02

OPEN ORIENTED

凹 凸 实 验 室

SeaJS从入门到原理

王彩暖

简介

SeaJS 是一个模块加载器，模块加载器需要实现两个基本功能：

* 实现模块定义规范，这是模块系统的基础。
* 模块系统的启动与运行。

为什么使用SeaJS

02

* 简单友好的模块定义规范
* 自然直观的代码组织方式
* Sea.js 提供常用插件
* 理论上，Sea.js 可以运行在任何浏览器引擎上。

SeaJS从入门到原理-主要内容

* 模块定义规范
* 模块系统的启动与运行
* 模块加载大体流程
* 与RequireJS的主要区别

CMD 规范

Modules/Transport 规范

CMD 规范

02

CMD 规范的前身是Modules/Wrappings规范

在 CMD 规范中，一个模块就是一个文件。

CMD 规范

define(factory)

```
define({ "foo": "bar" });
define(function(require, exports, module) {
    // 模块代码
});
```

Modules/Transport 规范

define(id?, deps?, factory)

id：模块标识。
deps：一个数组，表示模块依赖。

Modules/Transport 规范

如何理解：SeaJS 只支持 CMD 模块的话，没法实现 JS 文件的合并了？

Modules/Transport 规范

如何理解：SeaJS 只支持 CMD 模块的话，没法实现 JS 文件的合并了？

首先，认为CMD 规范 中一个模块就是一个文件，一个文件里面定义了两个，所以出现异常也不奇怪了。

Modules/Transport 规范


```
// a.js
define(function (require, exports) {
    exports.add = function (a, b) {
        return a + b;
    };
});
```

```
// b.js
define(function (require) {
    var a = require('./a');
    var c = a.add(1, 2);
    alert(c);
});
```

Modules/Transport 规范


```
// a.js
define(function (require, exports) {
    exports.add = function (a, b) {
        return a + b;
    };
});
// b.js
define(function (require) {
    var a = require('./a');
    var c = a.add(1, 2);
    alert(c);
});
```

index.js

Modules/Transport 规范

SeaJS 用这个 index.js文件的 URL 作为id ， 并缓存 id 与 模块之间的关系，导致只有最后一个定义的define会被识别，因为前面定义的模块被它覆盖了。

factory参数

require
exports
module

require：Function

* require是一个函数方法，用来获取其他模块提供的接口，而且是同步往下执行。require的模块不能被返回时，应该返回null。

require：Function

* require是一个函数方法，用来获取其他模块提供的接口，而且是同步往下执行。require的模块不能被返回时，应该返回null。

* require.async(id, callback?)：用来在模块内部异步加载模块，并在加载完成后执行指定回调。require的模块不能被返回时，callback应该返回null。callback接受返回的模块作为它的参数。

require：Function

* require是一个函数方法，用来获取其他模块提供的接口，而且是同步往下执行。require的模块不能被返回时，应该返回null。

* require.async(id, callback?)：用来在模块内部异步加载模块，并在加载完成后执行指定回调。require的模块不能被返回时，callback应该返回null。callback接受返回的模块作为它的参数。

* require.resolve(id)：不会加载模块，只返回解析后的绝对路径。

require：Function

* factory第一个参数必须命名为 require 。
* 不要重命名 require 函数，不要在任何作用域中给 require 重新赋值。
* require 的参数值必须是字符串直接量。

为什么那么死规定？！

require：Function

首先你要知道SeaJS 是如何知道一个模块的具体依赖的。SeaJS 通过 **factory.toString()** 拿到源码，再通过**正则匹配** require 的方式来得到依赖信息。这也是必须遵守 require 书写约定的原因。

require：Function

有时会希望可以使用 require 来进行条件加载，如下：

```
if (todayIsWeekend) {
  require("play");
} else {
  require("work");
}
```

在浏览器端中，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。

exports：Object

用来在模块内部对外提供接口。
exports 仅仅是 module.exports 的一个引用。

在 factory 内部给 exports 重新赋值时，并不会改变 module.exports 的值。因此给 exports 赋值是无效的，不能用来更改模块接口。

exports：Object


```
// 通过 exports 对外提供接口foo 属性
 exports.foo = 'bar';

 // 对外提供 doSomething 方法
 exports.doSomething = function() {};

 // 错误用法！！！
 exports = {
   foo: 'bar',
   doSomething: function() {}
 };
```

module：Object

* module.uri：模块解析后的uri
* module.dependencies：模块依赖
* module.exports：暴露模块接口数据，也可以通过 return 直接提供接口，因个人习惯使用。

对 module.exports 的赋值需要同步执行，慎重放在回调函数里，因为无法立刻得到模块接口数据。

module：Object

02

```
// 通过module.exports提供整个接口
 module.exports = {
   foo: 'bar',
   doSomething: function() {}
 };
```

模块标识id

模块标识id尽量遵循路径即 ID原则，减轻记忆模块 ID 的负担。

模块标识id会用在 require、 require.async 等加载函数中的第一个参数。

模块标识id

三种类型的标识：

* 相对标识：以 . 开头（包括.和..），相对标识**永远相对当前模块的 URI 来解析**。

模块标识id

三种类型的标识：

* 相对标识：**以 . 开头**（包括.和..），相对标识**永远相对当前模块的 URI 来解析**。

* 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 SeaJS配置 的 base 路径）**来解析。

模块标识id

三种类型的标识：

* 相对标识：以 . 开头（包括.和..），相对标识**永远相对当前模块的 URI 来解析**。

* 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 SeaJS配置 的 base 路径）**来解析。

* 普通路径：除了相对和顶级标识之外的标识都是普通路径，相对当前页面解析。

模块标识id

'./index’
'../index'
'index'
'/index'

模块标识id

可省略后缀.js，但是".css" 后缀不可省略。

SeaJS 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）。

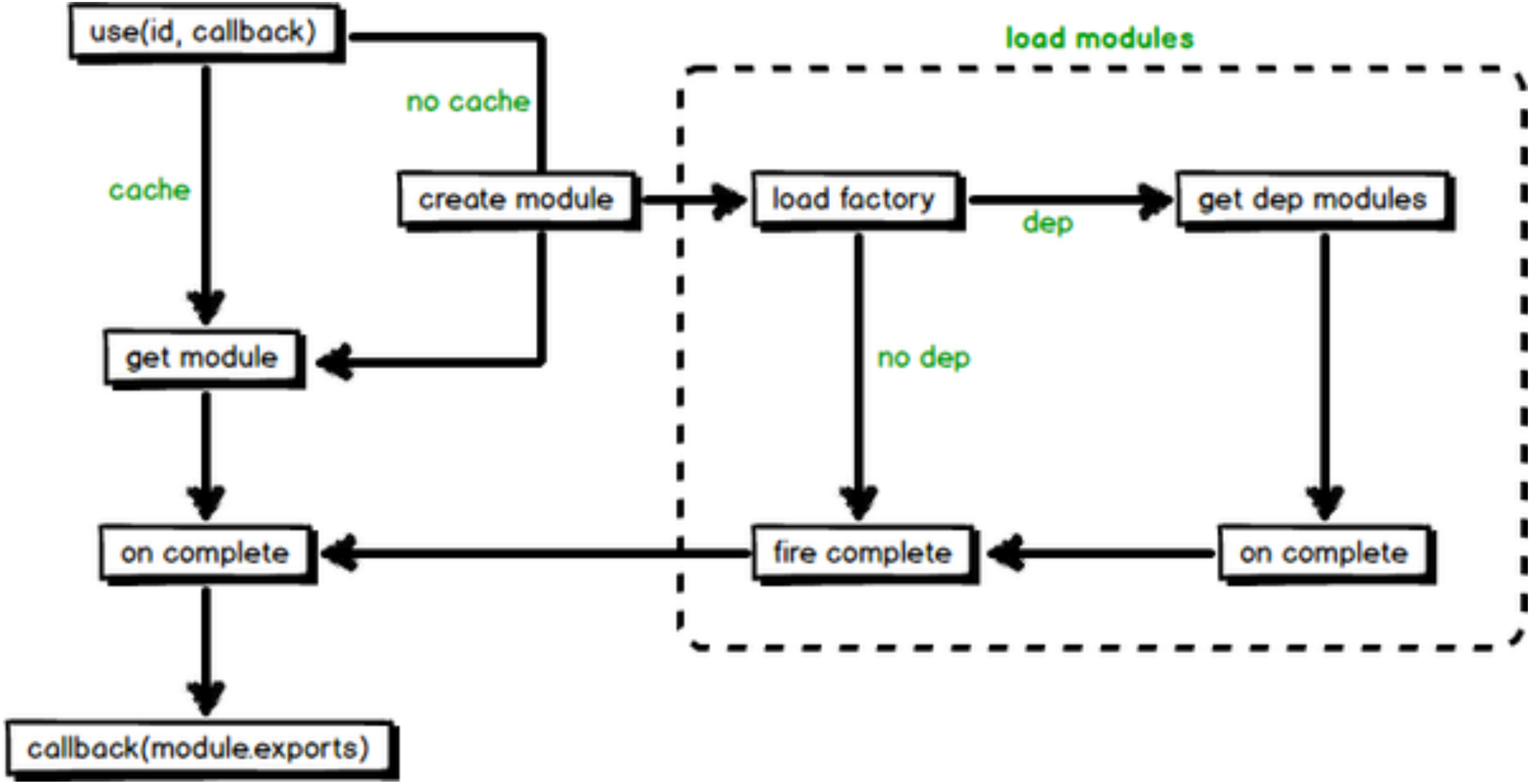
模块系统的启动与运行


```
<script type="text/javascript" src="../gb/sea.js"></
script>
<script>
  seajs.use('./index.js');
</script>
```

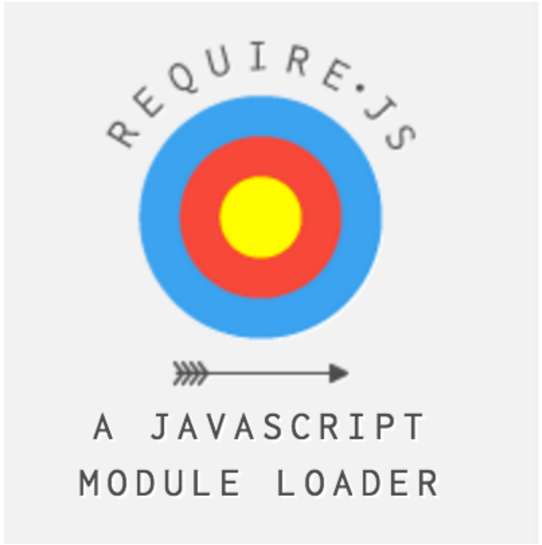
模块系统的启动与运行

* seajs.cache：Object，查阅当前模块系统中的所有模块信息。
* seajs.resolve：Function，利用模块系统的内部机制对传入的字符串参数进行路径解析。
* seajs.require：Function，全局的 require 方法，可用来直接获取模块接口。
* seajs.data：Object，查看 seajs 所有配置以及一些内部变量的值。

模块加载大体流程


use(id, callback)
no cache
cache
create module
load modules
load factory
dep
get dep modules
get module
no dep
on complete
fire complete
on complete
callback(module.exports)

与RequireJS的主要区别



与RequireJS的主要区别-**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

与RequireJS的主要区别-factory 的执行时机不同

SeaJS**按需执行依赖**避免浪费，但是require时才解析的行为对性能有影响。

SeaJS是异步加载模块的没错, 但执行模块的顺序也是**严格按照模块在代码中出现(require)的顺序**。

与RequireJS的主要区别-factory 的执行时机不同

RequireJS更遵从js异步编程方式，**提前执行依赖**，输出顺序取决于哪个 js 先加载完。

```
define(['a', 'b'], function(A, B) {
    //运行至此，a.js 和 b.js 已下载完成
    //A、B 两个模块已经执行完，直接可用
    return function () {};
});
```

与RequireJS的主要区别-factory 的执行时机不同

如果两个模块之间突然模块A依赖模块B：

SeaJS的懒执行可能有问题，而RequireJS不需要修改当前模块。

当模块A依赖模块B，模块B出错了：

如果是SeaJS，模块A执行了某操作，可能需要回滚。RequireJS因为尽早执行依赖可以尽早发现错误，不需要回滚。

与RequireJS的主要区别-聚焦点有差异

SeaJS努力成为浏览器端的模块加载器，RequireJS牵三挂四，兼顾Rhino 和 node，因此RequireJS比SeaJS的文件大。

与RequireJS的主要区别- **理念不一样**

RequireJS 有一系列插件，功能很强大，但破坏了模块加载器的纯粹性。SeaJS 则努力保持简单，并支持 CSS 模块的加载。

参考文献

seajs官网
http://www.zhihu.com/question/21157540
http://annn.me/how-to-realize-cmd-loader/
http://chaoskeh.com/blog/why-its-hard-to-combo-seajs-modules.html
https://github.com/cmdjs/specification/blob/master/draft/module.md
https://www.douban.com/note/283566440/
https://imququ.com/post/amd-simplified-commonjs-wrapping.html
https://lifesinger.wordpress.com/2011/05/17/the-difference-between-seajs-and-requirejs/

02

OPEN ORIENTED

凹 凸 实 验 室

THANKS FOR LISTENING

王彩暖