**后台小白前端入门－－RequireJS**

[](http://www.jianshu.com/u/199d841c785a)

[wuqke](http://www.jianshu.com/u/199d841c785a) 关注

2016.08.05 14:13 字数 4303 阅读 14834评论 10喜欢 46

**导语：**

之前一直有听说RequireJS，但是一直都没机会去了解，只知道它是一个给js做模块化的API。最近在做React，其组件化的思想和js模块化的思想不谋而合。就想在项目中应用React的同时，也把RequireJS加进来，看看会不会对页面加载或者开发有很好的效果。

**What is RequireJS？**

在说明什么是RequireJS之前，不得不提的就是Javascript模块化历史的背景。其实在早期，javascript作为一门新兴的脚本语言出现，有着庞大的愿景，它并不是作为一门仅仅针对客户端设计的语言。只是说后来web应用的流行，javascript作为浏览器端脚本语言而迅速传开，加上Netscape和微软的竞争将其过早的标准化。所以就导致了JS的诸多缺陷，其中一个就是模块化(但是你可以惊奇地发现其实javascript有将import,export等作为保留字，说明设计的时候其实是有考虑的，新的标准es6也让原生支持模块化了)。然后随着web应用越来越复杂，嵌入的javascript代码越来越多，还有node的兴起，模块化编程就变成了必须。

所以就有了后来Dojo工具包和Google的Closure库支持的模块系统。还有两个非常通用的标准规范，CommonJS和AMD。这里就不展开说了，我们只需要知道，实现CommonJS规范的API是同步加载模块的，而实现AMD规范的API是则是异步加载模块。  
所以理论上来说，AMD规范的非阻塞加载更加适合浏览器端。而RequireJS就是AMD规范的最好实现。抄一段官方文档对RequireJS的描述：

RequireJS 是一个JavaScript模块加载器。它非常适合在浏览器中使用, 它非常适合在浏览器中使用，但它也可以用在其他脚本环境, 就像 Rhino and Node. 使用RequireJS加载模块化脚本将提高代码的加载速度和质量。

**Why RequireJS？**

所以，知道了RequireJS是干什么的，也差不多知道为什么我们要使用RequireJS了。不过还是总结一下用RequireJS的好处吧。

* 异步“加载”。我们知道，通常网站都会把script脚本的放在html的最后，这样就可以避免浏览器执行js带来的页面阻塞。使用RequireJS，会在相关的js加载后执行回调函数，这个过程是异步的，所以它不会阻塞页面。
* 按需加载。通过RequireJS，你可以在需要加载js逻辑的时候再加载对应 的js模块，这样避免了在初始化网页的时候发生大量的请求和数据传输，或许对于一些人来说，某些模块可能他根本就不需要，那就显得没有必要。
* 更加方便的模块依赖管理。相信你曾经一定遇到过因为script标签顺序问题而导致依赖关系发生错误，这个函数未定义，那个变量undefine之类的。通过RequireJS的机制，你能确保在所有的依赖模块都加载以后再执行相关的文件，所以可以起到依赖管理的作用。
* 更加高效的版本管理。想一想，如果你还是用的script脚本引入的方式来引入一个jQuery2.x的文件，然后你有100个页面都是这么引用的，那当你想换成jQuery3.x，那你就不得不去改这100个页面。但是如果你的requireJS有在config中做jQuery的path映射，那你只需要改一处地方即可。
* 当然还有一些诸如cdn加载不到js文件，可以请求本地文件等其它的优点，这里就不一一列举了。

**RequireJS 使用**

**需要在页面中引入的文件**

<script data-main="js/main" src="xxx/xxxx/require.js"></script>

使用RequireJS，你只需要引入一个require.js即可。需要说明的是，一个比较好的实践，就是你的页面上面应该也只需要通过\<script\>标签引入这一个js即可。然后你这个页面的所有业务逻辑只需要在main.js里面写(data-main属性作用后面会有讲)就可以了。其它引用的依赖怎么办？当然是通过require按需引入啊！

**Require基本概述**

其实Requirejs整个源文件包括注释就2000来行，其对外暴露的变量其实就三个,**requirejs**,**require**,**define**。

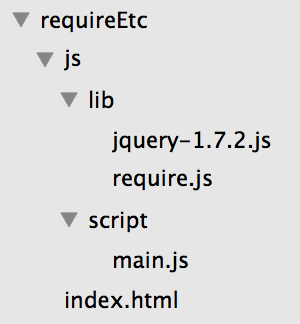
这其中**requirejs** 只是**require**的一个别名，目的是如果页面中有**require**其它实现了，你还是能通过使用requirejs来使用requireJS API的(本文中没有相关冲突，所以还是使用require)。

所以这意味着作为入门，你只需要掌握require,require.config,define这三样就可以了。

本文将以介绍require,require.config,data-main,define的顺序来介绍RequireJS。让比较简单的RequireJS更加简单，争取让大家只看这篇文章就能用好RequireJS。至于RequireJS是如何解决循环依赖,对于没有实现amd的模块如何通过shim来导出，如何在node中使用等问题。本文并没有提及，详细有需要可以去官方查阅。

**require**

首先，先不管三七二十一，我们先按照下面的方式创建一个这样的目录结构:



图片名称

然后require.js可以通过npm下载或者在官网获得。jquery同理，jquery需要下载***1.7.0***或以上的版本。然后把对应的代码拷入对应的文件中，给出余下两个文件的代码:

// js/script/index.html

<!DOCTYPE html>

<html>

<head>

<title>Require Demo 1</title>

</head>

<body>

<div>

<h1>Require Demo 1 -- usage of Require()</h1>

<button id="contentBtn">Click me</button>

<p id="messagebox"></p>

</div>

<script data-main="js/script/main" src="js/lib/require.js" type="text/javascript"></script>

</body>

</html>

// js/script/main.js

require.config(

{

paths: {

'jquery': '../lib/jquery-1.7.2'

}

}

);

require(['jquery'],function ($) {

$(document).on('click','#contentBtn',function(){

$('#messagebox').html('You have access Jquery by using require()');

});

});

先看index.html的代码，其实比较简单，页面上在js中会用到的就是一个button和一个p标签。然后整个页面就只是一个js文件是通过\<script\>标签加载的，就是require.js。注意到标签中有一个data-main属性，你现在只需要了解require.js会在加载完成以后通过回调方法去加载这个data-main里面的js文件，所以这个js文件被加载的时候，RequireJS已经加载执行完毕。

然后接着看main.js文件,里面被一个匿名立即执行函数所包括。在require.config(...)中，可以配置许多配置项，后面会有详细说明。上面在config中添加了一个path，在path配置了一个模块ID和路径的映射，这样在后续的所有函数中就可以直接通过模块ID来引入依赖，而不用再多次引入依赖多次输入路径带来的麻烦。

然后接着就是我们的require(...)函数了。上面的语法中require函数接受的第一个参数是，所依赖模块的一个数组。即使你只需要传入一个依赖，也需要把这个依赖放进数组中传入。如果你有如本例子中设置了模块ID和路径的映射，那你在传入依赖的时候就可以使用模块ID来代替路径，如果没有配置模块ID你当然也可以通过路径来引进对应的模块。接着是传入回调函数，当引入的依赖加载完毕后，这个回调函数就会被触发。如果你传入的依赖有注入变量（函数）,然后在回调函数中需要用到，你就需要按照顺序在回调函数的参数中添加别名，在本例子中可以通过别名$来使用jQuery的相关API。所以有注入的模块需要放在无注入或者不需要调用模块的模块前面，方便回调函数传入别名。例子中在回调函数中为id为contentBtn的button注册监听事件，如果触发，则往id为messagebox的p标签添加相应的内容。

另外还需要额外说明的是路径，不管是在配置中写路径还是直接在require函数中写路径，你都需要了解requireJS在不同情况下的相对路径。

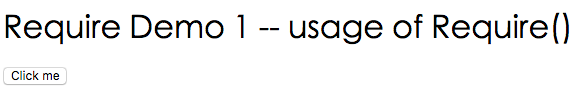
以下是相对路径的规则:

1.如果\<script\>标签引入require.js时没有指定data-main属性，则以引入该js的html文件所在的路径为根路径。

2.如果有指定data-main属性，也就是有指定入口文件，则以入口文件所在的路径为根路径。在本例子中也就是main.js所在的script文件夹就是根路径，这也是为什么配置jQuery的时候需要返回上层目录再进入lib目录才能找到jQuery文件。

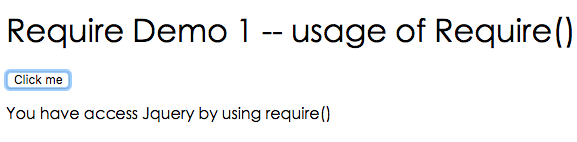
3.如果再require.config()中有配置baseUrl，则以baseUrl的路径为根路径。  
以上三条优先级逐级提升，如果有重叠，后面的根路径覆盖前面的根路径。

打开网页，然后你就应该看到这样的页面：



图片名称

点击按钮，有如下效果，说明通过RequireJS已载入Jquery，并且通过Jquery绑定了监听事件。



图片名称

**define**

讲完了如何引入模块，现在讲如何定义一个模块,require定义一个模块是通过 define = function (name, deps, callback)完成的,第一个参数是定义模块名，第二个参数是传入定义模块所需要的依赖，第三个函数则是定义模块的主函数，主函数和require的回调函数一样，同样是在依赖加载完以后再调用执行。  
先看个例子:

**当你没有任何依赖的时候,你可以这么写:**

// js/script/desc.js

define(function(){

return{

decs : 'this js will be request only if it is needed',

};

})

// 然后在main.js的添加如下代码

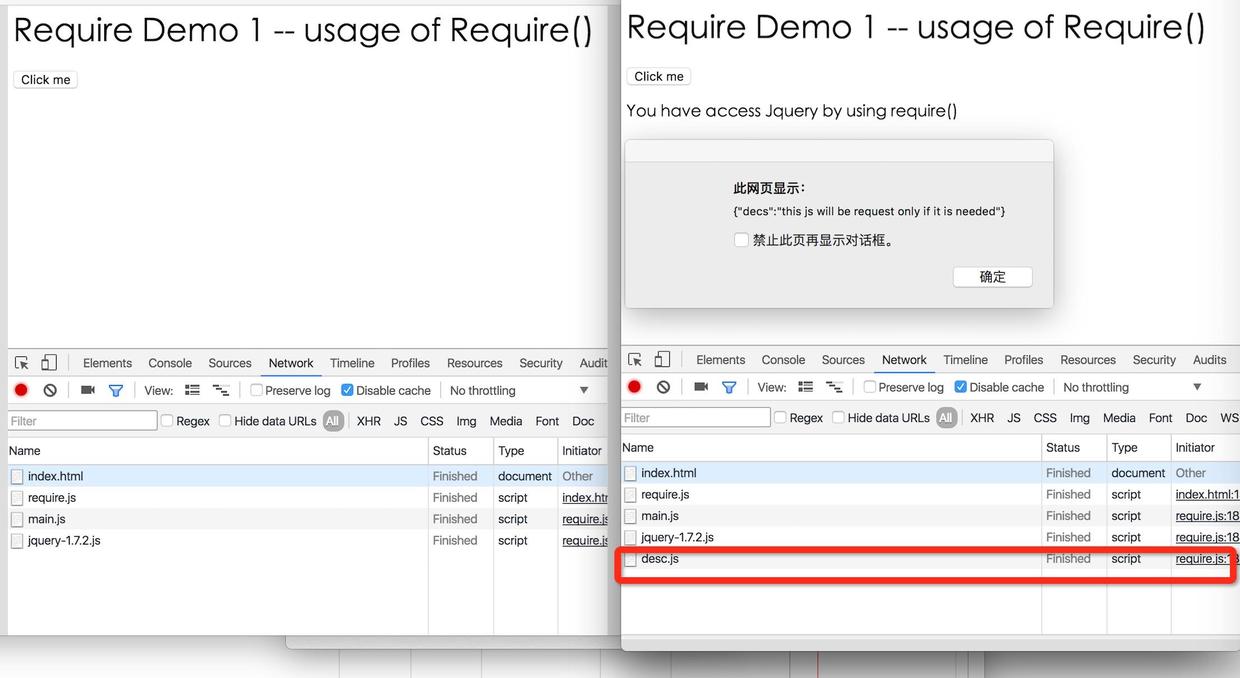
// js/script/main.js

$('#messagebox').html('You have access Jquery by using require()');

＋ require(['script/desc'],function(desc){

＋ alert(JSON.stringify(desc));

再次打开网页，打开network视图，点击按钮，通过require获得的desc模块就会alert出来，同时你会发现，desc.js是按需请求的，并不是在页面一开始的时候就请求的。



图片名称

**当你有相关依赖的时候,你可以这么写:**

// js/script/alertdesc.js

define(['script/desc'],function(desc){

return function (){

alert(JSON.stringify(desc));

};

})

// 然后在main.js的再做如下修改

// js/script/main.js

$('#messagebox').html('You have access Jquery by using require()');

－ require(['script/desc'],function(desc){

－ alert(JSON.stringify(desc));

＋ require(['script/alertdesc'],function(alertdesc){

＋ alertdesc();

**为什么我始终都没有使用name来定义自己的模块名:**

如果你细心，你可能会发现，刚刚define函数，有一个参数name是用来定义模块名的(也就是第一个传参)，为什么上面两个例子都没有用到。其实我确实可以添加模块名，如下:

// js/script/alertdesc.js

define(['script/desc'],function(desc){

.....

})

//Change To

define('/script/alertdesc',['script/desc'],function(desc){

.....

})

但是，这样做感觉不很有必要，因为如果哪一天我将这个文件转移到其他目录下，那我就得在这这里再修改一次模块名。官方其实也不推荐，用官方的说法是:让优化工具去自动生成这些模块名吧！

**require.config**

在上面一节介绍require（）函数的时候，我们已经接触过require.config(...)了。其实说白了，在require.config()做的一些修改会影响到全局require的一些特性。如上面的，你设置了baseUrl  
，其require的根路径就以这个路径为准，你在path中设置了模块ID与路径的映射，后面需要用到相关模块的时候直接使用模块ID代替路径就好了，设置map可以在不同路径下用相同的模块ID调用不同版本的模块。

其实这里并不打算对require.config()的具体配置展开来介绍，有需要可以直接去官网查阅相关配置信息加进来就好了。  
始终觉得require.config()应该抽出来，单独放在一个js文件里面，这样方便移植和重用。在github上看了些例子，找到一个比较好的放置require.config的地方，放在这里可以参考：

// 添加config.js

// js/script/config.js

define(function(){

require.config({

baseUrl: './js/',

paths: {

'jquery': 'lib/jquery-1.7.2'

}

});

});

// 替换main.js

// js/script/main.js

require(['config'],function(){

require(['jquery'],function ($) {

$(document).on('click','#contentBtn',function(){

$('#messagebox').html('You have access Jquery by using require()');

require(['script/alertdesc'],function(alertdesc){

alertdesc();

});

});

});

});

**data-main**

还记得刚刚的\<script\>引入RequireJS时标签中有一个data-main属性么？当require.js加载的时候会检查data-main属性，所以你可以在data-main指向的脚本(也就是本例子中的js/main.js)中设置模块加载的选项,然后在这个脚本加载第一个应用模块。注意，你在main.js中所设置的脚本是异步加载并通过回调来执行的，这意味着如果你在页面中有通过\<script\>引入其它的脚本，那不能保证在main.js里面做的配置会在其它脚本中生效。  
例如:

<script data-main="scripts/main" src="scripts/require.js"></script>

<script src="scripts/other.js"></script>

// scripts/main.js:

require.config({

paths: {

foo: 'libs/foo-1.1.3'

}

});

// scripts/other.js:

// 由于main.js会是在require.js异步加载完以后再通过回调去执行main.js的

// 所以other.js里面执行的这个require函数可能会发生在main.js的require.config执行之前

// 因此require.config会去尝试去加载"scripts/foo.js"，而不是"scripts/libs/foo-1.1.3.js"

require( ['foo'], function( foo ) {

});

这个例子是官方的。从这里也可以看出来，为什么如前文所说的。页面中最好只有一个入口点文件(属性data-main中引入的main.js)，然后这个入口点文件里引入或者编写配置，加载相关应用模块。  
当然你也可以像官方给的第二种方案，不设置入口点，然后在每个require回调中再引入相关配置，不过那样很麻烦而且不易于维护。这里就不给出例子了，有需要可以去官网看。

**总结**

以上就是关于关于RequireJS简单使用的介绍了，大家有需要可以直接看源码，大概就2000多行，不看具体实现，看它对几个函数声明的描述，对使用起来也是很有帮助的，你会发现有一些连官方文档都没提及到的一些特性(比如require()方法可以直接传入config配置作为第一个参数)。

另外，说一点小插曲，如果需要查阅RequireJS官方的API，有条件的还是建议直接访问英文官方文档。如果说中文的官方文档说还停留在老版本，翻译得比较生涩难懂就算了。一些很明显有错误的描述就真的是责任问题了。我在看中文文档的时候真是各种难移理解，后来直接看英文文档，则顺畅很多。不多说，贴张图让大家感受一下英文文档和中文文档对于waitSeconds的描述：

