定时器

JavaScript提供定时执行代码的功能，叫做定时器（timer），主要由setTimeout()和setInterval()这两个函数来完成。

setTimeout()

setTimeout函数用来指定某个函数或某段代码，在多少毫秒之后执行。它返回一个整数，表示定时器的编号，以后可以用来取消这个定时器。

```
var timerId = setTimeout(func|code, delay)
```

上面代码中，setTimeout函数接受两个参数，第一个参数`func|code`是将要推迟执行的函数名或者一段代码，第二个参数`delay`是推迟执行的毫秒数。

```
console.log(1);
setTimeout('console.log(2)',1000);
console.log(3);
```

上面代码的输出结果就是1，3，2，因为setTimeout指定第二行语句推迟1000毫秒再执行。

需要注意的是，推迟执行的代码必须以字符串的形式，放入setTimeout，因为引擎内部使用eval函数，将字符串转为代码。如果推迟执行的是函数，则可以直接将函数名，放入setTimeout。一方面eval函数有安全顾虑，另一方面为了便于JavaScript引擎优化代码，setTimeout方法一般总是采用函数名的形式，就像下面这样。

```
function f(){
  console.log(2);
}

setTimeout(f,1000);

// 或者

setTimeout(function (){console.log(2)},1000);
```

setInterval()

setInterval函数的用法与setTimeout完全一致，区别仅仅在于setInterval指定某个任务每隔一段时间就执行一次，也就是无限次的定时执行。

```
var i = 1
var timer = setInterval(function() {
  console.log(i++);
}, 1000);
```

上面代码表示每隔1000毫秒就输出一个2，直到用户点击了停止按钮。

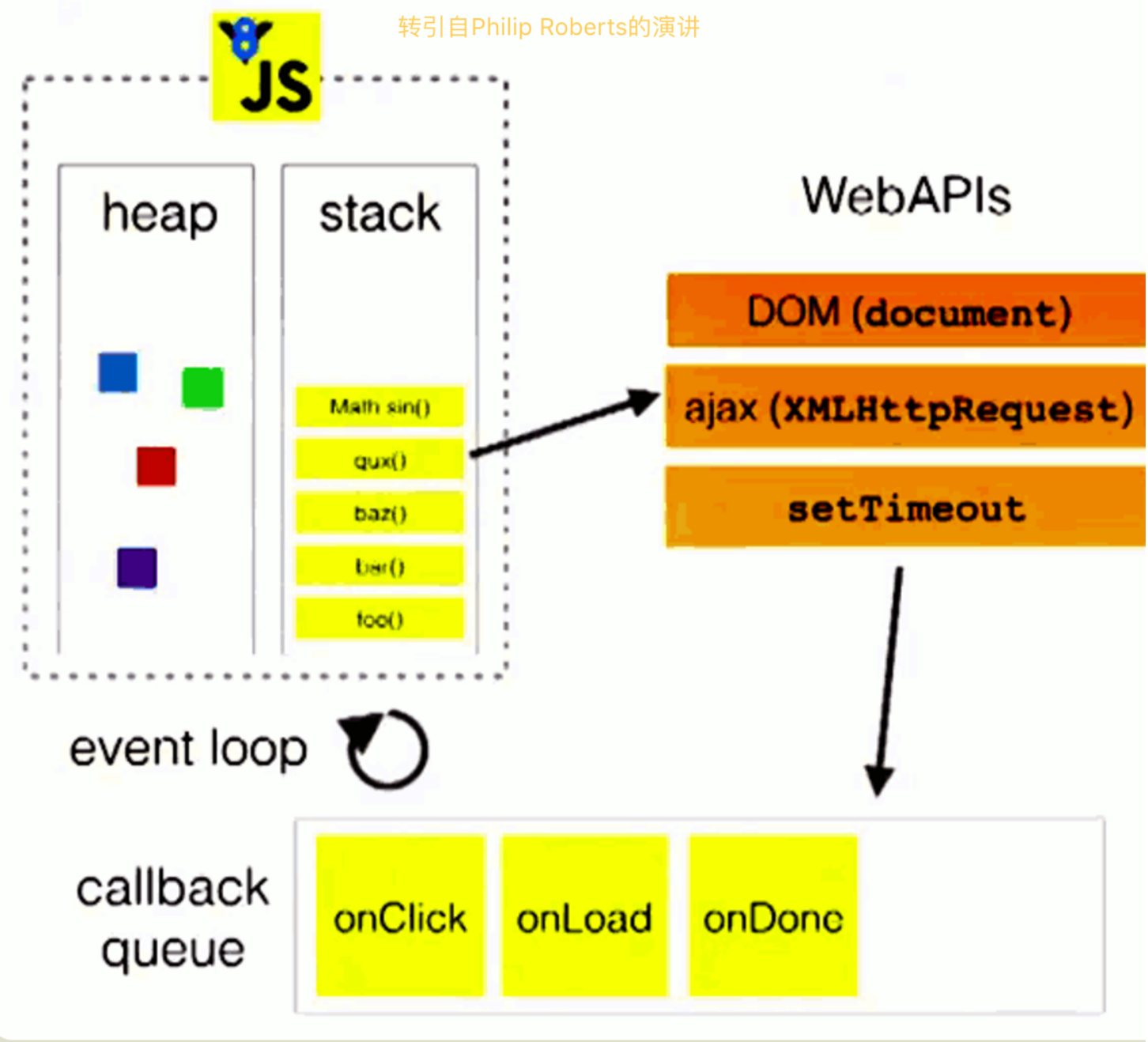
clearTimeout()，clearInterval()

setTimeout和setInterval函数，都返回一个表示计数器编号的整数值，将该整数传入clearTimeout和clearInterval函数，就可以取消对应的定时器。

```
var id1 = setTimeout(f,1000);
var id2 = setInterval(f,1000);

clearTimeout(id1);
clearInterval(id2);
```

单线程模型



参考文章

运行机制

setTimeout和setInterval的运行机制是，将指定的代码移出本次执行，等到下一轮Event Loop时，再检查是否到了指定时间。如果到了，就执行对应的代码；如果不到，就等到再下一轮Event Loop时重新判断。这意味着，setTimeout指定的代码，必须等到本次执行的所有代码都执行完，才会执行。

setTimeout的作用是将代码推迟到指定时间执行，如果指定时间为0，即setTimeout(f,0)，那么不会立刻执行

setTimeout(f,0)将第二个参数设为0，作用是让f在现有的任务（脚本的同步任务和“任务队列”中已有的事件）一结束就立刻执行。也就是说，setTimeout(f,0)的作用是，尽可能早地执行指定的任务。

测试题1：以下代码输出什么

```
var i=0;
for(var i=0; i<10; i++){
  setTimeout(function(){
      console.log(i)
  }, 1000)
}
```

测试题2：以下代码输出什么

```
var t = true;
setTimeout(function(){
  t = false;
}, 1000);

while(t){ }
console.log('end')
```

异步与回调

```
function f1(callback){
    setTimeout(function(){
        //做某件事，可能很久
        console.log('别急，开始执行f1')
        for(var i=0;i< 100000;i++){

        }
        console.log('f1执行完了')

        callback()
    }, 0);

}
function f2(){
    console.log('执行f2');
}
function f3(){
    console.log('执行f3');
}
f1(f2) //当f1执行完之后再执行 f2
f3()
```

函数节流

```
  var timer
function hiFrequency(){
    if(timer){
        clearTimeout(timer)
    }
      timer = setTimeout(function(){
          console.log('do something')
      }, 300)
}

hiFrequency()
hiFrequency()
hiFrequency()
```

改造

```
function throttle(fn, delay) {
    var timer = null
    return function(){
        clearTimeout(timer)
        timer = setTimeout(function(){
            fn(arguments)
        }, delay)
    }
}

function fn(){
    console.log('hello ')
}

var fn2 = throttle(fn, 1000)
fn2()
fn2()
fn2()
```