**CMD&AMD**

## 一、CMD 和AMD

**1 模块化**

随着浏览器功能越来越完善，前端已经不仅仅是切图做网站，前端在某些方面已经媲美桌面应用。越来越庞大的前端项目，越来越复杂的代码，前端开发者们对于模块化的需求空前强烈。随着浏览器功能越来越完善，前端已经不仅仅是切图做网站，前端在某些方面已经媲美桌面应用。越来越庞大的前端项目，越来越复杂的代码，前端开发者们对于模块化的需求空前强烈。后来node出现了，跟随node出现的还有commonjs，这是一种js模块化解决方案，像Node.js主要用于服务器的编程，加载的模块文件一般都已经存在本地硬盘，所以加载起来比较快，不用考虑异步加载的方式，CommonJS 加载模块是同步的，所以只有加载完成才能执行后面的操作。但是浏览器环境不同于Node，浏览器中获取一个资源必须要发送http请求，从服务器端获取，采用同步模式必然会阻塞浏览器进程出现假死现象。在这方面dojo曾经做了伟大尝试，早期dojo便是采用xhr+eval的方式，结果可想而知，阻塞现象是必然的。后来出现无阻塞加载脚本方式在开发中广泛应用，在此基础结合commonjs规范，前端模块化迎来了两种方案：AMD、CMD. 异步和管理依赖。

**AMD 是 RequireJS 在推广过程中对模块定义的规范化产出，CMD是SeaJS 在推广过程中被广泛认知。**

**2 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。

**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

**3 CMD**

通用模块定义（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。RequireJS 和 SeaJS 都是模块化框架的代表，AMD和CMD，是他们各自定义模块化的方式，大同小异，主要是代码风格和API不同。

**4 RequireJS、SeaJS 区别**

（1）两者定位有差异。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 服务器端

（2）两者遵循的标准有差异。RequireJS 遵循的是 AMD（异步模块定义）规范，SeaJS 遵循的是 CMD （通用模块定义）规范。规范的不同，导致了两者 API 的不同。SeaJS 更简洁优雅。

（3）两者社区理念有差异。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 不强推，而采用自主封装的方式来“海纳百川”，目前已有较成熟的封装策略。

（4）两者代码质量有差异。RequireJS 是没有明显的 bug，SeaJS 是明显没有 bug。

（5）两者对调试等的支持有差异。SeaJS 通过插件，可以实现 Fiddler 中自动映射的功能，还可以实现自动 combo 等功能，非常方便便捷。RequireJS 无这方面的支持。

（6）两者的插件机制有差异。RequireJS 采取的是在源码中预留接口的形式，源码中留有为插件而写的代码。SeaJS 采取的插件机制则与 Node 的方式一致：开放自身.

## 二、require.js

**1 requireJS 是用JavaScript编写的js框架**

主要功能是可以按不同的先后依赖关系对 javascript 等文件的进行加载工作，可简单理解为JS文件的加载器，它非常适合在浏览器中使用，它可以确保所依赖的JS文件加载完成之后再加载当前的JS文件，这在大量使用JS文件的项目中可确保各个JS文件的先后加载顺序，确保避免了以前因某些原因某个文件加载慢而导致其它加载快的文件需要依赖其某些功能而出现某函数或某变量找不到的问题，这点非常有用，也是 requireJS 的主要价值所在吧；

1,require会定义三个变量：define,require,requirejs，其中require ===requirejs，一般使用require更简短,页面加载同时不阻塞渲染;

2,define(function(){...}); 从名字就可以看出这个api是用来定义一个模块;

3,require(['aModule','bModule'],function(a,b){...})加载依赖模块并执行加载完后的回调函数;

4,在使用requirejs时，加载模块时不用写.js后缀，当然也是不能写后缀;

5,require.config({...}) 是用来配置模块加载位置和设置基本路径等;

6,加载requirejs脚本的script标签加入了data-main属性，这属性指定在加载完reuqire.js后，就用requireJS加载该属性值指定路径下的JS文件并运行，所以一般该JS文件称为主JS文件，类似C或Java语言中main函数的功能，然后把require.config的配置加该主文件后可使每一个页面都使用这个配置，然后页面中就可直接使用require来加载所有的短模块名即可，这样即可在一个地方配置所有地方使用，同时模块别名的使用和管理也都比较方便；例如： < script data-main="js/main" src="js/require.js">< /script>

7,当script标签指定data-main属性时，require会默认的将data-main指定的js为根路径;

8,require加载的模块一般都需要符合AMD规范,即使用define来申明模块;

9,加载非AMD规范的js，这时候就需要用到另一个功能：shim;

require.config({

paths:{

"underscore":"http://www.xxx.com/xpath/xFile.js"

}

,shim: {

"underscore" : {

exports : "\_";

}

}

});

**2 require.js优点**

防止js加载阻塞页面渲染,使用程序调用的方式加载js，防出现如下丑陋的场景

<script type="text/javascript" src="a.js"></script>

<script type="text/javascript" src="b.js"></script>

<script type="text/javascript" src="c.js"></script>

**3 require.js开发**

**（1）<script src="js/require.js" defer async="true" data-main="js/main" ></script>**

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。

data-main属性的作用是，指定网页程序的主模块。在上例中，就是js目录下面的main.js，这个文件会第一个被require.js加载。

require(['jquery', 'underscore', 'backbone'], function ($, \_, Backbone){

　　// some code here

});

require()函数接受两个参数。

第一个参数是一个数组，表示所依赖的模块，上例就是['moduleA', 'moduleB', 'moduleC']，即主模块依赖这三个模块；

第二个参数是一个回调函数，当前面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该函数，从而在回调函数内部就可以使用这些模块。

默认情况下，require.js假定这三个模块与main.js在同一个目录，文件名分别为 jquery.js，underscore.js和backbone.js，然后自动加载。

以下是自动以加载

require.config

({

　　　　baseUrl: "js/lib",

　　　　paths:

{

　　　　　　"jquery": "jquery.min",

　　　　　　"underscore": "underscore.min",

　　　　　　"backbone": "backbone.min"

　　　　}

});

define(function (){

　　　　var add = function (x,y){

　　　　　　return x+y;

　　　　};

　　　　return {

　　　　　　add: add

　　　　};

　　});

define(['myLib'], function(myLib)

{

　　　　function foo()

{

　　　　　　myLib.doSomething();

　　　　}

　　　　return

{

　　　　　　foo : foo

　　　　};

});

**（2）加载非规范的模块:**

require.config

({

　　　　shim:

{

　　　　　　'underscore':

{

　　　　　　　　exports: '\_'

　　　　　　},

　　　　　　'backbone':

{

　　　　　　　　deps: ['underscore', 'jquery'],

　　　　　　　　exports: 'Backbone'

　　　　　　}

　　　　}

});

require.js插件:

require.js还提供一系列插件，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。

shim:

{

　　　'jquery.scroll':

{

　　　　　deps: ['jquery'],

　　　　　exports: 'jQuery.fn.scroll'

　　　}

　}

text和image插件，则是允许require.js加载文本和图片文件。

define(

[

　　　　'text!review.txt',

　　　　'image!cat.jpg'

　　　　],

　　　　function(review,cat)

{

　　　　　　console.log(review);

　　　　　　document.body.appendChild(cat);

　　　　}

　　);

## 三、sea.js

和requireJS相似的，seaJS 是用JavaScript编写的js框架，主要功能是可以按不同的先后依赖关系对 javascript 等文件的进行加载工作，可简单理解为JS文件的加载器，它非常适合在浏览器中使用，它可以确保所依赖的JS文件加载完成之后再加载当前的JS文件，这在大量使用JS文件的项目中可确保各个JS文件的先后加载顺序，确保避免了以前因某些原因某个文件加载慢而导致其它加载快的文件需要依赖其某些功能而出现某函数或某变量找不到的问题，这点非常有用，也是 seaJS （遵守CMD） 的主要价值所在吧；但和 requireJS （遵守AMD规范）有所区别

1,seajs.config({...}); //用来对 Sea.js 进行配置。

2,seajs.use(['a','b'],function(a,b){...}); //用来在页面中加载一个或多个模块。

3,define(function(require, exports, module){...}); //用来定义模块。Sea.js 推崇一个模块一个文件，遵循统一的写法：

4,require(function(require){var a = require("xModule"); ... }); //require 用来获取指定模块的接口。

5,require.async, //用来在模块内部异步加载一个或多个模块。 例如：

define(function(require){

require.async(['aModule','bModule'],function(a,b){ // 异步加载多个模块，在加载完成时，执行回调

a.func();

b.func();

})

});

6,exports, //用来在模块内部对外提供接口。 例如：

define(function(require, exports){

exports.varName01 = 'varValue'; // 对外提供 varName01 属性

exports.funName01 = function(p1,p2){ // 对外提供 funName01 方法

....

}

});

7,module.exports, 与 exports 类似，用来在模块内部对外提供接口。例如：

define(function(require, exports, module) {

module.exports = { // 对外提供接口

name: 'a',

doSomething: function() {...};

};

});

**1 Seajs，一个Web模块加载框架**，追求简单、自然的代码书写和组织方式，：Sea.js 遵循 CMD 规范，模块化JS代码。依赖的自动加载、配置的简洁清晰，可以让程序员更多地专注编码。

**2 优缺点**

@优点：

1).提高可维护性。

2).模块化编程。

3).动态加载，前端性能优化

@缺点：

1).学习文档偏少且混乱，会更改团队使用JS的编写习惯，必须使用模块化编程。

2).不太适合团队目前的情况，多JS文件但少改动，动态加载优势和模块化优势不明显。

3). 需要配套使用SPM工具，JS的打包和管理工具。

**3 实际开发**

<script type="text/javascript" src="/uoms/app/js/seajs/sea.js"></script>

seajs.use

([ "webapp/award/awardDetail" ],

function(awardDetail)

{

awardDetail.init({});

}

);

define('webapp/award/awardDetail',

['webapp/base/apputils',"webapp/base/dateUtil"],

function(require, exports, module)

{

var app = require("webapp/base/apputils");

var dateUtil = require("webapp/base/dateUtil");

exports.init = function()

{

.....

}

exports.doBusiness = function()

{

.....

}

});

//配置js路径

seajs.config

({

alias:

{

"jquery":"../examples-master/sea-modules/jquery/jquery/1.10.1/jquery.js"

}

});

//加载模块

seajs.use('../js/seajs/init',function($)

{

$("#test\_div").click(function(){alert(1);});

});

//init.js

define(function(require, exports, module) =

{

var m1=require('M1');

exports.init=function(){m1.run(); }

});

//M1.js

define(function(require,exports,module)=

{

var m2=require('M2');

exports.run=function()

{

alert('M1');

m2.run();

}

});

define(function(require,exports,module)=

{

exports.run=function()

{

alert('M2');

}

});

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

<script>

seajs.use('./init', function(init)

{

init.init();

});

</script>