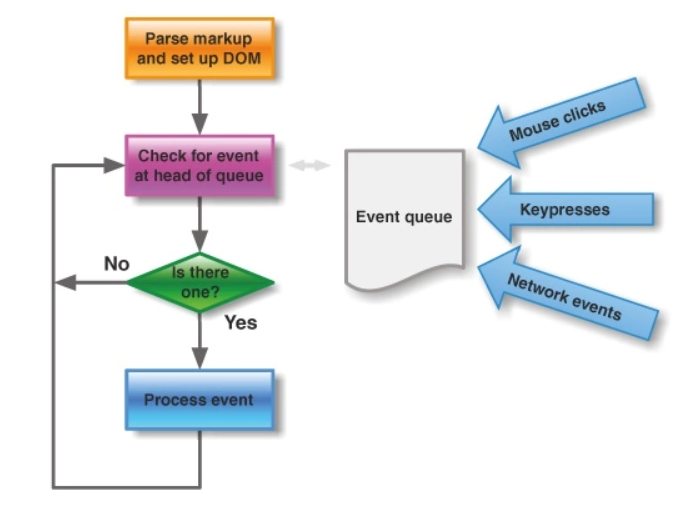
**JavaScript**是一门**单线程**解释型语言，但是**浏览器**本身是**多线程**的，**浏览器的内核**是**多线程**的，比如解释运行JavaScript的同时还在加载网络资源。

但是我们还有知道在**JavaScript里有异步编程**的说法，比如Ajax，setTimeout，setInterval或者ES6中的Promise，async，await。

**JavaScript事件循环和消息队列（浏览器环境）**

因为**JavaScript**是**单线程**的，而**浏览器**是**多线程**的，所以为了执行不同的同步异步的代码，**JavaScript运行**的环境**采用**里**事件循环**和**消息队列**来达到目的。  
**每个线程的任务执行顺序都是FIFO(先进先出)**  
在JavaScript运行的环境中，有**一个负责程序本身的运行**，作为**主线程**；另**一个负责主线程与其他线程的通信**，被称为**Event Loop 线程**。  
每当**主线程**遇到**异步的任务**，把他们移入到**Event Loop 线程**，然后**主线程继续运行**，等到**主线程完全运行完**之后，**再去Event Loop 线程拿结果**。  
而**每个异步任务**都包含着与它**相关联的信息**，比如**运行状态**，**回调函数**等。



由此我们可以知道，**同步任务**和**异步任务**会被分发到不同的线程去执行。  
现在我们就可以分析一下一下代码的运行结果了。

setTimeout(()=>{console.log("我才是第一");},0);

console.log("我是第一");

1. 因为setTimeout是异步的事件，所以主线程把它调入Event Loop线程进行注册。
2. 主线程继续执行console.log("我是第一");
3. 主线程执行完毕，从Event Loop 线程读取回调函数。再执行console.log("我才是第一");;

### setTimeout 和 setInterval

#### setTimeout

这里值得一提的是，setTimeout(callback,0)指的是主线程中的同步任务运行完了之后立刻由Event Loop 线程调入主线程。  
而计时是在调入Event Loop线程注册时开始的，此时setTimeout的回调函数执行时间与主线程运行结束的时间相关。  
关于setTimeout要补充的是，即便主线程为空，0毫秒实际上也是达不到的。根据HTML的标准，最低是4毫秒。

#### setInterval

需要注意的是，此函数是每隔一段时间将回调函数放入Event Loop线程。  
**一旦setInterval的回调函数fn执行时间超过了延迟时间ms，那么就完全看不出来有时间间隔了**

### micro-task（微任务） 与 macro-task（宏任务）

Event Loop线程中包含任务队列（用来对不同优先级的异步事件进行排序），而任务队列又分为macro-task（宏任务）与micro-task（微任务），在最新标准中，它们被分别称为task与jobs。

* macro-task大概包括：script(整体代码), setTimeout, setInterval, setImmediate, I/O, UI rendering。
* micro-task大概包括: process.nextTick, Promise, Object.observe(已废弃), MutationObserver(html5新特性)
* setTimeout/Promise等我们称之为任务源。而进入任务队列的是他们指定的具体执行任务（回调函数）。

来自不同的任务源的任务会进入到不同的任务队列中，而不同的任务队列执行过程如下：  
**执行过程如下：**

**JavaScript引擎首先执行整个进程，先按照单线程的方式执行，**

**当遇到宏任务macro-task或者微任务micro-task的时候将任务加入列队中，**

**将宏任务macro-task按照先后顺序加入宏任务macro-task列队中；  
将微任务micro-task按照先后顺序加入微任务micro-task列队中；**

**当主线程执行完毕后；  
JavaScript引擎首先从微任务micro-task中取出第一个任务，**

**然后按照顺序将微任务micro-task列队中的微任务micro-task程序执行完毕；  
执行完毕后，再像执行微任务一样将宏任务macro-task列队中的所有任务取出，按顺序全部执行；**

**直到队列中的任务都取完。**

**(先进先出)**

举个大例子

**console.log("start");**

**var promise = new Promise((resolve) => {**

**console.log("promise start..");**

**resolve("promise");**

**}); //3**

**promise.then((val) => console.log(val));**

**setTimeout(()=>{console.log("setTime1")},0);**

**console.log("test end...")**

这里我们按顺序来分析。

##### 第一轮

1. 整体script代码作为一个宏任务进入主线程，运行console.log("start");。
2. 然后遇到Promises直接运行console.log("promise start..")。
3. 然后遇到promise.then，存入到micro-task队列中。
4. 然后遇到setTimeout,存入到macro-task队列中。
5. 于然后运行console.log("test end...");
6. 在这一轮中，主线程运行结束，运行微任务micro-task队列中的 promise.then，输出promise

##### 第二轮

1. 取出宏任务macro-task队列中的setTimeout，运行console.log("setTime1");

##### 结果

输出的顺序就是

// start

// promise start

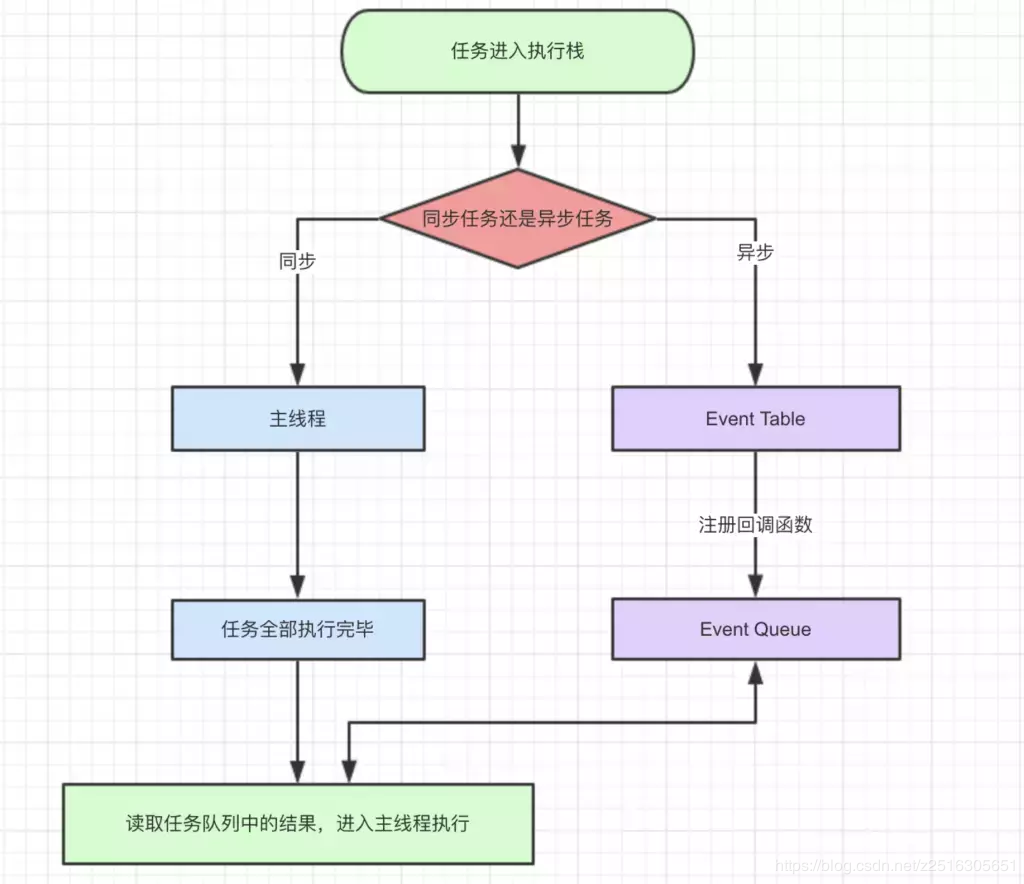
// test end...

// promise

//setTime1

主执行栈和任务队列

所有的任务可以分为同步任务和异步任务，同步任务，顾名思义，就是立即执行的任务，同步任务一般会直接进入到主线程中执行；而异步任务，就是异步执行的任务，比如ajax网络请求，setTimeout 定时函数等都属于异步任务，异步任务会通过任务队列( Event Queue )的机制来进行协调



同步和异步任务分别进入不同的执行环境，同步的进入主线程，即主执行栈，异步的进入 Event Queue 。主线程内的任务执行完毕为空，会去 Event Queue 读取对应的任务，推入主线程执行。 上述过程的不断重复就是我们说的 Event Loop (事件循环)。

宏任务与微任务

在执行过程中优先级：**主干代码 > 微任务 > 宏任务**

宏任务：setImmediate 、setTimeout/setInterval

微任务：nextTick、promise

从任务层面，在JS引擎中，我们可以按性质把任务分为两类，macrotask（宏任务）和 microtask（微任务）。

**宏任务**

(macro)task（又称之为宏任务），可以理解是每次执行栈执行的代码就是一个宏任务（包括每次从事件队列中获取一个事件回调并放到执行栈中执行）。

浏览器为了能够使得JS内部(macro)task与DOM任务能够有序的执行，会在一个(macro)task执行结束后，在下一个(macro)task 执行开始前，对页面进行重新渲染，流程如下：

(macro)task->渲染->(macro)task->...

1

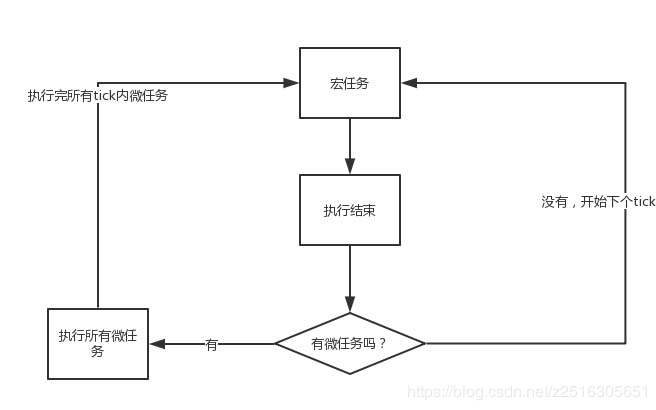
**微任务**

microtask（又称为微任务），可以理解是在当前 task 执行结束后立即执行的任务。也就是说，在 当前task任务后，下一个task之前，在渲染之前。所以它的响应速度相比setTimeout（setTimeout是task）会更快，因为无需等渲染。

也就是说，在某一个macrotask执行完后，就会将在它执行期间产生的所有microtask都执行完毕（在渲染前）。

microtask主要包含：Promise.then、MutaionObserver、process.nextTick(Node.js 环境)

它们的执行顺序如下：



在事件循环中，每进行一次循环操作称为tick，通过阅读规范可知，每一次 tick 的任务处理模型是比较复杂的，关键步骤总结如下：

执行一个宏任务（栈中没有就从事件队列中获取）

执行过程中如果遇到微任务，就将它添加到微任务的任务队列中

宏任务执行完毕后，立即执行当前微任务队列中的所有微任务（依次执行）

当前宏任务执行完毕，开始检查渲染，然后GUI线程接管渲染

渲染完毕后，JS线程继续接管，开始下一个宏任务（从事件队列中获取）

宏任务与微任务的优先级(浏览器)：

macrotask（按优先级顺序排列）: script(你的全部JS代码，“同步代码”）, setTimeout, setInterval, setImmediate, I/O,UI rendering

microtask（按优先级顺序排列）:process.nextTick,Promises（浏览器原生实现的 Promise）, Object.observe, MutationObserver（await后面的代码，Promise.then里面的代码也属于微任务）

JS引擎首先从macrotask queue中取出第一个任务，执行完毕后，将microtask queue中的所有任务取出，按顺序全部执行；

然后再从macrotask queue（宏任务队列）中取下一个，执行完毕后，再次将microtask queue（微任务队列）中的全部取出；

循环往复，直到两个queue中的任务都取完。

很多面试都遇到这段代码输出的顺序是什么这样的问题，下面就是这么一个简单的问题

