3. 模块化开发

模块化开发，**一个模块就是一个实现特定功能的文件，有了模块我们就可以更方便的使用别人的代码，要用什么功能就加载什么模块。**

模块化开发优点：

1 前后端更好的分离

2 避免变量污染，命名冲突

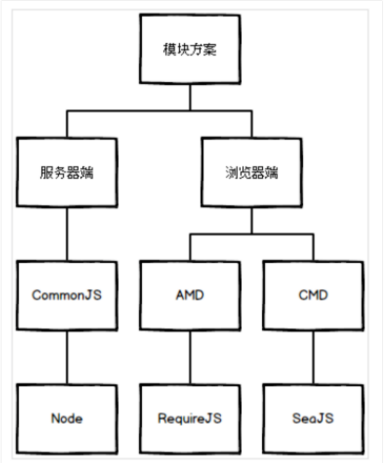
2 提高代码复用率

3 提高维护性，模块式的开发，一个文件就是一个模块，控制了文件的粒度，

每个模块可以专注于一个功能

4 依赖关系的管理，能更好的处理依赖

5 按需加载



CommonJS（常用在服务器端，同步的，如nodejs）

AMD（常用在浏览器端，异步的，如**requirejs**）（Asynchronous Module Definition）

CMD（常用在浏览器端，异步的，如**seajs)**

服务器端模块 在服务器端，所有的模块都存放在本地硬盘，可以同步加载完成，等待时间就是硬盘的读取时间。

浏览器端模块: 在浏览器端，所有的模块都放在服务器端，同步加载，等待时间取决于网速的快慢，可能要等很长时间，浏览器处于"假死"状态。因此，浏览器端的模块，不能采用"同步加载"（synchronous），只能采用"**异步加载**"（**asynchronous**）。

CommonJS语法

一 模块定义

根据commonJS规范，一个单独的文件是一个模块，每一个模块都是一个单独的作用域，也就是说，在该模块内部定义的变量，无法被其他模块读取，除非为global对象的属性。

二 模块暴露

模块只有一个出口，module.exports 对象，我们需要把模块希望输出的内容放入该对象。

三 模块引入

加载模块用require方法，该方法读取一个文件并且执行，返回文件内部的module.exports对象。

```
var name = 'Byron';

function printName(){
    console.log(name);
}

function printFullName(firstName){
    console.log(firstName + name);
}

module.exports = {
    printName: printName,
    printFullName: printFullName
```

然后加载模块

```
var nameModule = require('./myModel.js');
nameModule.printName();
```

AMD

Asynchronous Module Definition，异步模块定义。使用AMD规范进行页面开发需要用到对应的函数库RequireJS，实际上AMD是RequireJS在推广过程中对模块定义的规范化的产出。

requireJS主要解决两个问题：

1 多个js文件可能有依赖关系，被依赖的文件需要早于依赖它的文件加载到浏览器。

2 js加载的时候浏览器会停止页面渲染，加载文件愈多，页面失去响应的时间愈长。

```
//定义模块

define(['dependency'],function(){

  var name = 'Byron';
  function printName(){
     console.log(name);
}

  return {
     printName:printName
   }

})

//加载模块

require(['myModule'],function(my){
   my.printName();
})
```

语法：

requireJS定义了一个函数define，它是全局变量，用来定义模块。

define(id,dependencies,factory)

——id 可选参数，用来定义模块的标识，如果没有提供该参数，脚本文件名（去掉拓展名）

——dependencies 是一个当前模块依赖的模块名称数组

——factory 工厂方法，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。

在页面上使用require函数加载模块；

require([dependencies], function(){});

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；

——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

AMD推崇的是依赖前置，被提前罗列出来并会被提前下载并执行，后来做了改进，可以不用罗列依赖模块，允许在回调函数中就近使用require引入并下载执行模块。

CMD

即common module definition（通用模块定义）

CMD依赖sea.js，sj要解决的问题和rj一样，只不过在模块定义方式和模块加载时机上有所不同，cmd是sea.js在推广过程中的规范化产出，sea.js是另一种前端模块化工具，它的出现缓解了requireJS的几个痛点。

```
define(id, deps, factory)

因为CMD推崇一个文件一个模块，所以经常就用文件名作为模块id;
CMD推崇依赖就近，所以一般不在define的参数中写依赖，而是在factory中写。

factory有三个参数:
function(require, exports, module){}

一, require
require 是 factory 函数的第一个参数，require 是一个方法，接受 模块标识 作为唯一参数，用来获取其他模块提供的接口;

二, exports
exports 是一个对象，用来向外提供模块接口;

三, module
module 是一个对象，上面存储了与当前模块相关联的一些属性和方法。

demo
// 定义模块  myModule.js
define(function(require, exports, module) {
  var $ = require('jquery.js')
  $('div').addClass('active');
});

// 加载模块
seajs.use(['myModule.js'], function(my){

});
```

AMD与CMD区别

总结如下:

1.最明显的区别就是在模块定义时对依赖的处理不同。

AMD推崇依赖前置 在定义模块的时候就有声明其依赖的模块

CMD推崇就近依赖 只有在用到某模块的时候再去require

2.AMD依赖模块的执行顺序和书写顺序不一定一致；CMD模块的执行顺序和书写顺序是完全一致的。

AMD在加载模块完成后就会执行改模块，所有模块都加载执行完后会进入require的回调函数，执行主逻辑，看网络速度，哪个先下载下来，哪个先执行，但是主逻辑一定在所有依赖加载完成后才执行；CMD加载完某个依赖模块后并不执行，只是下载而已，在所有依赖模块加载完成后进入主逻辑，遇到require语句的时候才执行对应的模块，这样模块的执行顺序和书写顺序是完全一致的。

3.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，处理方式不通过）

这也是很多人说AMD用户体验好，因为没有延迟，依赖模块提前执行了，CMD性能好，因为只有用户需要的时候才执行的原因。