**[AngularJS 中的Promise --- $q服务详解](http://www.cnblogs.com/xing901022/p/4928147.html)**

先说说什么是Promise，什么是$q吧。Promise是一种异步处理模式，有很多的实现方式，比如著名的Kris Kwal's Q还有JQuery的Deffered。

**什么是Promise**

以前了解过Ajax的都能体会到回调的痛苦，同步的代码很容易调试，但是异步回调的代码，会让开发者陷入泥潭，无法跟踪，比如：

funA(arg1,arg2,function(){

funcB(arg1,arg2,function(){

funcC(arg1,arg2,function(){

xxxx....

})

})

})

本身嵌套就已经很不容易理解了，加上不知何时才触发回调，这就相当于雪上加霜了。

但是有了Promise这种规范，它能帮助开发者用同步的方式，编写异步的代码，比如在AngularJS中可以使用这种方式：

deferABC.resolve(xxx)

.then(funcSuccess(){},funcError(){},funcNotify(){});

当resolve内的对象成功执行，就会触发funcSuccess，如果失败就会触发funcError。有点类似

deferABC.resolve(function(){

Sunccess:funcSuccess,

error:funcError,

notify:funcNotify

})

再说的直白点，Promise就是一种对执行结果不确定的一种预先定义，如果成功，就xxxx；如果失败，就xxxx，就像事先给出了一些承诺。

**比如，小白在上学时很懒，平时总让舍友带饭，并且事先跟他说好了，如果有韭菜鸡蛋就买这个菜，否则就买西红柿炒鸡蛋；无论买到买不到都要记得带包烟。**

小白让舍友带饭()

.then(韭菜鸡蛋，西红柿炒鸡蛋)

.finally(带包烟)

**$q服务**

q服务是AngularJS中自己封装实现的一种Promise实现，相对与Kris Kwal's Q要轻量级的多。  
先介绍一下$q常用的几个方法：

* defer() 创建一个deferred对象，这个对象可以执行几个常用的方法，比如resolve,reject,notify等
* all() 传入Promise的数组，批量执行，返回一个promise对象
* when() 传入一个不确定的参数，如果符合Promise标准，就返回一个promise对象。

在Promise中，定义了三种状态：等待状态，完成状态，拒绝状态。

**关于状态有几个规定：**

* 1 状态的变更是不可逆的
* 2 等待状态可以变成完成或者拒绝

**defer()方法**

在$q中，可以使用resolve方法，变成完成状态；使用reject方法，变成拒绝状态。

下面看看 $q的简单使用：

<html ng-app="myApp">

<head>

<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />

<script src="http://apps.bdimg.com/libs/angular.js/1.2.16/angular.min.js"></script>

</head>

<body>

<div ng-controller="myctrl">

{{test}}

</div>

<script type="text/javascript">

var myAppModule = angular.module("myApp",[]);

myAppModule.controller("myctrl",["$scope","$q",function($scope, $ q ){

$scope.test = 1;//这个只是用来测试angularjs是否正常的，没其他的作用

var defer1 = $q.defer();

var promise1 = defer1.promise;

promise1

.then(function(value){

console.log("in promise1 ---- success");

console.log(value);

},function(value){

console.log("in promise1 ---- error");

console.log(value);

},function(value){

console.log("in promise1 ---- notify");

console.log(value);

})

.catch(function(e){

console.log("in promise1 ---- catch");

console.log(e);

})

.finally(function(value){

console.log('in promise1 ---- finally');

console.log(value);

});

defer1.resolve("hello");

// defer1.reject("sorry,reject");

}]);

</script>

</body>

</html>

其中defer()用于创建一个deferred对象，defer.promise用于返回一个promise对象，来定义then方法。then中有三个参数，分别是成功回调、失败回调、状态变更回调。

其中resolve中传入的变量或者函数返回结果，会当作第一个then方法的参数。then方法会返回一个promise对象，因此可以写成

xxxx

.then(a,b,c)

.then(a,b,c)

.then(a,b,c)

.catch()

.finally()

继续说说上面那段代码，then...catch...finally可以想想成java里面的try...catch...finally。

**all()方法**

这个all()方法，可以把多个primise的数组合并成一个。当所有的promise执行成功后，会执行后面的回调。回调中的参数，是每个promise执行的结果。  
当批量的执行某些方法时，就可以使用这个方法。

var funcA = function(){

console.log("funcA");

return "hello,funA";

}

var funcB = function(){

console.log("funcB");

return "hello,funB";

}

$q.all([funcA(),funcB()])

.then(function(result){

console.log(result);

});

执行的结果：

funcA

funcB

Array [ "hello,funA", "hello,funB" ]

**when()方法**

when方法中可以传入一个参数，这个参数可能是一个值，可能是一个符合promise标准的外部对象。

var funcA = function(){

console.log("funcA");

return "hello,funA";

}

$q.when(funcA())

.then(function(result){

console.log(result);

});

当传入的参数不确定时，可以使用这个方法。

hello,funA

# angular Promises详解（$q服务）

[](http://www.jianshu.com/u/c84a55462fdf)

作者 [MakingChoice](http://www.jianshu.com/u/c84a55462fdf)

2015.12.06 20:28\* 字数 1509 阅读 2187评论 0喜欢 3

这是我在翻阅资料的时候，在国外的thinkster上找到的一篇关于关于angular中promise的一篇资料教程，写的非常好，收益良多，我现在翻译一下，供大家学习交流，如果大家想深入学习，欢迎到thinkster注册，并且付费。  
下面是正文：  
angular的promise是由$q提供和构件的，$q提供了一个通过注册一个promise项目来异步执行的方法。Promise作为ES6的一部分规范形成了原生javascript。angular的$q服务提供了一个新的接口，方便与以后移植到ES6。  
code

<html>

<head>

<tittle>Promise fun</tittle>

</head>

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.0-beta.5/angular.min.js"></script>

<script src="app.js"></script>

<body ng-app="app">

</body>

</html>

function getData($timeout, $q) {

return function() {

var defer = $q.defer()

// simulated async function

$timeout(function() {

}, 2000)

return defer.promise

}

}

angular.module('app', [])

.factory('getData', getData)//把getData方法进入factory进行实例化，方便进去run方法进行配置

.run(function(getData) {//run方法用于快速配置，注入完成后启动（只能是实例化的服务或者参数可以配置）

var promise = getData()

})

为了简单起见，我们使用angular的$timeout服务模拟异步函数，尤其是讲解AJAX使用angular的$HTTP服务，是最常用使用promise的地方

## 异步对象

延迟对象是一个简单的对象当为了解决这个promise对象暴露出一个promise对象，它是使用$q.defer()方法来构建，并且暴露出三个主要的方法，resolve() reject()和notify();这个相关联的promise对象能够通过promise对象属性来访问.

function getData(){

return function(){

var defer=$q.defer();

$timeout(function(){

defer.resolve("data received");

},2000);

return defer.promise;

}

}

这里创建了一个promise对象，然后返回promise对象的属性，执行异步函数当函数完成后，我们解析这个延迟对象，这个resoleve()函数的参数会被传递到回调函数中。

## Promise对象

现在我们获得了一个promise对象（defer.promise），让我们注册一个回调函数，当异步函数完成后执行。使用then()方法，附加一个回调函数，用来返回promise对象用来打印返回的string对象。

.run(function(getData){

var promise=getData()

.then(function(){

console.log(string);

})

})

## 拒绝一个promise对象

我们已经见过如何成功解析一个promise对象，但是当异步函数失败的时候会发生什么，相比于使用Resolve()方法，我们称reject()方法如果当失败的时候使用。  
使用Math.random()来模拟失败然后返回这个promise对象，一半的机会。

function getData($timeout,$q){

var defer=$q.defer();

$timeout(function(){

if(Math.round(Math.rondom())){

defer.solve('data received!')

}else{

defer.reject('oh no an error! try again');

}

},2000)

return defer.promise;

}

then()方法的第二个参数是一个可选的发生错误时候的回调函数，当这个promise对象失败的时候调用。  
给then()传入第二个作为执行发生错误时的回调函数。

.run(function(getData){

var promise=getData()

.then(function(string){

console.log(string)

},function(error){

console.log(error);

})

})

## 使用$q构造函数

使用$q服务本身能够快速变换一个基于异步函数的回调函数，形成一种快速解决方案。  
重写这个模拟的异步函数用来返回一个promise对象使用$q函数。

function($timeout,$q){

return function(){

return $q(function(resolve,reject){

$timeout(function(){

if(Math.ronund(Math.random())){

resolve('data received!')

}else{

reject('oh no an error! try again')

}

},2000)

})

}

}

这个方法完成了同样的事情，手动的创建了一个延迟对象，

## finally()方法

promise对象只能成功执行函数或者失败回调函数能够被执行，但不是全部。但是当你需要确保这个方法执行忽略promise的结果。你可以通过使用finally()来注册这个方法，当对于一个已知状态的时候，很有用。  
使用这个finally()方法来打印相应时间间隔，当异步函数完成时。

.run(function(getData) {

var promise = getData()

.then(function(string) {

console.log(string)

}, function(error) {

console.error(error)

})

.finally(function() {

console.log('Finished at:', new Date())

})

})

现在你可以看到现在时间被打印出来，忽略了promise执行的状态。

## promise链

最有魅力的promise特性就是能够使用使用链链接起来。它允许数据通过这个链流动，并且每一步进行操作和突变。  
简单的示例：  
现在修改我们的异步函数，用来传递一个0-9之间的随机数，到resolve()函数

function getData($timeout, $q) {

return function() {

// simulated async function

return $q(function(resolve, reject) {

$timeout(function() {

resolve(Math.floor(Math.random() \* 10))

}, 2000)

})

}

}

当页面刷新，可以看到一个随机的interger数字在0-9之间跳动。  
为了开始链式调用，我们必须修改这个回调函数返回一个value。  
修改这个promise调用的回调函数用来返回一个随机数乘以2。

.run(function(getData) {

var promise = getData()

.then(function(num) {

console.log(num)

return num \* 2

})

})

现在我们能够链式调用另外一个回调函数对象使用then()方法，能够在第一个回调函数被执行后返回，这个值传入第二个回调函数被乘以2。

.run(function(getData) {

var promise = getData()

.then(function(num) {

console.log(num)

return num \* 2

})

.then(function(num) {

console.log(num) // = random number \* 2

})

})

这个例子虽然简单，但是说明了一个强大的概念，此外，不仅仅是从promise回调返回一个简单的值，能够返回新的promise对象。这个作用域链会暂停，知道返回这个回调成功。这样允许把多个异步回调串在一起。

下面是原文的连接，如果大家喜欢，可以到原网站学习。<https://thinkster.io/a-better-way-to-learn-angularjs/promises>  
本文可以随意转载，但需注明出处。[http://www.jianshu.com/p/df257dfbc8fe](http://www.jianshu.com/p/df257dfbc8fe" \t "_blank)  
这是我的github 文章分享的地方，欢迎交流。[https://github.com/vista5004/article](https://github.com/vista5004/article" \t "_blank)

[**Angular $q 完全指南**](http://blog.csdn.net/zhuyucheng123/article/details/50597291)

标签： [angular](http://www.csdn.net/tag/angular)[q](http://www.csdn.net/tag/q)

2016-01-27 23:33 1307人阅读 [评论](http://blog.csdn.net/zhuyucheng123/article/details/50597291#comments)(0) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/zhuyucheng123/article/details/50597291#report)

http://static.blog.csdn.net/images/category_icon.jpg分类：

angularJS*（2）* http://static.blog.csdn.net/images/arrow_triangle%20_down.jpg

目录[(?)](http://blog.csdn.net/zhuyucheng123/article/details/50597291" \o "系统根据文章中H1到H6标签自动生成文章目录)[[+]](http://blog.csdn.net/zhuyucheng123/article/details/50597291)

本文转自：[点击打开链接](http://segmentfault.com/a/1190000004317947" \t "_blank)

如果想使用 $http 或者其他异步操作, 那 $q 是必须要掌握的概念啦. Let's get started!

**如何理解$q, deferred object ?**

[形象的讲解angular中的$q与promise](http://www.cnblogs.com/asnowwolf/p/3689300.html)

假设有一个家具厂，而它有一个VIP客户张先生。

有一天张先生需要一个豪华衣柜，于是，他打电话给家具厂说我需要一个衣柜，回头做好了给我送来，这个操作就叫$q.defer，也就是延期，因为这个衣柜不是现在要的，所以张先生这是在发起一个可延期的请求。

同时，家具厂给他留下了一个回执号，并对他说：我们做好了会给您送过去，放心吧。这叫做promise，也就是承诺。

这样，这个defer算是正式创建了，于是他把这件事记录在自己的日记上，并且同时记录了回执号，这叫做deferred，也就是已延期事件。

现在，张先生就不用再去想着这件事了，该做什么做什么，这就是“异步”的含义。

假设家具厂在一周后做完了这个衣柜，并如约送到了张先生家（包邮哦，亲），这就叫做deferred.resolve(衣柜)，也就是“已解决”。而这时候张先生只要签收一下这个（衣柜）参数就行了，当然，这个“邮包”中也不一定只有衣柜，还可以包含别的东西，比如厂家宣传资料、产品名录等。整个过程中轻松愉快，谁也没等谁，没有浪费任何时间。

假设家具厂在评估后发现这个规格的衣柜我们做不了，那么它就需要deferred.reject(理由)，也就是“拒绝”。拒绝没有时间限制，可以发生在给出承诺之后的任何时候，甚至可能发生在快做完的时候。而且拒绝时候的参数也不仅仅限于理由，还可以包含一个道歉信，违约金之类的，总之，你想给他什么就给他什么，如果你觉得不会惹恼客户，那么不给也没关系。

假设家具厂发现，自己正好有一个符合张先生要求的存货，它就可以用$q.when(现有衣柜)来把这个承诺给张先生，这件事就立即被解决了，皆大欢喜，张先生可不在乎你是从头做的还是现有的成品，只会惊叹于你们的效率之高。

假设这个家具厂对客户格外的细心，它还可能通过deferred.notify(进展情况)给张先生发送进展情况的“通知”。

这样，整个异步流程就圆满完成，无论成功或者失败，张先生都没有往里面投入任何额外的时间成本。

好，我们再扩展一下这个故事：

张先生这次需要做一个桌子，三把椅子，一张席梦思，但是他不希望今天收到个桌子，明天收到个椅子，后天又得签收一次席梦思，而是希望家具厂做好了之后一次性送过来，但是他下单的时候又是分别下单的，那么他就可以重新跟家具厂要一个包含上述三个承诺的新承诺，这就是$q.all(桌子承诺，椅子承诺，席梦思承诺)，

这样，他就不用再关注以前的三个承诺了，直接等待这个新的承诺完成，到时候只要一次性签收了前面的这些承诺就行了。

**如何创建 promise -1**

$q 支持两种写法, 第一种是类似于ES6标准构造函数写法

$q(function resolver (resolve, reject) {})

注意:

+ ES6 写法并不支持 progress/notify 的回调函数

+ 在构造函数中抛异常也并不会显式的reject the promise

// var iWantResolve = false;

var iWantResolve = true;

function es6promise() {

return $q(function (resolve, reject) {

$timeout(function () {

if (iWantResolve) {

resolve("es6promise resolved");

} else {

reject("es6promise reject");

}

}, 1000)

})

}

**promise 的方法**

* promise.then(successCb, errCb, notifyCb)
* 其中successCb 将在 promise resolve 后被调用, errCb 将在 promise reject 后被调
* notifyCb 将在 deferred.notify 后被调用, 可以多次调用
* promise.catch == promise.then(null, errCb), 用于处理之前没有被处理的 rejected promise
* promise.finally 将最后被调用, 一般用于资源释放的清理操作

es6promise()

.then(function (data) {

console.log(data);

})

.catch(function (err) {

console.log(err);

});

// if(iWantResolve == true) output: es6promise resolved

// if(iWantResolve = false) output: es6promise reject

**如何创建 promise -2**

第二种是类似于 commonJS 的写法 $q.deferred()

function commonJsPromise() {

var deferred = $q.defer();

$timeout(function () {

deferred.notify("commonJS notify");

if (iWantResolve) {

deferred.resolve("commonJS resolved");

} else {

deferred.reject("commonJS reject");

}

}, 500);

return deferred.promise;

}

commonJsPromise()

.then(function /\*\* success callback\*\*/(data) {

console.log(data);

}, function /\*\* error callback \*\*/ (err) {

console.log(err);

}, function /\*\* progress callback \*\*/ (update) {

console.log(update);

});

// if(iWantResolve == true) output: commonJS notify commonJS resolved

// if(iWantResolve = false) output: commonJS notify commonJS reject

**$q.all**

* $q.all([promise1, promise1]) 接受一个包含若干个 promise 的数组,
* 等所有的 promise resolve 后, 其本身 resolve 包含上述结果的数组 [data1, data2]
* 如果上述 promise 有一个 reject, 那么$q.all() 会把这个 rejected promise 作为其 rejected promise (只有一个哦)
* progress/notify 的 callback 并没有用

$q.all([es6promise(), commonJsPromise()])

.then(function (dataArr) {

console.log("$q.all: ", dataArr);

}, function (err) {

console.log("$q.all: ", err)

}, function /\*\* unnecessary \*\*/ (update) {

console.log("$q.all", update);

});

// if(iWantResolve == true) output: $q.all: ["es6promise resolved", "commonJS resolved"]

// if(iWantResolve = false) output: $q.all: es6promise reject

**$q.reject, $q.when, $q.resolve**

* $q.reject() 立即返回一个rejected 的 promise, 在链式调用的时候很有用
* $q.resolve == $q.when(value, successCb, errorCb, progressCb)
* value 可能是一个 then-able 的 obj(即可以是 $q.defer() 返回的, 也可以是其他库产生的), 也可能是任意数据, 但是 $q.when 最终都会返回一个 promise
* $q.when 既可以写成上述的构造函数形式, 也可以写成 $q.when(value).then(fn, fn, fn) 的形式

$q.reject("instant reject")

.catch(function (err) {

console.log(err);

});

// output: instant reject

$q.when(commonJsPromise(),

function /\*\* success callback \*\*/(data) {

console.log("$q.when success callback function: " + data);

return "$q.when success callback return another value";

})

.then(function (data) {

console.log("$q.when then function:" + data);

});

// if(iWantResolve == true) output:

// $q.when success callback functionL: commonJS resolved

// $q.when then function:$q.when success callback return another value

// if(iWantResolve = false) output:

// $q.when err callback function: commonJS reject

// $q.when then function:undefined

$q.when("some value", function (data){

console.log(data);

})

// output: some value

**promise chains 链式调用**

任何在 successCb, errCb 中返回的非 $q.reject()对象, 都将成为一个 resolve 的 promise.   
所以可以出现如下语法 promise.then().then().then()

$q.when("1")

.then(function (data) {

console.log(data);

return $q.reject(2);

})

.catch(function (err) {

console.log(err);

return 3;

})

.then(function (data) {

console.log(data);

})

// output: 1 2 3

# 深入介绍angular中的Promise对象($q)用法

www.111cn.net 更新:2014-11-28 编辑:bluestart 来源:转载

今天小编为各位介绍一下深入介绍angular中的Promise对象($q)用法例子，希望此例子可以帮助到各位同学。

在用JQuery的时候就知道 promise 是 Js异步编程模式的一种模式，但是不是很明白他跟JQuery的deferred对象有什么区别。随着公司项目的进行，要跟后台接数据了，所以决定搞定它。

Promise

Promise是一种模式，以同步操作的流程形式来操作异步事件，避免了层层嵌套，可以链式操作异步事件。

我们知道，在编写[javascript](http://www.111cn.net/js_a/js.html)异步代码时，callback是最最简单的机制，可是用这种机制的话必须牺牲控制流、异常处理和函数语义化为代价，甚至会让我们掉进出现callback大坑，而promise解决了这个问题。

ES6中Promise、angularJS内置的AngularJS内置Q，以及when采用的都是Promises/A规范，如下：

每个任务都有三种状态：未完成(pending)、完成(fulfilled)、失败(rejected)。

pending状态：可以过渡到履行或拒绝状态。  
fulfilled状态：不能变为其他任何状态，而且状态不能改变，必须有value值。  
rejected状态：不能变为其他任何状态，而且状态不能改变，必须有reason。  
状态的转移是一次性的，状态一旦变成fulfilled（已完成）或者failed（失败/拒绝），就不能再变了。

|  |  |
| --- | --- |
| 代码如下 | 复制代码 |
| function okToGreet(name){     return name === 'Robin Hood'; }   function asyncGreet(name) {     var deferred = $q.defer();       setTimeout(function() {      // 因为这个异步函数fn在未来的异步执行，我们把代码包装到 $apply 调用中，一边正确的观察到 model 的改变         $scope.$apply(function() {             deferred.notify('About to greet ' + name + '.');               if (okToGreet(name)) {                 deferred.resolve('Hello, ' + name + '!');             } else {                 deferred.reject('Greeting ' + name + ' is not allowed.');             }         });     }, 1000);       return deferred.promise; }   var promise = asyncGreet('Robin Hood'); promise.then(function(greeting) {     alert('Success: ' + greeting); }, function(reason) {     alert('Failed: ' + reason); }, function(update) {     alert('Got notification: ' + update); }); | |

**Q Promise的基本用法**

上面代码表示， $q.defer() 构建的 deffered 实例的几个方法的作用。如果异步操作成功，则用resolve方法将Promise对象的状态变为“成功”（即从pending变为resolved）；如果异步操作失败，则用reject方法将状态变为“失败”（即从pending变为rejected）。最后返回 deferred.promise ，我们就可以链式调用then方法。

 JS将要有原生Promise，ES6中已经有Promise对象，[firefox](http://www.111cn.net/tags.php/firefox/" \t "_blank)和Chrome 32 beta版本已经实现了基本的Promise API

**AngularJs中的$q.defferd**

通过 调用 $q.defferd 返回deffered对象以链式调用。该对象将Promises/A规范中的三个任务状态通过API关联。

deffered API

**deffered 对象的方法**

resolve(value)：在声明resolve()处，表明promise对象由pending状态转变为resolve。  
reject(reason)：在声明resolve()处，表明promise对象由pending状态转变为rejected。  
notify(value) ：在声明notify()处，表明promise对象unfulfilled状态，在resolve或reject之前可以被多次调用。

**deffered 对象属性**

promise ：最后返回的是一个新的deferred对象 promise 属性，而不是原来的deferred对象。这个新的Promise对象只能观察原来Promise对象的状态，而无法修改deferred对象的内在状态可以防止任务状态被外部修改。

Promise API

当创建 deferred 实例时会创建一个新的 promise 对象,并可以通过 deferred.promise 得到该引用。

promise 对象的目的是在 deferred 任务完成时,允许感兴趣的部分取得其执行结果。

**promise 对象的方法**

then(errorHandler, fulfilledHandler, progressHandler)：then方法用来监听一个Promise的不同状态。errorHandler监听failed状态，fulfilledHandler监听fulfilled状态，progressHandler监听unfulfilled（未完成）状态。此外,notify 回调可能被调用 0到多次，提供一个进度指示在解决或拒绝（resolve和rejected）之前。  
[catch](http://www.111cn.net/tags.php/catch/" \t "_blank)(errorCallback) —— promise.then(null, errorCallback) 的快捷方式  
finally(callback) ——让你可以观察到一个 promise 是被执行还是被拒绝, 但这样做不用修改最后的 value值。 这可以用来做一些释放资源或者清理无用对象的工作,不管promise 被拒绝还是解决。 更多的信息请参阅 完整文档规范.  
通过then()方法可以实现promise链式调用。

|  |  |
| --- | --- |
| 代码如下 | 复制代码 |
| promiseB = promiseA.then(function(result) {     return result + 1;   });     // promiseB 将会在处理完 promiseA 之后立刻被处理,   // 并且其  value值是promiseA的结果增加1 | |

**$q的其他方法**

$q.when(value)：传递变量值，promise.then()执行成功回调  
$q.all(promises)：多个promise必须执行成功，才能执行成功回调，传递值为数组或哈希值，数组中每个值为与Index对应的promise对象。