Sea.js 学习笔记 - 羽翼飞扬

时间 2014-02-14 16:32:00  [博客园-所有随笔区](http://www.tuicool.com/sites/QbYvau)

原文  [http://www.cnblogs.com/yuyifeiyang/p/3549791.html](http://www.cnblogs.com/yuyifeiyang/p/3549791.html?utm_source=tuicool&utm_medium=referral)

主题 [SeaJS](http://www.tuicool.com/topics/11060081)[HTML](http://www.tuicool.com/topics/11060000)

Sea.js 追求简单、自然的代码书写和组织方式，具有以下核心特性：

* **简单友好的模块定义规范** ：Sea.js 遵循 [***CMD***](https://github.com/cmdjs/specification/blob/master/draft/module.md) 规范，可以像 [***Node.js***](http://nodejs.org/) 一般书写模块代码。
* **自然直观的代码组织方式** ：依赖的自动加载、配置的简洁清晰，可以让我们更多地享受编码的乐趣。

Sea.js 还提供常用插件，非常有助于开发调试和性能优化，并具有丰富的可扩展接口。

**一、引入sea.js**

    在调用 seajs.use 之前，需要先引入 sea.js 文件，推荐直接使用 script 标签同步引入：

<script src="path/to/sea.js"></script>

    (使用直接引入外部脚本的方式的话，config内base路径则为该script的src属性的绝对路径)

    为了满足某些场景下的性能优化需求，也可以将 sea.js 的源码内嵌：

<script> *// sea.js 的源码* </script>

    (注意：代码内嵌时，需要通过 seajs.config 手动配置 base 路径。)

**二、seajs配置说明：**

    seajs . config : (可进行配置seajs的路径寻找规则、模块别名映射、预加载模块、以及seajs的基础路径)

**seajs.config** 支持的属性有：

    ' alias ' : (当模块标识很长时，可以使用 alias 来简化，使用 alias ，可以让文件的真实路径与调用标识分开，有利于统一维护。)

    ' paths ' : (当目录比较深，或需要跨目录调用模块时，可以使用 paths 来简化书写， paths 配置也可以结合 alias 配置一起使用，让模块引用非常方便。)

    ' vars ' : (有些场景下，模块路径在运行时才能确定，这时可以使用 vars 变量来配置， vars 配置的是模块标识中的变量值，在模块标识中用 {key} 来表示变量。)

    ' map ' : (该配置可对模块路径进行映射修改，可用于路径转换、在线调试等。)

    ' preload ' : (使用 preload 配置项，可以在普通模块加载前，提前加载并初始化好指定模块。 preload 中的空字符串会被忽略掉。)

    ' debug ' : (值为 true 时，加载器不会删除动态插入的 script 标签。插件也可以根据 debug 配置，来决策 log 等信息的输出。)

    ' base ' : (Sea.js 在解析模块路径标识时，会基于 base 路径来解析。)

    ' charset ' : (获取模块文件时， <script> 或 <link> 标签的 charset 属性。 默认是 utf-8 )

**使用案例：**

**1. alias ：模块别名配置**

seajs.config({

'alias' : {

'jquery': 'jquery/jquery/1.10.1/jquery',

'app/biz': 'http://path/to/app/biz.js'

}

});

define(function(**require**, exports, **module**) {

var $ = **require**('jquery');

//=> 加载的是 http://path/to/base/jquery/jquery/1.10.1/jquery.js

var biz = **require**('app/biz');

//=> 加载的是 http://path/to/app/biz.js

});

**2. vars 配置的是模块标识中的变量值，在模块标识中用 {key} 来表示变量。**

seajs.config({

vars: {

'locale': 'zh-cn'

}

});

define(function(**require**, exports, **module**) {

var lang = **require**('./i18n/{locale}.js');

//=> 加载的是 path/to/i18n/zh-cn.js

});

**3.该配置可对模块路径进行映射修改，可用于路径转换、在线调试等。**

seajs.config({

map: [

[ '.js', '-debug.js' ]

]

});

define(function(**require**, exports, **module**) {

var a = **require**('./a');

//=> 加载的是 path/to/a-debug.js

});

**4.使用 preload 配置项，可以在普通模块加载前，提前加载并初始化好指定模块。**

*// 在老浏览器中，提前加载好 ES5 和 json 模块, (注：preload 中的空字符串会被忽略掉。)*

seajs.config({

preload: [

Function.prototype.bind ? '' : 'es5-safe',

**this**.JSON ? '' : 'json'

]

});

*// (注意：preload 中的配置，需要等到 use 时才加载。比如：)*

seajs.config({

preload: 'a'

});

*// 在加载 b 之前，会确保模块 a 已经加载并执行好*

seajs.**use**('./b');

*// preload 配置不能放在模块文件里面：*

seajs.config({

preload: 'a'

});

define(function(**require**, exports) {

*// 此处执行时，不能保证模块 a 已经加载并执行好，只有在调用 use或require.async时，preload里面预先标识的模块才会被加载。*

});

**5. 配置seajs寻找模块时的相对基础路径**

seajs.config({

'base' : '../sea-modules/'

});

*/\**

*这里需要注意的是在设置base属性的时候，./ 或 ../ 相关路径是根据当前html文件路径走的，如果html路径为：/var/www/html/index/*

*那么上面的代码设置的base路径就是：/var/www/html/sea-modules/，*

*如果不手动设置该项base属性的话，sea.js会进行正则匹配然后获得sea.js的目录路径，具体正则匹配和设置可查看源码。*

*\*/*

**三、seajs引入模块有两种方法：**

    1. seajs.use     : public API,    可在全局任何位置调用该方法(前提是保证seajs对象未被覆盖方法未被重写)

    2. seajs.async   : private API,   在执行全局函数define时，通常会传递一个函数进去当factory参数，然后seajs在执行该函数时，会传递三个参数:

    require (type:[ Function ];desc:[一个封装在内部的加载文件模块类]),

     exports (type[ Object ];desc:[该模块的一个属性]),

     module (type[ Object ];desc:[该模块自身的引用])

    第一种情况：seajs.use('abc');

    第二种情况： seajs.use(['a', 'b']);

    两种加载格式，但这里大家需要注意的是，Sea.js中加载一个文件就等于创建一个Module实例，每个实例都有以下属性

**this**.uri = uri

**this**.dependencies = deps || [] *// 依赖自身的模块ID*

**this**.exports = **null**

**this**.status = 0 *// 当前模块的状态(FETCHING: 1,SAVED: 2,LOADING: 3,LOADED: 4,EXECUTING: 5,EXECUTED: 6)*

**this**.\_waitings = {} *// 哪些模块依赖我？*

**this**.\_remain = 0 *// 卸载依赖项的数量*

    然后这个模块都被存储在一个叫 cachedMods 的Object当中，然后模块的uri就是键 模块自身实例就是值。

    比如第一种调用use情况的话，最终结果: cachedMods里的键为(seajs模块自生成的一个标识ID) 对应值为 Module实例; use函数第一个参数'abc'不是Module模块的标识id, 而是该模块的一个依赖, 最终这个依赖‘abc’会通过一系列函数提取出绝对路径 同样会被require方法生成(js|css)元素添加到DOM树当中。

**四、模块标识**

    模块标识是一个字符串，用来标识模块。在 require 、 require.async 等加载函数中，第一个参数都是模块标识。

Sea.js 中的模块标识是 [***CommonJS 模块标识***](http://wiki.commonjs.org/wiki/Modules/1.1.1) 的超集:

1. 一个模块标识由斜线（ / ）分隔的多项组成。
2. 每一项必须是小驼峰字符串、 . 或 .. 。
3. 模块标识可以不包含文件后缀名，比如 .js 。
4. 模块标识可以是 **相对** 或 **顶级** 标识。如果第一项是 . 或 .. ，则该模块标识是相对标识。
5. 顶级标识根据模块系统的基础路径来解析。
6. 相对标识相对 require 所在模块的路径来解析。

      注意: 符合上述规范的标识肯定是 Sea.js 的模块标识，但 Sea.js 能识别的模块标识不需要完全符合以上规范。 比如，除了大小写字母组成的小驼峰字符串，Sea.js 的模块标识字符串还可以包含下划线（ \_ ）和连字符（ - ）， 甚至可以是 http://、 https:// 、 file:/// 等协议开头的绝对路径。

**1.** **相对标识**   
    相对标识以 . 开头，只出现在模块环境中（define 的 factory 方法里面）。相对标识永远相对当前模块的 URI 来解析：

*// 在 http://example.com/js/a.js 的 factory 中：*

**require**.resolve('./b');

*// => http://example.com/js/b.js*

*// 在 http://example.com/js/a.js 的 factory 中：*

**require**.resolve('../c');

*// => http://example.com/c.js*

**2.** **顶级标识**   
  顶级标识不以点（.）或斜线（/）开始， 会相对模块系统的基础路径（即 Sea.js 的 base 路径）来解析：

*// 假设 base 路径是：http://example.com/assets/*

*// 在模块代码里：*

**require**.resolve('gallery/jquery/1.9.1/jquery');

*// => http://example.com/assets/gallery/jquery/1.9.1/jquery.js*

模块系统的基础路径即 base 的默认值，与 sea.js 的访问路径相关：   
如果 sea.js 的访问路径是： http://example.com/assets/sea.js   
则 base 路径为： http://example.com/assets/   
当 sea.js 的访问路径中含有版本号时，base 不会包含 seajs/x.y.z 字串。 当 sea.js 有多个版本时，这样会很方便。   
如果 sea.js 的路径是： http://example.com/assets/seajs/1.0.0/sea.js   
则 base 路径是：  http://example.com/assets/

当然，也可以手工配置 base 路径：

seajs.config({ base: 'http://code.jquery.com/' }); // 在模块代码里： **require**.resolve('jquery'); // => http://code.jquery.com/jquery.js

**3. 普通路径：**  
除了相对和顶级标识之外的标识都是普通路径。普通路径的解析规则，和 HTML 代码中的 <script src="..."></script> 一样，会相对当前页面解析。

*// 假设当前页面是 http://example.com/path/to/page/index.html*

*// 绝对路径是普通路径：*

**require**.resolve('http://cdn.com/js/a');

*// => http://cdn.com/js/a.js*

*// 根路径是普通路径：*

**require**.resolve('/js/b');

*// => http://example.com/js/b.js*

*// use 中的相对路径始终是普通路径：*

seajs.**use**('./c');

*// => 加载的是 http://example.com/path/to/page/c.js*

seajs.**use**('../d');

*// => 加载的是 http://example.com/path/to/d.js*

**提示：**  
    1.顶级标识始终相对 base 基础路径解析。   
    2.绝对路径和根路径始终相对当前页面解析。   
    3.require 和 require.async 中的相对路径相对当前模块路径来解析。   
    4.seajs.use 中的相对路径始终相对当前页面来解析。

**四、文件后缀的自动添加规则**

Sea.js 在解析模块标识时， 除非在路径中有问号（?）或最后一个字符是井号（#），否则都会自动添加 JS 扩展名（.js）。如果不想自动添加扩展名，可以在路径末尾加上井号（#）。

*// ".js" 后缀可以省略：*

**require**.resolve('http://example.com/js/a');

**require**.resolve('http://example.com/js/a.js');

*// => http://example.com/js/a.js*

*// ".css" 后缀不可省略：*

**require**.resolve('http://example.com/css/a.css');

*// => http://example.com/css/a.css*

*// 当路径中有问号（"?"）时，不会自动添加后缀：*

**require**.resolve('http://example.com/js/a.json?callback=define');

*// => http://example.com/js/a.json?callback=define*

*// 当路径以井号（"#"）结尾时，不会自动添加后缀，且在解析时，会自动去掉井号：*

**require**.resolve('http://example.com/js/a.json#');

*// => http://example.com/js/a.json*

**内容没有补全以及描述比较差劲，后续还会补充和更正完善，并当有时间的时候将seajs源码进行小小的解读。**