杰普软件科技有限公司

www.briup.com

Tel: (021)55660810
Fax: (021)55660802
Email:training@briup.com
Msn: training.sh@hotmail.com
Home:http://www.briup.com

地址：上海市闸北区万荣路 1188 **弄 F栋 6**号三层 **-** 上海服务外包科技园龙软园区
邮编： 200436

电话： 021-56657112
传真： 021-55661523-8003
电邮： training@briup.com
主页：
http://www.briup.com

Briup High-End IT Training

Node

第 1 章：入门

Brighten Your Way And Raise You Up.

- 下载

Node 也叫 **NodeJS**， **Node.js**，由 **Ryan-Dahl** 于 **2009** 年 **5** 月在 **GitHub** 发布了第一版。 **Node** 是一个 **JavaScript** 运行环境 **(runtime)** 。实际上它是对 **Google V8** 引擎进行了封装。官网 **(http://www.nodejs.org)** 是这样介绍 **Node** 的：一个搭建在 **ChromeJavaScript** 运行时上的平台，用于构建高速、可伸缩的网络程序。 **Node** 采用的事件驱动、非阻塞 **I/O** 模型，使它既轻量又高效，并成为构建运行在分布式设备上的数据密集型实时程序的完美选择。

目前 **Node** 在实际开发中主要作为开发基础环境存在，用于进行模块管理，编译 **es6** 代码，编译 **sass** 文件，运行各种 **Js** 脚本。

- 下载

进入官网 **https://nodejs.org/en/**，选取合适的版本进行下载

- 安装
  - **Linux**

    先将安装包解压

    然后进行环境变量的配置即可

  - **windows**

直接点击安装即可

基本使用

- 执行 **js** 脚本

使用 node 可以直接执行 js 脚本

例如当前有一个 js 脚本文件为 demo.js 可以使用 node 命令来执行 js 脚本。

$ node demo # 或者 $ node demo.js

- **REPL** 环境

在命令行键入 node 命令，后面没有文件名，就进入一个 Node.js 的 REPL 环境（Read–eval–print loop，”读取 - 求值 - 输出”循环），可以直接运行各种 JavaScript 命令。

```
$ node
>1+1
2
>
```

- 模块化

Node.js 采用模块化结构，按照 **CommonJS** 规范定义和使用模块。在 **Node** 中，以模块为单位划分所有功能，并且提供一个完整的模块加载机制，使得我们可以将应用程序划分为各个不同的部分，并且对这些部分进行很好的协同管理。通过将各种可重用的代码编写在各种模块中的方法，我们可以大大减少应用程序的代码量，提高应用程序开发效率以及应用程序的可读性。通过模块加载机制，我们也可以将各种第三方模块引入到我们的应用程序中。

CommonJS

JavaScript 是一种功能强大的面向对象语言，具有一些最快速的动态语言解释器。官方 **JavaScript** 规范定义了一些用于构建基于浏览器的应用程序的对象的 **api** 。但是，规范并没有定义一个用于对于构建更广泛的应用程序的标准库。 **CommonJS API** 将通过定义处理许多常见应用程序需求的 **API** 来填补这一空白，最终提供与 **Python** 、 **Ruby** 和 **Java** 一样丰富的标准库。其意图是应用程序开发人员能够使用 **CommonJS API** 编写应用程序，然后在不同的 **JavaScript** 解释器和主机环境中运行该应用程序。在兼容 **CommonJS** 的系统中，你可以使用 **JavaScript** 程序开发：

- 服务器端 **JavaScript** 应用程序
- 命令行工具
- 图形界面应用程序
- 混合应用程序（如， **Titanium** 或 **Adobe AIR** ）

➢ **Node** 应用由模块组成，采用 **CommonJS** 模块规范。

◆ 模块作用域

每个文件就是一个模块，有自己的作用域。在一个文件里面定义的变量、函数、类，都是私有的，对其他文件不可见

- 私有属性

```
// example.js

var x = 5;

var addX = function (value) { return value + x; };
```

上面代码中，变量 x 和函数 addX，是当前文件 example.js 私有的，其他文件不可见。

- 全局属性

如果想在多个文件分享变量，必须定义为 global 对象的属性。

```
global.warning = true;
```

◆ 模块交互

CommonJS 规范规定，每个模块内部， **module** 变量代表当前模块。这个变量是一个对象，它的 **exports** 属性（即 **module.exports** ）是对外的接口。加载某个模块，其实是加载该模块的 **module.exports** 属性。

- 定义模块

```
var x = 5;

var addX = function (value) { return value + x; };

module.exports.x = x;

module.exports.addX = addX;
```

- 模块加载

require 方法用于加载模块。

```
var example = require('./example.js');
console.log(example.x); // 5
console.log(example.addX(1)); // 6
```

模块对象

Node 内部提供一个 **Module** 构建函数。所有模块都是 **Module** 的实例。每个模块内部，都有一个 **module** 对象，代表当前模块。它有以下属性。

- **module.id** 模块的识别符，通常是带有绝对路径的模块文件名。
- **module.filename** 模块的文件名，带有绝对路径。
- **module.loaded** 返回一个布尔值，表示模块是否已经完成加载。
- **module.parent** 返回一个对象，表示调用该模块的模块。
- **module.children** 返回一个数组，表示该模块要用到的其他模块。
- **module.exports** 表示模块对外输出的值。

exports 变量

为了方便， **Node** 为每个模块提供一个 **exports** 变量，指向 **module.exports** 。这等同在每个模块头部，有一行这样的命令。

```
var exports = module.exports;
```

◆ path 模块

path 模块提供了一些工具函数，用于处理文件与目录的路径，使用如下方法引用：

var path = require('path');

path.basename() 该方法返回一个参数路径的最后一部分

path.dirname() 该方法返回一个 path 的目录名

path.extname() 该方法返回 path 的扩展名，即从 path 的最后一部分中的最后一个 .（句号）字符到字符串结束。

path.isAbsolute() 该方法会判定 path 是否为一个绝对路径。

path.join() 该方法使用平台特定的分隔符把全部给定的 path 片段连接到一起，并规范化生成的路径

path.normalize() 该方法会规范化给定的 path，并解析 '..' 和 '.' 片段

path.delimiter 该属性提供平台特定的路径分隔符

另外：

- __filename： 指向当前运行的脚本文件名。
- __dirname： 指向当前运行的脚本所在的目录。

◆ querystring 模块

querystring 模块提供了一些实用函数，用于解析与格式化 **URL** 查询字符串。使用如下方法引用

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

querystring.stringify(obj[, sep[, eq]]) 将对象转换为查询字符串

obj 要序列化成 URL 查询字符串的对象。

sep 用于界定查询字符串中的键值对的子字符串。默认为 '&'。

eq 用于界定查询字符串中的键与值的子字符串。默认为 '='。

querystring.parse(str[, sep[, eq]]) 将查询字符串转换为对象

◆ **url** 模块

url 模块提供了一些实用函数，用于 **URL** 处理与解析。 可以通过以下方式使用

var url = require('url');

url.parse()　　将一个 url 地址转换为一个对象

url.resolve()　　该方法会以一种 Web 浏览器解析超链接的方式把一个目标 URL 解析成相对于一个基础 URL

```
url.resolve('/one/two/three', 'four'); // '/one/two/four'
url.resolve('http://example.com/', '/one'); // 'http://example.com/one'
url.resolve('http://example.com/one', '/two'); // 'http://example.com/two'
```

杰普软件科技有限公司
www.briup.com

Tel: (021)55660810
Fax: (021)55660802
Email:training@briup.com
Msn: training.sh@hotmail.com
Home:http://www.briup.com

地址：上海市闸北区万荣路 1188 **弄 F栋 6**号三层 **-** 上海服务外包科技园龙软园区
邮编： 200436

电话： 021-56657112
传真： 021-55661523-8003
电邮： training@briup.com
主页：
http://www.briup.com

Briup High-End IT Training

第 2 章

服务器编程（http 模块）

Brighten Your Way And Raise You Up.

学习目标

- 掌握 **http** 的基本使用方法
- 掌握获取用户请求
- 回应用户请求

◆ 基本使用

Node 内置的 **HTTP** 功能使得它非常适合用来开发 **Web** 程序。 **Node** 的核心是一个强大的流式 **HTTP** 解析器，大概由 **1500** 行经过优化的 **C** 代码组成，是 **Node** 的作者 **Ryran Dahl** 写的，这个解析器跟 **Node** 开发给 **JavaScript** 的底层 **TCP-API** 相结合，提供了一个非常底层，但是非常灵活的 **HTTP** 服务器

1. 加载 **http** 模块

   ```
   var http = require('http');
   ```

2. 创建 **Http** 服务器

   ```
   http.createServer(fn(req,res));
   ```

   服务器每次收到 HTTP 请求后都会调用这个回调函数。这个回调函数会接受两个参数，第一个参数为 http.IncomingMessage 对象，此处代表一个客户端请求，第二个参数值为一个 http.ServerResponse 对象，代表一个服务器端响应对象。返回值为被创建的服务器对象

3. 设定监听

   ```
   server.listen(port[,host][,backlog][,callback]);
   ```

◆ 获取客户端请求

HTTP 服务器接收到的客户端请求时调用的回调函数中的第一个参数值为一个 **http.IncomingMessage** 对象，该对象用于读取客户端请求流中的数据，因此，当从客户端请求流中读取到新的数据时触发 **data** 事件，当读取完客户端请求流中的数据时触发 **end** 事件

- 属性

1. **req.method**

   表示客户端向服务器端发送请求时使用的方法，"GET"/"POST"

2. **req.url**

   表示客户端发送请求时使用的 URL 参数字符串，例如 "/","/post/new/?param=value"

3. **req.headers**

   表示客户端返回发送的请求对象，其中存放了客户端发送的所有请求头信息，包括各种 cookie 信息以及浏览器的各种信息

◆ 获取客户端请求

HTTP 服务器接收到的客户端请求时调用的回调函数中的第一个参数值为一个 **http.IncomingMessage** 对象，该对象用于读取客户端请求流中的数据，因此，当从客户端请求流中读取到新的数据时触发 **data** 事件，当读取完客户端请求流中的数据时触发 **end** 事件

- 方法

1. **req.pause();**

   暂停此 request 触发事件，对于控制上传非常有用

2. **req.resume()**

   恢复一个暂停的 request

◆ 获取客户端请求

HTTP 服务器接收到的客户端请求时调用的回调函数中的第一个参数值为一个 **http.IncomingMessage** 对象，该对象用于读取客户端请求流中的数据，因此，当从客户端请求流中读取到新的数据时触发 **data** 事件，当读取完客户端请求流中的数据时触发 **end** 事件

- 事件

1. **data**

   当接受到消息体中的一部分时候发出 data 事件

2. **end**

   每次完全接受完消息后都会触发一次

- 回应客户端请求

createServer 回调函数的第二个参数，由 **HTTP** 服务器内部创建的。

1. response.end([data][,encoding])

结束响应，这个方法会告诉服务器此响应的所有报文头以及报文体已经发出，服务器在此调用后认为这条信息已经发送完毕，这个方法必须对每个响应调用一次。如果指定的 data 参数，相当于调用了 response.write(data,encoding); 然后跟着调用了 response.end()

2. response.write(chunk[,encoding]);

发送响应内容， chunk 可以是一个字符串或者一个 buffer, 如果 chunk 是一个字符串，则第二个参数指定如何将这个字符串编码成字节流，默认编码为 "utf8"

3. response.writeHead(statusCode[,reasonPhrase,][headers]);

statusCode	用于指定一个三位的 HTTP 状态码 , 例如 404
reasonPhrase	字符串，用于指定该状态码的描述信息
headers	对象，用于指定服务器端创建的响应头对象

杰普软件科技有限公司

www.briup.com

Tel: (021)55660810
Fax: (021)55660802
Email:training@briup.com
Msn: training.sh@hotmail.com
Home:http://www.briup.com

地址：上海市闸北区万荣路 1188 **弄 F栋 6**号三层 **-** 上海服务外包科技园龙软园区
邮编： 200436

电话： 021-56657112
传真： 021-55661523-8003
电邮： training@briup.com
主页：
http://www.briup.com

Briup High-End IT Training

第 3 章：mysql 模块

Brighten Your Way And Raise You Up.

学习目标

- **mysql** 模块的安装与基本使用
- 利用 **mysql** 模块进行增删改查操作

mysql 模块的基本使用

- 安装 **mysql** 模块

  npm install mysql

- 基本使用方法

```
var mysql = require("mysql");                          // 加载 mysql 模块
 var connection = mysql.createConnection({    // 创建连接对象
            host     : 'localhost',
            user     : 'briup',
            password : '123321',
            database : 'web1701'
});
connection.connect();                                  // 进行数据库连接
connection.query('SELECT * from student', function (error, results) {
            if (error) throw error;                    // 执行查询
            console.log('The solution is: ', results);
});
connection.end();                                      // 关闭链接
```

- 连接参数
  - **host** **mysql** 服务所在主机地址，默认为 **localhost**
  - **port** **mysql** 服务端口号，默认为 **3306**
  - **user** 用户名
  - **password** 密码
  - **database** 所要连接的数据库
  - **charset** 建立连接所使用的编码方式，默认为 **UTF8_GENERAL_CI**
  - **connectTimeout** 连接超时毫秒数，默认为 **10000**

- 建立连接

connect 方法有一个参数为回调函数，该函数有一形参用来接收错误信息，如果连接失败 mysql 模块将错误信息传递给 err.

```
connection.connect(function(err) {
    if (err) {
        console.error('error connecting: ' + err.stack);
        return;
    }
    console.log('connected as id ' + connection.threadId);
});
```

在调用查询方法的时候连接会自动创建

```
connection.query('SELECT 1', function (error, results, fields) {
    if (error) throw error;
    // connected!
});
```

- 连接池

使用连接池管理所有的连接，这样可以方便共享单个连接。

```
var mysql = require('mysql');
var pool  = mysql.createPool({
  connectionLimit : 10,
  host            : 'example.org',
  user            : 'bob',
  password        : 'secret',
  database        : 'my_db'});
// 获取连接
pool.getConnection(function(err, connection) {
        connection.query('SELECT 1', function (error, results, fields) {
            connection.release();     // 释放连接
            connection.destroy();     // 当连接不需要的时候销毁连接
        });
});
```

- 关闭连接池

关闭连接池中所有连接。

```
pool.end(function (err) {
  // all connections in the pool have ended
});
```

- 连接池参数
  - **createConnection** 的所有参数
  - connectionLimit

    用于指定连接池中最大的连接数，默认为 10
  - queryLimit

    用于指定允许挂起的最大连接数，如果挂起连接数超过该数值，将抛出异常。默认为 0
  - waitForConnections

    指定当连接池中已经没有可用连接，且当前在用的连接数已等于连接池的最大连接数时所执行的处理。如果值为 true, 则挂起数据库连接请求，直到一个当前在用的数据库连接释放数据库连接。如果为 false, 抛出异常，默认为 true

- 中止连接

中止连接比较优雅的方式是调用 connection 对象的 end() 方法，该方法将保证所有的数据库操作结束后关闭与数据库的连接。 .

```
connection.end(function(err) {
  // The connection is terminated now
});
```

也可以使用 destroy 方法关闭连接 , 这将导致立即终止底层的套接字

```
connection.destroy();
```

- 数据库操作
  - 查询

可以调用 Connection,Pool 实例中的 query() 方法来执行查询。 query 方法的参数形式有以下几种:

query(sql, callback(error,results,fields))

query(sql, values, callback(error,results,fields))

query(options, callback(error,results,fields))

其中

sql 为用于执行数据库操作的 sql 语句

error 为在执行查询过程中出现的异常

results 为查询结果

fields 为 results 中的属性信息

values 为替换数组

options 为对象 {sql,values}

杰普软件科技有限公司
www.briup.com

Tel: (021)55660810
Fax: (021)55660802
Email:training@briup.com
Msn: training.sh@hotmail.com
Home:http://www.briup.com

地址：上海市闸北区万荣路 1188 **弄 F栋 6**号三层 **-** 上海服务外包科技园龙软园区
邮编： 200436

电话： 021-56657112
传真： 021-55661523-8003
电邮： training@briup.com
主页：
http://www.briup.com

Briup High-End IT Training

第 4 章

服务器编程（express 模块）

Brighten Your Way And Raise You Up.

- 介绍

虽然可以使用 **Node** 开发一个高性能的服务器应用程序，但是 **Node** 中只是提供了大量底层的功能，为了让开发者更快，更方便的开发出一个完整的应用程序， **Node** 社区以及 **Node** 的官方包网站（**https://npmjs.org**）提供了各种可用于实现高端功能的开发包，其中很多开发包本身已经实现了一个完整的开发框架。 **Express** 就是基础框架其中之一。 **Express** 是一个可以在 **Node.js** 中使用的 **MVC** 框架，该框架现在已经得到了广泛的利用，可以使用该框架中的各种特性更为方便快捷的开发出一个完整的 **Web** 应用程序。 **Express** 是一个可以在 **Node.js** 中使用的 **MVC** 框架，该框架现在已经得到了广泛的利用，可以使用该框架中的各种特性更为方便快捷的开发出一个完整的 **Web** 应用程序

- 使用

1. 安装

```
$ npm install express
```

2. 加载

```
var express = require('express');
```

3. 配置路由

```
var app = express();
app.get('/', function (req, res,next) { res.send('Hello World!')); // 路由
```

3. 设置监听

```
app.listen(3000, function () {console.log(" 端口号为 3000 的服务器已经开启 ")});
```

路由

路由（**Routing**）是 **Express** 框架中的一个重要概念，在 **Express** 框架中使用路由来根据客户端请求所提交的不同 **URL** 地址返回不同的服务器端响应结果。

- 单个路由

直接定义在服务器对象上

```
var app = express();
app.get(url,callback(req,resp){});          接收 get 请求
app.post(url,callback(req,resp){});         接收 post 请求
app.delete(url,callback(req,resp){}); 接收 delete 请求
app.put(url,callback(req,resp){});          接收 put 请求
app.all(url,callback(req,resp){});          接收所有类型的请求
```

- 其中回调函数中的参数 **req** 是用来获取请求信息的对象， **resp** 是用来回应请求信息的对象， **next** 用于调用下一个路由

- 路由
  - 路由中间件

在路由中间件对象上定义子路由

```
var app = express();
var userRoute = express.Router ();        创建路由中间件对象
userRoute.get(callback(){});              定义中间件
userRoute.post(callback(){});
userRoute.delete(callback(){});
userRoute.put(callback(){});
app.use("/user",userRoute);               使用中间件
```

- 其中回调函数中的参数 **req** 是用来获取请求信息的对象， **resp** 是用来回应请求信息的对象， **next** 用于调用下一个路由

- 路由
  - 路由中指定参数

app.get("/findBook/:id/:name",callback);

如果用户的请求是‘ http://localhost:8888/findBook/1001/terry’ 即可以完成匹配，通过 req.params 获取参数组成的对象

- 参数获取
  - **get** 请求中的参数获取

```
app.get("/login",function(req,resp){
    var obj = url.parse(req.url);
    var params = querystring.parse(obj.query);
    resp.write(JSON.stringify(req.query));
    resp.end();
});
```

  - **post** 请求中的参数获取

```
app.post("/login",function(req,resp){
    var buff = new Buffer(0);     req.on("data",function(data){
    buff += data;
    });
    req.on("end",function(){
    var params = querystring.parse(buff.toString());
    resp.send(JSON.stringify(params));
    });
});
```

◆ 回应请求

通过回调函数中的 response 对象回应请求

res.send([body]) 发送 http 请求， body 可以为流，字符串，对象，或者数组

res.json([body]) 与 send 方法参数相同，可以将其他类型参数转换为 json

res.end([data] [, encoding])结束响应进程

res.redirect(path) 重定向

- 静态文件托管

通过 **Express** 内置的 **express.static** 可以方便地托管静态文件，例如图片、**CSS** 、**JavaScript** 文件等。将静态资源文件所在的目录作为参数传递给 **express.static** 中间件就可以提供静态资源文件的访问了。例如，假设在 **public** 目录放置了图片、**CSS 和 JavaScript** 文件，你就可以

```
app.use(express.static('public'));
```

杰普软件科技有限公司
www.briup.com

Tel: (021)55660810
Fax: (021)55660802
Email:training@briup.com
Msn: training.sh@hotmail.com
Home:http://www.briup.com

地址：上海市闸北区万荣路 1188 **弄 F栋 6**号三层 **-** 上海服务外包科技园龙软园区
邮编： 200436

电话： 021-56657112
传真： 021-55661523-8003
电邮： training@briup.com
主页：
http://www.briup.com

Briup High-End IT Training

第 5 章 :npm

Brighten Your Way And Raise You Up.

学习目标

- 介绍 **npm**
- 掌握如何使用 **npm** 命令初始化模块
- 掌握如何本地安装，更新，删除依赖
- 掌握如何全局安装，更新，删除依赖
- 了解本地 **npm** 仓库 **(sinopia)**

介绍

Npm 是的 **Js** 开发者能够更方便的分享和复用以及更新代码，被复用的代码被称为包或者模块，一个模块中包含了一到多个 **js** 文件。在模块中一般还会包含一个 **package.json** 的文件，该文件中包含了该模块的配置信息。一个完整的项目，需要依赖很多个模块。一个完整的 **npm** 包含三部分

- **npm** 网站

  用于预览 **npm** 管理的包

- 注册机制

  用于上传包，使用数据库来维护包与上传者的信息。

- 客户端

  用于安装包

◆ 创建一个模块

Node.js 的模块是一种能够被发布到 **npm** 上的包。创建模块从创建 **package.json** 文件开始， **package.json** 是模块的配置文件。

- 可以使用 **npm init** 命令来初始化 **package.json** 文件

```
$ npm init
```

  - ✓ **name** 模块名称　　**version** 模块版本
  - ✓ **description** 描述信息　　**main** 指定模块入口文件
  - ✓ **Dependencies** 依赖关系　　**engines** 指定 **node** 版本
  - ✓ **devDependencies** 环境依赖或测试依赖
  - ✓ **optionalDependencies** 可选择依赖
  - ✓ **script** 定义当前模块脚本，使用 **npm run** 来运行所定义的脚本

- 使用 **-y** 参数创建默认 **package.json** 文件

```
$ npm init -y
```

安装 npm

npm 会随着 **node** 一起被安装到本地。可以使用以下命令来更新 **npm**

$ npm install npm@latest -g

- 安装淘宝镜像

由于默认 npm 的仓库在国外，下载起来很慢，可以使用淘宝镜像来加快下载速度。

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

- Registry 注册中心

  官方： https://registry.npmjs.org

  淘宝： https://registry.npm.taobao.org

  私有： http://localIP:port

- 修改 **npm** 权限

执行 npm 的时候有时会遇到权限不足的情况，可以通过以下方式进行修正。

$ npm config get prefix

$ sudo chown -R $(whoami) $(npm config get prefix)/{lib/node_modules,bin,share}

◆ 模块安装

如果想要仅在当前模块中使用某个第三方模块，就可以使用 **npm install** 的默认安装，默认安装即是本地安装；如果想要在命令行中使用模块，就需要进行全局安装。安装时，如果当前目录中没有 **node_modules** 目录， **npm** 就会创建一个该目录。

$ npm install <package_name>

- **$ npm install**

  安装所有项目依赖的模块，依赖的模块定义在 package.json 中

- **$ npm install**

  安装模块时，默认会将所安装的模块写入到 package.json 中的 dependencies 属性，通过添加一些参数改变这个特性。

  -D, --save-dev: Package will appear in your devDependencies.

  -O, --save-optional: Package will appear in your optionalDependencies.

  --no-save: Prevents saving to dependencies

  -E, --save-exact: Saved dependencies will be configured with an exact version rather than using npm's default semver range operator.

- 模块更新

全局更新依赖的模块

$ npm update <module_name>

- 模块删除

从 **node_modules** 中删除不需要的模块

$ npm uninstall -g <package_name>

不仅删除 **node_modules** 中的依赖，还需要删除 **package.json** 中的信息，可以使用 — **save** 参数

$ npm uninstall –save -g <package_name>

- 搭建本地 **npm** 仓库（**sinopia**）

1. 安装 **$ npm install -g sinopia**
2. 配置 **$ npm set registry http://localhost:4873/**
3. 添加用户 **$ npm adduser --registry http://localhost:4873/**
4. 发布模块 **$ npm publish <module_name>**
5. 启动 **$ sinopia**