探究点击事件在JavaScript事件循环中的表现

> 链接：https://github.com/liusaint/ls-blog/issues/20

JavaScript的事件循环event loop很多文章都写的非常详细了。这里也不多做介绍。 在很多文章中都有介绍鼠标事件Mouse Event是属于macrotask。本文探究一下Mouse Event在event loop的情况。点击事件是同步还是异步？点击事件何时加入事件队列？点击事件加入事件队列的是什么，是回调函数吗？

加入事件队列的两种时机

- 工作线程空闲时。 加入后瞬间执行。感觉不到等待。属于比较简单的情况，不会跟其他的任务打堆。
- 工作线程繁忙。加入事件队列后需等待执行。如果有多个事件加入队列，会有一个排序。

点击事件是异步还是同步?

- 代码模拟点击时的情况：

```
var btn = document.querySelector('button');
btn.onclick = function(){ console.log('1') }
btn.click();
console.log('2');
```

输出：

```
1
2
```

结论：模拟点击而非真实点击的时候，事件处理是同步操作。

- 真实鼠标点击的情况：

首先，我们用鼠标点的，根据常识，应该是异步的。不过我们依然要验证一下。 我们在线程繁忙的时候发起多次点击，查看执行顺序。 promise.then是属于microtask，事件循环中执行完一个macrotask会把所有的microtask执行一下。所以我们在线程繁忙的时候点击，在点击事件的回调中加入promise.then。如果执行了promise.then中的内容说明执行完了一次事件循环。

```
var btn = document.querySelector('button');
var promise = new Promise(function(resolve){
    resolve();
})
btn.onclick = function(){

    console.log('click');

    //页面失去响应2s。
    var time = new Date().getTime();
    while(new Date().getTime()-time < 2000){};
```

```
    promise.then(function(){
        console.log('then');
    })

}
```

连续点三次，输出：

```
click
then
click
then
click
then
```

分析： 一次点击->执行click回调->页面卡顿时，线程繁忙连续点击两次->加入promise.then->执行promise.then()->执行下个循环，click->加入promise.then->执行promise.then->执行下个循环，click->加入promise.then->执行promise.then。 如果三次点击是同步，应该先输出三次click,再输出三次promise。

结论：手动点击是异步的。

点击事件加入事件队列的时机

是我们点击的时间吗？我们来间接验证一下。setTimeout在事件循环中属于macrotask，跟点击事件是平级的。我们在点击事件的回调中中插入setTimeout，查看执行顺序，根据执行顺序我们可以推测加入事件循环的顺序。

```
        var btn = document.querySelector('button');
        btn.onclick = function(){

            console.log('click');
            //页面失去响应2s。
            var time = new Date().getTime();
            while(new Date().getTime()-time < 2000){};
            setTimeout(function(){
                console.log('timeout');
            })

        }
```

快速点击三次按钮，输出：

```
click *3
timeout *3
```

分析：线程空闲时点击第一次（加入事件队列）->执行click->线程繁忙->点击2次（加入2个点击事件到事件队列）->加入第一个setTimeout到队列-> 执行第二次点击的回调->加入第二个setTimeout到队列->执行第三次点击的回调->加入第三个setTimeout到队列->执行第一个setTimeout->执行第二个setTimeout->执行第三个setTimeout

结论：加入事件循环的时机是我们点击的时机。

点击事件加入事件队列的是什么，是事件的回调函数吗？

一开始我是这样认为的。既然加入了队列，那么肯定是回调函数咯。结果发现并不是。

如果加入事件队列的是回调函数，会发生什么?

之前一个项目中，测试说狂点保存会保存多次。那个按钮的防重复点击使用的是遮罩，也就是一点按钮，发ajax请求提交，就会出现一个遮罩把按钮挡住，就无法再次点击按钮了。 测试用的的4g内存的笔记本的IE9,那个页面从点击保存按钮到从页面中提取数据，到打开遮罩居然要好几秒。在打开遮罩之前页面一直处理无响应的状态，但是测试可以继续点按钮。我猜想是因为点击事件的回调加入了事件队列，所以后面会执行多次请求。 然后按这个思路把bug解决了。

后来有一天一个偶然的尝试，我发现上次那个bug解决的思路似乎不对，虽然bug也确实是解决了。。。 那个表单提交，点提交之后有一个异步操作，因为用了require.js,在异步操作里有耗时操作获取数据，然后才是打开遮罩，发起ajax请求。 简化场景：

```
    var btn = document.querySelector('button');

    btn.onclick = function(){

        console.log('click');

        setTimeout(function(){
            //页面失去响应2s。
            var time = new Date().getTime();
            while(new Date().getTime()-time < 2000){};
            btn.style.display = 'none';//遮罩，用隐藏按钮代替
            console.log('timeout');
        })
    }
```

连续点击三次，输出：

```
click
timeout
```

都只输出了一次。后面两次点击的时候页面处于无响应状态。但是按钮是存在的，并且代码也并没有执行到隐藏按钮那一句去。 由前面的推论，加入事件循环的时机应该是点击的时候。如果回调函数直接加入到事件队列中去了，那么后面还会有四次输出的。

分析：页面空闲->点击按钮（点击加入事件循环）->执行click->加入setTimeout->执行setTimeout->页面卡顿，线程繁忙，第二次点击（点击加入事件循环队列）->页面卡顿，线程繁忙，第三次点击（点击加入事件循环队列）->卡顿结束，按钮被隐藏->延迟点击第二次->延迟点击第三次 **推论**：加入事件队列的是点击本身，而不是回调。当这个点击事件进入队列的时候，会尝试在该地方去点一下，然后再看该地方的元素有没回调什么的。相当于把整个点击都延迟了。

顺带提一下那个bug之所以会提交多次，是因为那个页面进去的瞬间会加载很多富文本编辑器之类的，执行很多初始化代码，所以页面本身卡的时候点击那个本身带异步函数的按钮，就会出现这个异步操作被多次加入队列的情况。

再次验证：

对上面的结论进行再次验证。我们点击之后，用代码移动滚动条，把第二个按钮移动到第一个按钮我们点击的位置下面。

```
<body>
    <button class="a">打开控制台，快速点击三次</button>
    <br/>
    <button class="b" style="height: 1000px;">按钮b</button>
    <script>

        var btn = document.querySelector('.a');
        btn.onclick = function(){
            console.log('click a');
            setTimeout(function(){
                //页面失去响应2s。
                var time = new Date().getTime();
                while(new Date().getTime()-time < 2000){};
                window.scrollTo(100,500);//滚动到第二个按钮上。
                console.log('timeout');
            })

        }

        var btn1 = document.querySelector('.b');
        btn1.onclick = function(){
            console.log('click b');
        }
    </script>
</body>
```

连续点击a按钮三次，输出：

```
click a
timeout
click b * 2
```

我们点的时候虽然是在a按钮上点的，效果却成了点击b按钮的效果。相当于整个点击延迟了。跟我们上面的结论吻合。 **分析**：页面空闲->点击按钮a（点击加入事件循环）->执行click->加入setTimeout->执行setTimeout->页面卡顿，线程繁忙，第二次点击a（点击加入事件循环队列）->页面卡顿，线程繁忙，第三次点击a（点击加入事件循环队列）->卡顿结束，页面滚动，把按钮b移动到前三次点击的位置->延迟点击第二次，点到b->执行b的click回调->延迟点击第三次，点到b按钮->执行b的click回调。

结论

页面工作线程繁忙时点击按钮，点击事件会在点击页面的时候加入事件循环队列。 加入事件队列的并不是事件的回调函数，而是单纯的点击本身，可能包含了本次点击的位置信息等， 当事件循环轮到执行该事件循环的时候，根据记录的信息延迟点击， 这个时候可能点击到的已经不是鼠标点击时点击的按钮。如果按钮被遮住或隐藏，该按钮是点不到的，但是点击位置出现了其他按钮（移动过来一个按钮），也会执行其他按钮的回调。

最后，以上都是推测

都是我根据现象和已知信息进行的推论，我在网上并没有找到相关文献以及理论依据。 查了很久的资料都没有看到相关的。 如果有不对的地方，希望不吝赐教。