第3章 模块与包的管理与使用

Node.js开发实战教程

内容导航

Contents

Section 00

【学习目标】

（1）了解Node.js的模块系统；
（2）学会定义和导入模块。

Section 01

【为什么要使用模块】

- Node.js将可重用代码封装在各种模块中，减少应用程序的代码量，提高开发效率和代码的可读性。
- 模块打包代码的方式不会改变全局作用域，开发人员可以在被载入的模块文件中选择要对外暴露的方法和变量。
- 模块可以发布到npm仓库中与他人共享。
- 开发人员可以使用来自npm仓库的第三方模块，而不必担心某个模块会覆盖其他模块的变量和方法。

Section 02

【自定义模块】

- Node.js主要用于服务器端编程，无须考虑非同步加载的方式，选择CommonJS规范。
- 按照CommonJS规范要求，模块必须通过module.exports对象导出对外暴露的变量或接口，通过require()方法将其他模块的输出加载到当前模块作用域中。
- 在Node.js模块系统中，独立的模块有自己的作用域，其变量、方法等都是对其他文件不可见的。
- 典型的模块可以是一个包含exports对象属性定义的文件，可将exports看作是module.exports的简单引用形式。通过在exports对象上指定额外的属性，可以将方法和对象等添加到模块的根部。例如，创建一个名为hello.js的文件，添加代码来定义模块。

```
var show_day=new Array('星期一','星期二','星期三','星期四','星期五','星期六','星期日');
var nowTime = new Date();
var day=nowTime.getDay();
exports.sayHello = function(name) {
    console.log('你好！ ' + name);
    console.log('今天是' + show_day[day-1]);
};
```

Section 03

【导入模块】

- 使用require导入模块是同步I/O操作，尽量不要在I/O操作密集的地方使用require()方法。
- 通常只在程序最初加载时使用require，模块一般在文件顶部导入。

```
var hello = require('./hello');
hello.sayHello('小王');
```

Section 04

【使用module.exports定义模块】

- 如果希望模块成为某个类的实例，则应将要导出的对象赋值给module.exports属性。
- 使用module.exports对象可以对外提供单个变量、方法或者对象。
- 将模块定义为类的实例的示例

```
var show_day=new Array('星期一','星期二','星期三','星期四','星期五','星期六','星期日');
var nowTime = new Date();
var day=nowTime.getDay();
function Hello(name) {
    var name;
    this.setName = function(yourName){
        name = yourName;}
    this.sayHello = function() {
        console.log('你好！ ' + this.name);
        console.log('今天是' + show_day[day-1]);
    };
};
module.exports = Hello;
```

Section 04

【使用module.exports定义模块】

- 导入类实例模块的示例

```
var Hello = require('./hello1');
hello = new Hello();  //此处需要实例化类
hello.setName('小张');
hello.sayHello();
```

Section 05

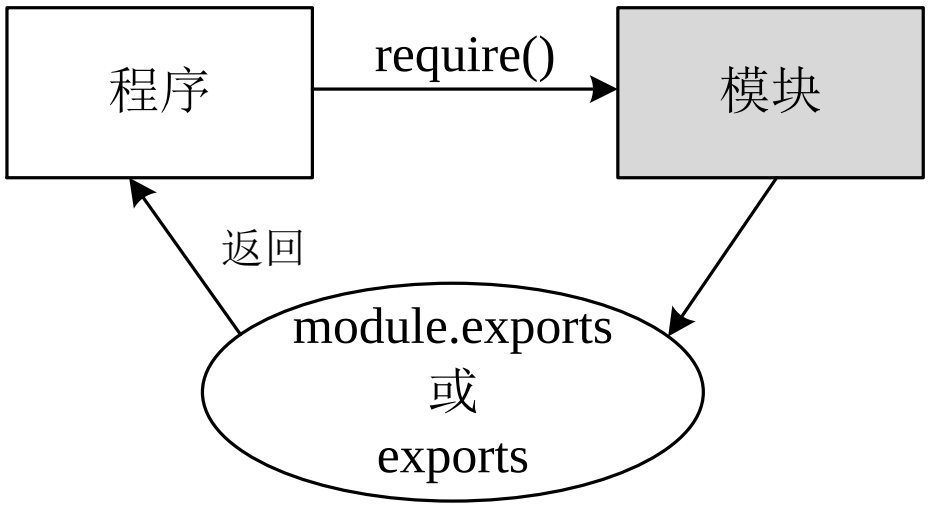
【exports与module.exports的关系】

- exports在模块的文件级别作用域内有效，它在模块被执行前被赋予module.exports的值。
- exports只是对module.exports的一个全局引用。
- 最终返回给调用程序的是module.exports而不是exports。

```
module.exports.hello = true; // 会被导出
exports = { hello: false };  // 不会被导出，仅在模块内有效
```

具体使用时一定要注意exports与module.exports之间的区别。使用前者导出的方法返回的是模块函数，可以直接调用；而由后者导出的方法返回的是一个类，需实例化为对象之后才可以调用。

模块工作机制



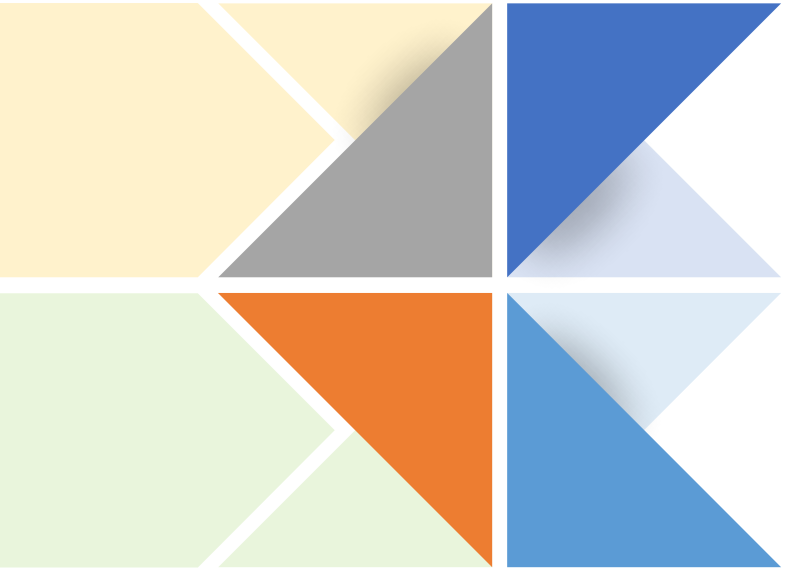
Section 06

【模块加载顺序】

- 从缓存中加载
- 加载核心模块
- 加载文件模块
- 文件夹作为模块
  - （1）加载package.json文件指定的文件
  - （2）加载index.js文件
  - （3）加载index.node文件
- 从node_modules目录加载
- 从全局目录加载
- 循环加载

内容导航

Contents



Section 00 【学习目标】

（1）了解Node.js的核心模块；
（2）掌握os、path、url等模块的用法。

Section 01 【os模块——提供基本的系统操作方法】

- Node.js的os模块提供一些操作系统相关的实用方法，导入该模块：

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

- 属性
- 方法
- 常量

Section 01

【os模块——提供基本的系统操作方法】

▶▶▶ 示例

程序代码	执行结果

const os = require('os');
console.log('操作系统类型：' + os.type());
console.log('操作系统平台：' + os.platform());
console.log('系统内存总量：' + os.totalmem() + " 字节");
console.log('空闲内存量：' + os.freemem() + " 字节");
console.log('CPU信息：');
console.log(os.cpus());
	C:\nodeapp\ch03\modulecom>node os_test.js
	操作系统类型：Windows_NT
	操作系统平台：win32
	系统内存总量：3220688896 字节
	空闲内存量：1413165056 字节
	CPU信息：
	[{ model: 'Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz',
	speed: 3500,
	times: { user: 136390, nice: 0, sys: 183796, idle: 1154859, irq: 6515 } },
	{ model: 'Intel(R) Core(TM) i7-4770K CPU @ 3.50GHz',
	speed: 3500,
	times: { user: 157203, nice: 0, sys: 134765, idle: 1182687, irq: 1531 } }]

Section 02

【util模块——提供实用工具】

- util模块主要用于支持Node.js内部API的需求。
- 提供的大部分实用工具可用于应用程序与模块开发。导入该模块：

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

util.inspect(object[, options])

- 返回对象的字符串表示，主要用于调试。
- 以下代码用于查看util对象的所有属性：

```
const util = require('util');
console.log(util.inspect(util, { showHidden: true, depth: null }));
```

util.format(format[, ...args])

- 返回一个格式化后的字符串。

```
util.format('%s:%s', '蓝天白云'); // 返回: '蓝天白云:%s ‘
util.format('%s:%s', '环境优美', '绿水青山', '蓝天白云'); // 返回'环境优美:绿水青山 蓝天白云 ‘
util.format(1, 2, 3); // 返回'1 2 3'
```

Section 03

【path模块——处理和转换文件路径】

- 提供了一些工具方法，用于处理文件与目录的路径。导入该模块：

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

不同风格的路径

- path模块最大的用处是解决多平台目录路径问题。
- path模块默认会根据Node.js应用程序运行的操作系统的不同而变化。
- 要想在任何操作系统上处理Windows文件路径时获得一致的结果，可以使用path模块的path.win32属性。
- Node.js在Windows系统上遵循单驱动器工作目录的理念。

属性

- path.delimiter：提供平台特定的路径分隔符，Windows上是“;”，POSIX上是“:”。
- path.sep：提供平台特定的路径分段分隔符。Windows上是“\”，POSIX上是“/”。

Section 03

【path模块——处理和转换文件路径】

▸▸▸ 方法

- path.normalize(path)：对路径进行规范化，并解析“..”和“.” 。
- path.dirname(path)：返回路径的目录名，类似于UNIX中的dirname命令。
- path.basename(path[, ext])：返回路径中的最后一部分，可选的ext参数表示文件扩展名。
- path.extname(path)：返回路径中文件的后缀名，即路径中最后一个“.”之后的部分。
- path.parse(path)：返回完整路径的一个对象。
- path.format(pathObject)：从一个对象表示的路径返回一个字符串表示的路径。
- path.resolve([...paths])：将一个路径或路径片段的序列解析为一个绝对路径。
- path.relative(from, to)：返回从参数from到to的相对路径（基于当前工作目录）。
- path.join([...paths])：使用平台特定的分隔符将路径片段序列连接到一起，并规范生成的路径。
- path.isAbsolute(path)：判定路径是否为一个绝对路径。

Section 03

【path模块——处理和转换文件路径】

▶▶▶ 示例

程序代码	执行结果

const path = require("path");
console.log('格式化路径: ');
console.log(path.normalize('/test/test1//2slashes///1slash/tab/..'));
console.log('连接路径: ');
console.log(path.join('/test', 'test1', '2slashes/1slash', 'tab', '..'));
console.log('获取绝对路径 : ' + path.resolve('index.js'));
console.log('获取扩展名 : ' + path.extname('index.js'));
	C:\nodeapp\ch03\modulecom>node path_test
	格式化路径: \test\test1\2slashes\1slash
	连接路径: \test\test1\2slashes\1slash
	获取绝对路径 :
	C:\nodeapp\ch03\modulecom\index.js
	获取扩展名 : .js

Section 04

【url模块——URL处理与解析】

- url模块提供了一些实用方法，用于URL处理与解析。。导入该模块：

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

▸▸▸ url模块的两套API

- Node.js特有的API（传统的URL API）主要用于兼容已有应用程序。

```
const url = require('url');
const myURL = url.parse('https://user:pass@sub.host.com:8080/p/a/t/h?query=string#hash')
```

- 新的应用程序应使用WHATWG API。

```
const { URL } = require('url');
const myURL = new URL('https://user:pass@sub.host.com:8080/p/a/t/h?query=string#hash');
```

Section 04

【url模块——URL处理与解析】

▸▸▸ WHATWG API的URL类提供的方法和属性

- URL类根据WHATWG URL标准实现。首先使用构造方法new URL(input[, base])创建URL对象，然后使用URL类提供的属性和方法来进行操作。
- 主要属性和方法列举如下。
  - url.protocol：获取及设置URL的协议（protocol）部分。
  - url.host：获取及设置URL的主机（host）部分，包括端口。
  - url.hostname：获取及设置URL的主机名（hostname）部分。
  - url.port：获取及设置URL的端口（port）部分。
  - url.pathname：获取及设置URL的路径（path）部分。
  - url.search：获取及设置URL的序列化查询（query）部分。
  - url.hash#：获取及设置URL的hash部分。例如http://example.org/foo#bar。
  - url.href：获取及设置序列化的URL，返回值与url.toString和url.toJSON的返回值相同。
  - url.toString()：返回序列化的URL。
  - url.toJSON()：返回序列化的URL，URL对象使用JSON.stringify()序列化时将自动调用该方法。

Section 04

【url模块——URL处理与解析】

WHATWG API的URLSearchParams类提供的方法

- 提供对URL查询字符串部分进行处理的方法。

```
const { URL, URLSearchParams } = require('url');
const myURL = new URL('https://example.org/?abc=123');
console.log(myURL.searchParams.get('abc'));
// 输出 123
```

传统的URL API

- 使用url.parse(urlString[, parseQueryString[, slashesDenoteHost]]) 解析URL字符串并创建一个URL对象。
- 使用传统的URL API提供的方法来进行操作。

Section 05

【querystring模块——URL查询字符串处理和解析】

- 提供一些实用方法用于解析与格式化URL查询字符串。导入该模块：

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

- querystring.parse(str[, sep[, eq[, options]]]) 将一个URL查询字符串解析成一个键值对的集合（相当于反序列化）。示例：解析查询字符串foo=bar&abc=xyz&abc=123的结果如下：

  ```
  {
    foo: 'bar',
    abc: ['xyz', '123']
  }
  ```

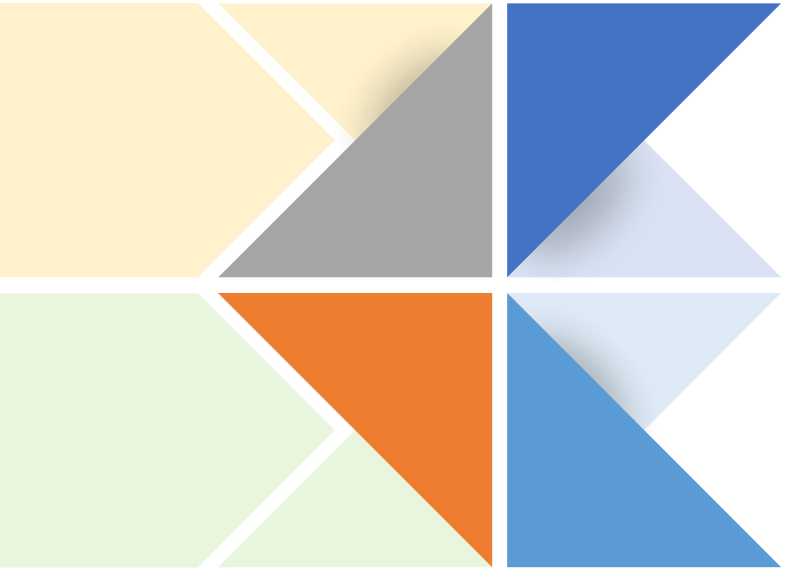
- querystring.stringify(obj[, sep[, eq[, options]]]) 将一个对象转换成URL查询字符串，是querystring.parse的逆运算。参数obj是要序列化成URL查询字符串的对象。示例：

  调用：querystring.stringify({ foo: 'bar', baz: ['qux', 'quux'], corge: '' });
  返回：“foo=bar&baz=qux&baz=quux&corge=”

- querystring.unescape(str)用于对字符串进行解码，通常提供给querystring.parse()使用。
- querystring.escape(str) 用于对字符串进行URL编码，主要提供给querystring.stringify() 使用。

内容导航

Contents



Section 00

【学习目标】

（1）认识npm、包与模块；

（2）掌握Node.js包的管理和使用。

Section 01

【什么是npm】

- npm——Node Package Manager
- npm组成部分
  - Web网站：用来查找包、设置配置文件以及管理npm应用的其他方面，例如，可以为自己的公司设置公共或私有包的访问管理。
  - 命令行接口：包管理器，大多数开发人员会通过它来使用包。
  - 注册中心：提供JavaScript软件及其元数据信息的大型公共数据库，也就是官方仓库。

Section 02

【理解包与模块】

- 包是由package.json文件描述的一个文件或目录。
- 模块是由Node.js的require()方法加载的任何文件或目录。

包的形式	模块的形式
(1) 一个文件夹，它包含由package.json文件描述的一个程序。 (2) 一个包含（1）的gzip压缩包。 (3) 一个可以解析到（2）的URL地址。 (4) 一个发布到官方仓库的带有（3）的<名称>@<版本>形式。 (5) 一个指向（4）的<名称>@<标签>形式。 (6) 一个符合（5）的<名称>@<标签>形式。 (7) 一个指向（1）的Git网址。	一个包含package.json文件（定义有main字段）的文件夹。 一个包含index.js文件的文件夹。 一个JavaScript文件。

Section 03

【npm包管理器】

- npm是整个Node.js社区最流行、支持第三方模块最多的包管理器。
- npm用来安装、共享和发布代码以及管理项目中的依赖。
- Node.js本身集成了npm，npm工具本身可以通过npm命令来升级版本，命令如下：

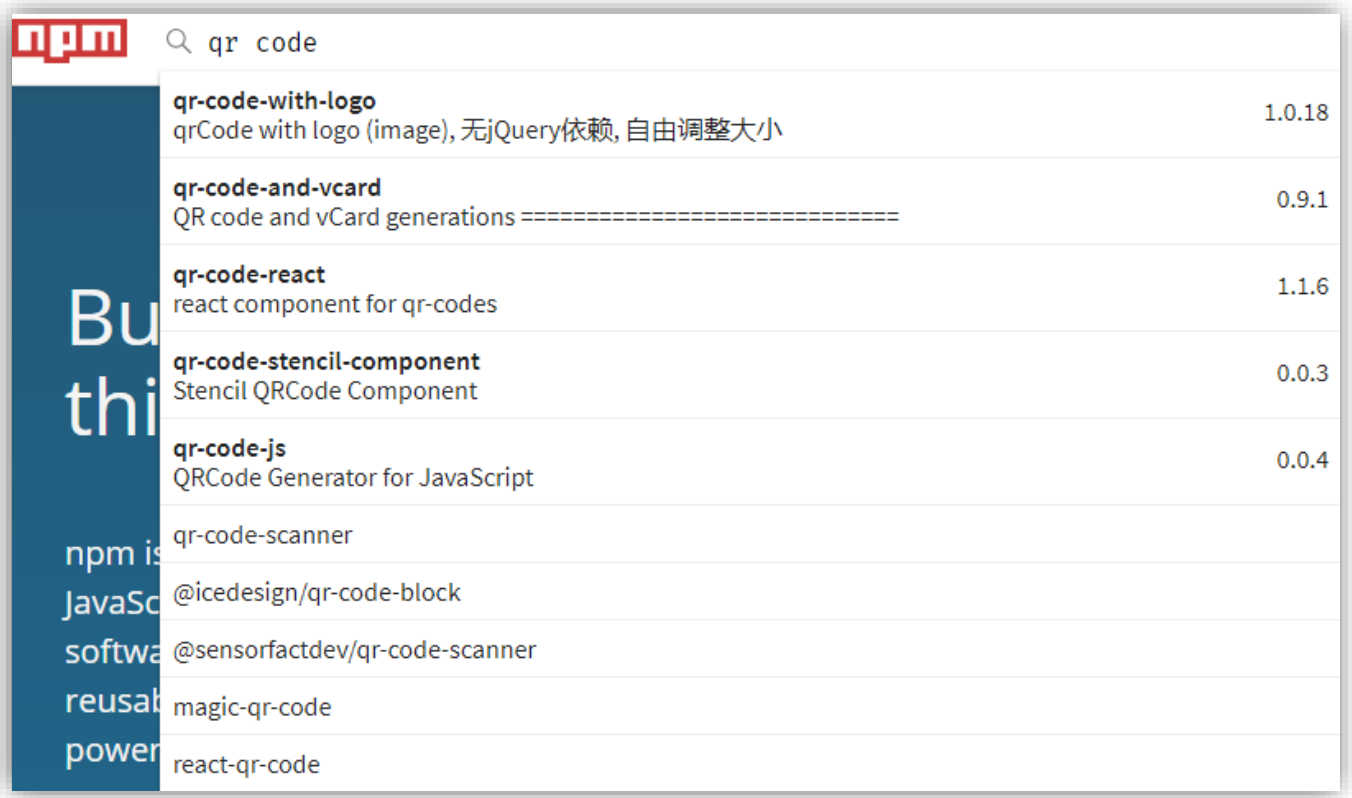
```
npm install npm -g
```

Section 04

【查找和选择包】

▶▶▶ 通过浏览器查找和选择包

- 查找包



Section 04

【查找和选择包】

▶▶▶ 通过浏览器查找和选择包

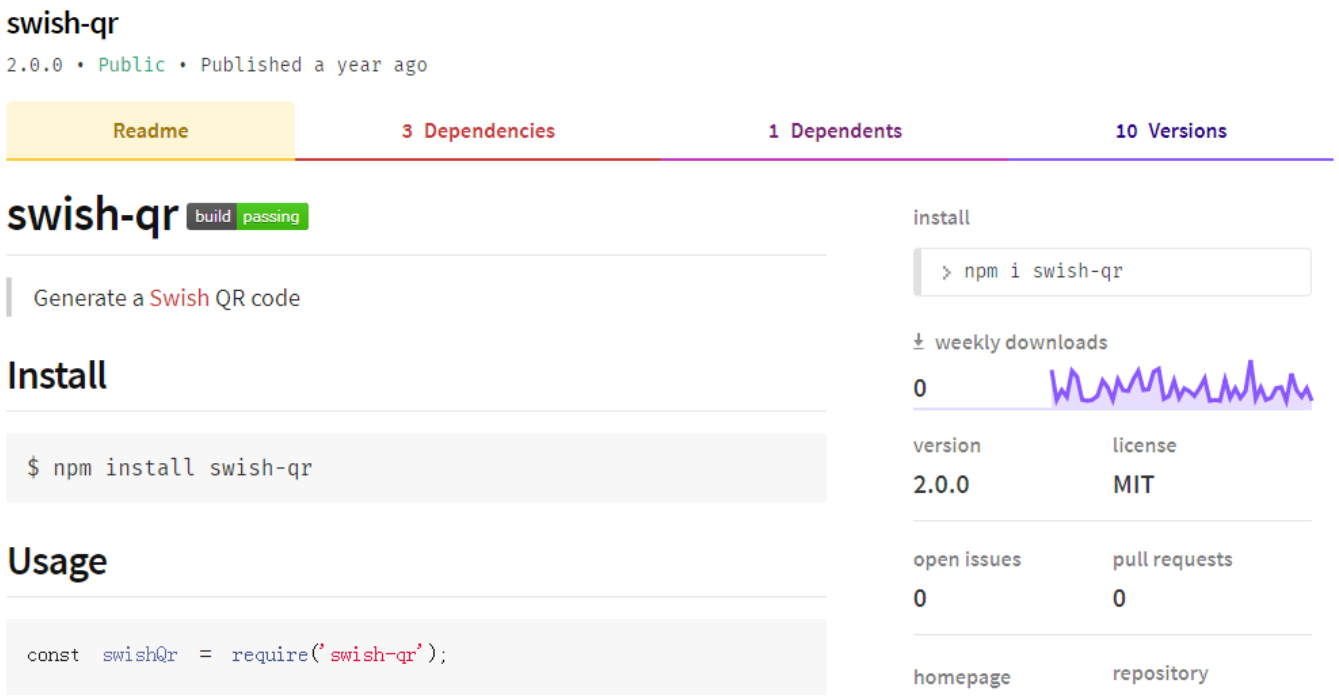
- 列出查询结果

Section 04

【查找和选择包】

▶▶▶ 通过浏览器查找和选择包

- 从若干类似的包中进行选择最合适的
- 查看包的详细信息



Section 04

【查找和选择包】

▶▶▶ 通过浏览器查找和选择包

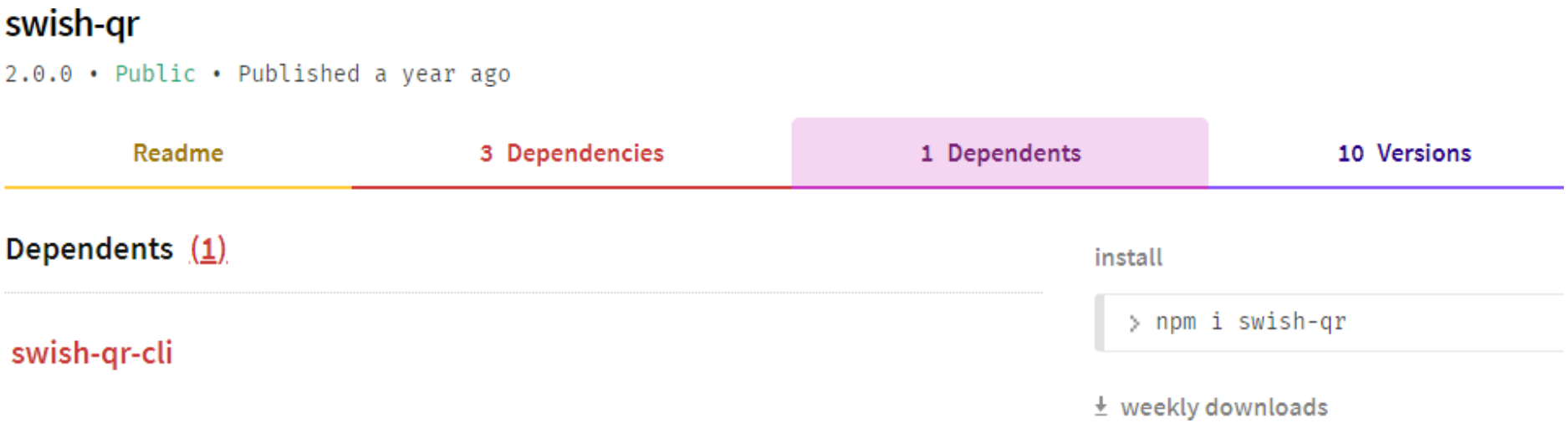
- 查看包的详细信息
  - 显示依赖包列表

Section 04

【查找和选择包】

▸▸▸ 通过浏览器查找和选择包

- 查看包的详细信息
  - 显示被依赖的包列表

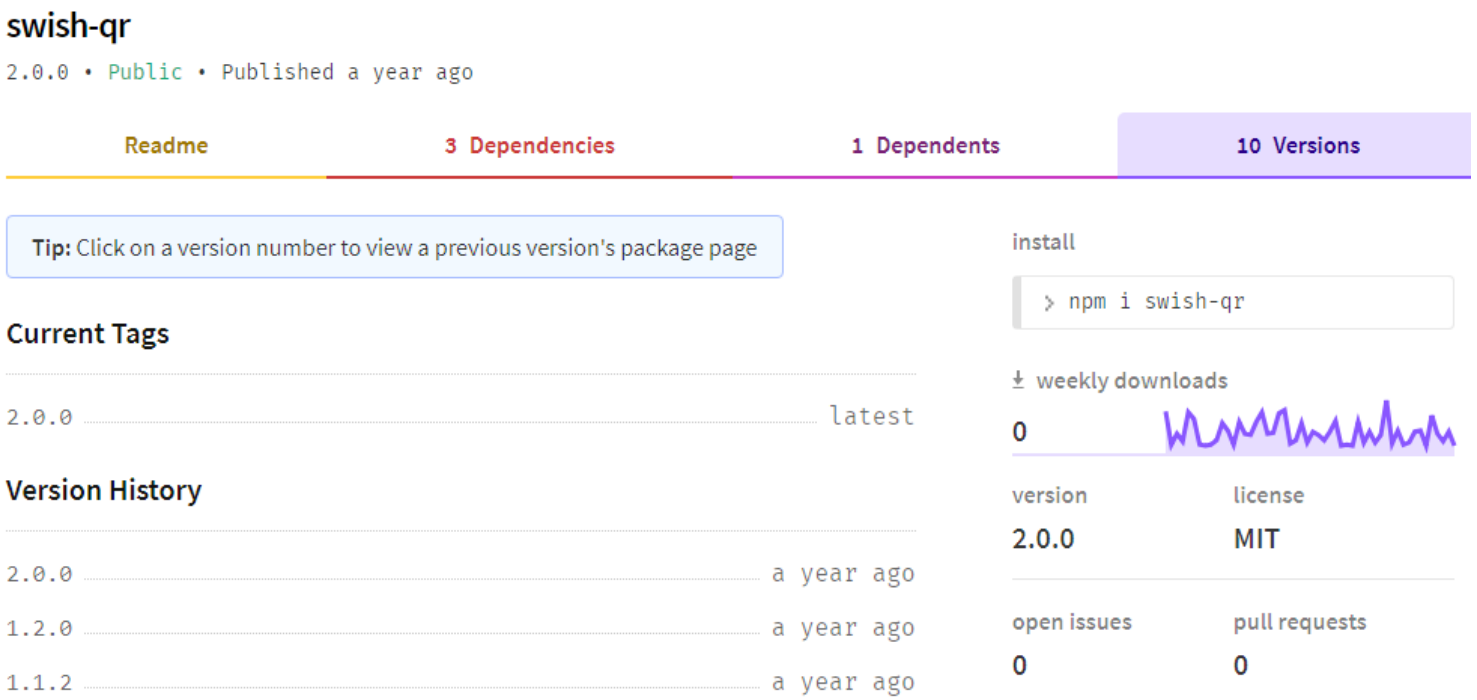


Section 04 【查找和选择包】

▶▶▶ 通过浏览器查找和选择包

➢ 查看包的详细信息

- 显示包的版本列表



Section 04 【查找和选择包】

▸▸▸ 使用npm命令行工具查找包

- 使用npm search命令搜索包

```
C:\nodeapp\ch03\pkgtest>npm search QR code
NAME                        | DESCRIPTION          | AUTHOR          | DATE       | VERSION  | KEYWORDS
qr-image                    | QR Code generator... | =alexeyten      | 2016-12-22 | 3.2.0    | qrcode qr code qr png svg i
@icedesign/qr-code-block    | 根据配置属性生成二... | =temper357...   | 2018-08-02 | 1.0.0    | ice react block
react-qr-svg                | React.js component... | =no23reason     | 2017-10-05 | 2.1.0    | react reactjs qr-code qr sv
swish-qr                    | Generate a Swish QR... | =gillstrom...  | 2017-06-30 | 2.0.0    | base64 generate swish qr
```

- 使用npm view命令进一步查看包信息

```
C:\nodeapp\ch03\pkgtest>npm view qr-image
qr-image@3.2.0 | MIT | deps: none | versions: 11
QR Code generator (png, svg, pdf, eps)
```

Section 04

【查找和选择包】

▶▶▶ 常用的包列表

- Web开发
- 基础拓展
- 代码组织
- 测试
- 项目管理
- 预编译

Section 05

【使用npm命令安装包】

▸▸▸ 安装本地包

npm install <包名>

▸▸▸ 确定安装包的版本

npm install lodash@4.2.0

▸▸▸ 安装全局包

npm install –g <包名>

▸▸▸ 查看安装信息

- 执行命令npm list可查看当前目录下已安装的包；
- 执行命令npm list –g可查看所有全局安装的包。

Section 06

【使用语义版本控制管理代码（包）】

▸▸▸ 概述

- 版本号的形式为X.Y.Z，X、Y和Z分别代表主版本号、次版本（副版本）号和补丁版本号。
- X、Y和Z必须为非负整数，禁止在数字前补零，每个数值都是递增的。

▸▸▸ 代码（包）发布的语义版本控制

代码状态	阶段	规则	示例
首次发布	新产品	从1.0.0开始	1.0.0
向后兼容的bug修复	补丁发布	更新Z位	1.0.1
向后兼容的新功能	次版本发布	更新Y位并将Z位置0	1.10
变动较大，向后不兼容	主版本发布	更新X位并将Y和Z位置0	2.0.0

Section 06

【使用语义版本控制管理代码（包）】

▸▸▸ 代码（包）使用的语义版本控制

- 在package.json文件中指定应用程序可接受的更新版本范围。
- 假使要以一个包的1.0.4版本作为起始版本，可以按照以下方式指定可接受的版本范围。
  - 补丁发布：1.0、1.0.x或～1.0.4。
  - 次版本发布：1、1.x或 ^1.0.4。
  - 主版本发布：*或x。

Section 07

【使用package.json文件管理本地安装包】

package.json文件的功能

- 列出当前项目所依赖的包。
- 通过语义化版本控制规则指定当前项目所使用的包的版本。
- 实现可重用的构建，更易于与其他开发人员分享包。

package.json文件的基本组成

- name：项目名称。
- version：采用x.x.x的形式，符合语义化版本控制规则。

Section 07

【使用package.json文件管理本地安装包】

▸▸▸ 创建package.json文件

- 创建默认的package.json

```
C:\nodeapp\ch03\pkgtest >npm init --yes
Wrote to C:\nodeapp\ch03\pkgtest\package.json:
{
  "name": "mynodeprj",                    // 项目版本（必需）
  "version": "1.0.0",                     // 项目版本（必需）
  "description": "",                      // 项目描述信息
  "main": "index.js",                     // 程序的主入口文件
  "dependencies": {                       // 项目的依赖包
    "lodash": "^4.17.11"
  },
  "devDependencies": {},     // 项目开发阶段的依赖包
  "scripts": {                            // 项目执行的脚本
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],                         // 项目的关键词，用于搜索
  "author": "",                           // 项目作者
  "license": "ISC"                        // 项目许可协议
}
```

Section 07

【使用package.json文件管理本地安装包】

▸▸▸ 创建package.json文件

- 命令行交互方式运行npm init命令

```
C:\nodeapp\ch03\pkgtest>npm init
This utility will walk you through creating a package.json file.
It only covers the most common items, and tries to guess sensible defaults.
See `npm help json` for definitive documentation on these fields
and exactly what they do.
Use `npm install <pkg>` afterwards to install a package and
save it as a dependency in the package.json file.
Press ^C at any time to quit.
package name: (mynodeprj)     #首先要求提供项目名称
```

- 可以使用npm set命令为npm init命令设置字段默认值

```
> npm set init.author.email "wombat@npmjs.com"  // 设置作者的邮箱
> npm set init.author.name "ag_dubs"                          // 设置作者的姓名
> npm set init.license "MIT"                                        // 设置许可
```

Section 07

【使用package.json文件管理本地安装包】

▸▸▸ 指定依赖包

- 向package.json文件中添加“dependencies”依赖（应用程序在生产环境中所需的）：

  npm install <包名> [--save-prod]

- 向package.json文件中添加“devDependencies”依赖(开发和测试所需的)：

  npm install <包名> --save-dev

▸▸▸ 手动编辑package.json

```
{
  "name": "my_package",
  "version": "1.0.0",
  "dependencies": {
    "my_dep": "^1.0.0"
  },
  "devDependencies" : {
    "my_test_framework": "^3.1.0"
  }
}
```

Section 08

【包的其他操作】

▸▸▸ 包的升级

npm update <包名>
npm update -g <包名>
npm update -g

▸▸▸ 包的卸载

npm uninstall <包名>
npm uninstall --save-dev <包名>
npm uninstall -g <包名>

▸▸▸ 清空npm本地缓存

npm cache clean

▸▸▸ 发布自己的包

- 需要在npm注册服务器上注册用户。
- 准备好包代码之后，使用npm publish命令进行发布。

Section 09

【使用淘宝npm镜像】

- 淘宝团队提供了完整的npmjs.org镜像，其版本同步频率为10分钟一次。
- 淘宝专门定制了cnpm命令行工具以代替npm，可以执行以下命令进行cnpm工具安装。

```
npm install -g cnpm --registry=https://registry.npm.taobao.org
```

- 安装完成后就可以使用cnpm来代替npm安装和管理npm包。
- cnpm的使用方法与npm相同，只需将npm改成cnpm。

内容导航

Contents

Section 00 【学习目标】

（1）综合运用模块与包的知识；
（2）学会使用第三方模块编写程序。
（3）构建一个简单的项目，利用第三方模块来抓取网页中的图片。

Section 01 【技术准备】

- 主要解决两个问题
  - 发起HTTP请求——使用request包
  - 解析获取的网页内容——使用cheerio包

Section 01 【技术准备】

request包及其模块

- 使用request模块的示例代码：

```
var request = require('request');          //引导入request模块
request('https://www.baidu.com', function (error, response, body) {
  console.log('error:', error);                  // 如果出错输出错误信息
  console.log('statusCode:', response && response.statusCode); // 收到响应输出状态码
  console.log('body:', body);                              // 输出百度首页的HTML内容
});
```

- request的一个重要特色就是对流的支持。
- request提供非常好用的错误处理机制。

Section 01 【技术准备】

▸▸▸ cheerio包及其模块

- cheerio主要用来提取所爬取的网页节点内容。
- 使用cheerio模块的例子：

```
const cheerio = require('cheerio')
const $ = cheerio.load('<h2 class="title">Hello world</h2>')
$('h2.title').text('Hello there!')
$('h2').addClass('welcome')
$.html()
//这将输出：<html><head></head><body><h2 class="title welcome">Hello there!</h2>
</body></html>
```

Section 02

【实现思路】

（1）通过request请求网页地址并获取返回结果，得到整个网页代码。

（2）在request的回调函数中调用cheerio对返回的网页文档内容基于DOM结构进行解析，提取其中图片文件的URL地址。

（3）利用request模块再次请求图片文件的URL地址，将获取的结果以流的方式保存到本地。

（4）为更好地组织代码，将使用cheerio模块解析HTML文档的功能封装为一个文件模块，在项目的主入口文件index.js中加载该文件模块。

Section 03

【创建项目目录并准备package.json文件】

```
C:\nodeapp\ch03\crawlimg>npm init
 (此处省略交互过程)
About to write to C:\crawlimg\package.json:
{
  "name": "crawlimg-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
Is this OK? (yes)
C:\crawlimg>
```

Section 04

【安装request和cheerio包】

- 安装request和cheerio包（利用npm淘宝镜像）：

```
C:\nodeapp\ch03\crawlimg>cnpm install request cheerio  --save
```

- 选项--save将把安装包的信息添加到pakage.json文件的“dependencies”中：

```
"dependencies": {
   "cheerio": "^1.0.0-rc.2",
   "request": "^2.88.0"
 }
```

Section 05

【定义一个模块用于解析网页文档】

```
const cheerio = require('cheerio');    //载入第三方cheerio模块
exports.getImg =   function(htmldom, callback) {   // exports对象上设定对外暴露的方法
    let $ = cheerio.load(htmldom);     //装载网页
    $('img').each(function(i, elem) {    //遍历其中的img标签
        let imgSrc = $(this).attr('src');    //获取每一个图片的URL
        callback(imgSrc, i);    //通过回调函数处理图片URL
    });
}
```

Section 06

【编写主入口文件index.js】

```
const request = require('request');  //载导入第三方request模块
const path = require('path');   //载导入path核心模块用于处理目录路径
const fs  = require('fs');   //载导入fs核心模块用于操作文件
const parse = require('./parse');  //载导入request文件模块用于解析网页图片地址
const workUrl = 'http://www.qingdoanews.com';  //指定要抓取图片的网页地址，可以改变
const imgDir = path.join(__dirname, 'downimages'); //获取图片保存路径
//定义下载图片的函数
function downImg(imgUrl, i) {
    console.log(imgUrl);    //控制台输出图片URL
    //获取图片文件的扩展名，利用split()函数将图片URL分割成一个字符串数组并取最后一个元素
    let ext = imgUrl.split('.').pop();
    //利用request请求网页图片并将图片保存到指定的图片目录，使用一个从读取流到写出流的管道（pipe）
    request.get(imgUrl).pipe(fs.createWriteStream(path.join(imgDir, i + '.' + ext),
    { 'encoding': 'utf8' }));
}
```

Section 06

【编写主入口文件index.js】

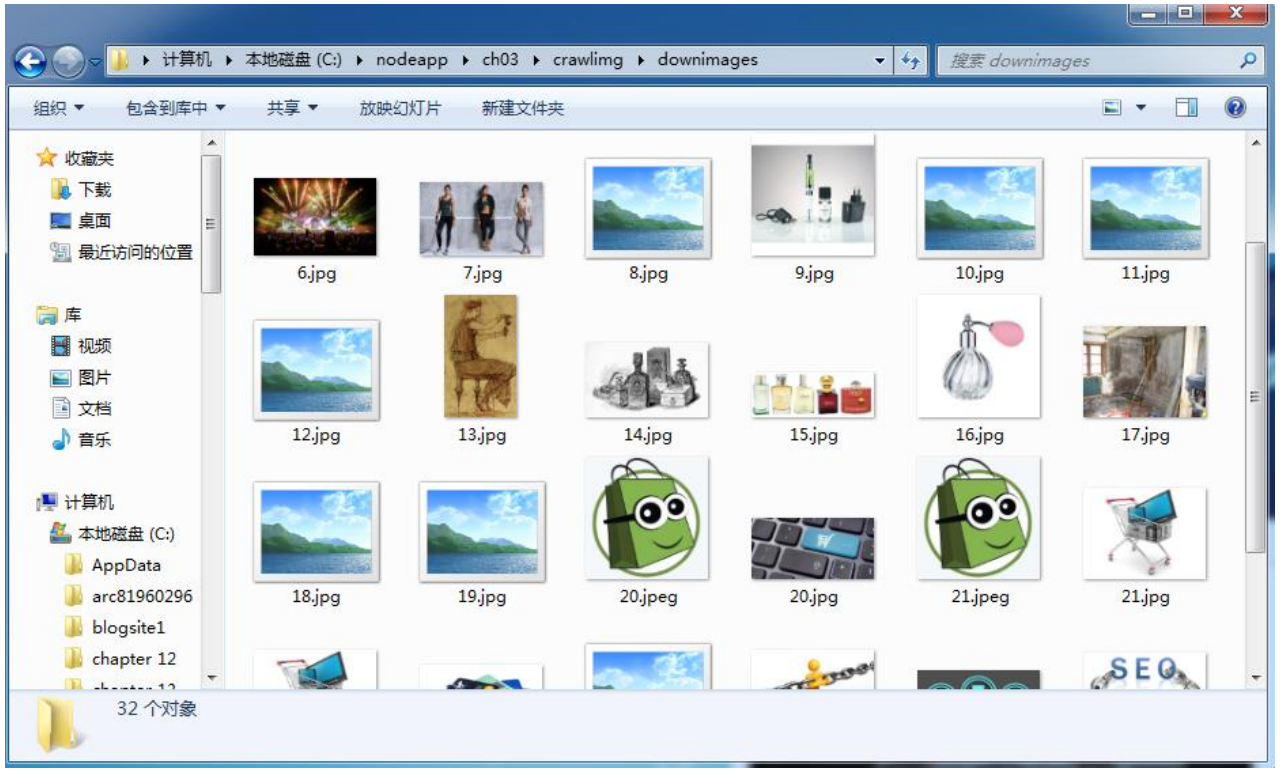
```
//下面是主程序，用于请求网页并对结果进行解析
request(workUrl, function(err, res, body) {
    if(!err && res) {
        console.log('start');
        parse.getImg(body,downImg);  //调用parse.getImg解析网页图片
        console.log("done");
    }
    else{
        console.log("error");
    }
});
```

Section 08

【运行程序】

```
C:\nodeapp\ch03\crawlimg>node index.js
start
https://qingdoanews.com/wp-content/uploads/2019/02/cropped-Qingdao.jpg
https://qingdoanews.com/wp-content/uploads/2019/02/cropped-Sverige-Kina.jpg
（部分省略）
https://qingdoanews.com/wp-content/uploads/2017/12/seo-blog-300x172.png
https://qingdoanews.com/wp-content/uploads/2017/12/SEO-Link.jpg
done
```

查看已下载的图片



本章小结

Summary

本章的主要内容是Node.js模块和包的相关知识和操作技能，包括模块的自定义和导入、模块的组织管理、模块的加载顺序、核心模块的使用、包的管理和使用，最后是一个使用第三方模块开发的案例。Node.js的内置模块远远不能满足开发需要，程序员必须熟练掌握第三方模块的使用。开发人员通过Node.js提供的npm包管理工具下载第三方提供的各种包，充分利用这些可重用代码，可以高效率地开发出功能强大的Node.js应用程序，这也正是Node.js开源系统的魅力之所在。

Thank you