脚本的加载是同步的，可以使用asyn和defer关键字使得加载异步，但可能因此在加载过程中丢失加载的顺序。另一个选择是将所有的脚本捆绑打包在一起，但在捆绑的时候你仍然需要把它们按照正确的顺序排序

AMD就是这样一种对模块的定义，使模块和它的依赖可以被异步的加载，但又按照正确的顺序

RequireJS是一个javascript文件和模块框架，它支持浏览器和像node.js之类的服务器环境

RquireJS所做的是，在你使用script标签加载你所定义的依赖时，将这些依赖通过head.appendChild()函数来加载他们，当依赖加载后，RequireJS计算出模块定义的顺序，并  
按正确的顺序进行调用。这意味着你需要做的仅仅是使用一个’根’来读取你需要的所有功能

RequireJS存在API

Define 该函数用户创建模块。每个模块拥有一个唯一的模块ID，它被用于RequireJS的运行时函数，define函数是一个全局函数，不需要使用requireJS命名空间。

Require 该函数用于读取依赖。同样它是一个全局函数，不需要使用requireJS命名空间

Config 该函数用于配置RequireJS

当RequireJS被加载的时候，它会使用data-main属性去搜寻一个脚本文件(它应该是与使用src加载RequireJS是相同的脚本)。Data-main需要给所有的脚本文件设置一个根路径。根据这个根路径，RequireJS将会去加载所有相关的模块。

另外一个方式定义根路径是使用配置函数，后面我们将会看到。requireJS假设所有的依赖都是脚本，那么当你声明一个脚本依赖的时候你不需要使用.js后缀

配置函数

如果你想改变RequireJS的默认配置来使用自己的配置，你可以使用require.config函数，config函数需要传入一个可选参数对象，这个可选参数对象包括了许多的配置参数选项，下面是一些你可以使用的配置：

baseUrl –用于加载模块的根路径

paths – 用于映射不存在根路径下面的模块路径

shims – 配置在脚本/模块外面并没有使用RequireJS的函数依赖并且初始化函数。假设underscore并没有使用RequireJS定义。但是你还是想通过RequireJS来使用它，那么就需要在配置中把它定义为shim

deps – 加载依赖关系数组

在这个例子中把路径设置为script/app，由lib开始的每一个模块都被配置在script/lib文件夹下面，backbone加载的是一个shim依赖

用RequireJS定义模块

模块是进行了内部实现封装，暴露接口和合理限制范围的对象。RequireJS提供了define函数用于定义模块。按章惯例每个javascript文件只应该定义一个模块。Define函数接受一个依赖数组和一个包含模块定义的函数。通常模块定义函数会把前面的数组中的依赖模块按顺序作为参数接收

使用require函数

在RequireJS中另外一个非常有用的函数是require函数。Require函数用于加载模块依赖但并不会创建一个模块。

利用requireJS进行依赖管理

1. javascript开发一直以来都遵循这种方式发展
2. 在家多份javascript文件需要多次HTTP请求，从而导致加载时间延长
3. Javascript依赖管理难以实施

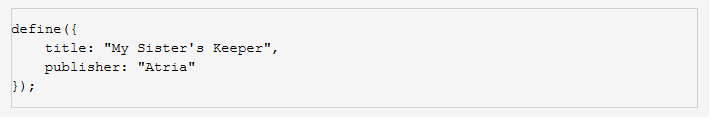
CommonJS工作的其中一个标准是模块。模块是一个自包含的代码段，这种说法定义了模块本身，通过设定模块标准，依赖管理变得更加简单。它并非保持某种必须维持秩序的隐含主列表，各模块只为自身依赖项。该映射可用来确定所需的资源，以及必须采用的加载顺序。

AMD

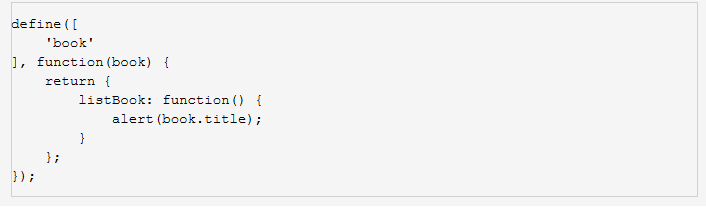
模块概念对于服务器端开发至关重要，因为它解决了如果根据依赖项定义加载模块这个问题，但浏览器端Javascript开发人员有些羡慕。为什么这么有效的几只只能局限于服务器？当然，浏览器需要异步而不是同步加载模块，但这并不意味着模块概念和依赖项定义无法适用。异步模块定义：它们从服务器端获取模块和依赖项定义API，并将其应用至浏览器的异步模块。

定义和请求模块

模块几乎总是在单一文件内定义。同样，单一文件仅包含单一的模块定义。在文件中心定义模块非常简单。



此代码使用define定义了书籍模块，它是一种由RequireJS显示的AMD函数。当您调用它时，相当于从本质说，”将我传递的内容注册为模块”，在这种情况下，该模块是一个以大括号开始和结束的书籍对象。默认情况下，RequireJS假设模块名称是一个文件路径，该文件路径遵循除扩展名外的下列基本URL，由于文件命名为book.js，其中book是默认的模块名称。当其他代码请求RequireJS获取book模块时，RequireJS将返回上文定义的对象。现在您可以在名为bookshelf.js的新文件中创建bookshelf模块，一遍了解如何在其内部请求书籍模块

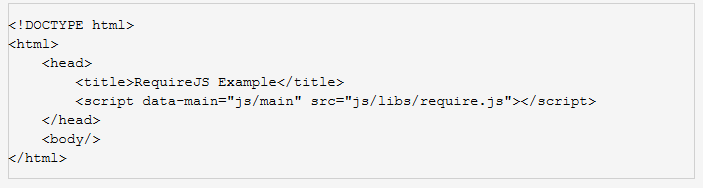


请注意，这个书籍模块有点不同。书籍模块不包含依赖项，因此要简单得多。此代码将大量书架依赖项传递至define，在这个情况下，唯一的依赖项是书籍。第二个参数是回调函数。如果尚未使用RequireJS注册该书籍模块(换句话说，book.js尚未加载至该应用程序)，RequireJS将从服务器获取它。

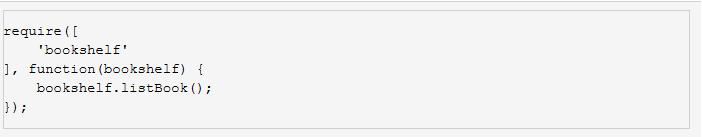
加载了book.js且注册了书籍模块后，RequireJS将执行回调函数并传入该模块(之前定义的书籍对象)作为参数。从技术角度而言，参数名称并不重要，您可以只是简单地命名为function(a1337Book)，也可以使用任何想用的名称，在这种情况下，使用参数名称与模块名称享福很有意义，因为保持一致有助于理解，最后，从此回调函数返回的任何对象均将会通过RequireJS注册为书架模块。在这种情况下，这是指带有listBook()方法的对象，刚刚我们就在使用它在书籍标题上调用alert。当加载多个模块时，RequireJS会尽量提高效率。例如，如果列出多个依赖项，则RequireJS将并行加载所有这些依赖项。

创建应用程序

为开始使用RequireJS及新模块，请创建一个基本HTML页面。下面就是这个页面的外观



毫不夸张地说，通过使用javascript操纵正文内容及加载HTML模版，无需向HTML文件添加任何其他元素即可构建大型但页面应用程序，您也可以使用RequireJS完成此操作。现在，只需要注意data-main属性。这将为RequireJS指出引导文件的所在位置js目录(假设主目录的扩展名为js)。下面是一个main.js文件示例



由于您指定此文件作为HTML文件的data-main，RequireJS将尽快加载并立即执行该文件。

您将会发现它与之前的模块定义具有某些相似之处，但此代码不调用define，而调用require。

Define()函数(当定义依赖项时至少需要)完成以下三个步骤

1. 加载指定的依赖项
2. 调用回调函数
3. 将回调函数返回的值注册为模块

Require()函数仅负责完成步骤1和2.main.js文件就是引导文件。我不需要使用RequireJS注册main模块，因为不会针对它调用任何其他，模块作为依赖项。

Main.js文件列出书架模块作为依赖项。假设尚未使用RequireJS注册书架模块，它将会加载bookshelf.js。当加载了bookshelf.js后，将会发现该书架将书籍模块列外依赖项。如果尚未注册书籍模块，则会加载book.js。操作完成后，书籍和书架均会使用RequireJS将其各自的对象注册为模块。执行main.js回调函数，书架从中通过。这时，您就可以利用书架执行想要执行的一切操作。如果需要，您还可以列出多个模块依赖项，它们会在使用RequireJS加载和注册后立即传入回调函数。

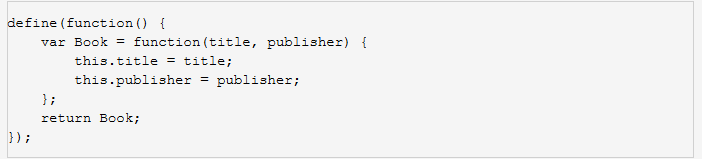
配置URL和路径

默认情况下，基本URL是指目录中包含引导文件的URL。在上面的示例中，main.js与book.js和bookshelf.js一起位于js目录。这意味着基本URL是/js/。您可以不把所有js文件直接放在js目录下，可以将book.js和bookshelf.js移至/js/model/。Main.js文件需要更新

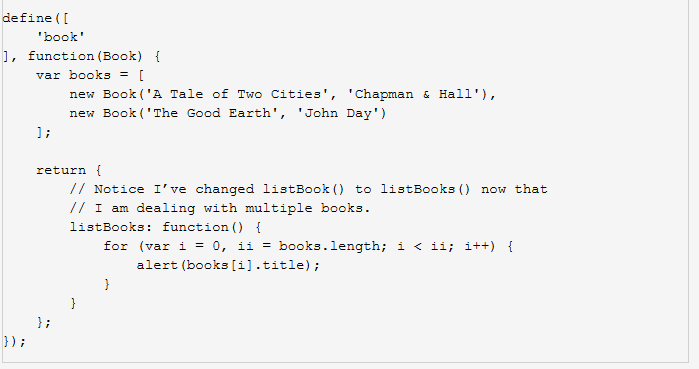


构造函数模块

到目前为止，您所看到的都是对象实例，书架是对象，书籍也是对象。事实上，模块通常就是构造函数(类似于经典语言中的类)。在本例中，您可能希望将书籍模块创建为构造函数。这样，书架即可创建并存储多个书籍对象。此时书籍模块如下所示



在这里请注意，您正在将函数传入define()而不是对象。当您传递函数而非普通对象时，RequireJS将执行该函数，届时该函数返回的所有元素均将成为模块。这里将返回书籍构造函数。现在书架模块显示如下：



优化部署

假设您已经将代码适当分解成粒度模块，则可能会具有数百个文件，除非进行某种优化，否则您的代码将提出数以百计的HTTP请求。

一般来说，您会设置优化器在部署流程期间运行。RequireJS优化器将通过扫描模块及依赖项代码发现应用程序内使用了哪些文件。然后缩小文件(缩小代码以保证真是文件较小)，并将它们连接起来(将它们粉碎再一起组成一个文件)。最后，输出一个包含所有应用程序代码的javascript文件。当您开展部署时，这个单一文件将替代main.js发挥作用。

当用户架子啊index.html时，它将反过来加载RequireJS，然后加载main.js文件。此时main.js文件将不仅包含常规main.js引导代码，而且还包含应用程序其余部分的所有小型级联代码。这是，文件中的所有模块均将使用RequireJS进行注册。当主目录开始探寻依赖项时，这些依赖项也开始寻找依赖项，RequireJS将会意识到所有必要模块均已加载，再次放弃从服务器加载它们。

当然，该优化器自身也包含一组选项。您可以深入几个不同的文件优化应用程序，呈现应用程序的各个部分而不是单个大文件。您还可以使用多个不同的压缩库。从串联中排除文件，甚至缩小CSS

Javascript文件加载:LABjs和RequireJS

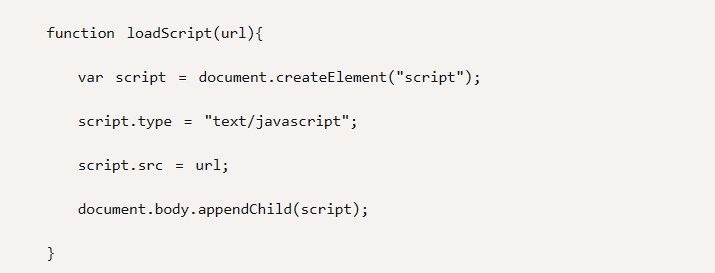
传统上，加载Javascript文件都是使用<script>标签

<script type=”text/javascript” src=”..”></script>

Script标签很方便，只要加入网页，浏览器就会读取并运行。但是，它存在一些严重的缺陷。

1. 严格的读取顺序：由于浏览器按照script在网页中出现的顺序，读取Javascript文件，然后立即运行，导致在多个文件互相依赖的情况下，依赖性最小的文件必须放在最前面，依赖性最大的文件必须放在最后面，否则代码会报错。
2. 性能问题：浏览器采用“同步模式”加载script标签，也就是说，页面会阻塞，等待Javascript文件加载完成，然后再运行后面的HTML代码，当存在多个script标签时，浏览器无法同时读取，必须读完一个再去读取另一个，造成读取时间大大延长，页面响应缓慢

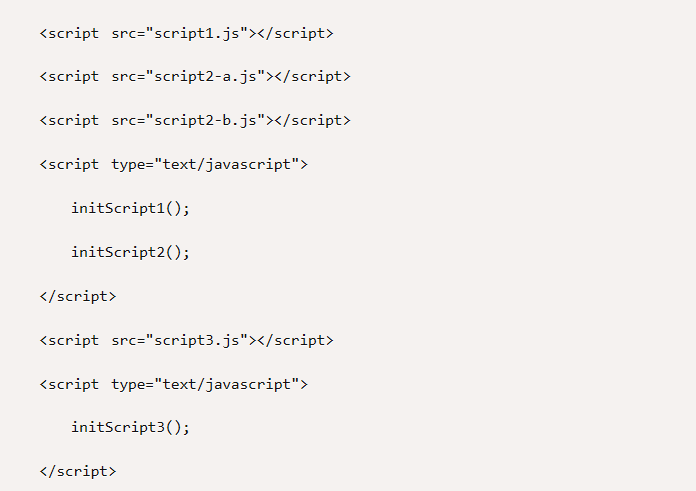
为了解决这些问题，可以使用DOM方法，动态加载Javascript文件



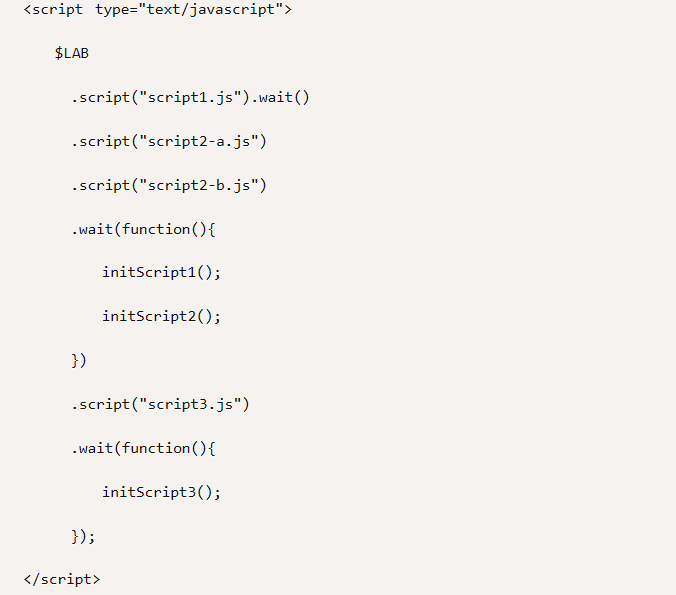
原理：浏览器即使创造出一个script标签，然后“异步”读取Javascript文件。这样不会造成页面堵塞，但会造成另外一个问题：这样加载的Javascript文件，不再原始的DOM结构之中，因此在DOM-ready事件和window.onload事件中指定的回调函数对它无效。

LABjs

例子

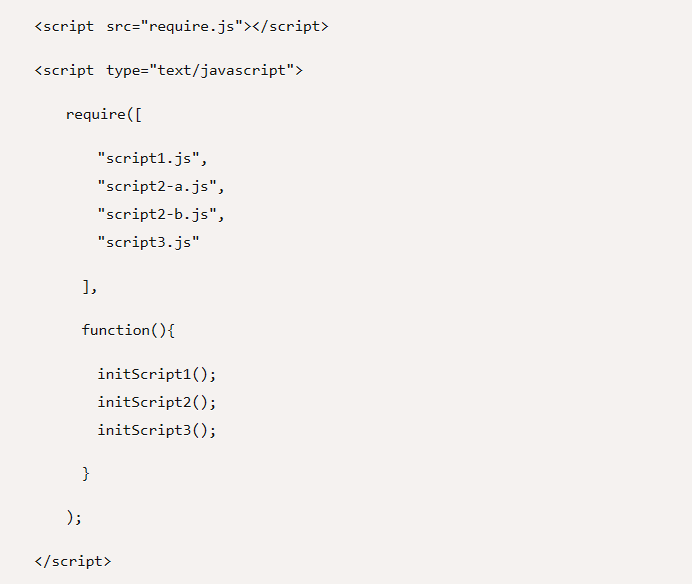


下面，用LABjs对其进行改写



首先，$LAB对象替代了script标签，然后.script()方法表示加载Javascript文件，不带参数的.wait()方法表示立即运行刚才加载的Javascript文件，在参数的.wait()方法也是立即运行刚才加载的Javascript文件，但是还运行参数中指定的函数。

接下来是RequireJS的改写



Require()接受两个参数，第一个数组表示索要加载的Javascript文件，第二个是加载完成后索要运行的回调函数。原生的require()不支持按次序加载，所以四个Javascript文件到底先加载哪个，无法事前知道，require()只保证这个四个文件全部加载完成之后，才会运行所指定的回调函数。

Require.js的加载

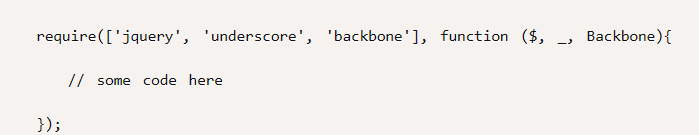
加载这个文件，也可能造成网页失去响应。解决办法有两个，一个是把它放在网页底部加载，另一个是写成下面这样：



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。

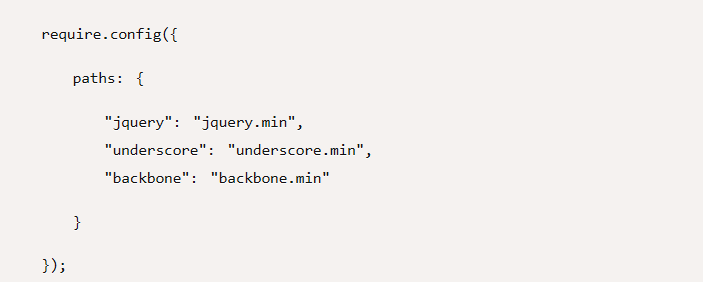
Require函数接收两个参数，第一个参数是一个数组，表示依赖的模块。第二个参数是一个回调函数，当前指定的模块都加载成功后，它将被调用。加载的模块会以参数形式传入函数，从而在回调函数内部就可以使用这些模块。

下面，我们看个实际的例子

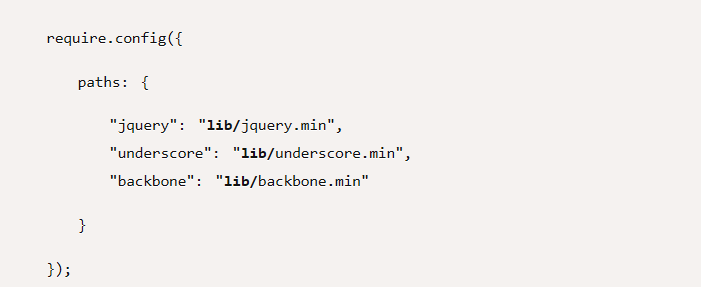


模块的加载

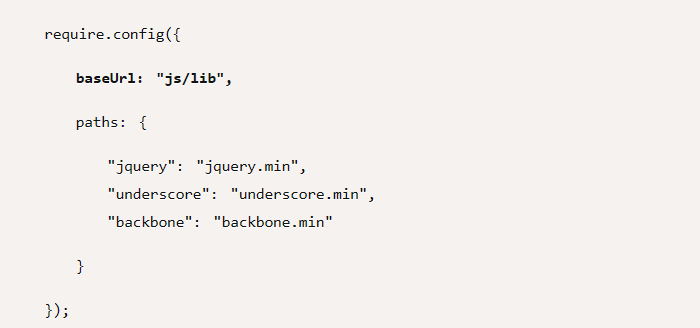
使用require.config方法，我们可以对模块的加载行为进行自定义。Require.config()就写在主模块(main.js)的头部。参数就是一个对象，这个对象的paths属性指定各个模块的加载路径。



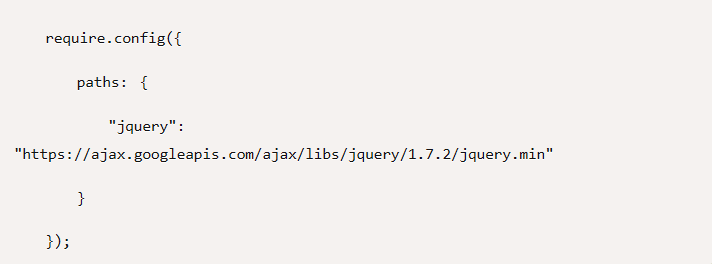
如果这些模块在其他目录，比如js/lib目录，则有两种写法。一种是逐一指定路径



另一种则是直接改变基目录



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

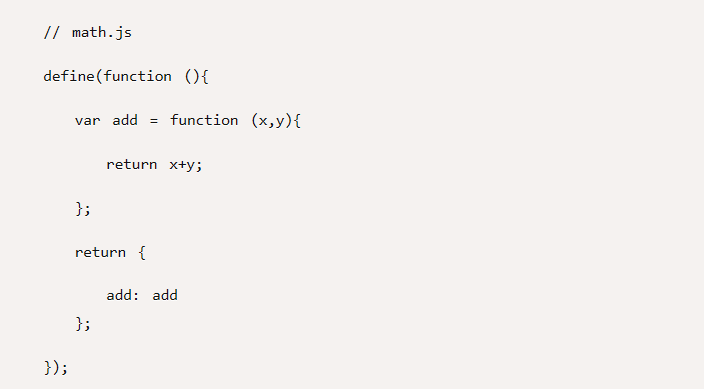


AMD模块的写法

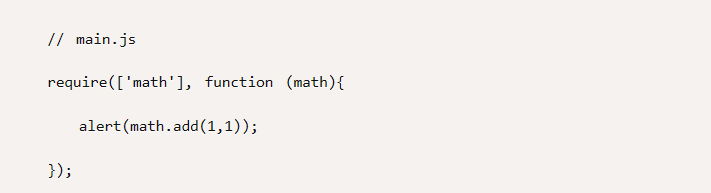
Require.js加载的模块，采用AMD规范。也就是说，模块必须按照AMD的规范来写。

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。

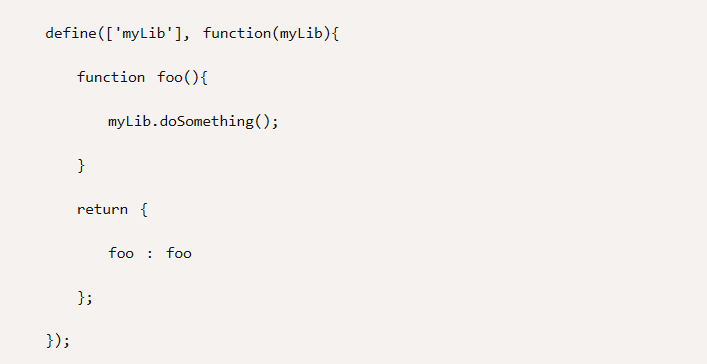
假定现在有一个main.js文件，它定义了一个math模块。那么math.js就要这样写：



加载方法如下



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



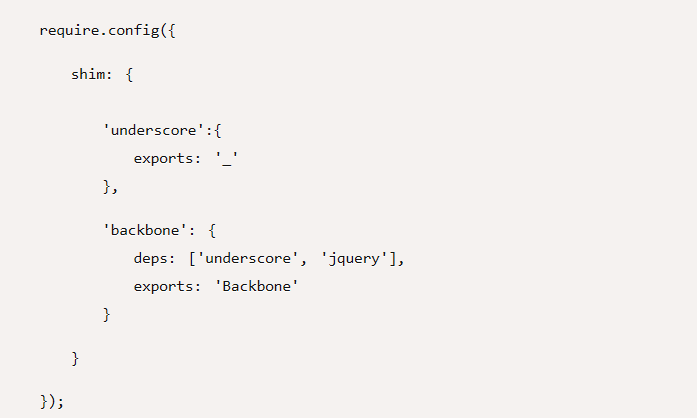
当require()函数加载上面这个模块的时候，就会先加载myLib.js文件

加载非规范的模块

理论上，requireJS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

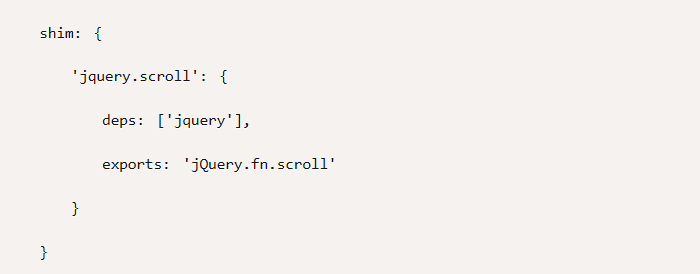
举例来说，underscore和backbone这两个库，都没有采用AMD规范编写。如果要加载它们的话，必须先定义它们的特征。



Require.config()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

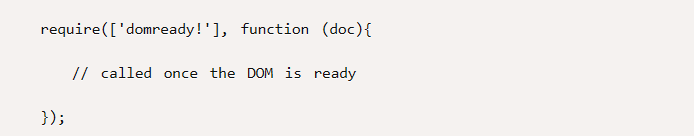
比如jQuery的插件可以这样定义



Require.js插件

Require.js还提供了一系列插件，实现一些特定的功能。

Domready插件，可以让回调函数在页面DOM结构加载完成后再运行



Text和image插件，则是允许require.js加载文本和图片文件



类似的插件还有json和mdown，用于加载json文件和markdown文件。

RequireJS和SeaJS的对比

RequireJS是一个Javascript文件和模块加载器，特别为浏览器优化，同时也可运行在Rhino和Node环境中

SeaJS是一个适用于浏览器端的Javascript模块加载器

RequireJS的定位中，除了是模块加载器，还是文件加载器，这决定了RequireJS需要类似LAB.js等script loader的功能

LABjs的核心功能是异步加载脚本并提供运行的依赖等待：



RequireJS API

1. 用法

RequireJS加载脚本的途径与传统script不同，它的目标是鼓励使用模块化的代码。因为这样可以更快地加载和更好地优化。在模块化的代码中会使用model IDs，而不是script中url。

RequireJS是相对于baseUrl来加载所有的代码，baseUrl通常设置为data-main属性中所引用的脚本的所在目录。Data-main是RequireJS中的一个特殊属性，脚本最初的加载就是它来触发的。

baseUrl也可以通过RequireJS config进行设置，如果没有明确设置baseUrl，并且data-main也没有使用，这时baseUrl就为运行RequireJS的html所在目录

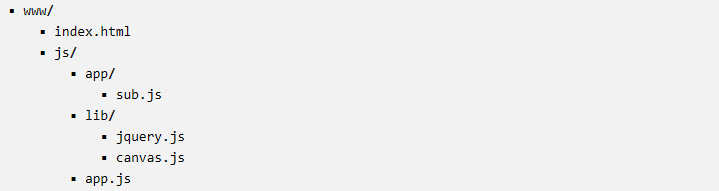
RequireJS默认假设我们依赖的都是脚本，所以module IDs不需要写’.js’后缀，requireJS将module ID转化成路径时会自动加上它。因此，在路径配置中，与传统的script相比，我们可以使用很简短的字符串。

有时你需要直接的引用一个脚本，不必使用’baseUrl+path’规则。此时的module ID要符合下面的规则

1. 以“.js”结尾
2. 以“/”开始
3. 包括URL协议，如“http:”或“https:”

一般来说，最好是用baseUrl和path来设置module IDs，这样做的话，在优化构建时，就可以更灵活地重命名和配置指向不同位置的路径。

同样的，为了避免大量的配置，最好要避免深层次的目录结构，你可以将所有脚本放到baseUrl目录下，或者将你的脚本的第三方的放在不同的文件夹下，例如下面这样：



在index.html中：



在app.js中：



理想情况下，通过define()定义的脚本会被当作为模块。但是，要获得没有通过define()定义的脚本如传统的全局脚本，就需要用到shim config，它可以正当地描述依赖关系

1.2模块的定义

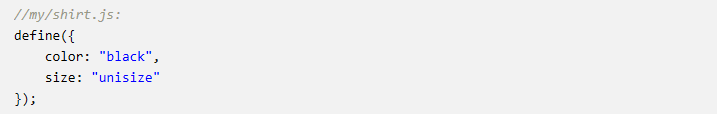
模块与传统的脚本文件不同，它定义了良好作用域的对象，它可以避免污染全局命名空间，可以明确地列出其依赖关系，并且处理依赖时不会设计到global对象，而是将这些依赖作为参数传递给定义模块的函数。RequireJS中的模块是 modulepattern的扩展，优势是引用其他模块时不需要使用global对象。

RquireJS中的模块会尽可能快地被加载，甚至是无序的方式，但是使用时是按照正确的依赖顺序。因为global对象没有被使用，所以在同一个页面中可以为一个模块加载不同的版本。

磁盘上的每一个文件只可以定义一个模块，但是通过优化工具可以将多个模块组织在一起即压缩到一个文件。

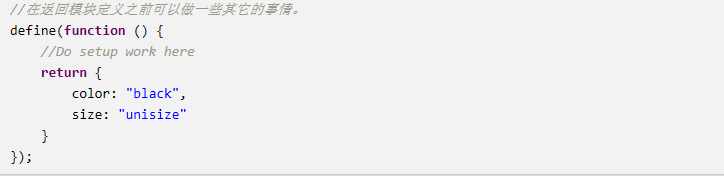
* + 1. 简单的Name Value

如果模块不存在任何依赖，只是一些Name/Value值，这时可以采用创建对象的语法来定义



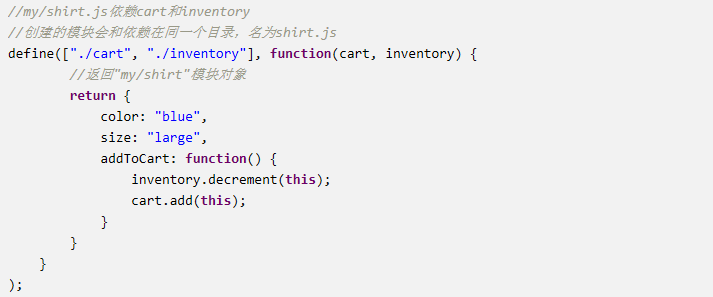
* + 1. 函数定义

如果模块不存在任何依赖，但是需要一个函数来做创建工作，这是可以给define()传递一个函数



* + 1. 函数定义和依赖关系

如果模块存在依赖关系，第一个参数是数组，用来存放依赖名称，第二个参数为函数，用来调用模块，一旦所有依赖加载好之后。该函数应该返回一个对象，就是这个对象定义了该模块。依赖数组作为参数传递给函数，二者的顺序会一一对应。



该例中，my/shirt模块被创建，它依赖my/cart和my/inventory。在磁盘上，文件的阻止结构为：

My/cart.js

My/inventory.js

My/shirt.js

上面函数中的’cart’和’inventory’参数与’./cart’和’./inventory’模块名对应

直到所有的依赖模块加载好之后，该函数才会被调用，并且通过“cart”和“inventory”参数来引用模块。

模块被定义成全局的是不建议这么做，因为有可能在同一个页面针对同一模块会加载不同的版本。并且，函数参数的顺序要与依赖数组对应。

函数中返回的对象定义了“my/shirt”模块。通过这种方式定义的模块，不再global对象中。

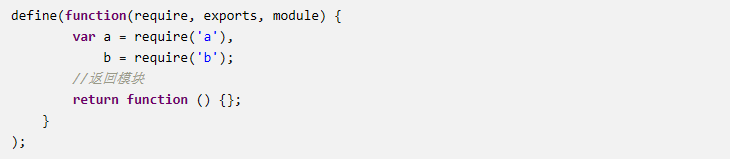
* + 1. 定义模块为函数

模块不一定非得返回对象，函数也是可以的



* + 1. 使用简单的CommonJS Wrapper定义模块

如果你希望重新使用之前通过CommonJS定义的模块，使用上面模块数组的方式比较困难，而且你项通过模块名来引用该模块对象，你可以使用简单的CommonJS Wrapper



这种包装依赖Function.prototype.toString()，来为函数内容提供一个有用的字符串值，但是在一些设备上，如PS3，和一些老的浏览器，它会工作不正常，使用优化器就可以将这些依赖存放为上面提到的数组的形式，这样可以兼容这些设备。

1．2.6 使用名称来定义模块

你或许会遇到一些定义，它包括模块名称，作为define()方法的第一个参数

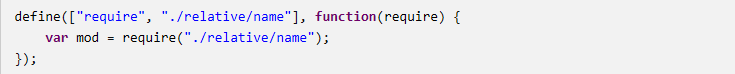


模块名称通常是由优化工具生成的，你可以明确指定模块名，但是这样做使得模块变得移植性很差-如果你将该文件移到其他文件夹，你得修改模块。通常不要自己命名模块，让优化工具去处理。优化工具会加上模块名，然后将各个模块合并为一个文件，这样就可以加载更快。

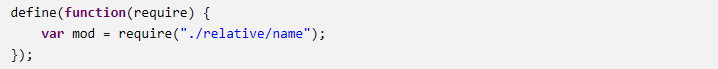
* + 1. 模块的其他知识

一个模块一个文件：每个javascript文件只定义一个模块，这样会给模块赋有name-to-path的查询算法。通过优化工具大量的模块会被合并到一个文件中，但是仅可以通过优化工具来合并。

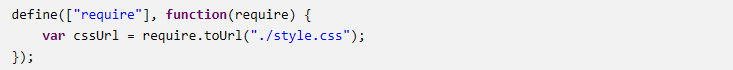
Define()中的相对模块名：例如require(“./relative/name”)可能会在define()函数体内被调用，这样就要求依赖require，以便relative name可以被正确解析。



更好的是采用translating CommonJS模块，可以更简洁地定义：



生成相对某一模块的URL：依赖require，然后调用require.toUrl()方法



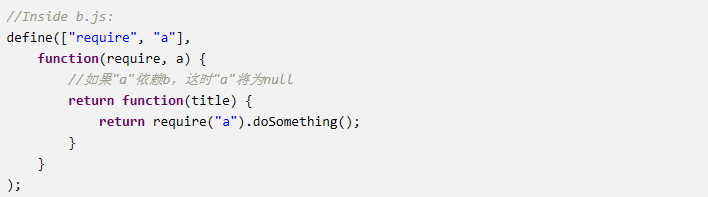
控制台调试：如果你在javascript控制台调用通过require([“module/name”,function(){}])加载的模块，你可以通过为require()指定模块名来进行加载：



注意：这只会在通过require([“module/name”])的异步加载的方式下工作，如果使用相对路径，例如:”./module/name”，这只会在define内执行

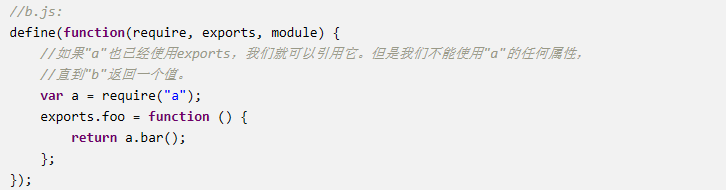
* + 1. 循环依赖

如果你定义了一个循环依赖(a依赖b，b依赖a)，在这种情况下，当b模块中的函数执行时，a的值为undefined。b可以在当模块使用require()定义之后去获取(一定要加上require的依赖，才能保证正确的上下文去寻找a)

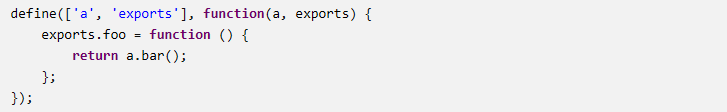


通常你不需要使用require()来获取模块，而是将模块作为参数传递给函数。循环依赖是少数情况，这意味着你需要重新去考虑设计，但是有时它是必须的，此时就需要用到require()

如果你对CommonJS模块比较熟悉，你可以使用exports为模块创建一个空的对象，它就可以立即被其他模块引用。为循环依赖的两端都使用这种做法，在其他模块中就可以很安全地使用。这种情况只适合为模块返回对象，而不是函数



如果使用数组形式，需要依赖”exports”



* + 1. 指定JSONP服务的依赖

JSONP是javascript调用服务的一种方式。它是跨域的，是通过script脚本的http get请求来建立的，在RequireJS中使用JSONP，需要指定callback参数的值为define，可以通过JSONP的url来得到值，就好像它是模块的定义。

下面是调用JSONP API端点的例子，在这个例子中，JSONP的回调函数就是为callback，所以callback=define，告诉API使用define()将JSON格式的返回对象包装起来。



JSONP的使用应该被限制在用JSONP服务来初始化程序的创建。如果JSONP服务超时，通过define()定义的其他模块可以能将不会被执行。所以这样的错误机制是不健全的。

JSONP的返回值只支持JSON对象。JSON返回数组，字符串或数字将不会执行。

此功能不应该使用在长轮询的JSONP连接-APIs用来处理实时数据流。这种类型的APIs应该在接收到响应后进行脚本清理，而RequireJS仅仅获取一次JSONP URL，以后在require()或者define()中使用相同的URL作为依赖时会得到一个缓存值。

在加载一个JSONP服务时，错误常常发生在超时，因为script tag没有提供网络问题更多的细节。为了错误检测，你可以重写require.onError()

* + 1. 取消模块的定义

全局函数require.undef()可以用来取消模块的定义。它会充值加载器内部的状态，会忽略之前对该模块的定义。

但是它不会再已经依赖模块的其他模块中把它移除。所以可以用在一些错误处理上，当没有其他模块依赖的情况下，或者之后的模块会依赖它。

如果你想为取消定义做更复杂的依赖关系图分析。

1. 内部机制

RequireJS加载的每个模块作为script tag，使用head.appendChild()。

在模块的定义时，RequireJS等到所有的依赖都加载完毕，会为函数的调用计算出正确的顺序，然后再函数中通过正确的顺序进行调用。

在拥有同步加载的服务器端javascript环境中使用RequireJS，重新定义require.load()会比较容易。构建系统也是如此，环境中的require.load方法可以在build/jslib/requirePatch.js中找到。

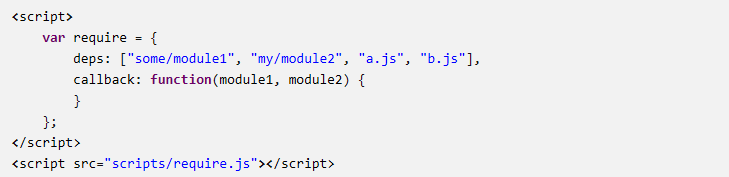
将来，它可能会被作为可选模块拉到require/dirctory，你可以在你的环境中加载，就会基于主机环境得到正确的加载行为。

3．配置

在最高级的HTML页面(或者是最高级的不是定义模块的script脚本)中使用require()，可以传一个配置对象：



你也可以在require.js未被加载之前，用全局变量require来定义配置对象，这些配置信息就会被自动应用。



注意:最好使用var require ={}，不要使用window.require = {} ，因为它在IE中工作不正常。

支持的配置选项

baseUrl：所有模块查询所基于的根路径。在上面的例子中，”my/module”对应的script中的src值就"/another/path/my/module.js“。baseUrl不会加载js文件，所以a.js和b.js会从相对该脚本当前目录直接加载。

如果在配置中没有明确地设置baseUrl，它的默认值就为加载require.js的html的目录所在的位置。如果指定了data-main属性值，则该路径就为baseUrl。

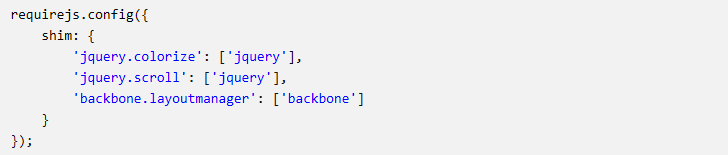
在加载require.js页面中baseUrl的URL可以是不同域的。RequireJS的脚本加载可以进行跨域。唯一的限制是加载文本的text插件：至少在开发环境中，他们的路径应该和当前页面在同一个域下。在使用优化工具之后，它就会在text插件资源里，你在其它的域可以引用text插件资源来使用该资源了。

Paths:是为没有直接在baseUrl中找到模块名而作的路径映射。路径的设置相对于baseUrl，除非使用”/”或者URL协议，用上面的例子说明：”some/module”的script的src为”/another/path/some/v1.0/module.js”,被用作模块名的路径中不应该包括.js，因为路径是针对文件夹的，将模块名映射为路径会自动加上’.js’。

Shim：可以为没有使用define来申明依赖的定义模块的那种传统的浏览器全局脚本进行配置和导出：



那些不需要导出任何模块值，而仅仅是jQuery或backbone的插件，可以在shim配置数组依赖：



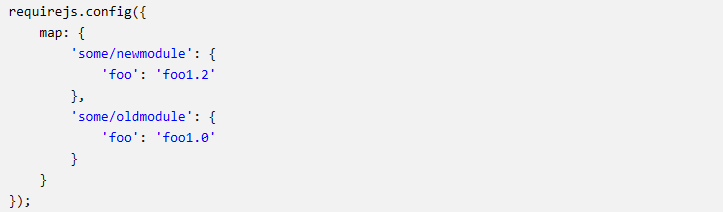
在IE中如果想要404检测，可以paths fallbacks或errbacks，这时需要exports一个字符串值，以便用来检测脚本是否被正确加载：



Map：对于给定的模块前缀，就不会使用提供的ID来加载模块了，是用新的ID来代替。

这种分类能力在大的项目中尤为重要，你或许会用到两个不同版本的foo模块，它们之间不会造成冲突。这是在基于上下文的多版本支持是不可能的。另外paths配置只是为模块ID创建根路径，而不是将一个模块ID映射到另一个模块。

Map例子



如果模块在磁盘存储的形式如下：

Foo1.0.js

Foo1.2.js

Some/

Newmodule.js

Oldmodule.js

“some/newmodule”通过require(‘foo’)会获得foo1.2.js，some/oldmodule会获得foo1.0.js

这个特性只会在调用define的AMD模块，并注册为匿名模块时才可以使用。

可以支持”\*”通配符，所有模块都会用该map配置，如果匹配了特殊配置，则使用该特殊配置，例如：

