链滴

jquery.Deferred promise 解决异步回调

作者：xuwangcheng14
原文链接：https://ld246.com/article/1484993737008
来源网站：链滴

感觉Jquery的Deferred 对象很有用并且掌握难度也有,花点时间还是值得研究下的。

注:博文转至 http://www.cnblogs.com/greatluoluo/p/5721746.html 作者张三的美丽家园

<hr />
<hr />

我们先来看一下编写AJAX编码经常遇到的几个问题:

1.由于AJAX是异步的，所有依赖AJAX返回结果的代码必需写在AJAX回调函数中。这就不可避免地成了嵌套，ajax等异步操作越多，嵌套层次就会越深，代码可读性就会越差。

<pre>

<pre class="prettyprint lang-js">$.ajax({
url: url,
data: dataObject,
success: function(){
console.log("I depend on ajax result.");
},
error: function(){}
});</pre>

</pre>

console.log("I will print before ajax finished.");
2.如果AJAX请求之间存在依赖关系，我们的代码就会形成Pyramid of Doom（金字塔厄运）。比如们要完成这样一件事：有4个供Ajax访问的url地址，需要先Ajax访问第1个，在第1个访问完成后，用到的返回数据作为参数再访问第2个，第2个访问完成后再第3个...以此到4个全部访问完成。按照这样写法，似乎会变成这样:

<pre>
<pre class="prettyprint lang-js">$.ajax({
url: url1,
success: function(data){
$.ajax({
url: url2,
data: data,
success: function(data){
$.ajax({
//...
});
}
});

}
});</pre>
</pre>

    3.考虑这种场景，假如我们同时发送两个Ajax请求，然后要在两个请求都成功返回后再做一件接下的事，想一想如果只按前面的方式在各自的调用位置去附加回调，这是不是很困难?

    可以看到：JavaScript中类似于AJAX这种异步的操作，会导致代码嵌套层次复杂，可读性差，有时候甚至是实现需求都非常困难。为了解决这种异步回调难的问题，CommonJS组织制定了异步模编程规范Promises/A。目前该规范已经有了很多的实现者，比如Q, when.js, jQuery.Deffered()等。们以jQuery.Deffered学习下Promise。

    Promise的状态

    Promise对象有3种可能的状态：肯定状态(resolved)、否定状态(rejected)、等待状态(pending)刚开始创建的Promise对象处于pending状态，只能从pending变成resolved或者是从pending变成rected状态。

<pre>
<pre class="prettyprint lang-js">var df1 = $.Deferred();
console.log(df1.state());//pending

var df2 = $.Deferred();
df2.resolve();//resolved
console.log(df2.state());

var df3 = $.Deferred();
df3.reject();
console.log(df3.state());//rejected</pre>
</pre>

    $.Deferred()创建一个延迟对象(也就是Promise对象)，deferred.state()可以获取Promise对象当所处的状态。deferred.resolve()和deferred.reject()则是用来改变Promise对象的状态。

    Promise添加回调函数

    Promise对象有3种状态，我们可以分别为这3种状态注册回调函数。当Promise处于某个状态的时

，会触发这个状态下注册的回调函数。

<pre>
<pre class="prettyprint lang-js">var df = $.Deferred();
df.done(function(){alert("success");});
df.fail(function(){alert("fail");});
df.progress(function(){alert("progress");});

df.notify();

df.resolve();

// df.reject();</pre>
</pre>

    done()、fail()、progress()分别注册resolved、rejected、pending状态下的回调函数。通过resolve()、reject()、notify()可以触发事先注册的回调函数。

    Promise是支持链式调用的，上面的代码可以写成下面的样子。

<pre>
<pre class="prettyprint lang-js">var df = $.Deferred();
df.done(function(){alert("success");})
.fail(function(){alert("fail");})
.progress(function(){alert("progress");});
Promise支持多个回调函数，会按照注册顺序调用。
var df = $.Deferred();
df.done(function(){alert("first");})
.fail(function(){alert("fail");});

df.done(function(){alert("second");});
df.done(function(){alert("third");});

df.resolve();</pre>
</pre>

    deferred.always()添加的回调函数，无论Promise是resolved状态还是rejected状态，都会被调用

<pre>
<pre class="prettyprint lang-js">var df1 = $.Deferred();
df1.always(function(type){alert(type);});
df1.resolve("resolve");

var df2 = $.Deferred();
df2.always(function(type){alert(type);});

df2.reject("reject");</pre>
</pre>

progress()和notify()能够用来实现进度条效果，因为notify()允许调用多次，而reject()和resolve()能调用一次。这个很好理解，因为一旦状态变成resolved或者是rejected，就不能再改变其状态，也有必要。

<pre>
<pre class="prettyprint lang-js">var df = $.Deferred();
df.done(function(){alert("success");});
df.fail(function(){alert("fail");});
df.progress(function(){alert("progress");});

// resolve()调用2次,但是只能触发1次success
df.resolve();
df.resolve();

var mudf = $.Deferred();
mudf.done(function(){alert("success");});
mudf.fail(function(){alert("fail");});
mudf.progress(function(){alert("progress");});
</pre>
</pre>

// 每次调用notify都会触发progress回调函数
<pre class="prettyprint lang-js">mudf.notify("%10");
mudf.notify("%20");</pre>

rejectWith()、resolveWith()、notifyWith()功能上和reject()、resolve()、notify()没有什么差别，要差别在于回调函数中的执行上下文(方法中的this)和参数形式。具体差别可以参考"JQuery.Callback系列一:api使用
// 老的ajax写法

<pre>
<pre class="prettyprint lang-js">$.ajax({
url: "test.html",
success: function(){
alert("success");
},
error:function(){
alert("error");
}
});</pre>
</pre>

// 使用promise后的写法

<pre>
<pre class="prettyprint lang-js">$.ajax("test.html")
.done(function(){})
.fail(function(){})
.done(function(){}
.fail(function(){});</pre>
</pre>

JQuery中的Deferred对象与Promis对象区别

JQuery.Deferred相关的API，有的返回的是Deferred对象，有的返回的是Promise对象。如done(、reject()等大部分函数返回的都是Deferred对象，$.when()和then()函数返回的是Promise对象。具可以参考JQuery API文档。

JQuery官方对Promise Objects的解释是:

This object provides a subset of the methods of the Deferred object (then, done, fail, always progress, state and promise) to prevent users from changing the state of the Deferred.

可以看到Promise对象其实就是Deferred对象的一部分，Deferred对象提供了notify、reject、resove等改变状态的方法，但是Promise对象没有提供这些方法。

文章开始提到的AJAX问题1~3，问题1可以很容易通过Promise得到解决。问题2和问题3是通过$.hen()和deferred.then()得到解决，由于这2个API相对来说复杂一些，以后的文章再分析这2个API。

详解"这篇文章中的fire()和fireWith()。

上面简单的介绍了Promise的使用方式，我们可以用Promise的方式来编写AJAX代码。可以很容易看出：使用Promise后代码嵌套层次少了，代码是纵向增长的，而不再是横向增长。而且使用Promis，可以指定多个ajax回调函数。

<p style="font-size:18px;vertical-align:baseline;text-align:justify;color:#2F2F2F;font-family:Geo

gia, 'Times New Roman', Times, 'Songti SC', SimSun, serif;background-color:#FFFFFF;">
jquery Deferred 快速解决异步回的问题

<pre>
<pre class="prettyprint lang-js">function ok(name){

var dfd = new $.Deferred();

callback:func(){

return dfd.resolve(response);

}

return dfd.promise();

}

$.when(ok(1),ok(2)).then(function(resp1,resp2){})</pre>

</pre>

//相关API 分成3类

1类：$.when(pro1,pro1) 将多个 promise 对象以and的关系 合并为1个

2类：promise 激发为 解决 deferred.resolve([args]) deferred.resolveWith(context, [args])

和 拒绝 .reject .rejectWith

context 上下文 替换 this 和通知 .notify .notifyWith

3类: 对激发的响应 解决时deferred.done(args) 拒绝时 deferred.fail() 通知时 deferred.progress()

不管 解决 或 拒绝 deferred.always()

deferred.then(doneCallbacks, failCallbacks [, progressCallbacks])

    promise(或者叫deferred 延迟对象如何获取？)

    var dfd = new $.Deferred(); return dfd.promise();

    返回promise当前状态

    deferred.state() pending(尚未完成) resolved rejected

    管道

    deferred.pipe([doneFilter], [failFilter])

<pre>
<pre class="prettyprint lang-js">var defer = $.Deferred()

var filtered = defer.pipe(null, function(value) {

return value * 3;

});

defer.reject(6);

filtered.fail(function(value) {

alert("Value is (3*6 =) 18: " + value);

});</pre>

</pre>