# 模块系统

1. EcmaScript语言

* 和浏览器不一样，在Node中没有BOM,DOM

1. 核心模块

* 文件操作的fs
* http服务的http
* url路径操作模块
* Path路径
* Os操作系统信息

1. 第三方模块

* Art-template
* 必须通过npm下载才可以使用

1. 自定义模块

* 自己创建的模块

# 什么是模块化

1. 文件作用域
2. 通信规则

* 加载 require
* 导出

# 三．commonJS模块规范

在Node中的JS还有一个很重要的概念，模块系统。

* 模块作用域
* 使用require方法用来加载模块
* 使用exports接口对象来导出模块中的成员

## 3-1. 加载 require

语法：

var 自定义名称 = require(‘模块’)

两个作用：

* 执行被加载模块中的代码
* 得到加载模块中的exports导出接口对象

## 3-2. require方法加载规则

* 核心模块 （本质也是文件，文件已经被编译到了二进制文件中了，只需要按名字来加载，例如：require(‘fs’)）
* 第三方模块（需要安装 例如：安装art-tempalte，第三方都需要通过require(‘包名’)的方式来加载才可以使用）

art-template加载过程：

1. 找到node\_modules目录；
2. node\_modules/art-template
3. node\_modules/art-template/package.json文件
4. node\_modules/art-template/package.json文件中的main属性
5. main属性中就记录了art-template的入口模块，默认index.js文件（如果没有package.json就直接找默认index.js文件，没有index.js文件就报错）
6. 加载使用这个第三方包，实际上加载的还是文件

* 自定义模块（需要使用目录的路径加载 例如：require(./a) ../a为上一级目录文件）

### 模块查找机制

#### 优先从缓存加载

加载过了的不会重复加载，可以拿到其中的接口对象，但是不会重复执行里面的代码 （c.js只加载了一次）



#### 核心模块

#### **路径形式的模块**

./xxx 当前路径

../xxx 上一级目录路径

/xxx `/`在这表示磁盘根路径

C：/xxx

#### **第三方模块**

第三方模块的标识就是第三方模块的名称（不可能有第三方模块和核心模块的名字是一样的）

通过npm下载

使用方式：var 名称 = require(‘npm install下载的包名')

## 3-3. 导出 exports接口对象

* Node中是模块作用域，默认文件中所有的成员只在当前文件模块有效
* 对于希望可以被其他模块访问的成员，我们就需要把这些公开的成员都挂载到exports接口对象中就可以了
* 导出多个成员（必须在对象中）：

exports.a=123;

exports.b=’hello’;

exports.c = function(){

Conlose.log(‘ccc’)

}

exports.d ={

Foo:’bar’

}

方法2：module.exports = {

foo:’dfg’,

a:234

}

* 导出单个成员（拿到的就是函数，字符串）必须为module.exports：

module.exports = ‘hello’;

module.exports = function(x,y){

Return x+y;

}

赋值 后面在数据会覆盖前面的数据

module也可以导出多个成员

module.exports = {

add: function(x,y){

Return x+y;

},

str:’hello’

}

exports和module.exports是等价的

console.log(exports===module.exports); //true

//module.exports.add=添加成员

exports.foo = 'bar'; 等价于 module.exports.foo = ‘bar’;

module.exports.add = function(x,y){

return x+y;

}

//module.exports=只是赋值

最终的返回值为module.exports

# 总结：

## 导出多个成员：

module.exports.xxx=xxx

module.exports ={

foo:’345’,

a:2334

}

exports.a=’dfd’

expoets.add=function(){}

## 导出单个成员(必须写为module.exports)：

module.exports=xxx

# 案例

//案例3

//{foo:'bar',a:123}

exports.foo = 'bar';

module.exports.a= 123;

//更换对象 导致exports != module.exports

exports = {

    a:234

}

//赋值 {foo:'haha',a:123}

module.exports.foo ='haha';

//exports和module.exports没有关系

exports.c=345;

//两个对象相等

exports=module.exports;

//{foo:'haha',a:675}

exports.a = 675;

//重新赋值 ，最终返回值为module.exports

module.exports = function(){

    console.log('dffg');

}

module.exports = {

    foo:'sdf',

    a:123

}

# exports和module.exports的区别

每一个模块中都有一个module对象

module对象中有一个exports对象

我可以把需要导出的成员都挂载到module.exports接口对象中，也就是以module.exports.xxx=xxx的方式

但是每次都要module.exports.xxx=xxx很麻烦，所以为了方便同时在每个模块了一个成员叫exports,所有exports===module.exports结果为true

所有，module.exports.xxx=xxx可以使用exports.xxx=xxx

当模块需要导出单个成员时必须使用module.exports=xxx的方式，不可以直接exports=xxx

因为每个模块最终向外return的是module.exports

而exports只是一个引用，所有exports=xxx重新赋值也不影响module.exports的结果

但是有一种赋值比较特殊exports===module.exports，这个是重新建立引用关系