Sea.js

模块定义

在SeaJS中，所有javascript文件都应该用模块的形式来书写，并且一个文件只包含一个模块。

Define

使用全局函数define来定义模块



Id

当前模块的唯一标识。该参数可选。如果没有指定，默认为模块所在文件的访问路径。如果指定的话，必须是顶级或绝对标识(不能是相对标识)

Dependencies

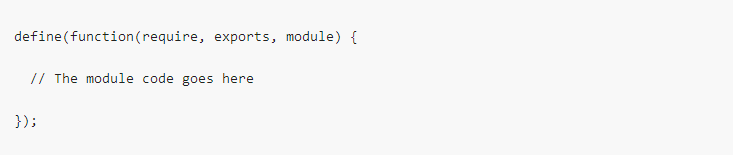
当前模块所依赖的模块，是一个由模块标识组成的数组。该参数可选。如果没有指定，模块加载器会从factory.toString()中解析出该数组。

注意：强烈推荐不要设定id和dependencies参数。在开发阶段，模块加载器会自动获取这两个参数。部署上线时，则可以通过优化工具来提取这两个参数。

Factory

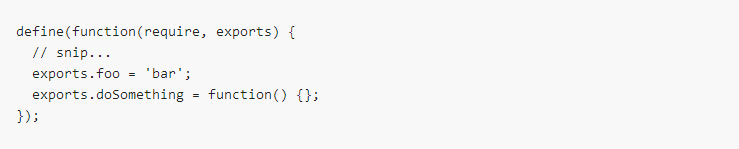
模块的工厂函数。模块初始化时，会调用且仅调用一次该工厂函数。Factory可以是函数，也可以是对象，字符串等任意值，这时module.exports会直接设置为factory值。

Factory函数在调用时，会始终传入三个参数：require,exports和module，这三个参数在所有模块代码里可用。

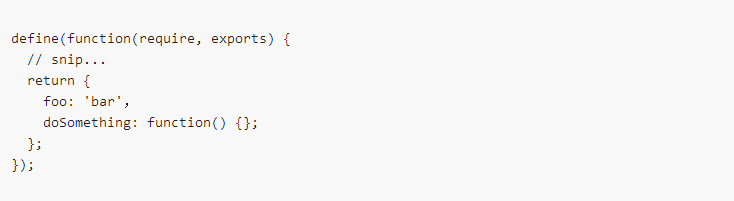


Exports

Exports用来向外提供模块的API



除了给exports对象增加成员，还可以使用return直接向外提供API。

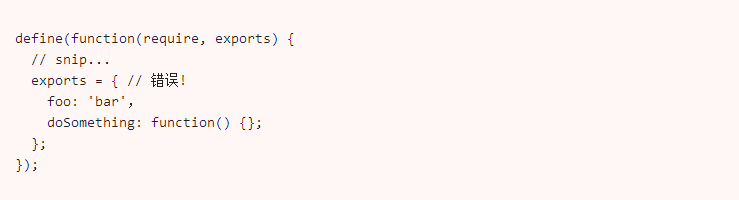


如果return语句是模块中的唯一代码，可简化为：



上面这种格式特别适合定义JSON数据。

注意：下面这种写法是错误的！



模块加载器不能获取到新赋给exports变量的值。请使用return或module.exports。

Require

Require函数用来访问其他模块提供的API



它接受模块标识作为唯一参数。

请记牢，为了使静态分析能成功获取到模块依赖信息，在书写模块时，需要遵循一些简单的规则

书写规范

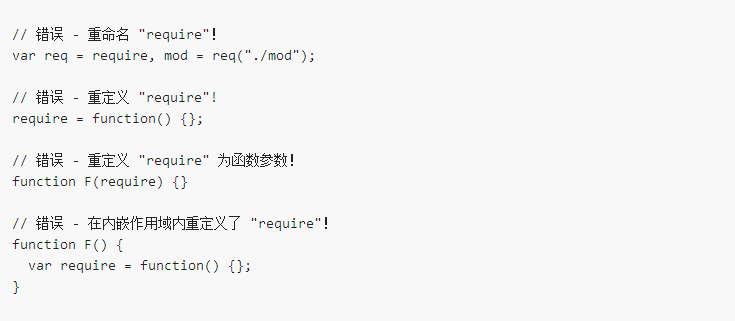
1. 正确拼写

在模块代码中，第一个参数必须命名为require



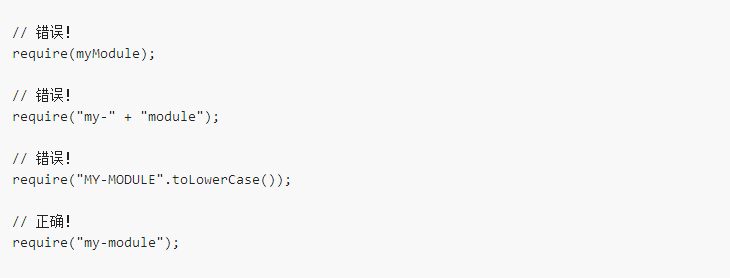
2．不要修改

不要重命名require函数，或在任何作用域中给require重新赋值



1. 使用直接量

Require的参数值必须是字符串直接量



在书写模块代码时，必须遵循这些规范。其实只要把require看做是语法关键字就好啦。

动态依赖的小提示

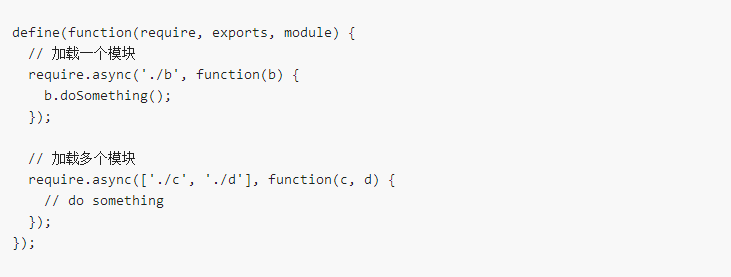
偶尔会希望可以使用require来进行条件加载：



但请牢记，从静态分析的角度来看，这个模块同时依赖play和work两个模块，加载器会把这两个模块文件都下载下来。这种情况下，推荐使用require.async来进行条件加载。

Require.async

该方法可用来异步加载模块，并在加载完成后执行回调函数。



Require.resolve

使用require()的内部机制来解析并返回模块路径。该函数不会加载模块，只返回解析后的路径。

Require.load

该方法可用来异步加载脚本，并在加载完成后，执行指定的回调函数。开发插件时，可以通过覆盖该方法来实现自定义的资源加载。

Require.constructor

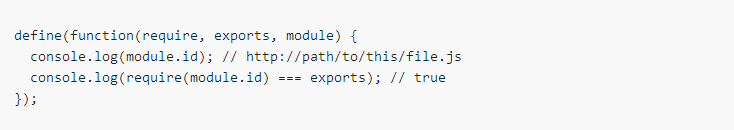
有时候，我们需要给所有require参数对象添加一些公用属性或方法。这时，使用require.constryctor来实现会非常方便。

Module

Module参数存储模块的元信息。拥有以下成员：

Module.id

当前模块的唯一标识。Require(module.id)必然返回此模块的exports。



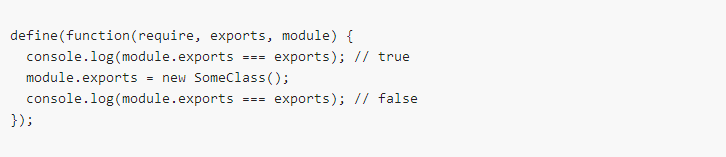
Module.dependencies

Module.dependencies是一个数组，标识当前模块的依赖列表。

该数组只读：模块加载完成后，修改该数组不会有任何效果。

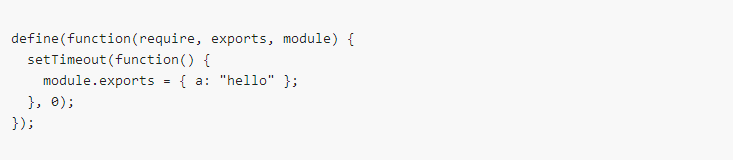
Module.exports

Exports对象由模块系统创建，这不能满足开发者的所有需求，有时候希望exports是某个类的实例。这时可用module.exports来实现

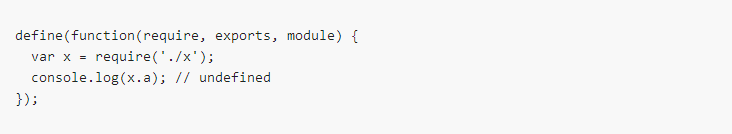


注意，对module.exports的赋值需要同步执行，它不能放在回调函数里。下面这样是不行的：

X.js



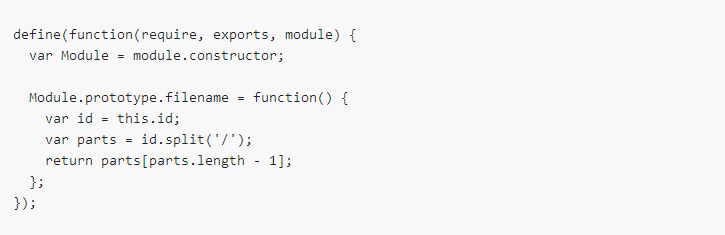
y.js



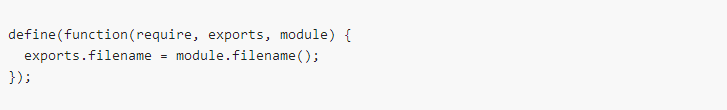
Module.constructor

有时候，我们需要给所有module参数对象添加一些公用属性或方法。在这种情况下，使用module.constructor可以很好地满足需求。

Extend.js



1. js



前言：

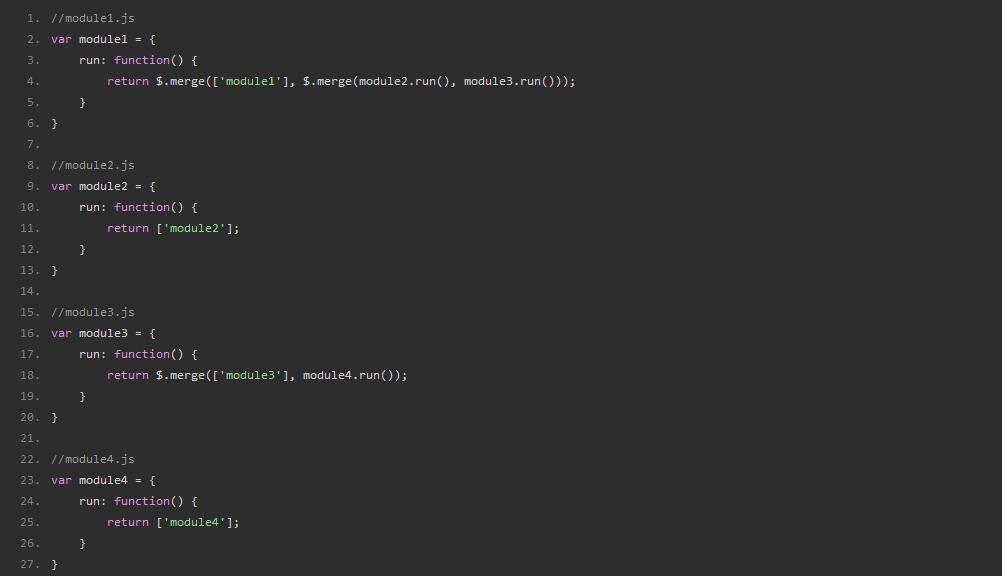
SeaJS是一个遵循CommonJS规范的javascript模块加载框架，可以实现javascript的模块开发及加载机制。与jQuery等javascript框架不同，SeaJS不会扩展封装语言特性，而只是实现javascript的模块化及按模块加载。SeaJS的主要目的是令javascript开发模块化并可以轻松愉悦进行加载，将前端工程师从繁重的javascript文件及对象依赖处理中解放出来，可以专注代码本身的逻辑。

SeaJS可以与jQuery这类框架完美集成。使用SeaJS可以提高javascript代码的可读性和清晰度，解决目前javascript编程中普遍纯在的依赖关系混乱和代码纠缠等问题，方便代码的编写和维护。

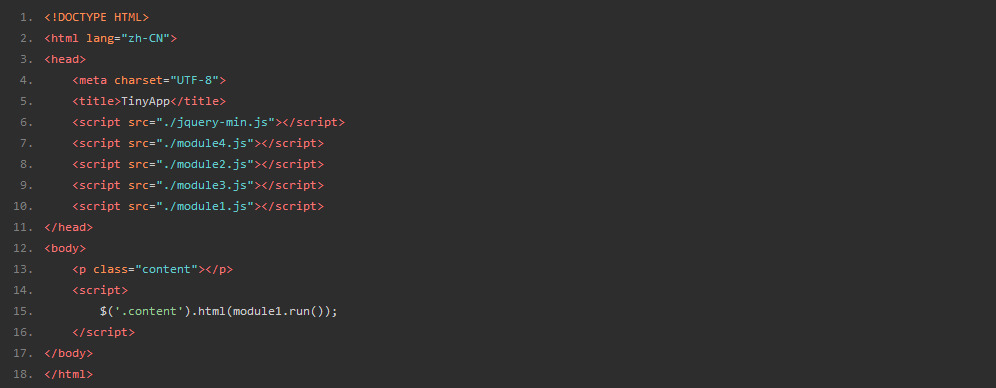
传统模式VSSeaJS模块化

假设我们现在正在开发一个web应用，我们决定在应用使用jQuery框架。应用的首页会用到module1.js,module1.js依赖module2.js和module3.js,同时module3.js依赖module4.js

传统开发



此时index.html需要引用module1.js及其所有下层依赖(注意顺序)

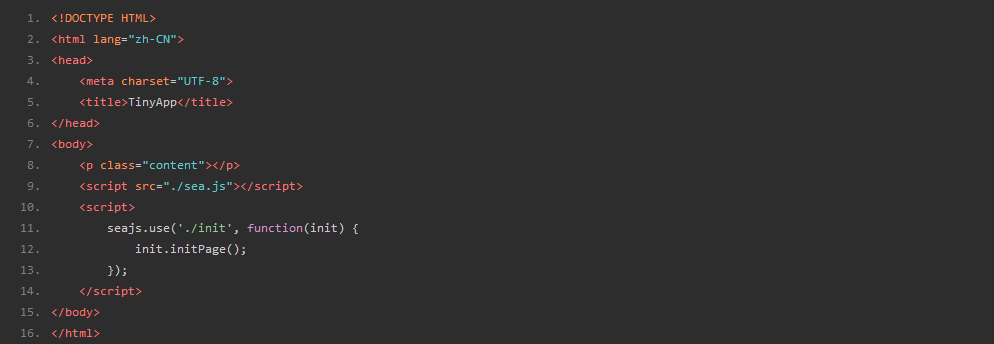


随着项目的进行，js文件会越来越多，依赖关系也会越来越复杂，使得js代码和html里的script列表变得难以维护。

SeaJS模块化开发

下面看看如何使用SeaJS实现相同的功能。

首先是index.html：



可以看到tml页面不再需要引入所有依赖的js文件，而只是引入一个sea.js，sea.js会处理所有依赖，加载相应的js文件，加载策略可以选择在渲染页面时一次性加载所有js文件，也可以按需加载(用到时才加载响应js)，具体加载策略使用方法下文讨论。

Index.html加载了init模块，并使用此模块的initPage方法初始化页面数据，这里先不讨论代码细节。

SeaJS的特性：

1. Html页面不用再维护长的script标签列表，只要引入一个sea.js即可
2. Js代码以模块进行组织，各个模块通过require引入自己依赖的模块。

SeaJS基本开发原则

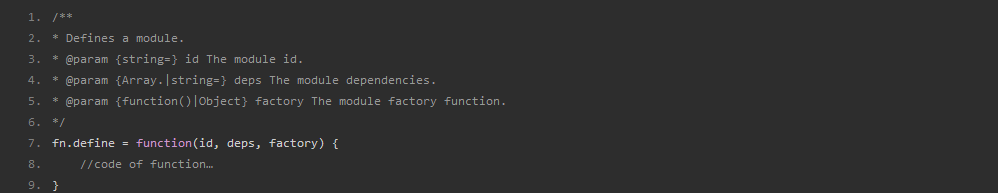
在讨论SeaJS的具体使用前，先介绍一下SeaJS的模块化理念和开发原则。

使用SeaJS开发javascript的基本原则就是：一切皆为模块。引入Sea.js，编写javascript代码就变成了编写一个又一个模块，SeaJS中模块的概念有点类似面向对象中的类模块可以拥有数据和方法，数据和方法可以定义为公共或私有，公共数据和方法可以提供别的模块调用。另外，每个模块应该都定义在一个单独js文件中，即一个对应一个模块。

模块的定义及编写

模块定义函数define

SeaJS中使用”define”函数定义一个模块。因为SeaJS的文档并没有关于define的完整参考，所以我们阅读了SeaJS源代码，发现define可以接收三个参数。



上面是我从SeaJS源码中摘录出来的，define可以接收的参数分别是模块ID，依赖模块数组及工厂函数。我阅读源代码后发现define对于不同参数个数的解析规则如下：

如果只是一个参数，则赋值给factory

如果有两个参数，第二个赋值为factory；第一个如果是array则赋值给deps，否则赋值给id。

如果有三个参数，则分别赋值给id，deps和factory。

但是，包括SeaJS的官方实例在内几乎所有用到define的地方都只传递一个工程函数金曲，类似与如下代码：



个人建议遵循SeaJS官方实例的标准，用一个参数的define定义模块。那么id和deps会怎么处理呢？

Id是一个模块的标识字符串，define只有一个参数时，id会被默认赋值为此js文件的绝对路径。如example.com下的a.js文件中使用define定义模块，则这个模块的ID会赋值为<http://example.com/a.js>，没有特别的必要建议不要传入id。deps一般也不需要传入，需要用到的模块用requir加载即可。

工厂函数factory解析

工厂函数是模块的主题和重点。在只传递一个参数给define时(推荐写法)，这个参数就是工厂函数，此时工厂函数的三个参数分别是：

1. require – 模块加载函数，用于记载依赖模块
2. exports – 接口点，将数据或方法定义在其上则将其暴露给外部调用
3. module – 莫的元数据

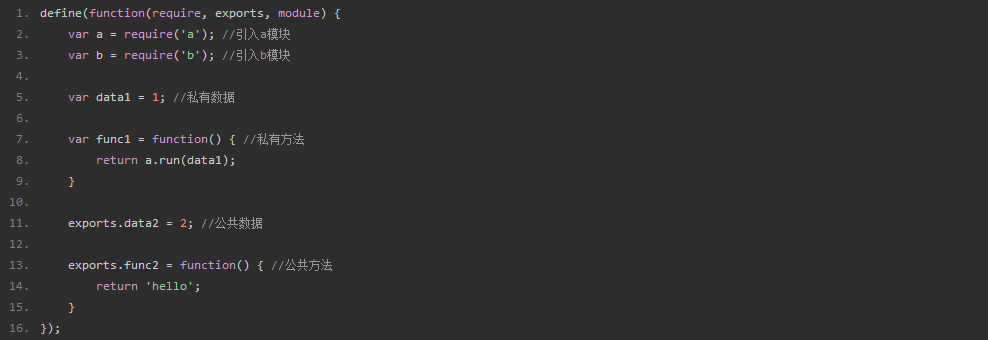
这三个参数可以根据需要选择是否需要显示指定。

下面说一下module。Module是一个对象，存储了模块的元信息，具体如下：

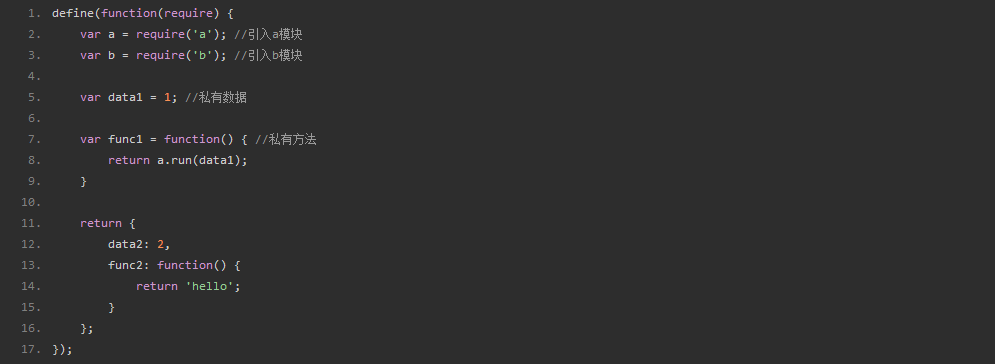
1. module.id – 模块的ID
2. module.dependencies – 一个数组，存储了此模块依赖的所有模块的ID列表。
3. module.exports – 与exports指向同一个对象。

三种编写莫的模式

第一种定义模块的模式是基于exports的模式



上面是一种比较“正宗”的模块定义模式。除了将公共数据和方法附加在exports上，也可以直接返回一个对象表示模块，如下面的代码与上面的代码功能相同：



如果模块定义没有其他代码，只返回一个对象，还可以如下简化写法



第三种方法对于定义纯JSON数据的模块非常适合。

模块的载入和引用

模块的寻址算法

上文说过一个模块对应的一个js文件，而载入模块时一般都是提供一个字符串参数告诉载入函数需要的模块，所以就需要有一套从字符串标识到实际模块所在文件路径的解析算法。SeaJS支持如下标识：

绝对地址-------给出js文件的绝对路径。

如



就代表载入<http://example.com/a.js>

相对地址----用相对调用载入函数所在js文件的相对地址寻找模块。

例如在<http://example/js/b.js>中载入



则砸入<http://example.com/c.js>。

基址地址—如果载入字符串标识既不是绝对路径也不是以”./”开头，则相对Seajs全局配置中的”base”来寻址，这种方法稍后讨论。

注意上面在载入模块时都不用传递后缀名”.js”，SeaJS会自动添加”.js”。但是下面三种情况下不会添加：

载入css时，如



路径中含有？时，如



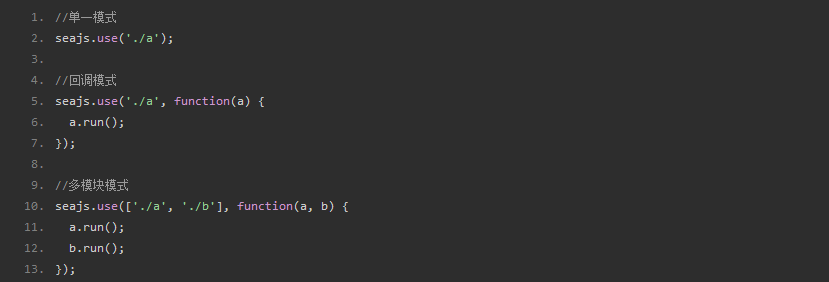
路径以”#”结尾时，如：



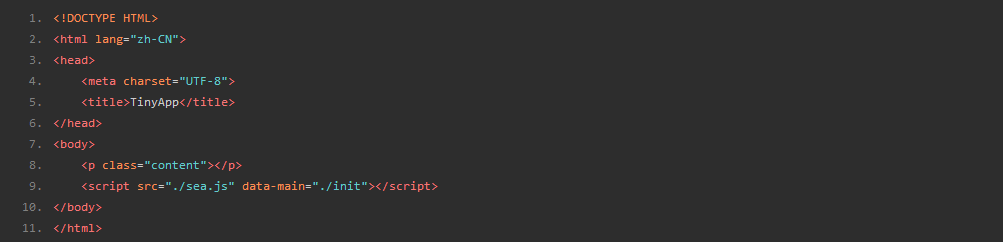
根据应用场景的不同，SeaJS提供了三个载入模块的API，分别是SeaJS.use,require和require.async，下面分别介绍。

SeaJS.use

seaJS.use主要用于载入入口模块。入口模块相当于c程序的main函数，同时也是整个模块依赖树的根。上面在TinyApp小例子中，init就是入口模块。seaJS用法如下



一般SeaJS.use只用在页面载入入口模块，SeaJS会顺着入口模块解析所有依赖模块并将它们加载。如果入口模块只有一个，也可以通过给引入SeaJS的script标签加入data-main属性来省略SeaJS.use，例如，上面TinyApp的index.html也可以改为如下写法：



这种写法令html更加简洁。

Require

Require是Seajs主要的模块加载方法，当在一个模块中需要用到其他模块时一般用require加载：



这里简要介绍一下Seajs的自动加载机制。上文说过，使用seajs后html只要包含sea.js即可，那么其他js文件是如何加载进来的呢？SeaJS会首先下载入口模块，然后顺着入口模块使用正则表达式匹配代码中所有的require，再根据require中的文件路径标识下载相应的js文件，对下载来的js文件再迭代进行类似操作。整个过程类似图的遍历操作(因为可能存在交叉循环依赖所以整个依赖数据结构是一个图而不是树)

明白了上面这一点，下面的规则就很好理解了：

传给require的路径标识必须是字符串字面量，不能是表达式，如下面使用require的方法是错误的



这都会造成Seajs无法进行正确的正则匹配以下下载相应的js文件。

Require.async

上文说过Seajs会在html页面打开时通过静态分析一次性记载所有需要的js文件，如果项要某个js在用到时才下载，可以使用require.async：



这样只有在用到这个模块时，对应的js文件才会被下载，也就实现了javascript代码的按需加载。

SeaJS的全局配置

SeaJS提供了一个seajs.config方法可以设置全局配置，接受一个标识全局配置的配置对象。具体使用方法如下：



其中base标识基址寻址的基址路径。例如base设置为<http://example.com/3-party/>，则



会载入<http://example.com/3-party/jquery.js>

Alias可以对较长的常用路径设置缩写

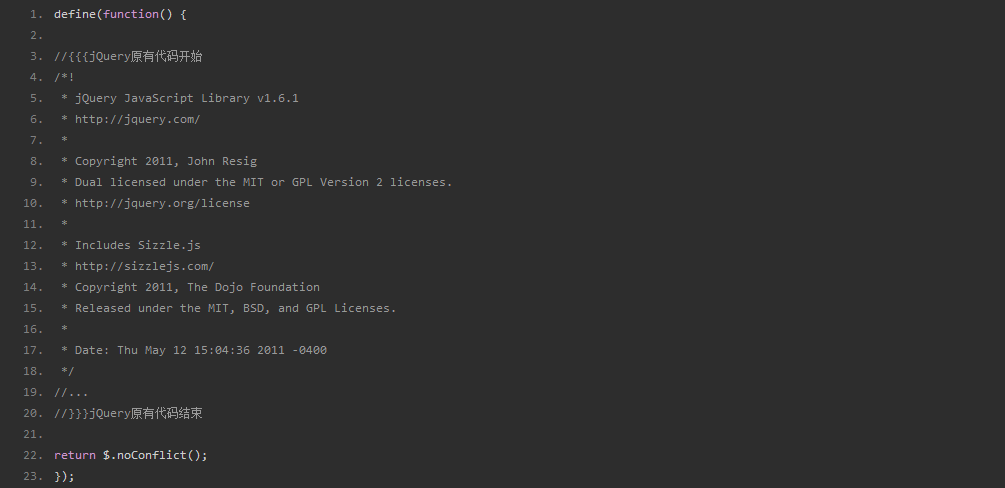
Charset 标识下载js时script标签的charset属性。

Timeout标识下载文件的最大时长，以毫秒为单位。

Debug 标识是否工作在调试模式下

SeaJS如歌与现有js库配合使用

要将现有js库如jQuery与SeaJS一起使用，只需根据SeaJS的模块定义规则对现有库进行一个封装。例如，下面是对jQuery的封装方法：



SeaJS项目的打包部署

SeaJS本来集成了一个打包部署工具spm，后来作者为了更kiss一点，将spm拆出了SeaJS而成为一个单独的项目。Spm的核心思想是将所有模块的代码都合并压缩后并入入口模块，由于SeaJS本身的特性，html不需要做任何改动就可以很方便的在开发环境和生产环境间切换。但是由于spm目前并灭有发布正式版本，所以本文不打算详细介绍，有兴趣的朋友可以参看其github项目主页

其实，由于每个项目所用的JS合并和压缩工具不尽相同，所以spm可能并不是完全适合每一个项目。在了解Seajs原理后，完全可以自己写一个符合自己项目特征的合并打包脚本。

一个完整的例子

上文说了那么多，知识点比较分散，所以最后我打算用一个完整的SeaJS列子把这些知识点串起来，方便朋友们归纳回顾。这个例子包含如下文件：

Index.html --- 主页面

Sea.js – SeaJS脚本

Init.js – init模块，入口模块，依赖data，jQuery，style三个模块。由主页面载入

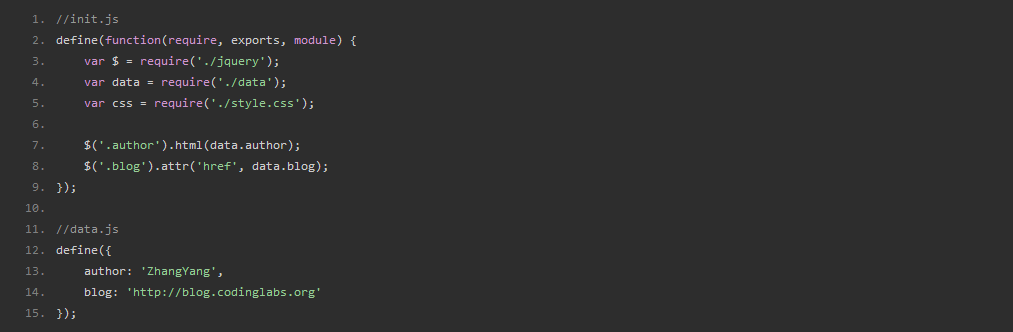
Data.js – data模块，纯json数据模块，由init载入

Style.css – css样式表，作为style模块由init载入。

Sea.js和jQuery。Js的代码属于库代码，这里只给出自己编写的文件代码



Javascript：



Css：

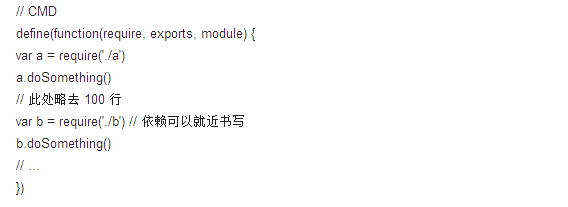


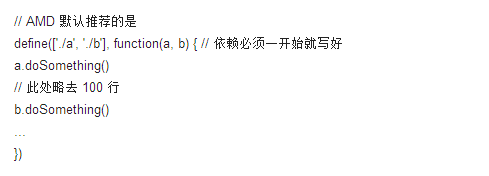
AMD和CMD的区别有哪些？

这些规范的目的都是为了javascript的模块化开发，特别是在浏览器端的。目前这些规范的实现都能达成浏览器端模块化开发的目的。

区别：

1. 对于依赖的模块，AMD是提前执行，CMD是延迟执行。不过RequireJS从2.0开始，也改成可以延迟执行(根据写法不同，处理方式不同)。CMD推崇as lazy as possible
2. CMD推崇依赖就近，AMD推崇依赖前置。看代码：





虽然AMD也支持CMD的写法，同时还支持require作为依赖传递，但RequireJS作者默认最喜欢上面的写法，也是官方文档里默认的模块定义写法。

1. AMD的API默认是一个当多个用，CMD的API严格区分，推崇职责单一。比如AMD里，require分全局require和局部require，都叫require。CMD里，没有全局require，而是根据模块系统的完备性，提供sea.use来实现模块系统的加载。CMD里，每一个API都简单纯粹。
2. 还有一些细节差异，具体看这个规范的一定，就不多说了。
3. AMD需要等待所有js都加载下来以后，才能正确执行所有依赖，CMD是什么时候用到，什么时候加载。

Javascript模块规范：AMD和CMD的区别

定义：

异步模块定义(AMD)是Asynchronous Module Definition的缩写，是RequireJS推广过程中对模块定义的规范化产出

通过模块定义(CMD)是Common Module Definition的缩写，是SeaJS在推广过程中对模块定义的规范化产出

不同之处

两者的主要区别如下：

1. 定位有差异：RequireJS想成为浏览器端的模块加载器，同时也想成为Rhino/Node等环境的模块加载器。SeaJS则专注于web浏览器端，同时通过node扩展的方式可以很方便跑在Node环境中。
2. 遵循的规范不同：RequireJS遵循AMD(异步模块定义)规范。SeaJS遵循CMD(通用模块定义)规范。规范的不同，导致了两者API不同。SeaJS更贴近CommonJS Modules/1.1和node Modules 规范
3. 推广理念有差异：RequireJS在尝试让第三方类库修改自身来支持RequireJS，目前只有少数社区采纳。SeaJS不强退，采用在自主封装的方式来”海纳百川”，目前已有比较成熟的封装策略。
4. 对开发调试的支持有差异：SeaJS非常关注代码的开发调试，有nocache、debug等用于调试的插件。requireJS无这方面的明显支持。
5. 插件机制不同：RequireJS采取的是在源码中预留接口的形式，插件类型比较单一。SeaJS采取的是通过事件机制，插件类型更丰富
6. 还有不少差异，涉及具体使用方式和源码实现，欢迎有兴趣者研究并发表看法，总之如果说RequireJS是prototype类库的话，则Sea.js致力于成为jQuery类库



Base:有一个默认的值，它是sea.js在你项目中的位置，也就是说在默认情况下，所有模块都是相对sea.js的路径来解析。

当设置了base后，你require或者require.async一个模块的时候，如果是采用顶级标识(不以.和/开头)来载入，那么base设定的路径对模块起作用，而以.和/开头载入的模块，读取的模块是以相对当前模块的路径来解析。注意：base还会对alias配置起作用，这一点作者并没有提到。

Alias：

别名，很好理解。有时候项目比较大，各模块间划分比较细，目录文件比较多，深，这个时候可以给某一个模块或者一组模块起一个别名，用来简化模块的标识，同时也方便模块的迁移。

Preload：

Preload是配置需要提前加载并初始化的模块，或者预加载的文件。根据作者的原话，提前加载的模块，在use的时候才会加载，但是本人测试的时候，只是建立config文件后并没有use模块，但是在控制面板里preload设置的模块和文件均会发送请求并加载，不知是否后续版本有变动。

还有一点要注意，在preload文件的时候(特别是像jq库)要考虑其他文件内容对preload的文件依赖。比如，reload里预加载jq库文件(注意是库文件，不是改动后的jq模块)，然后某一个模块里require('jqeury-plug/fancybox/jquery.fancybox-1.3.4.pack')这个时候，需要考虑解析和加载的问题了，由于在本地速度相当快，有的时候就是fancybox加载了开始执行的时候jq库文件却还没加载好，结果就导致错误出现了，关于使用jq模块和jq库的问题后面再讨论。

Debug

Debug有3个值可选：0|1|2

0：普通状态，你就当啥也没发生

1：调试状态，combo会失效，其他和0的时候一样

2：无缓存状态，所有请求都会自动在后面添加一个时间戳，其他和1的时候一样

值为2的时候在2.0.0里没有表现出来。

Charset

Map

简单的说是文件映射，在上面config.js里做的映射起作用就是控制版本号，而除了这个功能外，在开启seajs-debug状态下map将发挥更大的作用。