笔记3.

Node中的模块系统

> 使用 Node 编写应用程序主要就是在使用：

- EcmaScript 语言
  - 和浏览器不一样，在 Node 中没有 BOM、DOM
- 核心模块
  - 文件操作 `fs`
  - http 服务的 `http` 模块
  - `url` 路径操作模块
  - `path` 路径处理模块
  - `os` 操作系统信息
- 第三方模块
  - `art-template`
  - 必须通过 `npm` 来下载才可以使用
- 自己写的模块
  - 创建的文件

> 什么是模块化

- 文件作用域
- 通信规则
  - 加载 require
  - 导出

> commonJS 模块规范

在 Node 中的 JavaScript 还有一个很重要的概念：模块系统

- 模块作用域
- 使用 require 方法加载模块
- 使用 exports 接口对象来导出模块中的成员

> 加载 `require`

语法：

```
var 自定义模块名称 = require('模块')
```

两个作用：

- 执行被加载模块中的代码
- 得到被加载模块中的 `exports` 导出接口对象

> 导出 `exports`

- Node 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到的成员，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 = 12
exports.b = 'he good'
exports.c = function(){
    console.log('c')
}
exports.d = {
    name:'it',
    age:12,
    like:'打球'
}
```

导出单个成员（拿到的数据是：函数、字符串、数字...）：

```
module.exports = 12
```

一下情况会被覆盖：

```
module.exports = 12

// 以这个为准，后者会覆盖前者
module.exports = function(x,y){
    return x+y
}
```

也可以这样来导出多个成员：

```
module.exports = {
    name:'Jack',
    age:12,
    hobbies:['篮球','乒乓球','羽毛球'],
    father:'John',
    mother:'Milen',
    add:function(x,y){
        return x+y
    }
}
```

`exports` 和 `module.exports` 的区别

- exports 和 module.exports 的区别
  - 每个模块中都有一个 module 对象
  - module 对象中有一个 exports 对象
  - 我们可以把需要导出的成员都挂载到 module.exports 接口对象中
  - 也就是：`moudle.exports.xxx = xxx` 的方式
  - 但是每次都 `moudle.exports.xxx = xxx` 很麻烦，点儿的太多了
  - 所以 Node 为了你方便，同时在每一个模块中都提供了一个成员叫：`exports`
  - `exports === module.exports` 结果为 `true`s
  - 所以对于：`moudle.exports.xxx = xxx` 的方式 完全可以：`expots.xxx = xxx`
  - 当一个模块需要导出单个成员的时候，这个时候必须使用：`module.exports = xxx` 的方式
  - 不要使用 `exports = xxx` 不管用
  - 因为每个模块最终向外 `return` 的是 `module.exports`
  - 而 `exports` 只是 `module.exports` 的一个引用
  - 所以即便你为 `exports = xx` 重新赋值，也不会影响 `module.exports`
  - 但是有一种赋值方式比较特殊：`exports = module.exports` 这个用来重新建立引用关系的

> require 方法加载规则

- 核心模块 +
- 第三方模块 +
- 自己写的文件模块 +