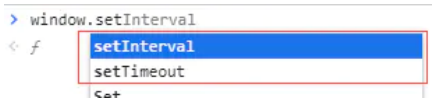
一篇深度理解setInterval和setTimeout以及JS执行机制

jCodeLife
0.164 2020.08.16 12:20:40 字数 2,460 阅读 23

编辑文章

setInterval和setTimeout是定义在window上的两个函数



这两个函数接受两个参数：

1. 第一个参数：接受一个回调函数 callback
2. 第二个参数：表示推迟执行的毫秒数。 time

```
//每隔100毫秒输出一个1
setInterval(function(){
  console.log(1);
},100);
//100毫秒后输出1，仅输出一次
setTimeout(function(){
  console.log(1);
},100);
```

不同的是：

1. setTimeout表示定时器，在指定毫秒数后执行回调函数，仅执行一次；而setInterval表示定时循环器，每间隔指定毫秒数就执行一遍回调函数，若是计时器不被清除，回调函数会执行无数遍
2. 清除定时器时，setInterval对应clearInterval()，setTimeout对应clearTimeout()

clearInterval和clearTimeout也都是定义在window上的函数，接受一个参数表示定时器的名字，根据这个名字清除对应的定时器

使用定时器时注意几点：

1. 若在创建定时器时没有名字，则定时器无法清除；
2. 定时器定义在全局对象window上，内部函数this指向的window；
3. setInterval里面传递的毫秒数只会在第一次的时候识别，之后不能改了
4. **setTimeout，setInterval是异步任务**

有些同学可能还不知道异步任务是什么？

其实要深入理解异步，还得理解js的执行机制
我们先来看个例子吧

```
//下面代码的执行结果是什么
var p = new Promise(resolve => {
    console.log(4);
    resolve(5);
});
function func1() {
    console.log(1)
}
function func2() {
    setTimeout(() => {
        console.log(2)
    });
```

推荐阅读

Vue干货小技巧——学会再也不做加班狗
阅读 1,314

超级基础却又超级容易出错的前端面试题（1）
阅读 89

学会这6个强大的CSS选择器，将真正帮你写出干净的CSS代码！
阅读 258

面试了几个前端，给爷整哭了！
阅读 29,589

vue优化页面
阅读 818

```
        console.log(6)
    });
}
func2();
```

答案：4 1 3 5 6 2

不知道你答对了没？

如果答对了，说明你对JS执行机制有较深的了解，继续往下看可以复习和巩固；如果不知道或者答错了，也别伤心，本文下面部分会给你答案

我们知道js的一大特点就是单线程，同一时间只能做一件事情。但是为什么要设计成单线程呢？

其实跟它的用途有关，js作为浏览器的脚本语言，主要用途就是于用户互动和操作DOM。想想如果被设计成多线程，一个线程在某个DOM节点上添加内容，另外一个线程在删除这个节点，那浏览器该听谁的？

所以，为了避免多线程带来的复杂问题，从设计之初就决定了JS就是单线程，这称为这门语言的核心特性，将来也不会改变。

为了利用多核CUP的计算能力，HTML5提出了Web Worker标准，允许JS 创建多个线程，但是子线程完全受主线程的控制，且子线程不能操作DOM。所以，这个新标准并没有改变js单线程的本质。

单线程就意味着，所有任务需要排队，前一个任务结束，才会执行后一个任务。如果前一个任务耗时很长，后一个任务就不得不一直等着。但是如果有些任务很慢时（比如Ajax操作从网络读取数据），我还是要等结果在执行后一个任务吗？这样不好吧

于是，有了一种异步任务。

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

所以其实js多线程的实现就是通过异步的方式来实现的

运行机制如下：

（1）所有同步任务都在主线程上执行，形成一个执行栈（Call 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。

执行栈用于组织JS代码，保障JS代码的有序执行。每当调用一个函数时，都会将该函数压入执行栈中，执行完弹出，接着调用下一个函数

第二步中，异步任务的运行结果其实背后借助了浏览器的其他线程

浏览器内核常驻的线程：

1. js引擎线程

   用于解释执行js代码、用户输入、网络请求等

2. GUI渲染线程

推荐阅读

Vue干货小技巧——学会再也不做加班狗
阅读 1,314

超级基础却又超级容易出错的前端面试题（1）
阅读 89

学会这6个强大的CSS选择器，将真正帮你写出干净的CSS代码！
阅读 258

面试了几个前端，给爷整哭了！
阅读 29,589

vue优化页面
阅读 818

4. 定时触发器线程
   setInterval、setTimeout等待时间结束后，会把执行函数推入任务队列中
5. 浏览器事件处理线程
   将click、mouse等交互事件发生后，将要执行的回调函数放入到事件队列中

任务队列"是一个先进先出的数据结构，排在前面的事件，优先被主线程读取。
主线程空了，才会再去读取"任务队列"，但是任务队列在不同的宿主环境中有所差异，大部分宿主环境会将任务队列分成macrotask（宏任务） 和 microtask（微任务）

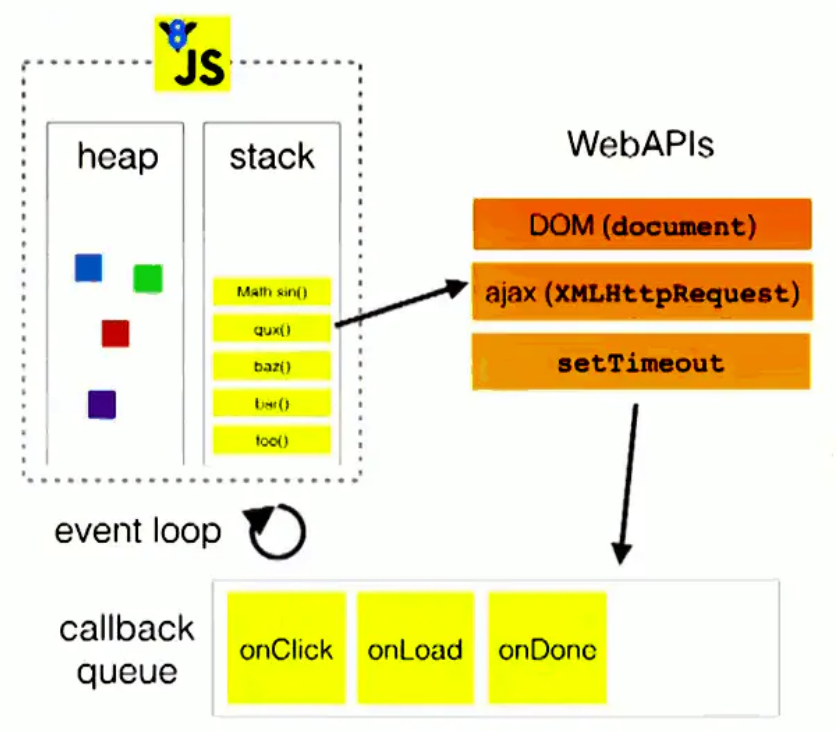
宏任务主要包含：script(整体代码)、setTimeout、setInterval、I/O、UI 交互事件、setImmediate(Node.js 环境)

微任务主要包含：Promise then、async await、MutaionObserver、process.nextTick(Node.js 环境)

当执行栈清空时，JS引擎首先会将微任务中的所以任务依次执行结束，如果没有微任务了，则执行宏任务。

"主线程的读取过程基本上是自动的，只要执行栈一清空，"任务队列"上第一位的事件就自动进入主线程。但是定时器首先要检查是否到了执行时间，到了规定的时间，才进入主线程执行，执行完再去任务队列中读取下一个事件，这个过程是不断循环的，我们把这种循环的机制称为Event Loop（事件循环），即

主线程运行的时候，产生堆（heap）和栈（stack），栈中的代码调用各种外部API(即各种函数)，它们在"任务队列"中加入各种事件（click，load，done）。只要栈中的代码执行完毕，主线程就会去读取"任务队列"，依次执行那些事件所对应的回调函数。



再来看定时器
如果将setTimeout()的第二个参数设为0，表示当执行栈清空以后，立即执行（0毫秒间隔）指定的回调函数。

```
setTimeout(function(){console.log(1);}, 0);
console.log(2);
```

行一次。

需要注意的是，setTimeout()只是将事件插入了"任务队列"，必须等到当前代码（执行栈以及排在前面的微任务）执行完，主线程才会去执行定时器中指定的回调函数。要是当前代码耗时很长，有可能要等很久，所以并没有办法保证，回调函数一定会在setTimeout()指定的时间执行。

最后回来看看上面的那道题目：

```
//下面代码的执行结果是什么
var p = new Promise(resolve => {
    console.log(4);
    resolve(5);
});
function func1() {
    console.log(1)
}
function func2() {
    setTimeout(() => {
        console.log(2)
    });
    func1();
    console.log(3);
    p.then(resolved => {
        console.log(resolved)
    }).then(() => {
        console.log(6)
    });
}
func2();
```

执行

第一步：new Promis压放入执行栈中，然后执行里面的代码，打印4，执行resolve(5);注意这里跟new普通函数一样是正常执行的，不会加入到宏任务中；

第二步：执行func2()，

遇到setTimeout，将它放入宏任务中；

接着执行func1();打印出1

接着console.log(3);打印3

遇到Promise对象执行then()时，这里是异步操作，会将里面回调函数放入微任务中，等待执行

当执行栈被清空时，执行微任务中的console.log(resolved)，打印出5，接着再去微任务中找到下个事件，打印出6；

当微任务清空后，再执行宏任务，即setTimeout到时间后会答应出2，

所以最后答案为：`4 1 3 5 6 2`

参考资料：

深入理解定时器系列第一篇——理解setTimeout和setInterval
setTimeout和setInterval的深入理解
UI多线程-深入剖析Js执行机制
深入理解JavaScript事件循环机制
2分钟了解 JavaScript Event Loop
JavaScript微任务与宏任务、异步、事件循环与消息队列理解
深度剖析JavaScript事件循环机制（未完善）
为什么javascript是单线程？
JavaScript 运行机制详解：再谈Event Loop—作者：阮一峰
浏览器内核常驻线程

还没有人赞赏，支持一下

jCodeLife 书山有路勤为径，学海无涯苦作舟
总资产14 (约0.99元) 共写了12.3W字 获得231个赞 共23个粉丝

写下你的评论...

全部评论 0 只看作者 关闭评论 按时间倒序 按时间正序

被以下专题收入，发现更多相似内容 投稿管理

+收入我的专题 领跑计划笔记

推荐阅读

Vue干货小技巧——学会再也不做加班狗
阅读 1,314

超级基础却又超级容易出错的前端面试题（1）
阅读 89

学会这6个强大的CSS选择器，将真正帮你写出干净的CSS代码！
阅读 258

面试了几个前端，给爷整哭了！
阅读 29,589

vue优化页面
阅读 818