JavaScript是单线程

所有任务可以分成两种，一种是同步任务（synchronous），另一种是异步任务（asynchronous）。同步任务指的是，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；

异步任务指的是，不进入主线程、而进入"任务队列"（task queue）的任务，只有"任务队列"通知主线程，某个异步任务可以执行了，该任务才会进入主线程执行

return

整个 JavaScript 并不会停止执行，仅仅是函数停止。JavaScript 将继续执行代码，从调用函数的地方

具体来说，异步执行的运行机制如下。

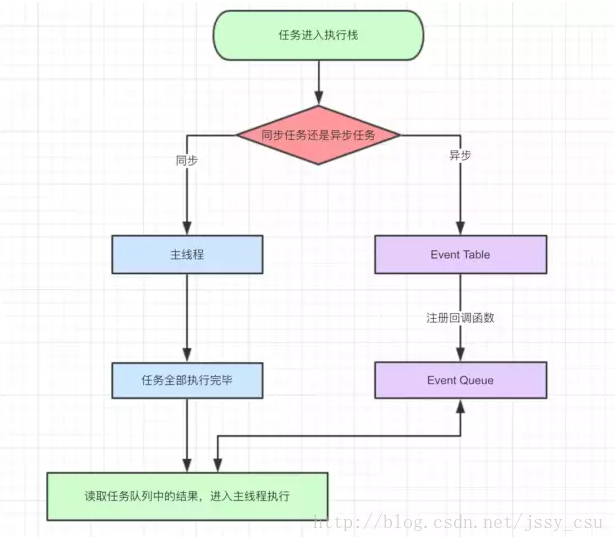
（同步执行也是如此，因为它可以被视为没有异步任务的异步执行。）

（1）所有同步任务都在主线程上执行，形成一个执行栈（execution context stack）。

（2）主线程之外，还存在一个"任务队列"（task queue）。只要异步任务有了运行结果，就在"任务队列"之中放置一个事件。

（3）一旦"执行栈"中的所有同步任务执行完毕，系统就会读取"任务队列"，看看里面有哪些事件。那些对应的异步任务，于是结束等待状态，进入执行栈，开始执行。

（4）主线程不断重复上面的第三步。



同步和异步任务分别进入不同的执行”场所”，同步的进入主线程，异步的进入Event Table并注册函数。

当指定的事情完成时，Event Table会将这个函数移入Event Queue。

主线程内的任务执行完毕为空，会去Event Queue读取对应的函数，进入主线程执行。

上述过程会不断重复，也就是常说的Event Loop(事件循环)。

let data = [];

$.ajax({

url:www.javascript.com,

data:data,

success:() => {

console.log('发送成功!');

}

})

console.log('代码执行结束');

上面是一段简易的ajax请求代码：

ajax进入Event Table，注册回调函数success。

执行console.log(‘代码执行结束’)。

ajax事件完成，回调函数success进入Event Queue

主线程从Event Queue读取回调函数success并执行。

回调函数就是一个参数，将这个函数作为参数传到另一个函数里面，当那个函数执行完之后，再执行传进去的这个函数。这个过程就叫做回调

主函数的事先干完，回头再调用传进来的那个函数

主函数不用等待回调函数执行完，可以接着执行自己的代码。所以一般回调函数都用在耗时操作上面

**setTimeout(fn,0)的意思就是不用再等多少秒了，只要主线程执行栈内的同步任务全部执行完成，栈为空就马上执行**