## [JavaSript模块规范 - AMD规范与CMD规范介绍](http://blog.chinaunix.net/uid-26672038-id-4112229.html)

## JavaSript模块化

    在了解AMD，CMD规范前，还是需要先来简单地了解下什么是模块化，模块化开发？

    模块化是指在解决某一个复杂问题或者一系列的杂糅问题时，依照一种分类的思维把问题进行系统性的分解以之处理。模块化是一种处理复杂系统分解为代码结构更合理，可维护性更高的可管理的模块的方式。可以想象一个巨大的系统代码，被整合优化分割成逻辑性很强的模块时，对于软件是一种何等意义的存在。对于软件行业来说：解耦软件系统的复杂性，使得不管多么大的系统，也可以将管理，开发，维护变得“有理可循”。

    还有一些对于模块化一些专业的定义为：模块化是软件系统的属性，这个系统被分解为一组高内聚，低耦合的模块。那么在理想状态下我们只需要完成自己部分的核心业务逻辑代码，其他方面的依赖可以通过直接加载被人已经写好模块进行使用即可。

首先，既然是模块化设计，那么作为一个模块化系统所必须的能力：

    1. 定义封装的模块。

    2. 定义新模块对其他模块的依赖。

    3. 可对其他模块的引入支持。

    好了，思想有了，那么总要有点什么来建立一个模块化的规范制度吧，不然各式各样的模块加载方式只会将局搅得更为混乱。那么在JavaScript中出现了一些非传统模块开发方式的规范 CommonJS的模块规范，AMD（Asynchronous Module Definition），CMD（Common Module Definition）等。

## **AMD 与 RequireJS**

### **AMD**

    Asynchronous Module Definition，用白话文讲就是 异步模块定义，对于 JSer 来说，异步是再也熟悉不过的词了，所有的模块将被异步加载，模块加载不影响后面语句运行。所有依赖某些模块的语句均放置在回调函数中。

    AMD规范定义了一个自由变量或者说是全局变量 define 的函数。

**define( id?, dependencies?, factory );**  

    AMD规范  <https://github.com/amdjs/amdjs-api/wiki/AMD>

    第一个参数 id 为字符串类型，表示了模块标识，为可选参数。若不存在则模块标识应该默认定义为在加载器中被请求脚本的标识。如果存在，那么模块标识必须为顶层的或者一个绝对的标识。

    第二个参数，dependencies ，是一个当前模块依赖的，已被模块定义的模块标识的数组字面量。

    第三个参数，factory，是一个需要进行实例化的函数或者一个对象。

    创建模块标识为 alpha 的模块，依赖于 require， export，和标识为 beta 的模块

1. define("alpha", [ "require", "exports", "beta" ], function( require, exports, beta ){
2. export.verb = function(){
3. return beta.verb();
4. // or:
5. return require("beta").verb();
6. }
7. });

    一个返回对象字面量的异步模块

1. define(["alpha"], function( alpha ){
2. return {
3. verb : function(){
4. return alpha.verb() + 1 ;
5. }
6. }
7. });

    无依赖模块可以直接使用对象字面量来定义

1. define( {
2. add : function( x, y ){
3. return x + y ;
4. }
5. } );

    类似与 CommonJS 方式定义

1. define( function( require, exports, module){
2. var a = require('a'),
3. b = require('b');
4. exports.action = function(){};
5. } );

**require();**

    require API 介绍 <https://github.com/amdjs/amdjs-api/wiki/require>

    在 AMD 规范中的 require 函数与一般的 CommonJS中的 require 不同。由于动态检测依赖关系使加载异步，对于基于回调的 require 需求强烈。

**局部 与 全局 的require**

    局部的 require 需要在AMD模式中的 define 工厂函数中传入 require。

1. define( ['require'], function( require ){
2. // ...
3. } );
4. or：
5. define( function( require, exports, module ){
6. // ...
7. } );

    局部的 require 需要其他特定的 API 来实现。

    全局的 require 函数是唯一全局作用域下的变量，像 define一样。全局的 require 并不是规范要求的，但是如果实现全局的 require函数，那么其需要具有与局部 require 函数 一样的以下的限定：

    1. 模块标识视为绝对的，而不是相对的对应另一个模块标识。

    2. 只有在异步情况下，require的回调方式才被用来作为交互操作使用。因为他不可能在同步的情况下通过 require(String) 从顶层加载模块。

    依赖相关的API会开始模块加载。如果需要有互操作的多个加载器，那么全局的 reqiure 应该被加载顶层模块来代替。

1. require(String)
2. define( function( require ){
3. var a = require('a'); // 加载模块a
4. } );
5. require(Array, Function)
6. define( function( require ){
7. require( ['a', 'b'], function( a,b ){ // 加载模块a b 使用
8. // 依赖 a b 模块的运行代码
9. } );
10. } );
11. require.toUrl( Url )
12. define( function( require ){
13. var temp = require.toUrl('./temp/a.html'); // 加载页面
14. } );

    amdjs 的API   <https://github.com/amdjs/amdjs-api/wiki>

### **RequireJS**

    官网 <http://www.requirejs.org/>

    API <http://www.requirejs.org/docs/api.html>

    RequireJS 是一个前端的模块化管理的工具库，遵循AMD规范，它的作者就是AMD规范的创始人 James Burke。所以说RequireJS是对AMD规范的阐述一点也不为过。

    RequireJS 的基本思想为：通过一个函数来将所有所需要的或者说所依赖的模块实现装载进来，然后返回一个新的函数（模块），我们所有的关于新模块的业务代码都在这个函数内部操作，其内部也可无限制的使用已经加载进来的以来的模块。

1. <script data-main='scripts/main' src='scripts/require.js'></script>

    那么scripts下的main.js则是指定的主代码脚本文件，所有的依赖模块代码文件都将从该文件开始异步加载进入执行。

**defined用于定义模块，RequireJS要求每个模块均放在独立的文件之中。按照是否有依赖其他模块的情况分为独立模块和非独立模块。**

    1. 独立模块，不依赖其他模块。直接定义：

1. define({
2. method1: function(){},
3. method2: function(){}
4. });

    也等价于

1. define(function(){
2. return{
3. method1: function(){},
4. method2: function(){}
5. }
6. });

    2. 非独立模块，对其他模块有依赖。

1. define([ 'module1', 'module2' ], function(m1, m2){
2. ...
3. });

    或者：

1. define( function( require ){
2. var m1 = require( 'module1' ),
3. m2 = require( 'module2' );
4. ...
5. });

    简单看了一下RequireJS的实现方式，其 require 实现只不过是将 function 字符串然后提取 require 之后的模块名，将其放入依赖关系之中。

**require方法调用模块**

    在require进行调用模块时，其参数与define类似。

1. require( ['foo', 'bar'], function( foo, bar ){
2. foo.func();
3. bar.func();
4. } );

    在加载 foo 与 bar 两个模块之后执行回调函数实现具体过程。

    当然还可以如之前的例子中的，在define定义模块内部进行require调用模块

1. define( function( require ){
2. var m1 = require( 'module1' ),
3. m2 = require( 'module2' );
4. ...
5. });

    define 和 require 这两个定义模块，调用模块的方法合称为AMD模式，定义模块清晰，不会污染全局变量，清楚的显示依赖关系。AMD模式可以用于浏览器环境并且允许非同步加载模块，也可以按需动态加载模块。

## **CMD 与 seaJS**

### **CMD**

    在CMD中，一个模块就是一个文件，格式为：

**define( factory );**

    全局函数define，用来定义模块。

    参数 factory  可以是一个函数，也可以为对象或者字符串。

    当 factory 为对象、字符串时，表示模块的接口就是该对象、字符串。

    定义JSON数据模块：

1. define({ "foo": "bar" });

    通过字符串定义模板模块：

1. define('this is {{data}}.');

    factory 为函数的时候，表示模块的构造方法，执行构造方法便可以得到模块向外提供的接口。

1. define( function(require, exports, module) {
2. // 模块代码
3. });

**define( id?, deps?, factory );**

    define也可以接受两个以上的参数，字符串id为模块标识，数组deps为模块依赖：

1. define( 'module', ['module1', 'module2'], function( require, exports, module ){
2. // 模块代码
3. } );

    其与 AMD 规范用法不同。

**require** 是 factory 的第一个参数。

    require( id );

    接受模块标识作为唯一的参数，用来获取其他模块提供的接口：

1. define(function( require, exports ){
2. var a = require('./a');
3. a.doSomething();
4. });

    require.async( id, callback? );

    require是同步往下执行的，需要的异步加载模块可以使用 require.async 来进行加载：

1. define( function(require, exports, module) {
2. require.async('.a', function(a){
3. a.doSomething();
4. });
5. });

    require.resolve( id )

    可以使用模块内部的路径机制来返回模块路径，不会加载模块。

**exports** 是 factory 的第二个参数，用来向外提供模块接口。

1. define(function( require, exports ){
2. exports.foo = 'bar'; // 向外提供的属性
3. exports.do = function(){}; // 向外提供的方法
4. });

    当然也可以使用 return 直接向外提供接口。

1. define(function( require, exports ){
2. return{
3. foo : 'bar', // 向外提供的属性
4. do : function(){} // 向外提供的方法
5. }
6. });

    也可以简化为直接对象字面量的形式:

1. define({
2. foo : 'bar', // 向外提供的属性
3. do : function(){} // 向外提供的方法
4. });

    与nodeJS中一样需要注意的是，一下方式是错误的：

1. define(function( require, exports ){
2. exports = {
3. foo : 'bar', // 向外提供的属性
4. do : function(){} // 向外提供的方法
5. }
6. });

    需要这么做

1. define(function( require, exports, module ){
2. module.exports = {
3. foo : 'bar', // 向外提供的属性
4. do : function(){} // 向外提供的方法
5. }
6. });

    传入的对象引用可以添加属性，一旦赋值一个新的对象，那么值钱传递进来的对象引用就会失效了。开始之初，exports 是作为 module.exports 的一个引用存在，一切行为只有在这个引用上 factory 才得以正常运行，赋值新的对象后就会断开引用，exports就只是一个新的对象引用，对于factory来说毫无意义，就会出错。

**module** 是factory的第三个参数，为一个对象，上面存储了一些与当前模块相关联的属性与方法。

        module.id 为模块的唯一标识。

        module.uri 根据模块系统的路径解析规则得到模块的绝对路径。

        module.dependencies 表示模块的依赖。

        module.exports 当前模块对外提供的接口。

### **seaJS**

    官网 <http://seajs.org/docs/>

    API快速参考 <https://github.com/seajs/seajs/issues/266>

    sea.js 核心特征：

        1. 遵循CMD规范，与NodeJS般的书写模块代码。

        2. 依赖自动加载，配置清晰简洁。

    兼容 Chrome 3+，Firefox 2+，Safari 3.2+，Opera 10+，IE 5.5+。

**seajs.use**

    用来在页面中加载一个或者多个模块

1. // 加载一个模块
2. seajs.use('./a');
3. // 加载模块，加载完成时执行回调
4. seajs.use('./a'，function(a){
5. a.doSomething();
6. });
7. // 加载多个模块执行回调
8. seajs.use(['./a','./b']，function(a , b){
9. a.doSomething();
10. b.doSomething();
11. });

    其define 与 require 使用方式基本就是CMD规范中的示例。

## **AMD 与 CMD 区别到底在哪里？**

    看了以上 AMD，requireJS 与 CMD， seaJS的简单介绍会有点感觉模糊，总感觉较为相似。因为像 requireJS 其并不是只是纯粹的AMD固有思想，其也是有CMD规范的思想，只不过是推荐 AMD规范方式而已， seaJS也是一样。

    下面是玉伯对于 AMD 与 CMD 区别的解释：

    AMD 是 RequireJS 在推广过程中对模块定义的规范化产出。

    CMD 是 SeaJS 在推广过程中对模块定义的规范化产出。

    类似的还有 CommonJS Modules/2.0 规范，是 BravoJS 在推广过程中对模块定义的规范化产出还有不少??

    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：

1. // CMD
2. define(function(require, exports, module) {
3. var a = require('./a')
4. a.doSomething()
5. // 此处略去 100 行
6. var b = require('./b') // 依赖可以就近书写
7. b.doSomething()
8. // ...
9. })
10. // AMD 默认推荐的是
11. define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好
12. a.doSomething()
13. // 此处略去 100 行
14. b.doSomething()
15. // ...
16. })

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

    3. 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 seajs.use 来实现模块系统的加载启动。CMD 里，每个 API 都简单纯粹。

    4. 还有一些细节差异，具体看这个规范的定义就好，就不多说了。

另外，SeaJS 和 RequireJS 的差异，可以参考：<https://github.com/seajs/seajs/issues/277>

## **总结**

    本文主要是介绍了一下 AMD CMD的规范，顺便简单的讲述了一下 requireJS 与 seaJS。讲的较为笼统，下面的扩展阅读可以更好的帮助你理解模块化以及各个规范。

**扩展阅读：**

AMD规范文档 <https://github.com/amdjs/amdjs-api/wiki/AMD>  
amdjs 的 require 接口文档 <https://github.com/amdjs/amdjs-api/wiki/require>  
amdjs 的接口文档 <https://github.com/amdjs/amdjs-api/wiki>

RequireJS官网接口文档  <http://www.requirejs.org/docs/api.html>

模块系统 <https://github.com/seajs/seajs/issues/240>

前端模块化开发的价值 <https://github.com/seajs/seajs/issues/547>

前端模块化开发那点历史 <https://github.com/seajs/seajs/issues/588>

CMD 模块定义规范 <https://github.com/seajs/seajs/issues/242>

SeaJS API快速参考 <https://github.com/seajs/seajs/issues/266>

从 CommonJS 到 Sea.js <https://github.com/seajs/seajs/issues/269>

RequireJS和AMD规范  [http://javascript.ruanyifeng.com/tool/requirejs.html](http://..ruanyifeng.com/tool/requirejs.html)

CommonJS规范  [http://javascript.ruanyifeng.com/nodejs/commonjs.html](http://..ruanyifeng.com/nodejs/commonjs.html)

Javascript模块化编程 [http://www.ruanyifeng.com/blog/2012/10/javascript\_module.html](http://www.ruanyifeng.com/blog/2012/10/._module.html)

Javascript模块化编程 <http://www.ruanyifeng.com/blog/2012/10/asynchronous_module_definition.html>

知乎  AMD 和 CMD 的区别有哪些？ <http://www.zhihu.com/question/20351507>

JavaScript模块化开发 - CommonJS规范 [http://www.feeldesignstudio.com/2013/09/javascript-module-pattern-commonjs](http://www.feeldesignstudio.com/2013/09/.-module-pattern-commonjs)   
JavaScript模块化开发 - AMD规范 [http://www.feeldesignstudio.com/2013/09/javascript-module-pattern-amd](http://www.feeldesignstudio.com/2013/09/.-module-pattern-amd)  
模块化设计 <http://baike.baidu.com/view/189730.htm>   
模块化  <http://baike.baidu.com/view/182267.htm>