Promise是一个对象，是一个容器，用于异步计算，表示未来才会结束的一件事的结果，它可以将异步操作队列化，按照期望的顺序执行，返回预期的结果。

Promise产生的原因？  
JS里面包含大量的异步操作。异步主要包括两方面，一是事件监听与响应，比如监听onclick事件。二是回调，先调用外部函数，然后传入一个回调函数，外部函数执行完了就会调用传入的函数。

//事件监听

function start(){

//dosomething

}

$("#click").on('click',start);

//回调

$.ajax('http://www.baidu.com',{

success:function(res){

//dosomething

}

})

1. 稍有不慎，就有可能陷入回调地域（callback hell），全都是括号，代码后期不好维护。
2. 更深层次的是，我们在回调中不能正常使用return和throw。
3. 甚至可以说它剥夺了我们的堆栈。
4. function findLargest(dir,callback){//callback是最外层的调用
5. fs.readdir(dir,function(err,files){
6. if(err) return callback(err);//这里没有办法正常的使用try-catch
7. })
8. }

因为function(err,files)是当读dir完成的时候再调用，他和fs.readdir不在同那个一个执行栈，执行到fs.readdir完成时，它会退栈，它会认为我的事已经干完了，然后findLargest退栈，系统会再调用function(err,files)，这是一个全新的栈。主线程与事件队列。

new Promise(

//执行器

function(resolve,reject){

//一段耗时很长的异步操作

resolve();//成功

reject();//失败

}).then(function A(){

// 成功

},function B(){

//失败

})

Promise是一个代理对象，通过引入一个回调，避免更多回调。

Pending fulfilled rejected三个状态。

Promise状态改变，触发then（），每一个then都会返回一个新的Promise实例。

Promise状态一经改变，不会再改变。

简单例子：

console.log("here we go")

new Promise(resolve=>{

setTimeout(() => {

resolve('hello')

}, 2000);

}).then(value=>{

console.log(value+" world")

})

如果在then（）中不返回新的Promise实例，会怎么样呢？

它就会默认去执行下一个环节。如果什么都不返回，会默认返回undefined。

Promise.resolve（）？  
返回一个fulfilled的Promise实例。

如果参数为空，返回一个状态为fulfilled的Promise实例。

如果参数是一个跟Promise无关的值，同上，不过fullfiled响应函数会得到这个参数。

如果参数为Promise实例，则直接返回这个参数。

把回调包装成Promise，可读性更好。封装Ajax后：

function Ajax(ajaxOptions){

return new Promise(function(resolve,reject){

var xhr=new XMLHttpRequest();

xhr.open(ajaxOptions.url,ajaxOptions.method,ajaxOptions.async);

xhr.send();

xhr.onloadend=function(){

if(xhr.response.status===200){

resolve(xhr);

}else{

reject(xhr);

}

}

xhr.onerror=function(){

reject(xhr);

}

})

}

Ajax(ajaxOptions).then(function(xhr){

console.log(xhr.response.status);

},function(xhr){

console.log("Error"+xhr.response.status);

});

Promise的神奇之处在于他找回了return和catch。

Then中可以返回另一个Promise、一个同步值或者一个错误。