在实际的使用当中，有非常多的应用场景我们不能立即知道应该如何继续往下执行。最重要也是最主要的一个场景就是ajax请求。通俗来说，由于网速的不同，可能你得到返回值的时间也是不同的，这个时候我们就需要等待，结果出来了之后才知道怎么样继续下去。

//简单的原生ajax实现方式

var url = 'https://hq.tigerbrokers.com/fundamental/finance\_calendar/’;   
var result;  
  
var XHR = new XMLHttpRequest();  
XHR.open('GET', url, true);  
XHR.send();  
  
XHR.onreadystatechange = function() {  
 if (XHR.readyState == 4 && XHR.status == 200) {  
 result = XHR.response;  
 console.log(result);  
 }  
}

在ajax的原生实现中，利用了onreadystatechange事件，当该事件触发并且符合一定条件时，才能拿到我们想要的数据，之后我们才能开始处理数据。

但是如果这个时候，我们还需要做另外一个ajax请求，这个新的ajax请求的其中一个参数，得从上一个ajax请求中获取，这个时候我们就不得不如下这样做：

var url = 'https://hq.tigerbrokers.com/fundamental/finance\_calendar';  
var result;  
  
var XHR = new XMLHttpRequest();  
XHR.open('GET', url, true);  
XHR.send();  
  
XHR.onreadystatechange = function() {  
 if (XHR.readyState == 4 && XHR.status == 200) {  
 result = XHR.response;  
 console.log(result);  
  
 // 伪代码  
 var url2 = 'http:xxx.yyy.com/zzz?ddd=' + result.someParams;  
 var XHR2 = new XMLHttpRequest();  
 XHR2.open('GET', url, true);  
 XHR2.send();  
 XHR2.onreadystatechange = function() {  
 ...  
 }  
 }  
}

当出现三个甚至更多ajax请求依赖上一个请求时，我们的代码就陷入了无限回调之中。

这个时候我们使用promise来解决这个问题。

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

控制方法执行顺序：当我们想要确保某代码在谁谁之后执行时，我们可以利用函数调用栈，将我们想要执行的代码放入回调函数中。

// 一个简单的封装  
 function want() {  
 console.log('这是你想要执行的代码');  
 }  
   
 function fn(want) {  
 console.log('这里表示执行了一大堆各种代码');  
   
 // 其他代码执行完毕，最后执行回调函数  
 want && want();  
 }  
  
 fn(want);

确保我们想要的代码压后执行，除了利用函数调用栈的执行顺序之外，我们还可以利用队列机制。

function want() {  
 console.log('这是你想要执行的代码');  
}  
  
function fn(want) {  
// 将想要执行的代码放入队列中，根据事件循环的机制，我们就不用非得将它放到最后面了，由你自由选择  
 want && setTimeout(want, 0);  
 console.log('这里表示执行了一大堆各种代码');  
}  
  
fn(want);

另外一种如果浏览器支持promise队列的话我们就可以把他放在promise队列中进行处理

function want() {  
 console.log('这是你想要执行的代码');  
}  
  
function fn(want) {  
 console.log('这里表示执行了一大堆各种代码');  
  
 // 返回Promise对象  
 return new Promise(function(resolve, reject) {  
 if (typeof want == 'function') {  
 resolve(want);  
 } else {  
 reject('TypeError: '+ want +'不是一个函数')  
 }  
 })  
}  
  
fn(want).then(function(want) {  
 want();  
})  
  
fn('1234').catch(function(err) {  
 console.log(err);  
})

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

使用promise对ajax进行简单的封装

var url = 'https://hq.tigerbrokers.com/fundamental/finance\_calendar';

// 封装一个get请求的方法

function getJSON(url) {

return new Promise(function(resolve, reject) {

var XHR = new XMLHttpRequest();

XHR.open('GET', url, true);

XHR.send();

XHR.onreadystatechange = function() {

if (XHR.readyState == 4) {

if (XHR.status == 200) {

try {

var response = JSON.parse(XHR.responseText);

resolve(response);

} catch (e) {

reject(e);

}

} else {

reject(new Error(XHR.statusText));

}

}

}

})

}

getJSON(url).then(resp => console.log(resp));