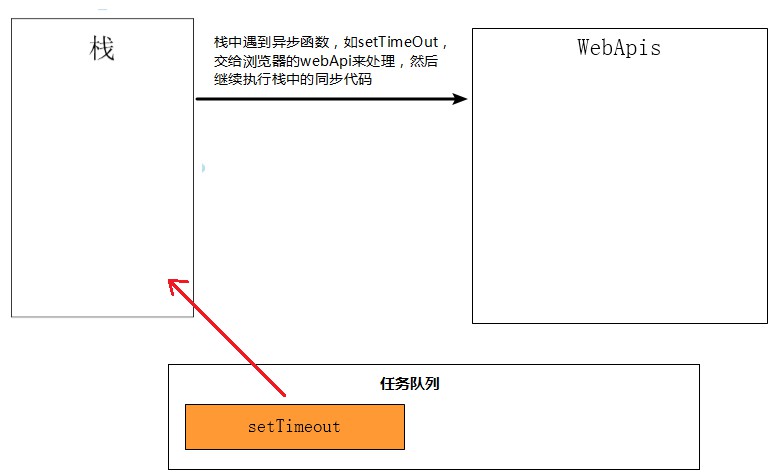
由于函数par()内部执行遇到了异步函数setTimeout,异步函数则会由浏览器的Runtime运行环境的工作线程来处理，等定时器设置的时间到达就会被放到任务队列中，此时栈的同步任务继续执行。



接着在执行par函数中的console.log(3),控制台打印结果为**3** ,此时栈的代码执行完毕后，会按照栈的特点进行

**先进后出，后进先出**顺序进行出栈。出栈顺序：**先函数par()----》后函数foo()----》最后函数bar**。

最后只剩下异步任务，由主线程去获取任务队列中的任务放在栈中去执行。也可以认为栈中的同步代码执行总是在读取异步任务之前执行。



最后执行setTimeout中的回调函数：结果控制台输出为2。

setTimeout(function(){  
 console.log(2);  
},0);

所以最终代码的运行结果为132。

# 小结

* js引擎是单线程执行js代码，同步任务在栈中按顺序执行，如果某一个同步任务没有执行完毕，则后面的代码将会处于阻塞等待状态
* 栈中若执行遇到了异步任务（如定时器、异步Ajax、事件），会将此异步任务通过浏览器对应的工作线程来处理。
* 工作线程中的所有异步任务均会按照设定的时间进行等待，时间一到会被加入任务队列。如果是异步ajax,则等待其返回结果在加入到任务队列
* 当栈中为空时，会通过事件循环来一个个获取任务队列中的任务放到栈中进行逐个运行。即栈中的同步任务总是在读取异步任务之前执行
* 定时器设置的时间不一定按照设定的时间进行执行，这得取决于栈中同步任务耗费的时间。因为栈中执行的同步任务如果耗费很长时间，则会影响到异步任务回调函数的执行。