



黑马程序员™  
www.itheima.com

传智播客旗下  
高端IT教育品牌

# Node.js 基础

# 目录 Contents

- ◆ Node开发概述
- ◆ Node运行环境搭建
- ◆ Node.js快速入门

# 1. Node开发概述

## 1.1 为什么要学习服务器端开发基础

- 能够和后端程序员更加紧密的配合
- 网站业务逻辑前置，学习前端技术需要后端技术支撑 (Ajax)
- 扩宽知识视野，能够站在更高的角度审视整个项目

# 1. Node开发概述

## 1.2 服务器端开发要做的事情

- 实现网站的业务逻辑
- 数据的增删改查

# 1. Node开发概述

## 1.3 为什么选择Node

- 使用JavaScript语法开发后端应用
- 一些公司要求前端工程师掌握Node开发
- 生态系统活跃，有大量开源库可以使用
- 前端开发工具大多基于Node开发

# 1. Node开发概述

## 1.4 Node是什么

**Node**是一个基于Chrome V8引擎的JavaScript**代码运行环境**。



node-v8.11.2-x  
64.msi



node-v8.11.2-x  
86.msi

### 运行环境

- 浏览器（软件）能够运行JavaScript代码，浏览器就是JavaScript代码的运行环境
- Node（软件）能够运行JavaScript代码，Node就是JavaScript代码的运行环境

# 目录 Contents

- ◆ Node开发概述
- ◆ Node运行环境搭建
- ◆ Node.js快速入门

## 2. Node运行环境搭建




### 2.1 Node.js运行环境安装

官网: <https://nodejs.org/en/>

**10.13.0 LTS**  
Recommended For Most Users

**11.1.0 Current**  
Latest Features

- LTS = Long Term Support 长期支持版 稳定版
- Current 拥有最新特性 实验版

| LTS<br>Recommended For Most Users  | Current<br>Latest Features   |   |
|--|--|---|
| <br>Windows Installer<br>node-v8.12.0-x64.msi | <br>macOS Installer<br>node-v8.12.0.pkg | <br>Source Code<br>node-v8.12.0.tar.gz |

Windows Installer (.msi)

Windows Binary (.zip)

macOS Installer (.pkg)

macOS Binary (.tar.gz)

Linux Binaries (x86/x64)

Linux Binaries (ARM)

Source Code

|                     |        |       |
|---------------------|--------|-------|
| 32-bit              | 64-bit |       |
| 32-bit              | 64-bit |       |
| 64-bit              |        |       |
| 64-bit              |        |       |
| 32-bit              | 64-bit |       |
| ARMv6               | ARMv7  | ARMv8 |
| node-v8.12.0.tar.gz |        |       |



## 2. Node运行环境搭建

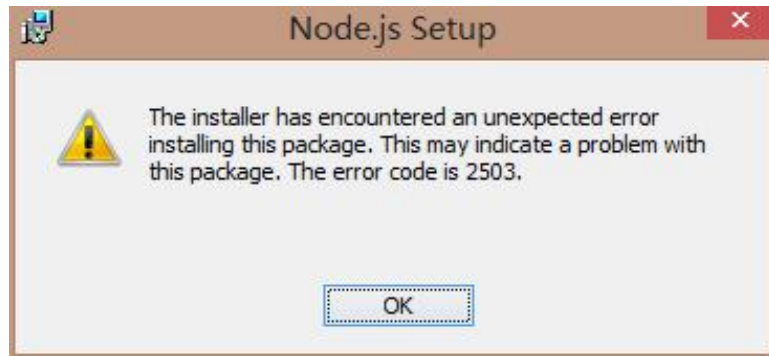
### 2.2 Node环境安装失败解决办法

#### 1. 错误代号2502、2503

失败原因：系统帐户权限不足。

解决办法：

1. 以管理员身份运行powershell命令行工具
2. 输入运行安装包命令 `msiexec /package node安装包位置`



## 2. Node运行环境搭建

### 2.2 Node.js环境安装失败解决办法

#### 2. 执行命令报错

失败原因：Node安装目录写入环境变量失败

解决办法：将Node安装目录添加到环境变量中



```
Windows PowerShell
版权所有 (C) Microsoft Corporation。保留所有权利。

PS C:\Users\itheima> node -v
node : 无法将“node”项识别为 cmdlet、函数、脚本文件或可运行程序的名称。请检查名称的拼写，如果包括路径，请确保路径正确，然后再试一次。
所在位置 行:1 字符: 1
+ node -v
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (node:String) [], CommandNotFoundException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\Users\itheima>
```



# 目录

# Contents

- ◆ Node开发概述
- ◆ Node运行环境搭建
- ◆ Node.js快速入门

## 3. Node.js快速入门

### 3.1 Node.js 的组成

- JavaScript 由三部分组成, **ECMAScript**, **DOM**, **BOM**。
- Node.js是由**ECMAScript**及**Node 环境**提供的一些**附加API**组成的, 包括文件、网络、路径等等一些更加强大的 API。

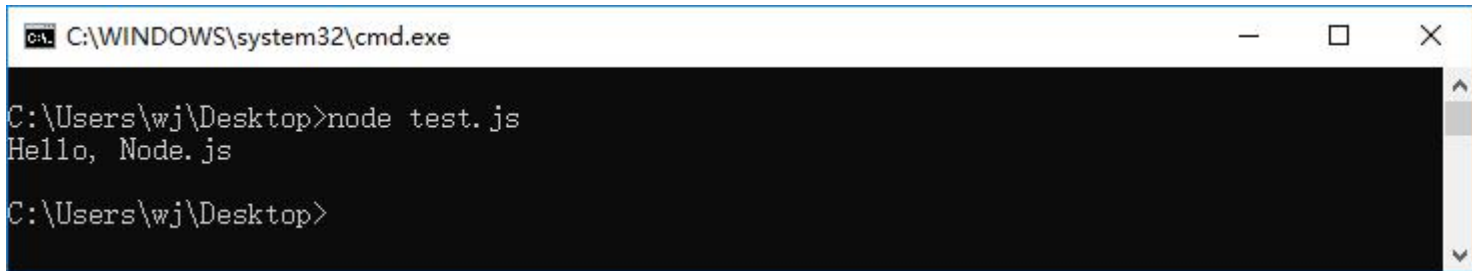


# ■ 3. Node.js快速入门

## 3.2 Node.js基础语法

所有ECMAScript语法在Node环境中都可以使用。

在Node环境下执行代码，使用Node命令执行后缀为.js的文件即可



```
Ca. C:\WINDOWS\system32\cmd.exe

C:\Users\wj\Desktop>node test.js
Hello, Node.js

C:\Users\wj\Desktop>
```

## 3. Node.js快速入门

### 3.3 Node.js全局对象global

在浏览器中全局对象是window，在Node中全局对象是global。

Node中全局对象下有以下方法，可以在任何地方使用，global可以省略。

- console.log() 在控制台中输出
- setTimeout() 设置超时定时器
- clearTimeout() 清除超时时定时器
- setInterval() 设置间歇定时器
- clearInterval() 清除间歇定时器

传智播客旗下  
高端IT教育品牌



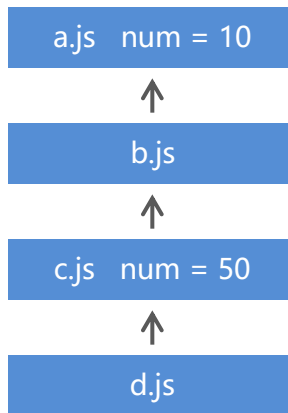
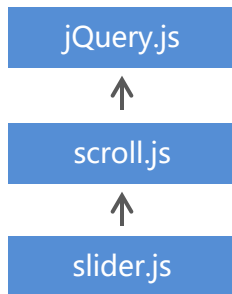
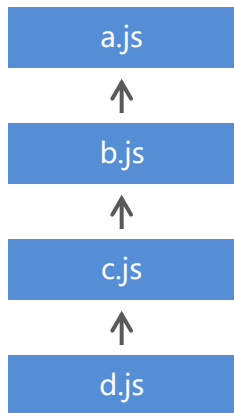
# 目录 Contents

- ◆ Node.js模块化开发
  - ◆ 系统模块
  - ◆ 第三方模块
  - ◆ package.json文件
  - ◆ Node.js中模块的加载机制

# 1. Node.js模块化开发

## 1.1 JavaScript开发弊端

JavaScript在使用时存在两大问题，**文件依赖**和**命名冲突**。



# ■ 1. Node.js模块化开发

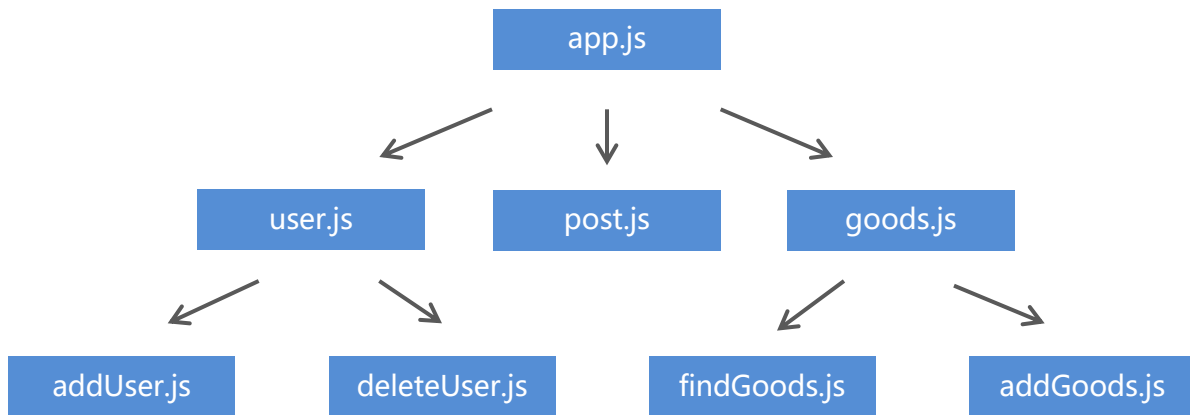
## 1.2 生活中的模块化开发



# 1. Node.js模块化开发

## 1.3 软件中的模块化开发

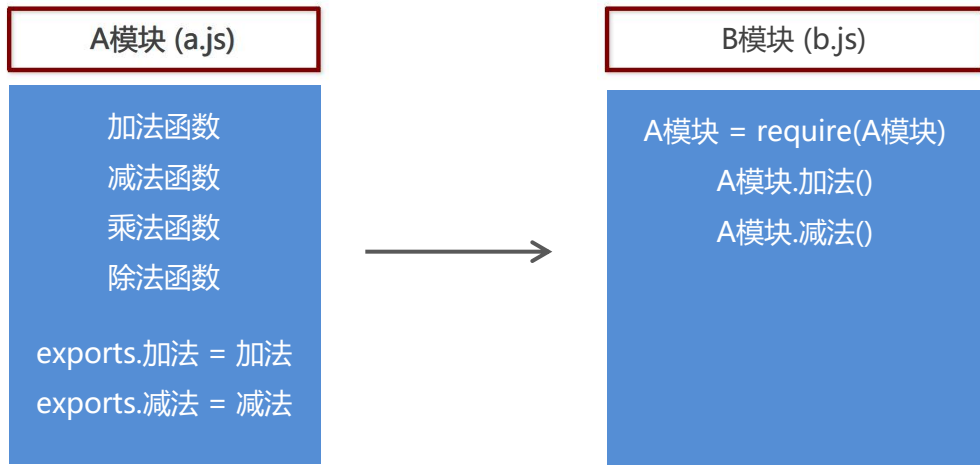
一个功能就是一个模块，多个模块可以组成完整应用，抽离一个模块不会影响其他功能的运行。



# 1. Node.js模块化开发

## 1.4 Node.js中模块化开发规范

- Node.js规定一个JavaScript文件就是一个模块，模块内部定义的变量和函数默认情况下在外部无法得到
- 模块内部可以使用exports对象进行成员导出，使用require方法导入其他模块。



# 1. Node.js模块化开发

## 1.5 模块成员导出

```
// a.js
// 在模块内部定义变量
let version = 1.0;
// 在模块内部定义方法
const sayHi = name => `您好, ${name}`;
// 向模块外部导出数据
exports.version = version;
exports.sayHi = sayHi;
```

# 1. Node.js模块化开发

## 1.6 模块成员的导入

```
// b.js  
// 在b.js模块中导入模块a  
let a = require('./b.js');  
// 输出b模块中的version变量  
console.log(a.version);  
// 调用b模块中的sayHi方法 并输出其返回值  
console.log(a.sayHi('黑马讲师'));
```

导入模块时后缀可以省略

# 1. Node.js模块化开发

## 1.7 模块成员导出的另一种方式

```
module.exports.version = version;  
module.exports.sayHi = sayHi;
```

**exports**是**module.exports**的别名(地址引用关系), 导出对象最终以**module.exports**为准



# 1. Node.js模块化开发

## 1.8 模块导出两种方式的联系与区别

exports

module.exports

```
exports.version = version;  
module.exports.version = version;
```

```
module.exports = {  
  name: 'zhangsan',  
}
```

{  
 version: 1.0,  
 sayHi: sayHi  
}

{  
 name: 'lisi' ,  
 age: 50  
}

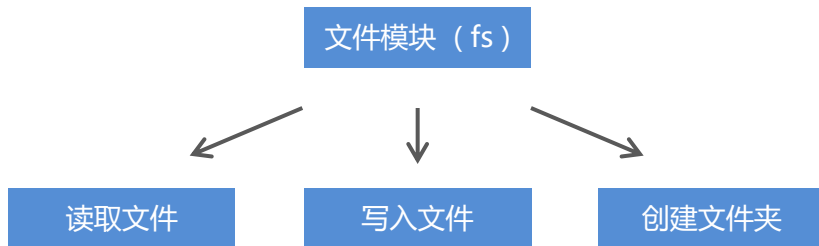
# 目录 Contents

- ◆ Node.js模块化开发
  - ◆ 系统模块
  - ◆ 第三方模块
  - ◆ package.json文件
  - ◆ Node.js中模块的加载机制

## 3. 系统模块

### 3.1 什么是系统模块

Node运行环境提供的API. 因为这些API都是以模块化的方式进行开发的, 所以我们又称Node运行环境提供的API为系统模块



## 3. 系统模块

### 3.2 系统模块fs 文件操作

f: file 文件, s: system 系统, 文件操作系统。

```
const fs = require('fs');
```

#### 读取文件内容

```
fs.readFile('文件路径/文件名称', ['文件编码'], callback);
```

## 3. 系统模块

### 3.2 系统模块fs 文件操作

#### 写入文件内容

```
fs.writeFile('文件路径/文件名称', '数据', callback);
```

```
const content = '<h3>正在使用fs.writeFile写入文件内容</h3>';  
fs.writeFile('../index.html', content, err => {  
  if (err !== null) {  
    console.log(err);  
    return;  
  }  
  console.log('文件写入成功');  
});
```

### 3.3 系统模块path 路径操作

#### 为什么要进行路径拼接

- 不同操作系统的路径分隔符不统一
- /public/uploads/avatar
- Windows 上是 \ /
- Linux 上是 /

### 3.4 路径拼接语法

```
path.join('路径', '路径', ...)
```

```
// 导入path模块
```

```
const path = require('path');
```

```
// 路径拼接
```

```
let finalPath = path.join('itcast', 'a', 'b', 'c.css');
```

```
// 输出结果 itcast\a\b\c.css
```

```
console.log(finalPath);
```

### 3.5 相对路径VS绝对路径

- 大多数情况下使用绝对路径，因为相对路径有时候相对的是命令行工具的当前工作目录
- 在读取文件或者设置文件路径时都会选择绝对路径
- 使用\_\_dirname获取当前文件所在的绝对路径



# 目录 Contents

- ◆ Node.js模块化开发
- ◆ 系统模块
- ◆ 第三方模块
- ◆ package.json文件
- ◆ Node.js中模块的加载机制

## ■ 4. 第三方模块

### 4.1 什么是第三方模块

别人写好的、具有特定功能的、我们能直接使用的模块即第三方模块，由于第三方模块通常都是由多个文件组成并且被放置在一个文件夹中，所以又名包。

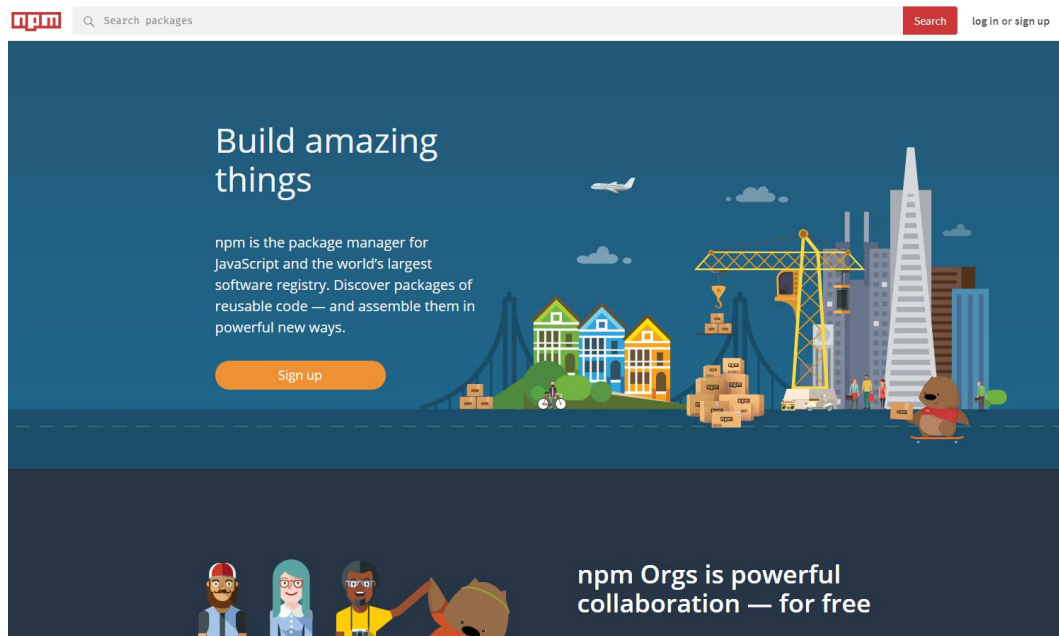
#### 第三方模块有两种存在形式：

- 以js文件的形式存在，提供实现项目具体功能的API接口。
- 以命令行工具形式存在，辅助项目开发

## 4. 第三方模块

### 4.2 获取第三方模块

npmjs.com: 第三方模块的存储和分发仓库



## 4. 第三方模块

### 4.2 获取第三方模块

npm (node package manager) : node的第三方模块管理工具

- 下载: npm install 模块名称
- 卸载: npm unintall package 模块名称

全局安装与本地安装

- 命令行工具: 全局安装
- 库文件: 本地安装



## 4. 第三方模块

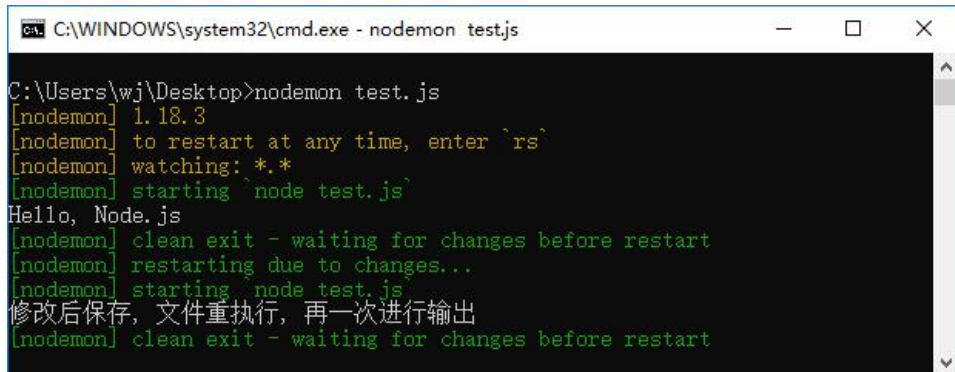
### 4.3 第三方模块 nodemon

nodemon是一个命令行工具，用以辅助项目开发。

在Node.js中，每次修改文件都要在命令行工具中重新执行该文件，非常繁琐。

#### 使用步骤

1. 使用npm install nodemon -g 下载它
2. 在命令行工具中用nodemon命令替代node命令执行文件



```
C:\WINDOWS\system32\cmd.exe - nodemon test.js

C:\Users\wj\Desktop>nodemon test.js
[nodemon] 1.18.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching: *.*
[nodemon] starting `node test.js`
Hello, Node.js
[nodemon] clean exit - waiting for changes before restart
[nodemon] restarting due to changes...
[nodemon] starting node test.js
修改后保存，文件重执行，再一次进行输出
[nodemon] clean exit - waiting for changes before restart
```

## 4. 第三方模块

### 4.4 第三方模块 nrm

nrm ( npm registry manager ): npm下载地址切换工具

npm默认的下载地址在国外, 国内下载速度慢

#### 使用步骤

1. 使用 `npm install nrm -g` 下载它
2. 查询可用下载地址列表 `nrm ls`
3. 切换npm下载地址 `nrm use` 下载地址名称

国外: npmjs.com



国内: npm.taobao.org



开发者



## 4. 第三方模块

### 4.5 第三方模块 Gulp

基于node平台开发的前端构建工具

将机械化操作编写成任务, 想要执行机械化操作时执行一个命令行命令任务就能自动执行了  
用机器代替手工, 提高开发效率。



## 4. 第三方模块

### 4.6 Gulp能做什么

- 项目上线，HTML、CSS、JS文件压缩合并
- 语法转换（es6、less ...）
- 公共文件抽离
- 修改文件浏览器自动刷新





## 4. 第三方模块

### 4.7 Gulp使用

1. 使用npm install gulp下载gulp库文件
2. 在项目根目录下建立gulpfile.js文件
3. 重构项目的文件夹结构 src目录放置源代码文件 dist目录放置构建后文件
4. 在gulpfile.js文件中编写任务.
5. 在命令行工具中执行gulp任务

# 1. Node.js中的模块化开发

## 4.8 Gulp中提供的方法

- gulp.src(): 获取任务要处理的文件
- gulp.dest(): 输出文件
- gulp.task(): 建立gulp任务
- gulp.watch(): 监控文件的变化

```
const gulp = require('gulp');  
// 使用gulp.task()方法建立任务  
gulp.task('first', () => {  
  // 获取要处理的文件  
  gulp.src('./src/css/base.css')  
  // 将处理后的文件输出到dist目录  
  .pipe(gulp.dest('./dist/css'));  
});
```



## 4. 第三方模块

### 4.9 Gulp插件

- gulp-htmlmin : html文件压缩
- gulp-cssso : 压缩css
- gulp-babel : JavaScript语法转化
- gulp-less: less语法转化
- gulp-uglify : 压缩混淆JavaScript
- gulp-file-include 公共文件包含
- browsersync 浏览器实时同步

# 目录 Contents

- ◆ Node.js模块化开发
- ◆ 系统模块
- ◆ 第三方模块
- ◆ package.json文件
- ◆ Node.js中模块的加载机制

## 6. package.json文件

### 6.1 node\_modules文件夹的问题

1. 文件夹以及文件过多过碎，当我们将项目整体拷贝给别人的时候，传输速度会很慢很慢。
2. 复杂的模块依赖关系需要被记录，确保模块的版本和当前保持一致，否则会导致当前项目运行报错

## 6. package.json文件

### 6.2 package.json文件的作用

项目描述文件，记录了当前项目信息，例如项目名称、版本、作者、github地址、当前项目依赖了哪些第三方模块等。

使用 **npm init -y** 命令生成。

## 6. package.json文件

### 6.3 项目依赖

- 在项目的开发阶段和线上运营阶段，都需要依赖的第三方包，称为项目依赖
- 使用npm install 包名命令下载的文件会默认被添加到 package.json 文件的 dependencies 字段中

```
{  
  "dependencies": {  
    "jquery": "^3.3.1"  
  }  
}
```

## 6. package.json文件

### 6.4 开发依赖

- 在项目的开发阶段需要依赖，线上运营阶段不需要依赖的第三方包，称为开发依赖
- 使用 `npm install 包名 --save-dev` 命令将包添加到 `package.json` 文件的 `devDependencies` 字段中

```
{  
  "devDependencies": {  
    "gulp": "^3.9.1"  
  }  
}
```



## ■ 6. package.json文件

### 6.5 package-lock.json文件的作用

- 锁定包的版本，确保再次下载时不会因为包版本不同而产生问题
- 加快下载速度，因为该文件中已经记录了项目所依赖第三方包的树状结构和包的下载地址，重新安装时只需下载即可，不需要做额外的工作

# 目录 Contents

- ◆ Node.js模块化开发
- ◆ 系统模块
- ◆ 第三方模块
- ◆ package.json文件
- ◆ Node.js中模块的加载机制

## 5. Node.js中模块加载机制

### 5.1 模块查找规则-当模块拥有路径但没有后缀时

```
require('./find.js');
```

```
require('./find');
```

1. require方法根据模块路径查找模块，如果是完整路径，直接引入模块。
2. 如果模块后缀省略，先找同名JS文件再找同名JS文件夹
3. 如果找到了同名文件夹，找文件夹中的index.js
4. 如果文件夹中没有index.js就会去当前文件夹中的package.json文件中查找main选项中的入口文件
5. 如果找指定的入口文件不存在或者没有指定入口文件就会报错，模块没有被找到

## ■ 5. Node.js中模块加载机制

### 5.2 模块查找规则-当模块没有路径且没有后缀时

```
require('find');
```

1. Node.js会假设它是系统模块
2. Node.js会去node\_modules文件夹中
3. 首先看是否有该名字的JS文件
4. 再看是否有该名字的文件夹
5. 如果是文件夹看里面是否有index.js
6. 如果没有index.js查看该文件夹中的package.json中的main选项确定模块入口文件
7. 否则找不到报错