JavaScript执行环境是**单线程**的、非阻塞的，它是解释型、弱类型、动态脚本语言。

解释与编译：计算机不能直接理解高级语言，只能直接理解**机器语言**，所以必须要把高级语言翻译成机器语言，计算机才能执行高级语言编写的程序。一个是编译，一个是解释，一个需要编译器，一个需要解释器（Js引擎）。两种方式只是翻译的时间不同。编译型语言写的程序执行之前，需要一个专门的编译过程。解释则不同，解释性语言的程序不需要编译，省了道工序，解释性语言在运行程序的时候才翻译（即时编译）。这样解释性语言每执行一次就要翻译一次，效率比较低。解释是一句一句的翻译。

编译型——优点：执行效率高，缺点：跨平台性差

解释型——优点：跨平台性好，缺点：执行效率低

单线程决定了它同一时间只能做一件事情 如果有多件事件就必须排队，后面的事情必须等前面的事情完成后才能执行

为什么JavaScript是单线程：JavaScript作为浏览器脚本语言，主要用途是与用户互动，以及操作DOM。这就决定了它只能是单线程 ，否则会带来很多复杂的同步问题。为了避免复杂性，这个特性我相信将来也不会改变。

最新的HTML5提出了Web Worker标准，允许JS脚本创建多个线程，但是子线程完全受主线程控制，不能操作DOM。所以JS还是单线程。

线程和进程的关系：

进程：操作系统分配的占有CPU资源的最小单位。拥有独立的地址空间。

线程：安排CPU执行的最小单位。同一个进程下的所有线程，共享进程的地址空间。

简单讲，计算机就像工厂，进程是个大车间，计算机内部有很多个这样的大车间。线程是工人，每一个车间里的工人至少有一个。

关系：进程中包含着至少一个线程。在进程创建之初，就会包含一个线程，这个线程会根据需要，调用系统库函数去创建其他线程。但需要注意的是，这些线程之间是没有层级关系的，他们之间协同完成工作。在整个进程完成工作之后，其中的线程会被销毁，释放资源。

单线程：单线程就是一个进程中只有一个线程。程序顺序执行，前面的执行完，才会继续执行后面的程序。

多线程：多线程就是一个进程中只有多个线程。在进程内部进行线程间的切换，由于每个线程执行的时间片很短，所以在感觉上是并行的。

虽然JavaScript是单线程，但是浏览器是事件驱动的、异步的、多线程的。

浏览器至少有四个线程（不同浏览器会有差异）： js引擎线程、界面渲染线程、事件触发线程、http请求线程。

在JavaScript中 所有的任务可以分为两种：一种是同步任务（synchronous）另一种是异步任务（asynchronous）  
同步任务指: 在主线程上排队执行的任务，只有前一个任务执行完毕后，才能执行下一个任务  
异步任务指: 不进入主线程(不阻塞主线程)，而进入任务队列（task queue）的任务，**等主线程的任务全部执行完成后** 主线程会通过event loop(事件循环) 去询问任务队列中是否有可以被执行的任务了

如果有可以被执行的任务 **这个时候这个任务就会被放进主线程执行。**

下面是它们的运行机制：  
1.所有同步任务都在主线程上，形成一个执行栈  
2.主线程之外还有一个“任务队列”(也叫消息队列)，只要异步任务有了运行结果 ，就在任务队列中放一个事件（进入任务队列的是回调函数或事件）  
3.当执行栈中所有的任务执行完了，就去看看任务队列中有没有需要执行的事件 ，如果有的话，就结束它们的等待，进入执行栈 ，开始执行。  
4.主线程不断重读上面三步  
（这里还是单线程，只是多了一个任务队列）

Event Loop: 主线程从"任务队列"中读取事件，这个过程是循环不断的，所以整个的这种运行机制又称为Event Loop（事件循环）。这是计算机系统的一种运行机制。  
JavaScript语言就采用这种机制，来解决单线程运行带来的一些问题。JS会创建一个类似于while(true)的循环，每执行一次循环体的过程称为Tick，每次Tick的过程就是查看是否有待处理事件，如果有则取出**相关事件**及**回调函数**放入执行栈中由主线程执行，待处理的事件会存储在一个任务队列中，也就是每次Tick都会查看任务队列中是否有需要执行的任务；实际上事件循环是指主线程重复从消息队列取出消息、执行的过程。当消息队列为空时，就会等待直到消息队列变成非空，而且主线程只有在将当前的消息执行完成后，才会去取下一个消息。

宏任务与微任务执行机制：事件循环只有一个，但任务队列可能有多个，任务队列可分为宏任务（macro-task）和微任务（micro-task）。执行一个宏任务，过程中如果遇到微任务，就将其放到微任务的“任务队列”中；当前宏任务执行完成后，会查看微任务的“任务队列”，并将里面全部的微任务依次执行完。

宏任务（macro-task）：整体代码script、XHR回调、事件回调（鼠标键盘事件）、setImmediate、setTimeout、setInterval、indexedDB数据库操作等I/O、UI rendering；参与了事件循环的异步任务。

微任务（micro-task）：process.nextTick、Promise.then、Object.observer(已经被废弃)、MutationObserver(html5新特性)；没有参与事件循环的“异步”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。

回调函数(callback)：就是那些会被主线程挂起来的代码，前面所说的点击事件$(selectot).click(function)中的function就是一个回调函数，异步任务必须指定回调函数，当主线程开始执行异步任务，就是执行对应的回调函数，例如ajax的success，complete，error也都指定了各自的回调函数，这些函数就会加入任务队列中，等待执行；

setTimeout/ setinterva：setTimeout是延时将事件插入了"任务队列"，setinterva是定时将事件插入了"任务队列"，但都必须等主线程（执行栈）执行完，才去执行它指定的回调函数。要是当前代码耗时很长，有可能要等很久，回调函数不一定会在它们指定的时间执行。setTimeout/setinterva类似异步，但不是异步 。

