模块化的历史

非模块化存在的问题

问题1：命名重名问题，解决

问题2：被依赖的要先加载，解决

require的思想——解决了命名重名和依赖的文件要先加载

思想1：

思想2：

解决3：第三方框架

require.js，sea.js，es6.js

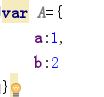
**1. 非模块化存在的问题**

我们做项目，一个js要引入里一个js,这个js又引入另外一个js，这个如果是非模块化，就要在HTML里面一个个引入js，并且js的加载顺序还要考虑这些js的命名问题。所以有以下两个问题。

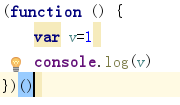
**问题1：命名重名问题**，解决

**解决1**. 可以用a\_a，b\_b，c\_c

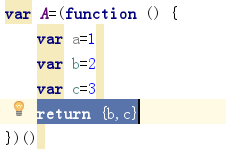
**解决2**. 可以用每个js创建一个对象



**解决3**. 闭包的形式，并且执行闭包的函数



**解决4**. 闭包和对象结合——不用将所有数据都返回处理，按需返回出需要的数据



问题2：被依赖的要先加载——不用script加载，按需加载

a.js依赖b.js，b.js依赖c.js

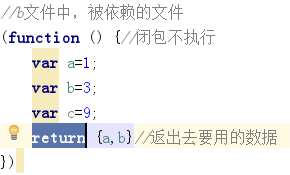
用require的思想来解决

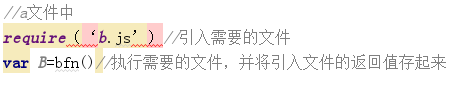
**2. require的思想——**解决了命名重名和依赖的文件要先加载

思想1：

被依赖的文件b.js，不需要命名空间，也不需要将闭包函数执行，将需要返回的数据存在一个对象中，并且返回出去。

a.js，加载文件b.js，执行闭包，返回的就是b.js中闭包的返回值





思想2：

被依赖的文件b.js，不需要命名空间，也不需要将闭包函数执行，传入一个参数，这个参数是对象，b.js需要传出去的数据存在对象下面，并且返回出去这个参数对象。

a.js，加载文件，require中创建一个对象，exports，然后把该对象传入到加载的文件中的函数内，执行被加载的函数。



解决3：第三方框架

require.js，sea.js，es6.js