# 高级4

## 题目1： 为什么要使用模块化？


在模块化还未规范化之前，有三种模仿模块化的方法，分别是函数声明，对象返回，和立即执行函数封装，之后才有模块化的规范，在ES6中，也对“类”和“模块”有定义，不过要主流浏览器都支持时才能广泛使用。之所以使用模块化是因为：

解决命名冲突；
方便依赖性管理；
提高代码的维护性和可读性；
每个模块都是单独文件，解耦提高代码复用性；

## 题目2： CMD、AMD、CommonJS 规范分别指什么？有哪些应用

**AMD：**
是”Asynchronous Module Definition”的缩写，意思就是”异步模块定义”。采用异步方式加载模块，模块的加载不影响它后面语句的运行。所有依赖这个模块的语句，都定义在一个回调函数中，等到加载完成之后，这个回调函数才会运行。
语法：`define(id, dependences, factory);`

**CommonJS：**
用于服务器端模块化，有一个全局性方法require()，用于加载模块。

语法：

```
require(dependences) // 加载依赖模块
exports.factory = function(){ // ...};  // 使用”exports”对象来做为输出的唯一表示。
```

**CMD：**
CMD（Common Module Definition）是 SeaJS推广过程中产生的。和AMD不同的是，它并不是异步加载，而是松散加载，只有当需要加载模块的时候，再用require方法引用模块。

语法：`define(factory);`

```
// CMD
// math.js
define(function(requires, exports, module) {
exports.add = function(x, y) {
  return x + y;
};
});
// inc.js
define(function(requires, exports, module) {
var add = require('math').add;
exports.inc = function(val) {
  return add(val, 1);
};
});
// program.js
define(function(require, exports, module) {
var inc = require('inc').inc;
var a = 1;
inc(a); // 2

module.id = "program";
});
```

**require.js：**

```
// 加载模块设置
requirejs.config({

  baseUrl: 'js/libs',                  // 指明模块的默认路径

  paths: {
    'jquery': 'jquery.min',            // 每个模块都是JS文件，即HTTP请求多
    'underscore': 'underscore.min',    // 可以用require.js的优化工具合并模块
    'backbone': 'backbone.min',
    'jquery.scroll': 'jquery.scroll.min'
    'xxx': 'https://xxx.com/libs/xxx/1.7.2/xxx.min'
  }
  shim: {                              // 为非规范的模块定义特征，接受配置对象
    'underscore': {
      exports: '_'                     // exports为外部调用模块时所用的名称
    },
    'backbone': {
      deps: ['underscore', 'jquery'], // deps数组表明模块的依赖性
      exports: 'Backbone'
    },
    'jquery.scroll': {
```

```
      deps: ['jquery'],
      exports: 'jQuery.fn.scroll'
    }
  }
});

// 加载AMD模块，并用回调函数操作
requirejs(['jquery', 'underscore', 'backbone'], function($, _, Backbone) {
  //...
});

// 写AMD模块

// 1. 不依赖其他模块，例如：math.js
define(function() {
  var add = function(x, y) {
    return x + y;
  };
  return {
    add: add
  };
});
// 2. 依赖其他模块，需指明依赖数组
define(['mylib'], function(mylib) {
  function foo() {
    mylib.doSomething();
  }
  return {
    foo: foo
  };
});

// require.js提供了一些插件

//1. domready插件，回调函数在页面DOM结构加载完成时运行
require(['domready!'], function(doc) {
  //...
});
//2. text和image插件，允许require.js加载文件和图片文件
define(['text!review.txt', 'image!dog.jpg'], function(review, dog) {
  console.log(review);
  document.body.appendChild(dog);
});
// 3. 类似的插件还有json和mdown，加载json文件和markdown文件
```

**实例**

创建html文件，引入require.js库，设置data-main属性：

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title></title>
  </head>
  <body>
    <script src="http://apps.bdimg.com/libs/require.js/2.1.11/require.min.js" d
ata-main="main"></script>
  </body>
</html>
```

在html目录下新建js文件夹，创建main.js：

```
// main.js
require(['starting']);
console.log("starting");
```

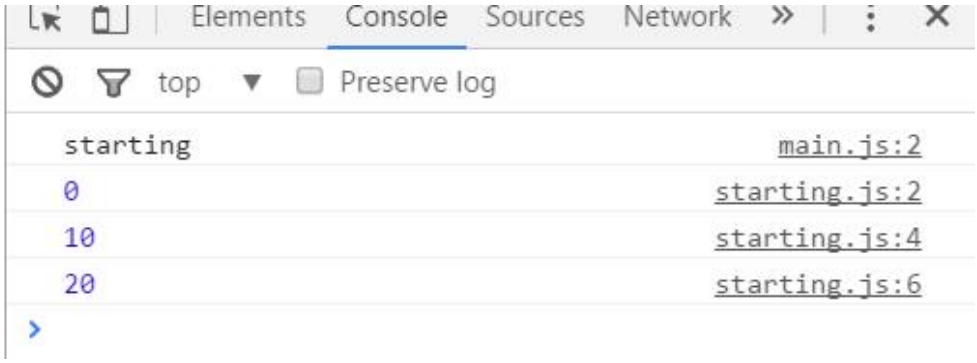
starting.js：

```
//starting.js

define(['inc'],function(inc){
   console.log(inc.getRes());
   inc.add1();
   console.log(inc.getRes());
   inc.add1();
   console.log(inc.getRes());
});
```

inc.js：

```
// inc.js

define(function() {

  var a = 0;

  var inc = {

    add1: function() {
      return a+=10;
    },

    getRes: function() {
      return a;
    }
  };
```

```
  return inc;
});
```




# 题目3： 使用 requirejs 完善入门任务15，包括如下功能：

1. 首屏大图为全屏轮播
2. 有回到顶部功能
3. 图片区使用瀑布流布局（图片高度不一），下部有加载更多按钮，点击加载更多会加载更多数据(数据在后端 mock)
4. 使用 r.js 打包应用
5.启动r.js: node r.js -o build.js
6.每次改动都要重新打包
7.路径指向很容易出错

1.预览地址
2.代码地址
3.点击加载瀑布流截图(使用XAMPP虚拟):