我对event-loop的理解

使用javascript（以后简称js）已经有两年了，现在已经开始学习nodejs。但是一直以来对于js的运行机制都没有一个清晰的概念，最近花了两天的时间研究了一下其event-loop机制，特来和大家分享一下我对它的理解。

**Javascript特点是单线程**

想了很久，觉得还是应该从线程说起。js的一大特点是单线程，我们都知道，对于单线程的程序来说，如果要处理多个任务，那么这些任务就需要排队等待处理。如果一个任务处理时间较长，那其它的任务等待的时间就会延长。对于程序来说，比较耗时间的是I/O的处理，在I/O处理过程中，CPU是闲置的，这样对CPU来说是一种资源的浪费。因此javascript的设计者将其运行机制设计为：当I/O设备在运行的时候，主线程会将等待I/O的任务先挂起，接着去执行后面等待的任务，当I/O设备返回结果的时候，主线程再去执行先前被挂起的任务。

因此js设计者使用了“event loop”实现了以上的机制。

**event loop 原理**

event loop 翻译成中文就是“事件循环”，其中有个关键词就是事件。对于js程序来说，系统会为其分配两块空间，一块儿事执行栈——供主线程执行同步的程序，另一块儿是事件队列——用于存放异步事件。

事件是通过回调函数来实现的，每一个事件都会对应一个回调函数，在事件队列里保存的事件信息就是回调函数的信息。首先主线程会先去实现执行栈里的程序，当执行栈里的程序都执行完以后，再去事件队列里查询可执行的事件，当有的事件得到返回结果以后会通知主线程，然后主线程会取出队列中相应的回调函数到执行栈中执行。因为主线程是总是循环不断的去事件队列里取事件，因此取名为“事件循环”。

下面我们来看一个例子

var Ajaxreq = **new** XMLHttpRequest();

Ajaxreq.open(‘get’,url);

Document.getElementById(“id”).onclick=function(){}

Ajaxreq.onreadystatechange=**function**(){};

Ajaxreq.send();

Document.getElementById(“id2”).write();

执行栈

Write()

事件队列

send()

Event loop

onclick

onreadystatechange

open()

XMLHttpRequest()

上例中有两个事件，一个是ajax，对应回调函数为 onreadystatechange = function() 另一个是onclick事件。按照上面我们讲述的event loop的运行机制，我们可以分析一下这个例子。

首先主线程会去实例化一个ajax对象，然后以get的方式请求服务器（open()和send()），然后执行write（）函数。当这些都执行完以后，主线程会去检测事件队列里的 onreadystatechange 和 onclick 是否可执行。我们知道onclick被触发的诱因是鼠标点击，如果鼠标不点击，那么会一直被监听而不执行。而onreadystatechange的触发诱因是服务端有返回结果，那么此事件的回调函数才会被执行。所以说主线程在执行完执行栈里的程序以后会一直去监听事件队列里的事件，当服务端返回结果的时候，onreadystatechange的回调函数被调进执行栈执行，执行完以后再回去监听事件队列，当有鼠标点击的时候（当然是在相应元素上面点击），onclick的回调函数会被调进执行栈执行。

所以说上述例子，不论onreadystatechange和onclick写在什么位置，执行结果都是相同的，也就是说等同于下述代码

var Ajaxreq = **new** XMLHttpRequest();

Document.getElementById(“id”).onclick=function(){}

Ajaxreq.onreadystatechange=**function**(){};

Ajaxreq.open(‘get’,url);

Ajaxreq.send();

Document.getElementById(“id2”).write();

在这里需要注意的是，在队列中的各个事件之间是没有先后顺序的，哪个事件先有结果，那个事件的回调函数先被执行。

**Nodejs中的event loop**

根据event loop的原理，我们回到刚开始提到的I/O问题上面。上面说过I/O是很花费事件的，所以说js会将操作I/O的事件置于事件队列中，主线程会继续执行其他的程序，当I/O设备有结果的时候，该事件才会被重新调用，这样就充分利用了CPU资源。当然，对于前端js来说，I/O操作用到的还是少的，不过前端有花费事件比较长的是向服务端发请求，所以出现了ajax 技术解决了这一问题。对于I/O操作在nodejs中用的比较多，对于mysql，sqlserver等数据库的读写需要用到I/O，因为nodejs也是单线程的，所以同样也是采用event loop的机制。举个nodejs操作mysql的例子

**var *ms*** = require('mysql');

**var *mscon*** = ***ms***.createConnection(***“数据库链接信息”***);

**var** query = ***mscon***.query("INSERT INTO 表名 SET ?",  
  
 {  
 **message**:”**message”**,  
 **id**:**id**,  
 **userid**:用户id,  
 **addtime**:时间戳,  
 },  
 **function**(errs,result){  
 **if**(errs) **throw** errs;

console.log(“操作成功”);

}

);

同样是采用了事件回调函数。因此，如果部署的好的话，nodejs几乎不会出现堵塞的现象。

以上就是我对javascript中event loop机制的理解，不知道准不准确，如果什么地方有错误，还请大家指出，大家共同讨论，共同提高。