## 模块化进化史教程 **1.** 全局function模式  
 **\*** module1.js  
 ```  
 //数据  
 let data = 'atguigu.com'  
   
 //操作数据的函数  
 function foo() {  
 console.log(`foo() ${data}`)  
 }  
 function bar() {  
 console.log(`bar() ${data}`)  
 }  
 ```  
 **\*** module2.js  
 ```  
 let data2 = 'other data'  
   
 function foo() { //与另一个模块中的函数冲突了  
 console.log(`foo() ${data2}`)  
 }  
 ```  
 **\*** test1.html  
 ```  
 <script type="text/javascript" src="module1.js"></script>  
 <script type="text/javascript" src="module2.js"></script>  
 <script type="text/javascript">  
   
 let data = "修改后的数据"  
 foo()  
 bar()  
 </script>  
 ```  
 **\*** 说明:  
 **\*** 全局函数模式: 将不同的功能封装成不同的全局函数  
 **\*** 问题: Global被污染了, 很容易引起命名冲突  
**2.** namespace模式  
 **\*** module1.js  
 ```  
 let myModule = {  
 data: 'atguigu.com',  
 foo() {  
 console.log(`foo() ${this.data}`)  
 },  
 bar() {  
 console.log(`bar() ${this.data}`)  
 }  
 }  
 ```  
 **\*** module2.js  
 ```  
 let myModule2 = {  
 data: 'atguigu.com2222',  
 foo() {  
 console.log(`foo() ${this.data}`)  
 },  
 bar() {  
 console.log(`bar() ${this.data}`)  
 }  
 }  
 ```  
 **\*** test2.html  
 ```  
 <script type="text/javascript" src="module2.js"></script>  
 <script type="text/javascript" src="module22.js"></script>  
 <script type="text/javascript">  
 myModule.foo()  
 myModule.bar()  
   
 myModule2.foo()  
 myModule2.bar()  
   
 myModule.data = 'other data' //能直接修改模块内部的数据  
 myModule.foo()  
   
 </script>  
 ```  
 **\*** 说明  
 **\*** namespace模式: 简单对象封装  
 **\*** 作用: 减少了全局变量  
 **\*** 问题: 不安全  
**3.** IIFE模式  
 **\*** module3.js  
 ```  
 (function (window) {  
 //数据  
 let data = 'atguigu.com'  
   
 //操作数据的函数  
 function foo() { //用于暴露有函数  
 console.log(`foo() ${data}`)  
 }  
   
 function bar() {//用于暴露有函数  
 console.log(`bar() ${data}`)  
 otherFun() //内部调用  
 }  
   
 function otherFun() { //内部私有的函数  
 console.log('otherFun()')  
 }  
   
 //暴露行为  
 window.myModule = {foo, bar}  
 })(window)  
 ```  
 **\*** test3.html  
 ```  
 <script type="text/javascript" src="module3.js"></script>  
 <script type="text/javascript">  
 myModule.foo()  
 myModule.bar()  
 //myModule.otherFun() //myModule.otherFun is not a function  
 console.log(myModule.data) //undefined 不能访问模块内部数据  
 myModule.data = 'xxxx' //不是修改的模块内部的data  
 myModule.foo() //没有改变  
   
 </script>  
 ```  
 **\*** 说明:  
 **\*** IIFE模式: 匿名函数自调用(闭包)  
 **\*** IIFE : immediately-invoked function expression(立即调用函数表达式)  
 **\*** 作用: 数据是私有的, 外部只能通过暴露的方法操作  
 **\*** 问题: 如果当前这个模块依赖另一个模块怎么办?  
**4.** IIFE模式增强  
 **\*** 引入jquery到项目中  
 **\*** module4.js  
 ```  
 (function (window, $) {  
 //数据  
 let data = 'atguigu.com'  
   
 //操作数据的函数  
 function foo() { //用于暴露有函数  
 console.log(`foo() ${data}`)  
 $('body').css('background', 'red')  
 }  
   
 function bar() {//用于暴露有函数  
 console.log(`bar() ${data}`)  
 otherFun() //内部调用  
 }  
   
 function otherFun() { //内部私有的函数  
 console.log('otherFun()')  
 }  
   
 //暴露行为  
 window.myModule = {foo, bar}  
 })(window, jQuery)  
 ```   
 **\*** test4.html  
 ```  
 <script type="text/javascript" src="jquery-1.10.1.js"></script>  
 <script type="text/javascript" src="module4.js"></script>  
 <script type="text/javascript">  
 myModule.foo()  
 </script>  
 ```  
 **\*** 说明  
 **\*** IIFE模式增强 : 引入依赖  
 **\*** 这就是现代模块实现的基石  
   
**5.** 页面加载多个js的问题  
 **\*** 页面:  
 ```  
 <script type="text/javascript" src="module1.js"></script>  
 <script type="text/javascript" src="module2.js"></script>  
 <script type="text/javascript" src="module3.js"></script>  
 <script type="text/javascript" src="module4.js"></script>  
 ```  
 **\*** 说明  
 **\*** 一个页面需要引入多个js文件  
 **\*** 问题:  
 **\*** 请求过多  
 **\*** 依赖模糊  
 **\*** 难以维护  
 **\*** 这些问题可以通过现代模块化编码和项目构建来解决

## Node.js模块化教程 **1.** 下载安装node.js  
**2.** 创建项目结构  
 ```  
 |-modules  
 |-module1.js  
 |-module2.js  
 |-module3.js  
 |-app.js  
 |-package.json  
 {  
 "name": "commonJS-node",  
 "version": "1.0.0"  
 }  
 ```  
**3.** 下载第三方模块  
 **\*** npm install uniq --save  
**4.** 模块化编码  
 **\*** module1.js  
 ```  
 module.exports = {  
 foo() {  
 console.log('moudle1 foo()')  
 }  
 }  
 ```  
 **\*** module2.js  
 ```  
 module.exports = function () {  
 console.log('module2()')  
 }  
 ```  
 **\*** module3.js  
 ```  
 exports.foo = function () {  
 console.log('module3 foo()')  
 }  
   
 exports.bar = function () {  
 console.log('module3 bar()')  
 }  
 ```  
 **\*** app.js   
 ```  
 /\*\*  
 **1.** 定义暴露模块:  
 module.exports = value;  
 exports.xxx = value;  
 **2.** 引入模块:  
 var module = require(模块名或模块路径);  
 \*/  
 "use strict";  
 //引用模块  
 let module1 = require('./modules/module1')  
 let module2 = require('./modules/module2')  
 let module3 = require('./modules/module3')  
   
 let uniq = require('uniq')  
 let fs = require('fs')  
   
 //使用模块  
 module1.foo()  
 module2()  
 module3.foo()  
 module3.bar()  
   
 console.log(uniq(**[**1, 3, 1, 4, 3**]**))  
   
 fs.readFile('app.js', function (error, data) {  
 console.log(data.toString())  
 })  
 ```  
**5.** 通过node运行app.js  
 **\*** 命令: node app.js  
 **\*** 工具: 右键-->运行

## Browserify模块化使用教程 **1.** 创建项目结构  
 ```  
 |-js  
 |-dist //打包生成文件的目录  
 |-src //源码所在的目录  
 |-module1.js  
 |-module2.js  
 |-module3.js  
 |-app.js //应用主源文件  
 |-index.html  
 |-package.json  
 {  
 "name": "browserify-test",  
 "version": "1.0.0"  
 }  
 ```  
**2.** 下载browserify  
 **\*** 全局: npm install browserify -g  
 **\*** 局部: npm install browserify --save-dev  
**3.** 定义模块代码  
 **\*** module1.js  
 ```  
 module.exports = {  
 foo() {  
 console.log('moudle1 foo()')  
 }  
 }  
 ```  
 **\*** module2.js  
 ```  
 module.exports = function () {  
 console.log('module2()')  
 }  
 ```  
 **\*** module3.js  
 ```  
 exports.foo = function () {  
 console.log('module3 foo()')  
 }  
   
 exports.bar = function () {  
 console.log('module3 bar()')  
 }  
 ```  
 **\*** app.js (应用的主js)  
 ```  
 //引用模块  
 let module1 = require('./module1')  
 let module2 = require('./module2')  
 let module3 = require('./module3')  
   
 let uniq = require('uniq')  
   
 //使用模块  
 module1.foo()  
 module2()  
 module3.foo()  
 module3.bar()  
   
 console.log(uniq(**[**1, 3, 1, 4, 3**]**))  
 ```  
**\*** 打包处理js:  
 **\*** browserify js/src/app.js -o js/dist/bundle.js  
**\*** 页面使用引入:  
 ```  
 <script type="text/javascript" src="js/dist/bundle.js"></script>   
 ```

## AMD require.js使用教程 **1.** 下载require.js, 并引入  
 **\*** 官网: **http://www.requirejs.cn/  
 \*** github : **https://github.com/requirejs/requirejs  
 \*** 将require.js导入项目: js/libs/require.js   
**2.** 创建项目结构  
 ```  
 |-js  
 |-libs  
 |-require.js  
 |-modules  
 |-alerter.js  
 |-dataService.js  
 |-main.js  
 |-index.html  
 ```  
**3.** 定义require.js的模块代码  
 **\*** dataService.js  
 ```  
 define(function () {  
 let msg = 'atguigu.com'  
   
 function getMsg() {  
 return msg.toUpperCase()  
 }  
   
 return {getMsg}  
 })  
 ```  
 **\*** alerter.js  
 ```  
 define(**[**'dataService', 'jquery'**]**, function (dataService, $) {  
 let name = 'Tom2'  
   
 function showMsg() {  
 $('body').css('background', 'gray')  
 alert(dataService.getMsg() + ', ' + name)  
 }  
   
 return {showMsg}  
 })  
 ```  
**4.** 应用主(入口)js: main.js  
 ```  
 (function () {  
 //配置  
 require.config({  
 //基本路径  
 baseUrl: "js/",  
 //模块标识名与模块路径映射  
 paths: {  
 "alerter": "modules/alerter",  
 "dataService": "modules/dataService",  
 }  
 })  
 //引入使用模块  
 require( ['alerter'], function(alerter) {  
 alerter.showMsg()  
 })  
 })()  
 ```  
   
**5.** 页面使用模块:  
 <script data-main="js/main" src="js/libs/require.js"></script>  
   
**------------------------------------------------------------------------  
  
6.** 使用第三方基于require.js的框架(jquery)  
 **\*** 将jquery的库文件导入到项目:   
 **\*** js/libs/jquery-1.10.1.js  
 **\*** 在main.js中配置jquery路径  
 ```  
 paths: {  
 'jquery': 'libs/jquery-1.10.1'  
 }  
 ```  
 **\*** 在alerter.js中使用jquery  
 ```  
 define(['dataService', 'jquery'], function (dataService, $) {  
 var name = 'xfzhang'  
 function showMsg() {  
 $('body').css({background : 'red'})  
 alert(name + ' '+dataService.getMsg())  
 }  
 return {showMsg}  
 })  
 ```  
**------------------------------------------------------------------------  
  
7.** 使用第三方不基于require.js的框架(angular/angular-messages)  
 **\*** 将angular.js和angular-messages.js导入项目  
 **\*** js/libs/angular.js  
 **\*** js/libs/angular-messages.js  
 **\*** 在main.js中配置  
 ```  
 (function () {  
 require.config({  
 //基本路径  
 baseUrl: "js/",  
 //模块标识名与模块路径映射  
 paths: {  
 //第三方库  
 'jquery' : 'libs/jquery-1.10.1',  
 'angular' : 'libs/angular',  
 'angular-messages' : 'libs/angular-messages',  
 //自定义模块  
 "alerter": "modules/alerter",  
 "dataService": "modules/dataService"  
 },  
 /\*  
 配置不兼容AMD的模块  
 exports : 指定导出的模块名  
 deps : 指定所有依赖的模块的数组  
 \*/  
 shim: {  
 'angular' : {  
 exports : 'angular'  
 },  
 'angular-messages' : {  
 exports : 'angular-messages',  
 deps : ['angular']  
 }  
 }  
 })  
 //引入使用模块  
 require( ['alerter', 'angular', 'angular-messages'], function(alerter, angular) {  
 alerter.showMsg()  
 angular.module('myApp', ['ngMessages'])  
 angular.bootstrap(document,["myApp"])  
 })  
 })()  
 ```  
 **\*** 页面:  
 ```  
 <form name="myForm">  
 用户名: <input type="text" name="username" ng-model="username" ng-required="true">  
 <div style="color: red;" ng-show="myForm.username.$dirty&&myForm.username.$invalid">用户名是必须的</div>  
 </form>  
 ```

## ES6-Babel-Browserify使用教程 **1.** 定义package.json文件  
 ```  
 {  
 "name" : "es6-babel-browserify",  
 "version" : "1.0.0"  
 }  
 ```  
**2.** 安装babel-cli, babel-preset-es2015和browserify  
 **\*** npm install babel-cli browserify -g  
 **\*** npm install babel-preset-es2015 --save-dev   
**3.** 定义.babelrc文件  
 ```  
 {  
 "presets": ["es2015"]  
 }  
 ```  
**4.** 编码  
 **\*** js/src/module1.js  
 ```  
 export function foo() {  
 console.log('module1 foo()');  
 }  
 export let bar = function () {  
 console.log('module1 bar()');  
 }  
 export const DATA\_ARR = [1, 3, 5, 1]  
 ```  
 **\*** js/src/module2.js  
 ```  
 let data = 'module2 data'  
   
 function fun1() {  
 console.log('module2 fun1() ' + data);  
 }  
   
 function fun2() {  
 console.log('module2 fun2() ' + data);  
 }  
   
 export {fun1, fun2}  
 ```  
 **\*** js/src/module3.js  
 ```  
 export default {  
 name: 'Tom',  
 setName: function (name) {  
 this.name = name  
 }  
 }  
 ```  
 **\*** js/src/app.js  
 ```  
 import {foo, bar} from './module1'  
 import {DATA\_ARR} from './module1'  
 import {fun1, fun2} from './module2'  
 import person from './module3'  
   
 import $ from 'jquery'  
   
 $('body').css('background', 'red')  
   
 foo()  
 bar()  
 console.log(DATA\_ARR);  
 fun1()  
 fun2()  
   
 person.setName('JACK')  
 console.log(person.name);  
 ```  
**5.** 编译  
 **\*** 使用Babel将ES6编译为ES5代码(但包含CommonJS语法) :

babel js/src -d js/lib

要直接使用babel,必须全局安装 npm install babel-cli -g  
 **\*** 使用Browserify编译js :

browserify js/lib/app.js -o js/lib/bundle.js  
**6.** 页面中引入测试  
 ```  
 <script type="text/javascript" src="js/lib/bundle.js"></script>  
 ```  
**7.** 引入第三方模块(jQuery)  
 1). 下载jQuery模块:   
 **\*** npm install jquery@1 --save  
 2). 在app.js中引入并使用  
 ```  
 import $ from 'jquery'  
 $('body').css('background', 'red')  
 ```

# 总结

**## JS模块化  
\*** 模块化的理解  
**\*** 什么是模块?  
 **\*** 将一个复杂的程序依据一定的规则(规范)封装成几个块(文件), 并进行组合在一起  
 **\*** 块的内部数据/实现是私有的, 只是向外部暴露一些接口(方法)与外部其它模块通信  
**\*** 一个模块的组成  
 **\*** 数据--->内部的属性  
 **\*** 操作数据的行为--->内部的函数  
**\*** 模块化  
 **\*** 编码时是按照模块一个一个编码的, 整个项目就是一个模块化的项目  
**\*** 模块化的进化过程  
 **\*** 全局function模式 :   
 **\*** 编码: 全局变量/函数  
 **\*** 问题: 污染全局命名空间, 容易引起命名冲突/数据不安全  
 **\*** namespace模式 :   
 **\*** 编码: 将数据/行为封装到对象中  
 **\*** 解决: 命名冲突(减少了全局变量)  
 **\*** 问题: 数据不安全(外部可以直接修改模块内部的数据)  
 **\*** IIFE模式/增强  
 **\*** IIFE : 立即调用函数表达式--->匿名函数自调用  
 **\*** 编码: 将数据和行为封装到一个函数内部, 通过给window添加属性来向外暴露接口  
 **\*** 引入依赖: 通过函数形参来引入依赖模块  
 ```  
 (function(window, module2){  
 var data = 'atguigu.com'  
 function foo() {  
 module2.xxx()  
 console.log('foo()'+data)  
 }  
 function bar() {  
 console.log('bar()'+data)  
 }  
   
 window.module = {foo}  
 })(window, module2)  
 ```  
**\*** 模块化规范  
 **\*** CommonJS  
 **\*** Node.js : 服务器端  
 **\*** Browserify : 浏览器端 也称为js的打包工具  
 **\*** 基本语法:  
 **\*** 定义暴露模块 : exports  
 ```  
 exports.xxx = value  
 module.exports = value  
 ```  
 引入模块 : require  
 ```  
 var module = require('模块名/模块相对路径')  
 ```  
 **\*** 引入模块发生在什么时候?  
 **\*** Node : 运行时, 动态同步引入  
 **\*** Browserify : 在运行前对模块进行编译/转译/打包的处理(已经将依赖的模块包含进来了),   
 运行的是打包生成的js, 运行时不存在需要再从远程引入依赖模块  
 **\*** AMD : 浏览器端  
 **\*** require.js  
 **\*** 基本语法  
 **\*** 定义暴露模块: define(**[**依赖模块名**]**, function(){return 模块对象})  
 **\*** 引入模块: require(**[**'模块1', '模块2', '模块3'**]**, function(m1, m2){//使用模块对象})  
 **\*** 配置:   
 ```  
 require.config({  
 //基本路径  
 baseUrl : 'js/',  
 //标识名称与路径的映射  
 paths : {  
 '模块1' : 'modules/模块1',  
 '模块2' : 'modules/模块2',  
 'angular' : 'libs/angular',  
 'angular-messages' : 'libs/angular-messages'  
 },  
 //非AMD的模块  
 shim : {  
 'angular' : {  
 exports : 'angular'  
 },  
 'angular-messages' : {  
 exports : 'angular-messages',  
 deps : ['angular']  
 }  
 }  
 })  
 ```  
 **\*** CMD : 浏览器端  
 **\*** sea.js  
 **\*** 基本语法  
 **\*** 定义暴露模块:   
 ```  
 define(function(require, module, exports){  
 通过require引入依赖模块  
 通过module/exports来暴露模块  
 exports.xxx = value  
 })  
 ```  
 **\*** 使用模块seajs.use(**[**'模块1', '模块2'**]**)  
 **\*** ES6  
 **\*** ES6内置了模块化的实现  
 **\*** 基本语法  
 **\*** 定义暴露模块 : export  
 **\*** 暴露一个对象:   
 ```  
 export default 对象  
 ```  
 **\*** 暴露多个:   
 ```  
 export var xxx = value1  
 export let yyy = value2  
   
 var xxx = value1  
 let yyy = value2  
 export {xxx, yyy}  
 ```  
   
 **\*** 引入使用模块 : import  
 **\*** default模块:  
 ```  
 import xxx from '模块路径/模块名'  
 ```  
 **\*** 其它模块  
 ```  
 import {xxx, yyy} from '模块路径/模块名'  
 import \* as module1 from '模块路径/模块名'  
 ```  
 **\*** 问题: 所有浏览器还不能直接识别ES6模块化的语法   
 **\*** 解决:  
 **\*** 使用Babel将ES6--->ES5(使用了CommonJS) ----浏览器还不能直接支行  
 **\*** 使用Browserify--->打包处理----浏览器可以运行