# 第三章 Node编程基础

Node不像大多数开源平台那样，它很容易设置，对内存和硬盘空间没有过多要求。也不需要复杂的集成开发环境或构建系统。但掌握一些基础知识对你的进步会有很大帮助。本章要解决Node开发新手要面对的两个难题：

A.如何组织代码；

B.怎么做异步编程。

大多数经验丰富的程序员都非常熟悉组织代码的问题。按照概念将逻辑组织成类和函数。将包含类和函数的文件组织到源码树的目录中。最后代码被组织到程序和库中。Node的模块系统提供了强大的代码组织机制，本章就要教你如何利用它组织代码。

要领会和掌握异步编程可能需要花些时间。你对程序逻辑应该如何执行的认识要有模式上的转变。在同步编程中，你在写下一行代码时就知道它前面的代码都会先于它执行。然而在异步开发中，程序逻辑乍一看可能就像鲁贝·戈德堡机一样复杂而又滑稽。俗话说，磨刀不误砍柴工，在开始开发大型项目之前，应该学一下怎么才能优雅的控制程序行为。

## 3.1 Node功能的组织及重用

在创建程序时，不管是用Node还是什么，经常会出现不可能把所有代码放到一个文件中的情况，当出现这种情况时，传统的方式是按逻辑相关性对代码分组，将包含大量代码的单个文件分解成多个文件。

Node模块打包是为了重用，但它们不会改变全局作用域。Node模块允许你从被引入文件中选择要暴露给程序的函数和变量。如果模块返回的函数或变量不止一个，那它可以通过设定exports对象的属性来指明它们。但如果模块只返回一个函数或变量，则可以设定module.exports属性。

Node的模块系统避免了对全局作用域的污染，从而也就避免了命名冲突，并简化了代码的重用。

### 3.1.1 创建模块

模块既可能是一个文件，也可能是包含一个或多个文件的目录，如果模块是个目录，Node通常会在这个目录下找一个加index.js的文件作为模块的入口。

典型的模块是一个包含exports对象属性定义的文件，这些属性可以是任意类型的数据，比如字符串、对象和函数。

为了演示如何创建基本的模块，我们在一个名为currency.js的文件中添加一些做货币转换的函数。这个文件如下面的代码清单所示，其中有两个函数，分别对加元和美元进行互换。

定义一个Node模块

Var canadianDollar=0.91;

function roundTwoDecimals(amount){

return Math.round(amount\*100)/100;

}

exports.canadianToUS=function(canadian){

return roundTwoDecimals(us/canadianDollar);

}

exports.USToCanadian=function(us){

return roundTwoDecimals(us/canadianDollar);

}

Exports对象上只设定了两个属性。也就是说引入这个模块的代码只能访问到canadianToUS和USToCanadian这两个函数。而变量canadianDollar作为私有变量仅作用在canadianToUS和USToCanadian的逻辑内部，程序不能直接访问它。

使用这个新模块要用到Node的require函数，该函数以你要用的模块的路径为参数。Node以同步的方式寻找它，定位到这个模块并加载文件中的内容。

下面这个是test-currency.js中的代码，它require了currency.js模块：

var currency=require(‘./currency’);

console.log(’50 Canadian dollars equals this amount of US dollars:’);

console.log(currency.canadianToUS(50));

console.log(’30 US dollars equals this amount of Canadian dollars:’);

consale.log(currency.USToCanadian(30));

引入一个以./开头的模块意味着，如果你准备创建的程序脚本test-currency.js在currency\_app目录下，那你的currency.js模块文件应该也放在currency\_app目录下。在引入时，.js扩展名可以忽略。

### 3.1.2 用module.exports微调模块的创建

尽管用函数和变量组装exports对象能满足大多数的模块创建需要，但有时你可能需要用不同的模型创建该模块。

如果只需要从模块中得到一个函数，那从require中返回一个函数的代码要比返回一个对象的代码更优雅。

要创建只返回一个变量或者函数的模块，你可能会以为只要把exports设定成你想返回的东西就行。但这样是不行的，因为Node觉得不能用任何其他对象、函数或变量给exports赋值。要创建只返回一个变量或函数的模块，需要把exports换成module.exports.用module.exports可以对外提供单个变量、函数或者对象。

### 3.1.3 用node\_modules重用模块

要求模块在文件系统中使用相对路径存放，对于组织程序特定的代码很有帮助，但对于想要在程序间共享或跟其他人共享代码却用处不大。Node中有一个独特的模块引入机制，可以不必知道模块在文件系统中的具体位置。这个机制就是使用node\_modules目录.

## 3.2 异步编程技术

如果你做过Web前端编程，并且遇到过界面事件触发的逻辑，那你就做过异步编程。服务端异步编程也一样：事件发生会触发响应逻辑。在Node的世界里流行两种响应逻辑管理方式：回调和事件监听。

回调通常用来定义一次性响应的逻辑。比如对于数据库查询，可以指定一个回调函数来确定如何处理查询结果。这个回调函数可能会显示数据库查询结果，根据这些结果做些计算，或者以查询结果为参数执行另一个回调函数。

事件监听器，本质上也是一个回调，不同的是，它跟一个概念实体（事件）相关联。例如，当有人在浏览器中点击鼠标时。鼠标点击是一个需要处理的事件。在Node中，当有HTTP请求过来时，HTTP服务器会发出一个请求事件。你可以监听那个请求事件，并添加一些响应逻辑。在下面这个例子中，每当有请求发出时，服务器就会调用handleRequest函数：

Server.on(‘request’,handleRequest);

一个NodeHTTP服务器实例就是一个事件发射器，一个可以继承、能够添加事件发射及处理能力的类。Node的很多核心功能都继承自EventEmitter，你也能创建自己的事件发射器。

Node有两种常用的响应逻辑处理方式，我们已经用其中一种构建了响应逻辑，现在该了解一下它是如何实现得了，所以接下来要学习如下内容：

A.如何用回调处理一次性事件；

B.如何用事件监听器响应重复性事件；

C.异步编程的几个难点。

### 3.2.1 用回调处理一次性事件

回调是一个函数，它被当做参数传给异步函数，它描述了异步操作完成之后要做什么。回调在Node开发中用得很频繁，比事件发射器用得多，并且用起来也很简单。

为了在程序中演示回调的用法，我们来做一个简单到的HTTP服务器，让它实现如下功能：

A.异步获取存放在JSON文件中的文章标题；

B.异步获取简单的HTML模板；

C.把那些标题组装到HTML页面里；

D.把HTML页面发送给用户。

JSON文件会被格式化成一个包含文章标题的字符串数组，内容如下所示。

[

“标题一标题一标题一标题一标题一标题一”,

“标题二标题二标题二标题二标题二标题二”,

“标题三标题三标题三标题三标题三标题三”

]

HTML模板文件（template.html）,如下所示，结构很简单，可以插入文章的标题。

<!DOCTYPE html>

<html>

<head>

<meta charset="UTF-8">

<title></title>

</head>

<body>

<h1>Last Posts</h1>

<ul>

<li>%</li>

</ul>

</body>

</html>

获取JSON文件中的标题并渲染Web页面的代码如下所示

var http=require('http');

var fs=require('fs');

http.createServer(function(req,res){

if(req.url=='/'){

fs.readFile('./titles.json',function(err,data){

if(err){

console.log(err);

res.end("server not");

}else{

var titles=JSON.parse(data.toString());

fs.readFile('./template.html',function(err,data){

if(err){

console.error(err);

res.end('Server Error');

}else{

var tmpl=data.toString();

var html=tmpl.replace('%',titles.join('</li><li>'));

res.writeHead(200,{'Content-Type':'text/html'});

res.end(html);

}

});

}

})

}

}).listen(8001,"127.0.0.1");

### 3.2.2 用事件发射器处理重复性事件

事件发射器会触发事件，并且在那些事件被触发时能够处理它们。一些重要的Node API组件，比如HTTP服务器、TCP服务器和流，都被做成了事件发射器。你也可以创建自己的事件发射器。

下面的代码清单实现了一个echo服务器。每当有客户端连接上来时，它就会创建一个socket。socket是个事件发射器，可以用on方法添加监听器响应data事件。只要socket上有新数据