前端开发中，起初只要在script标签中嵌入几十上百行代码就能实现一些基本的交互效果，后来js得到重视，应用也广泛起来了，jQuery，Ajax，Node.Js，MVC，MVVM等的助力也使得前端开发得到重视，也使得前端项目越来越复杂，然而，JavaScript却没有为组织代码提供任何明显帮助，甚至没有类的概念，更不用说模块（module）了，那么什么是模块呢？

一个模块就是实现特定功能的文件，有了模块，我们就可以更方便地使用别人的代码，想要什么功能，就加载什么模块。模块开发需要遵循一定的规范，否则就都乱套了。

根据AMD规范，我们可以使用define定义模块，使用require调用模块。

目前，通行的js模块规范主要有两种：CommonJS和AMD。

AMD 即Asynchronous Module Definition，中文名是“异步模块定义”的意思。它是一个在浏览器端模块化开发的规范，服务器端的规范是CommonJS。

模块将被异步加载，模块加载不影响后面语句的运行。所有依赖某些模块的语句均放置在回调函数中。

AMD 是 RequireJS 在推广过程中对模块定义的规范化的产出。

**define() 函数**

AMD规范只定义了一个函数 define，它是全局变量。函数的描述为：

define(id?, dependencies?, factory);

**参数说明**：

id：指定义中模块的名字，可选；如果没有提供该参数，模块的名字应该默认为模块加载器请求的指定脚本的名字。如果提供了该参数，模块名必须是“顶级”的和绝对的（不允许相对名字）。

依赖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。

依赖参数是可选的，如果忽略此参数，它应该默认为["require", "exports", "module"]。然而，如果工厂方法的长度属性小于3，加载器会选择以函数的长度属性指定的参数个数调用工厂方法。

工厂方法factory，模块初始化要执行的函数或对象。如果为函数，它应该只被执行一次。如果是对象，此对象应该为模块的输出值。

**模块名的格式**

模块名用来唯一标识定义中模块，它们同样在依赖性数组中使用：

1. 模块名是用正斜杠分割的有意义单词的字符串
2. 单词须为驼峰形式，或者"."，".."
3. 模块名不允许文件扩展名的形式，如“.js”
4. 模块名可以为 "相对的" 或 "顶级的"。如果首字符为“.”或“..”则为相对的模块名
5. 顶级的模块名从根命名空间的概念模块解析
6. 相对的模块名从 "require" 书写和调用的模块解析

**使用 require 和 exports**

创建一个名为"alpha"的模块，使用了require，exports，和名为"beta"的模块:

define("alpha", ["require", "exports", "beta"], function (require, exports, beta) {

exports.verb = function() {

return beta.verb();

//Or:

return require("beta").verb();

}

});

CMD（Common Module Definition） 通用模块定义。该规范明确了模块的基本书写格式和基本交互规则。该规范是在国内发展出来的。AMD是依赖关系前置，CMD是按需加载。

在 CMD 规范中，一个模块就是一个文件。代码的书写格式如下：

define(factory);

factory 为函数时，表示是模块的构造方法。执行该构造方法，可以得到模块向外提供的接口。factory 方法在执行时，默认会传入三个参数：require、exports 和 module：

define(function(require, exports, module) {

// 模块代码

});

require是可以把其他模块导入进来的一个参数，而export是可以把模块内的一些属性和方法导出的。

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。

对于依赖的模块，AMD 是提前执行，CMD 是延迟执行。

AMD:提前执行（异步加载：依赖先执行）+延迟执行

CMD:延迟执行（运行到需加载，根据顺序执行）

CMD 推崇依赖就近，AMD 推崇依赖前置。看如下代码：

// CMD

define(function(require, exports, module) {

var a = require('./a')

a.doSomething()

// 此处略去 100 行

var b = require('./b') // 依赖可以就近书写

b.doSomething()

// ...

})

// AMD 默认推荐的是

define(['./a', './b'], function(a, b) { // 依赖必须一开始就写好

a.doSomething()

// 此处略去 100 行

b.doSomething()

...

})

另外一个区别是：

AMD:API根据使用范围有区别，但使用同一个api接口

CMD:每个API的职责单一

AMD的优点是：异步并行加载，在AMD的规范下，同时异步加载是不会产生错误的。

CMD的机制则不同，这种加载方式会产生错误，如果能规范化模块内容形式，也可以