# 模块化

## ECMAScript标准的缺陷

• 没有模块系统

• 标准库较少

• 没有标准接口

• 缺乏管理系统

## 模块化

• 如果程序设计的规模达到了一定程度，则必须对其进行模块化。

• 模块化可以有多种形式，但至少应该提供能够将代码分割为多个源文件的机制。

• CommonJS 的模块功能可以帮我们解决该问题。

# CommonJS规范

## CommonJS规范

CommonJS规范的提出，主要是为了弥补当前JavaScript没有模块化标准的缺陷。CommonJS规范为JS指定了一个美好的愿景，希望JS能够在任何地方运行。

CommonJS对模块的定义十分简单：模块引用；模块定义； 模块标识

## 模块引用

• 在规范中，定义了require()方法，这个方法接手模块标识，以此将一个模块引入到

当前运行环境中。

• 模块引用的示例代码：– var math = require('math');

## 模块定义

### 模块作用域

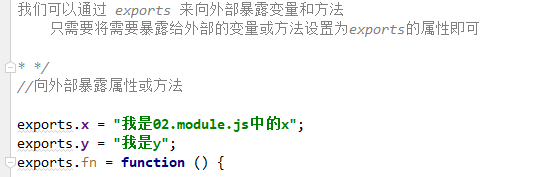
在运行环境中，提供了exports对象用于导出当前模块的方法或者变量，并且它是唯

一的导出的出口。

在模块中还存在一个module对象，它代表模块自身，而exports是module的属性。

在Node中一个文件就是一个模块。

### exports暴露



## 模块标识

### *require()引入*

模块标识其实就是模块的名字，也就是传递给require()方法的参数，它必须是符合驼峰命名法的字符串，或者是以.、..开头的相对路径、或者绝对路径。

*在node中，通过require()函数来引入外部的模块  
 require()可以传递一个文件的路径作为参数，node将会自动根据该路径来引入外部模块，这里路径，如果使用相对路径，必须以.或..开头*

*使用require()引入模块以后，该函数会返回一个对象，这个对象代表的是引入的模块*



### 独立

模块的定义十分简单，接口也十分简洁。每个模块具有独立的空间，它们互不干扰，在引用时也显得干净利落。

# Node的模块实现

## 概念

• Node中虽然使用的是CommonJS规范，但是其自身也对规范做了一些取舍。

• 在Node中引入模块，需要经历如下3个步骤：

路径分析

文件定位

编译执行

• 在Node中，模块分为三类：一类是底层由C++编写的内建模块，一类是Node提供的核

心模块；还有一类是用户编写的模块，称为文件模块。

## 模块标识类型

*我们使用require()引入外部模块时，使用的就是模块标识，我们可以通过模块标识来找到指定的模块，模块分成两大类*

### 核心模块

*核心模块  
 - 由node引擎提供的模块  
 - 核心模块的标识就是，模块的名字*

### 文件模块

*文件模块  
 - 由用户自己创建的模块  
 - 文件模块的标识就是文件的路径（绝对路径，相对路径）  
 相对路径使用.或..开头*

# exports

## exports

### 概念（被包裹）

*在node中有一个全局对象 global，它的作用和网页中window类似  
 在全局中创建的变量都会作为global的属性保存  
 在全局中创建的函数都会作为global的方法保存*

### 参数详解

*当node在执行模块中的代码时，它会首先在代码的最顶部，添加如下代码  
 function (exports, require, module, \_\_filename, \_\_dirname) {  
  
 在代码的最底部，添加如下代码  
 }  
  
 实际上模块中的代码都是包装在一个函数中执行的，并且在函数执行时，同时传递进了5个实参  
 exports  
 - 该对象用来将变量或函数暴露到外部  
  
 require  
 - 函数，用来引入外部的模块  
  
 module  
 - module代表的是当前模块本身  
 - exports就是module的属性  
 - 既可以使用 exports 导出，也可以使用module.exports导出  
  
 \_\_filename  
 C:\Users\lilichao\WebstormProjects\class0705\01.node\04.module.js  
 - 当前模块的完整路径  
  
 \_\_dirname  
 C:\Users\lilichao\WebstormProjects\class0705\01.node  
 - 当前模块所在文件夹的完整路径*

## exports 和module.exports

Exports只能单个导出，区别来源就是值传递和引用传递

### 使用 exports

- 例子：

exports.属性 = 属性值;

exports.方法 = 函数;

### 使用module.exports

- 例子：

module.exports.属性 = 属性值;

module.exports.方法 = 函数;

module.exports = {};