**requirejs能带来什么好处**

　　（1）实现js文件的异步加载，避免网页失去响应；

　　（2）管理模块之间的依赖性，便于代码的编写和维护。

官方对requirejs的描述：

*RequireJS is a JavaScript file and module loader. It is optimized for in-browser use, but it can be used in other JavaScript environments, like Rhino and Node. Using a modular script loader like RequireJS will improve the speed and quality of your code.*

大致意思：

在浏览器中可以作为js文件的模块加载器，也可以用在Node和Rhino环境，balabala...。这段话描述了requirejs的基本功能"模块化加载"，什么是模块化加载？我们要从之后的篇幅中一一解释

先来看一段常见的场景，通过示例讲解如何运用requirejs

**正常编写方式**

index.html:

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript" src="a.js"></script>

</head>

<body>

<span>body</span>

</body>

</html>

a.js:

function fun1(){

alert("it works");

}

fun1();

可能你更喜欢这样写

(function(){

function fun1(){

alert("it works");

}

fun1();

})()

第二种方法使用了块作用域来申明function防止污染全局变量，本质还是一样的，当运行上面两种例子时不知道你是否注意到，alert执行的时候，html内容是一片空白的，即<span>body</span>并未被显示，当点击确定后，才出现，这就是**JS阻塞浏览器渲染**导致的结果。

**requirejs写法**

当然首先要到requirejs的网站去下载js -> [requirejs.rog](http://requirejs.org/)  
index.html:

<!DOCTYPE html>

<html>

<head>

<script type="text/javascript" src="require.js"></script>

<script type="text/javascript">

require(["a"]);

</script>

</head>

<body>

<span>body</span>

</body>

</html>

a.js:

define(function(){

function fun1(){

alert("it works");

}

fun1();

})

浏览器提示了"it works"，说明运行正确，但是有一点不一样，这次浏览器并不是一片空白，body已经出现在页面中，目前为止可以知道requirejs具有如下优点：

1. 防止js加载阻塞页面渲染
2. 使用程序调用的方式加载js，防出现如下丑陋的场景

<script type="text/javascript" src="a.js"></script>

<script type="text/javascript" src="b.js"></script>

<script type="text/javascript" src="c.js"></script>

<script type="text/javascript" src="d.js"></script>

<script type="text/javascript" src="e.js"></script>

<script type="text/javascript" src="f.js"></script>

<script type="text/javascript" src="g.js"></script>

<script type="text/javascript" src="h.js"></script>

<script type="text/javascript" src="i.js"></script>

<script type="text/javascript" src="j.js"></script>

**基本API**

require会定义三个变量：define,require,requirejs，其中require === requirejs，一般使用require更简短

* define 从名字就可以看出这个api是用来定义一个模块
* require 加载依赖模块，并执行加载完后的回调函数

前一篇中的a.js：

define(function(){

function fun1(){

alert("it works");

}

fun1();

})

通过define函数定义了一个模块，然后再页面中使用：

require(["js/a"]);

来加载该模块(*注意require中的依赖是一个数组，即使只有一个依赖，你也必须使用数组来定义*)，requir API的第二个参数是callback，一个function，是用来处理加载完毕后的逻辑，如：

require(["js/a"],function(){

alert("load finished");

})

**加载文件**

之前的例子中加载模块都是本地js，但是大部分情况下网页需要加载的JS可能来自本地服务器、其他网站或CDN，这样就不能通过这种方式来加载了，我们以加载一个jquery库为例：

require.config({

paths : {

"jquery" : ["http://libs.baidu.com/jquery/2.0.3/jquery"]

}

})

require(["jquery","js/a"],function($){

$(function(){

alert("load finished");

})

})

这边涉及了require.config，require.config是用来配置模块加载位置，简单点说就是给模块起一个更短更好记的名字，比如将百度的jquery库地址标记为jquery，这样在require时只需要写["jquery"]就可以加载该js，本地的js我们也可以这样配置：

require.config({

paths : {

"jquery" : ["http://libs.baidu.com/jquery/2.0.3/jquery"],

"a" : "js/a"

}

})

require(["jquery","a"],function($){

$(function(){

alert("load finished");

})

})

通过paths的配置会使我们的模块名字更精炼，paths还有一个重要的功能，就是可以配置多个路径，如果远程cdn库没有加载成功，可以加载本地的库，如：

require.config({

paths : {

"jquery" : ["http://libs.baidu.com/jquery/2.0.3/jquery", "js/jquery"],

"a" : "js/a"

}

})

require(["jquery","a"],function($){

$(function(){

alert("load finished");

})

})

这样配置后，当百度的jquery没有加载成功后，会加载本地js目录下的jquery

1. 在使用requirejs时，加载模块时不用写.js后缀的，当然也是不能写后缀
2. 上面例子中的callback函数中发现有$参数，这个就是依赖的jquery模块的输出变量，如果你依赖多个模块，可以依次写入多个参数来使用：

require(["jquery","underscore"],function($, \_){

$(function(){

\_.each([1,2,3],alert);

})

})

如果某个模块不输出变量值，则没有，所以尽量将输出的模块写在前面，防止位置错乱引发误解

**全局配置**

上面的例子中重复出现了require.config配置，如果每个页面中都加入配置，必然显得十分不雅，requirejs提供了一种叫"主数据"的功能，我们首先创建一个main.js：

require.config({

paths : {

"jquery" : ["http://libs.baidu.com/jquery/2.0.3/jquery", "js/jquery"],

"a" : "js/a"

}

})

然后再页面中使用下面的方式来使用requirejs：

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

解释一下，加载requirejs脚本的script标签加入了data-main属性，这个属性指定的js将在加载完reuqire.js后处理，我们把require.config的配置加入到data-main后，就可以使每一个页面都使用这个配置，然后页面中就可以直接使用require来加载所有的短模块名

data-main还有一个重要的功能，当script标签指定data-main属性时，require会默认的将data-main指定的js为根路径，是什么意思呢？如上面的data-main="js/main"设定后，我们在使用require(['jquery'])后(不配置jquery的paths)，require会自动加载js/jquery.js这个文件，而不是jquery.js，相当于默认配置了：

require.config({

baseUrl : "js"

})

**第三方模块**

通过require加载的模块一般都需要符合AMD规范即使用define来申明模块，但是部分时候需要加载非AMD规范的js，这时候就需要用到另一个功能：shim，shim解释起来也比较难理解，shim直接翻译为"垫"，其实也是有这层意思的，目前我主要用在两个地方  
1. 非AMD模块输出，将非标准的AMD模块"垫"成可用的模块，例如：在老版本的jquery中，是没有继承AMD规范的，所以不能直接require["jquery"],这时候就需要shim，比如我要是用underscore类库，但是他并没有实现AMD规范，那我们可以这样配置

require.config({

shim: {

"underscore" : {

exports : "\_";

}

}

})

这样配置后，我们就可以在其他模块中引用underscore模块：

require(["underscore"], function(\_){

\_.each([1,2,3], alert);

})

1. 插件形式的非AMD模块，我们经常会用到jquery插件，而且这些插件基本都不符合AMD规范，比如jquery.form插件，这时候就需要将form插件"垫"到jquery中：

require.config({

shim: {

"underscore" : {

exports : "\_";

},

"jquery.form" : {

deps : ["jquery"]

}

}

})

也可以简写为：

require.config({

shim: {

"underscore" : {

exports : "\_";

},

"jquery.form" : ["jquery"]

}

})

这样配置之后我们就可以使用加载插件后的jquery了

require.config(["jquery", "jquery.form"], function($){

$(function(){

$("#form").ajaxSubmit({...});

})

})

好了，requirejs的基本配置大致就是这么多，还有一些扩展的功能会在之后的篇幅中提到

**一、为什么要用require.js？**

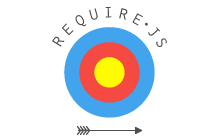
最早的时候，所有Javascript代码都写在一个文件里面，只要加载这一个文件就够了。后来，代码越来越多，一个文件不够了，必须分成多个文件，依次加载。下面的网页代码，相信很多人都见过。

　　<script src="1.js"></script>  
　　<script src="2.js"></script>  
　　<script src="3.js"></script>  
　　<script src="4.js"></script>  
　　<script src="5.js"></script>  
　　<script src="6.js"></script>

这段代码依次加载多个js文件。

这样的写法有很大的缺点。首先，加载的时候，浏览器会停止网页渲染，加载文件越多，网页失去响应的时间就会越长；其次，由于js文件之间存在依赖关系，因此必须严格保证加载顺序（比如上例的1.js要在2.js的前面），依赖性最大的模块一定要放到最后加载，当依赖关系很复杂的时候，代码的编写和维护都会变得困难。

require.js的诞生，就是为了解决这两个问题：



　　（1）实现js文件的异步加载，避免网页失去响应；

　　（2）管理模块之间的依赖性，便于代码的编写和维护。

**二、require.js的加载**

使用require.js的第一步，是先去官方网站[下载](http://requirejs.org/docs/download.html)最新版本。

下载后，假定把它放在js子目录下面，就可以加载了。

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

有人可能会想到，加载这个文件，也可能造成网页失去响应。解决办法有两个，一个是把它放在网页底部加载，另一个是写成下面这样：

　　<script src="js/require.js" **defer async="true"** ></script>

async属性表明这个文件需要异步加载，避免网页失去响应。IE不支持这个属性，只支持defer，所以把defer也写上。

加载require.js以后，下一步就要加载我们自己的代码了。假定我们自己的代码文件是main.js，也放在js目录下面。那么，只需要写成下面这样就行了：

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

data-main属性的作用是，指定网页程序的主模块。在上例中，就是js目录下面的main.js，这个文件会第一个被require.js加载。由于require.js默认的文件后缀名是js，所以可以把main.js简写成main。

**三、主模块的写法**

上一节的main.js，我把它称为"主模块"，意思是整个网页的入口代码。它有点像C语言的main()函数，所有代码都从这儿开始运行。

下面就来看，怎么写main.js。

如果我们的代码不依赖任何其他模块，那么可以直接写入javascript代码。

　　// main.js

　　alert("加载成功！");

但这样的话，就没必要使用require.js了。真正常见的情况是，主模块依赖于其他模块，这时就要使用AMD规范定义的的require()函数。

　　// main.js

　　require(['moduleA', 'moduleB', 'moduleC'], function (moduleA, moduleB, moduleC){

　　　　// some code here

　　});

require()函数接受两个参数。第一个参数是一个数组，表示所依赖的模块，上例就是['moduleA', 'moduleB', 'moduleC']，即主模块依赖这三个模块；第二个参数是一个回调函数，当前面指定的模块都加载成功后，它将被调用。加载的模块会以参数形式传入该函数，从而在回调函数内部就可以使用这些模块。

require()异步加载moduleA，moduleB和moduleC，浏览器不会失去响应；它指定的回调函数，只有前面的模块都加载成功后，才会运行，解决了依赖性的问题。

下面，我们看一个实际的例子。

假定主模块依赖jquery、underscore和backbone这三个模块，main.js就可以这样写：

　　require(['jquery', 'underscore', 'backbone'], function ($, \_, Backbone){

　　　　// some code here

　　});

require.js会先加载jQuery、underscore和backbone，然后再运行回调函数。主模块的代码就写在回调函数中。

**四、模块的加载**

上一节最后的示例中，主模块的依赖模块是['jquery', 'underscore', 'backbone']。默认情况下，require.js假定这三个模块与main.js在同一个目录，文件名分别为jquery.js，underscore.js和backbone.js，然后自动加载。

使用require.config()方法，我们可以对模块的加载行为进行自定义。require.config()就写在主模块（main.js）的头部。参数就是一个对象，这个对象的paths属性指定各个模块的加载路径。

　　require.config({

　　　　paths: {

　　　　　　"jquery": "jquery.min",  
　　　　　　"underscore": "underscore.min",  
　　　　　　"backbone": "backbone.min"

　　　　}

　　});

上面的代码给出了三个模块的文件名，路径默认与main.js在同一个目录（js子目录）。如果这些模块在其他目录，比如js/lib目录，则有两种写法。一种是逐一指定路径。

　　require.config({

　　　　paths: {

　　　　　　"jquery": "**lib/**jquery.min",  
　　　　　　"underscore": "**lib/**underscore.min",  
　　　　　　"backbone": "**lib/**backbone.min"

　　　　}

　　});

另一种则是直接改变基目录（baseUrl）。

　　require.config({

**baseUrl: "js/lib",**

　　　　paths: {

　　　　　　"jquery": "jquery.min",  
　　　　　　"underscore": "underscore.min",  
　　　　　　"backbone": "backbone.min"

　　　　}

　　});

如果某个模块在另一台主机上，也可以直接指定它的网址，比如：

　　require.config({

　　　　paths: {

　　　　　　"jquery": "https://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min"

　　　　}

　　});

require.js要求，每个模块是一个单独的js文件。这样的话，如果加载多个模块，就会发出多次HTTP请求，会影响网页的加载速度。因此，require.js提供了一个[优化工具](http://requirejs.org/docs/optimization.html)，当模块部署完毕以后，可以用这个工具将多个模块合并在一个文件中，减少HTTP请求数。

**五、AMD模块的写法**

require.js加载的模块，采用AMD规范。也就是说，模块必须按照AMD的规定来写。

具体来说，就是模块必须采用特定的define()函数来定义。如果一个模块不依赖其他模块，那么可以直接定义在define()函数之中。

假定现在有一个math.js文件，它定义了一个math模块。那么，math.js就要这样写：

　　// math.js

　　define(function (){

　　　　var add = function (x,y){

　　　　　　return x+y;

　　　　};

　　　　return {

　　　　　　add: add  
　　　　};

　　});

加载方法如下：

　　// main.js

　　require(['math'], function (math){

　　　　alert(math.add(1,1));

　　});

如果这个模块还依赖其他模块，那么define()函数的第一个参数，必须是一个数组，指明该模块的依赖性。

　　define(['myLib'], function(myLib){

　　　　function foo(){

　　　　　　myLib.doSomething();

　　　　}

　　　　return {

　　　　　　foo : foo

　　　　};

　　});

当require()函数加载上面这个模块的时候，就会先加载myLib.js文件。

**六、加载非规范的模块**

理论上，require.js加载的模块，必须是按照AMD规范、用define()函数定义的模块。但是实际上，虽然已经有一部分流行的函数库（比如jQuery）符合AMD规范，更多的库并不符合。那么，require.js是否能够加载非规范的模块呢？

回答是可以的。

这样的模块在用require()加载之前，要先用require.config()方法，定义它们的一些特征。

举例来说，underscore和backbone这两个库，都没有采用AMD规范编写。如果要加载它们的话，必须先定义它们的特征。

　　require.config({

　　　　shim: {  
  
　　　　　　'underscore':{  
　　　　　　　　exports: '\_'  
　　　　　　},

　　　　　　'backbone': {  
　　　　　　　　deps: ['underscore', 'jquery'],  
　　　　　　　　exports: 'Backbone'  
　　　　　　}

　　　　}

　　});

require.config()接受一个配置对象，这个对象除了有前面说过的paths属性之外，还有一个shim属性，专门用来配置不兼容的模块。具体来说，每个模块要定义（1）exports值（输出的变量名），表明这个模块外部调用时的名称；（2）deps数组，表明该模块的依赖性。

比如，jQuery的插件可以这样定义：

　　shim: {

　　　　'jquery.scroll': {

　　　　　　deps: ['jquery'],

　　　　　　exports: 'jQuery.fn.scroll'

　　　　}

　　}