jQuery的deferred对象详解

这个功能很重要，未来将成为jQuery的核心方法，它彻底改变了如何在jQuery中使用ajax。为了实现它，jQuery的全部ajax代码都被改写了

**jQuery的deferred对象详解**

作者：阮一峰

**一、什么是deferred对象？**

开发网站的过程中，我们经常遇到某些耗时很长的javascript操作。其中，既有异步的操作（比如ajax读取服务器数据），也有同步的操作（比如遍历一个大型数组），它们都不是立即能得到结果的。

通常的做法是，为它们指定回调函数（callback）。即事先规定，一旦它们运行结束，应该调用哪些函数。

但是，在回调函数方面，jQuery的功能非常弱。为了改变这一点，jQuery开发团队就设计了[deferred对象](http://api.jquery.com/category/deferred-object/" \t "_blank)。

**简单说，deferred对象就是jQuery的回调函数解决方案。**在英语中，defer的意思是"延迟"，所以deferred对象的含义就是"延迟"到未来某个点再执行。

它解决了如何处理耗时操作的问题，对那些操作提供了更好的控制，以及统一的编程接口。它的主要功能，可以归结为四点。下面我们通过示例代码，一步步来学习。

**二、ajax操作的链式写法**

首先，回顾一下jQuery的ajax操作的传统写法：

　　$.ajax({

　　　　url: "test.html",

　　　　success: function(){  
　　　　　　alert("哈哈，成功了！");  
　　　　},

　　　　error:function(){  
　　　　　　alert("出错啦！");  
　　　　}

　　});

（运行[代码示例1](http://jsfiddle.net/ruanyf/pdQYH/" \t "_blank)）

在上面的代码中，$.ajax()接受一个对象参数，这个对象包含两个方法：success方法指定操作成功后的回调函数，error方法指定操作失败后的回调函数。

$.ajax()操作完成后，如果使用的是低于1.5.0版本的jQuery，返回的是XHR对象，你没法进行链式操作；如果高于1.5.0版本，返回的是deferred对象，可以进行链式操作。

现在，新的写法是这样的：

　　$.ajax("test.html")

**.done(function(){ alert("哈哈，成功了！"); })**

**.fail(function(){ alert("出错啦！"); });**

（运行[代码示例2](http://jsfiddle.net/ruanyf/dYKLJ/" \t "_blank)）

可以看到，[done()](http://api.jquery.com/deferred.done/" \t "_blank)相当于success方法，[fail()](http://api.jquery.com/deferred.fail/" \t "_blank)相当于error方法。采用链式写法以后，代码的可读性大大提高。

**三、指定同一操作的多个回调函数**

deferred对象的一大好处，就是它允许你自由添加多个回调函数。

还是以上面的代码为例，如果ajax操作成功后，除了原来的回调函数，我还想再运行一个回调函数，怎么办？

很简单，直接把它加在后面就行了。

　　$.ajax("test.html")

　　.done(function(){ alert("哈哈，成功了！");} )

　　.fail(function(){ alert("出错啦！"); } )

**.done(function(){ alert("第二个回调函数！");} );**

（运行[代码示例3](http://jsfiddle.net/ruanyf/sQYjs/" \t "_blank)）

回调函数可以添加任意多个，它们按照添加顺序执行。

**四、为多个操作指定回调函数**

deferred对象的另一大好处，就是它允许你为多个事件指定一个回调函数，这是传统写法做不到的。

请看下面的代码，它用到了一个新的方法[$.when()](http://api.jquery.com/jQuery.when/)：

**$.when($.ajax("test1.html"), $.ajax("test2.html"))**

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

（运行[代码示例4](http://jsfiddle.net/ruanyf/CdKjn/" \t "_blank)）

这段代码的意思是，先执行两个操作$.ajax("test1.html")和$.ajax("test2.html")，如果都成功了，就运行done()指定的回调函数；如果有一个失败或都失败了，就执行fail()指定的回调函数。

**五、普通操作的回调函数接口（上）**

deferred对象的最大优点，就是它把这一套回调函数接口，从ajax操作扩展到了所有操作。也就是说，任何一个操作----不管是ajax操作还是本地操作，也不管是异步操作还是同步操作----都可以使用deferred对象的各种方法，指定回调函数。

我们来看一个具体的例子。假定有一个很耗时的操作wait：

　　var wait = function(){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！");

　　　　};

　　　　setTimeout(tasks,5000);

　　};

我们为它指定回调函数，应该怎么做呢？

很自然的，你会想到，可以使用$.when()：

　　$.when(wait())

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

（运行[代码示例5](http://jsfiddle.net/5wzrt/" \t "_blank)）

但是，这样写的话，done()方法会立即执行，起不到回调函数的作用。原因在于$.when()的参数只能是deferred对象，所以必须对wait()进行改写：

　　var dtd = $.Deferred(); // 新建一个deferred对象

　　var wait = function(dtd){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！");

**dtd.resolve();** // 改变deferred对象的执行状态

　　　　};

　　　　setTimeout(tasks,5000);

**return dtd;**

　　};

现在，wait()函数返回的是deferred对象，这就可以加上链式操作了。

　　$.when(wait(dtd))

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

（运行[代码示例6](http://jsfiddle.net/gfFPj/" \t "_blank)）

wait()函数运行完，就会自动运行done()方法指定的回调函数。

**六、deferred.resolve()方法和deferred.reject()方法**

如果仔细看，你会发现在上面的wait()函数中，还有一个地方我没讲解。那就是[dtd.resolve()](http://api.jquery.com/deferred.resolve" \t "_blank)的作用是什么？【当**deferred.resolve()方法和deferred.reject()方法执行完等于改变deferred的状态 立马执行done或者fail方法**】

例子1：<script type="text/javascript">

var dtd = $.Deferred(); // 新建一个Deferred对象

alert("执行完毕！1");

**dtd.resolve();**

　　var wait = function(dtd){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！2");

　　　　　　 // 改变Deferred对象的执行状态

　　　　};

　　　　setTimeout(tasks,5000);

　　　　return dtd;

　　};

　　$.when(wait(dtd))

　　.done(function(){ alert("哈哈，成功了！3"); })

　　.fail(function(){ alert("出错啦！4"); });

</script> **执行顺序1->3->2**

例子2：

<script type="text/javascript">

var dtd = $.Deferred(); // 新建一个Deferred对象

alert("执行完毕！1");

　　var wait = function(dtd){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！2");

**dtd.resolve();**

　　　　　　 // 改变Deferred对象的执行状态

　　　　};

　　　　setTimeout(tasks,5000);

　　　　return dtd;

　　};

　　$.when(wait(dtd))

　　.done(function(){ alert("哈哈，成功了！3"); })

　　.fail(function(){ alert("出错啦！4"); });

</script> **执行顺序1->2->3**

要说清楚这个问题，就要引入一个新概念"执行状态"。jQuery规定，deferred对象有三种执行状态----未完成，已完成和已失败。如果执行状态是"已完成"（resolved）,deferred对象立刻调用done()方法指定的回调函数；如果执行状态是"已失败"，调用fail()方法指定的回调函数；如果执行状态是"未完成"，则继续等待，或者调用[progress()](http://api.jquery.com/deferred.progress/" \t "_blank)方法指定的回调函数（jQuery1.7版本添加）。

前面部分的ajax操作时，deferred对象会根据返回结果，自动改变自身的执行状态；但是，在wait()函数中，这个执行状态必须由程序员手动指定。dtd.resolve()的意思是，将dtd对象的执行状态从"未完成"改为"已完成"，从而触发done()方法。

类似的，还存在一个[deferred.reject()](http://api.jquery.com/deferred.reject" \t "_blank)方法，作用是将dtd对象的执行状态从"未完成"改为"已失败"，从而触发fail()方法。

　　var dtd = $.Deferred(); // 新建一个Deferred对象

　　var wait = function(dtd){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！");

**dtd.reject(); // 改变Deferred对象的执行状态**

　　　　};

　　　　setTimeout(tasks,5000);

　　　　return dtd;

　　};

　　$.when(wait(dtd))

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

（运行[代码示例7](http://jsfiddle.net/bhDjd/" \t "_blank)）

**七、deferred.promise()方法**

上面这种写法，还是有问题。那就是dtd是一个全局对象，所以它的执行状态可以从外部改变。

请看下面的代码：

　　var dtd = $.Deferred(); // 新建一个Deferred对象

　　var wait = function(dtd){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！");

　　　　　　dtd.resolve(); // 改变Deferred对象的执行状态

　　　　};

　　　　setTimeout(tasks,5000);

　　　　return dtd;

　　};

　　$.when(wait(dtd))

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

**dtd.resolve();**

（运行[代码示例8](http://jsfiddle.net/nBFse/" \t "_blank)）

我在代码的尾部加了一行dtd.resolve()，这就改变了dtd对象的执行状态，因此导致done()方法立刻执行，跳出"哈哈，成功了！"的提示框，等5秒之后再跳出"执行完毕！"的提示框。

为了避免这种情况，jQuery提供了[deferred.promise()](http://api.jquery.com/deferred.promise/" \t "_blank)方法。它的作用是，在原来的deferred对象上返回另一个deferred对象，后者只开放与改变执行状态无关的方法（比如done()方法和fail()方法），屏蔽与改变执行状态有关的方法（比如resolve()方法和reject()方法），从而使得执行状态不能被改变。

请看下面的代码：

　　var dtd = $.Deferred(); // 新建一个Deferred对象

　　var wait = function(dtd){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！");

　　　　　　dtd.resolve(); // 改变Deferred对象的执行状态

　　　　};  
  
　　　　setTimeout(tasks,5000);

**return dtd.promise(); // 返回promise对象**

　　};

**var d = wait(dtd); // 新建一个d对象，改为对这个对象进行操作**

　　$.when(d)

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

**d.resolve(); // 此时，这个语句是无效的**

（运行[代码示例9](http://jsfiddle.net/Yur4R/)）

在上面的这段代码中，wait()函数返回的是promise对象。然后，我们把回调函数绑定在这个对象上面，而不是原来的deferred对象上面。这样的好处是，无法改变这个对象的执行状态，要想改变执行状态，只能操作原来的deferred对象。

不过，更好的写法是[allenm](http://blog.allenm.me/2012/01/jquery_deferred_promise_method/" \t "_blank)所指出的，将dtd对象变成wait(）函数的内部对象。

　　var wait = function(dtd){

**var dtd = $.Deferred(); //在函数内部，新建一个Deferred对象**

　　　　var tasks = function(){

　　　　　　alert("执行完毕！");

　　　　　　dtd.resolve(); // 改变Deferred对象的执行状态

　　　　};  
  
　　　　setTimeout(tasks,5000);

　　　　return dtd.promise(); // 返回promise对象

　　};

**$.when(wait())**

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

（运行[代码示例10](http://jsfiddle.net/q9TvT/" \t "_blank)）

**八、普通操作的回调函数接口（中）**

另一种防止执行状态被外部改变的方法，是使用deferred对象的建构函数$.Deferred()。

这时，wait函数还是保持不变，我们直接把它传入$.Deferred()：

**$.Deferred(wait)**

　　.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

（运行[代码示例11](http://jsfiddle.net/ruanyf/CucGp/" \t "_blank)）

jQuery规定，$.Deferred()可以接受一个函数名（注意，是函数名）作为参数，$.Deferred()所生成的deferred对象将作为这个函数的默认参数。

**九、普通操作的回调函数接口（下）**

除了上面两种方法以外，我们还可以直接在wait对象上部署deferred接口。

　　var dtd = $.Deferred(); // 生成Deferred对象

　　var wait = function(dtd){

　　　　var tasks = function(){

　　　　　　alert("执行完毕！");

　　　　　　dtd.resolve(); // 改变Deferred对象的执行状态

　　　　};

　　　　setTimeout(tasks,5000);

　　};

**dtd.promise(wait);**

　　wait.done(function(){ alert("哈哈，成功了！"); })

　　.fail(function(){ alert("出错啦！"); });

　　wait(dtd);

（运行[代码示例12](http://jsfiddle.net/ruanyf/PF7Xf/" \t "_blank)）

这里的关键是dtd.promise(wait)这一行，它的作用就是在wait对象上部署Deferred接口。正是因为有了这一行，后面才能直接在wait上面调用done()和fail()。

**十、小结：deferred对象的方法**

前面已经讲到了deferred对象的多种方法，下面做一个总结：

　　（1） [$.Deferred()](http://api.jquery.com/category/deferred-object/) 生成一个deferred对象。

　　（2） [deferred.done()](http://api.jquery.com/deferred.done/" \t "_blank) 指定操作成功时的回调函数

　　（3） [deferred.fail()](http://api.jquery.com/deferred.fail/" \t "_blank) 指定操作失败时的回调函数

　　（4） [deferred.promise()](http://api.jquery.com/deferred.promise/" \t "_blank) 没有参数时，返回一个新的deferred对象，该对象的运行状态无法被改变；接受参数时，作用为在参数对象上部署deferred接口。

　　（5） [deferred.resolve()](http://api.jquery.com/deferred.resolve/" \t "_blank) 手动改变deferred对象的运行状态为"已完成"，从而立即触发done()方法。

　　（6）[deferred.reject()](http://api.jquery.com/deferred.reject/) 这个方法与deferred.resolve()正好相反，调用后将deferred对象的运行状态变为"已失败"，从而立即触发fail()方法。

　　（7） [$.when()](http://api.jquery.com/jQuery.when/) 为多个操作指定回调函数。

除了这些方法以外，deferred对象还有二个重要方法，上面的教程中没有涉及到。

　　（8）[deferred.then()](http://api.jquery.com/deferred.then/)

有时为了省事，可以把done()和fail()合在一起写，这就是then()方法。

　　$.when($.ajax( "/main.php" ))

**.then(successFunc, failureFunc );**

如果then()有两个参数，那么第一个参数是done()方法的回调函数，第二个参数是fail()方法的回调方法。如果then()只有一个参数，那么等同于done()。

　　（9）[deferred.always()](http://api.jquery.com/deferred.always/)

这个方法也是用来指定回调函数的，它的作用是，不管调用的是deferred.resolve()还是deferred.reject()，最后总是执行。

　　$.ajax( "test.html" )

　　.always( function() { alert("已执行！");} );