http://www.nodebeginner.org/index-zh-cn.html

Node.js事实上就是另外一种上下文，它允许在后端（脱离浏览器环境）运行JavaScript代码

**一个完整的基于Node.js的web应用**

* 用户可以通过浏览器使用我们的应用。
* 当用户请求*http://domain/start*时，可以看到一个欢迎页面，页面上有一个文件上传的表单。
* 用户可以选择一个图片并提交表单，随后文件将被上传到*http://domain/upload*，该页面完成上传后会把图片显示在页面上。

### 应用不同模块分析

* 我们需要提供Web页面，因此需要一个*HTTP服务器*
* 对于不同的请求，根据请求的URL，我们的服务器需要给予不同的响应，因此我们需要一个*路由*，用于把请求对应到请求处理程序（request handler）
* 当请求被服务器接收并通过路由传递之后，需要可以对其进行处理，因此我们需要最终的*请求处理程序*
* 路由还应该能处理POST数据，并且把数据封装成更友好的格式传递给请求处理入程序，因此需要*请求数据处理功能*
* 我们不仅仅要处理URL对应的请求，还要把内容显示出来，这意味着我们需要一些*视图逻辑*供请求处理程序使用，以便将内容发送给用户的浏览器
* 最后，用户需要上传图片，所以我们需要*上传处理功能*来处理这方面的细节

## 构建应用的模块

### 一个基础的HTTP服务器

只要把不同功能的代码放入不同的模块中，保持代码分离还是相当简单的。

这种方法允许你拥有一个干净的主文件（main file），你可以用Node.js执行它；同时你可以拥有干净的模块，它们可以被主文件和其他的模块调用。

把主文件叫做index.js或多或少是个标准格式。把服务器模块放进叫server.js的文件里则很好理解。

Server.js:

var http = require("http");  
  
http.createServer(function(request, response) {  
  response.writeHead(200, {"Content-Type": "text/plain"});  
  response.write("Hello World");  
  response.end();  
}).listen(8888);

启动服务器并侦听8888端口：

var http = require("http");  
  
var server = http.createServer();  
server.listen(8888);

最有趣的部分是 createServer() 的第一个参数，一个函数定义。因为在JavaScript中，函数和其他变量一样都是可以被传递的。

### 进行函数传递

function say(word) {  
  console.log(word);  
}  
  
function execute(someFunction, value) {  
  someFunction(value);  
}  
  
execute(say, "Hello")

我们可以直接在另一个函数的括号中定义和传递这个函数：

fcuntion execute(somefunction,value){

somefunction(value);

}

execute(function(word){console.log(word)},"Hello");

在 execute 接受第一个参数的地方直接定义了我们准备传递给 execute 的函数。用给这个函数起名字，这也是为什么它被叫做 匿名函数 。

在JavaScript中，一个函数可以作为另一个函数接收一个参数。我们可以先定义一个函数，然后传递，也可以在传递参数的地方直接定义函数。

### 函数传递是如何让HTTP服务器工作的

向 createServer 函数传递了一个匿名函数。

用这样的代码也可以达到同样的目的：

var http = require("http");  
  
function onRequest(request, response) {  
  response.writeHead(200, {"Content-Type": "text/plain"});  
  response.write("Hello World");  
  response.end();  
}  
  
http.createServer(onRequest).listen(8888);

为什么要用这种方式呢？

Node.js/JavaScript的事件驱动设计

我们给某个方法传递了一个函数，这个方法在有相应事件发生时调用这个函数来进行 回调 。server1.js

### 服务器是如何处理请求的

当回调启动，我们的 onRequest() 函数被触发的时候，有两个参数被传入： request 和 response

### 服务端的模块放在哪里

我们现在在 server.js 文件中有一个非常基础的HTTP服务器代码，而且我提到通常我们会有一个叫 index.js 的文件去调用应用的其他模块（比如 server.js 中的HTTP服务器模块）来引导和启动应用。

Node.js中自带了一个叫做“http”的模块，我们在我们的代码中请求它并把返回值赋给一个本地变量。这把我们的本地变量变成了一个拥有所有 http 模块所提供的公共方法的对象。

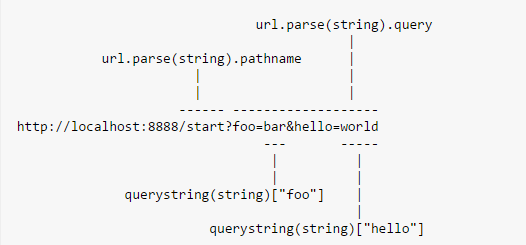
对于不同的URL请求，服务器应该有不同的反应。直接在回调函数onRequest() 中做这件事情。处理不同的HTTP请求在我们的代码中是一个不同的部分，叫做“路由选择”——那么，我们接下来就创造一个叫做 路由 的模块吧。

### 如何来进行请求的“路由”

我们要为路由提供请求的URL和其他需要的GET及POST参数，随后路由需要根据这些数据来执行相应的代码（这里“代码”对应整个应用的第三部分：一系列在接收到请求时真正工作的处理程序）。

因此，我们需要查看HTTP请求，从中提取出请求的URL以及GET/POST参数。这一功能应当属于路由还是服务器（甚至作为一个模块自身的功能）确实值得探讨，但这里暂定其为我们的HTTP服务器的功能.

我们需要的所有数据都会包含在request对象中，该对象作为onRequest()回调函数的第一个参数传递。但是为了解析这些数据，我们需要额外的Node.JS模块，它们分别是url和querystring模块.



现在我们来给onRequest()函数加上一些逻辑，用来找出浏览器请求的URL路径:

var http = require("http");  
var url = require("url");  
  
function start() {  
  function onRequest(request, response) {  
    var pathname = url.parse(request.url).pathname;  
    console.log("Request for " + pathname + " received.");  
    response.writeHead(200, {"Content-Type": "text/plain"});  
    response.write("Hello World");  
    response.end();  
  }  
  
  http.createServer(onRequest).listen(8888);  
  console.log("Server has started.");  
}  
  
exports.start = start;

现在我们可以来编写路由了，建立一个名为router.js的文件，添加以下内容：

function route(pathname) {  
  console.log("About to route a request for " + pathname);  
}  
  
exports.route = route;

首先，我们来扩展一下服务器的start()函数，以便将路由函数作为参数传递过去：

var http = require("http");  
var url = require("url");  
  
function start(route) {  
  function onRequest(request, response) {  
    var pathname = url.parse(request.url).pathname;  
    console.log("Request for " + pathname + " received.");  
 **route(pathname);**    response.writeHead(200, {"Content-Type": "text/plain"});  
    response.write("Hello World");  
    response.end();  
  }  
  
  http.createServer(onRequest).listen(8888);  
  console.log("Server has started.");  
}  
  
exports.start = start;

我们会相应扩展index.js，使得路由函数可以被注入到服务器中：

var server = require("./server");  
**var router = require("./router");**  
  
server.start(router.route);

### 路由给真正的请求处理程序

应用程序需要新的部件，因此加入新的模块 . 我们来创建一个叫做requestHandlers的模块，并对于每一个请求处理程序，添加一个占位用函数，随后将这些函数作为模块的方法导出：

function start() {  
  console.log("Request handler 'start' was called.");  
}  
  
function upload() {  
  console.log("Request handler 'upload' was called.");  
}  
  
exports.start = start;  
exports.upload = upload;

在这里我们得做个决定：是将requestHandlers模块硬编码到路由里来使用，还是再添加一点依赖注入？虽然和其他模式一样，依赖注入不应该仅仅为使用而使用，但在现在这个情况下，使用依赖注入可以让路由和请求处理程序之间的耦合更加松散，也因此能让路由的重用性更高。

这意味着我们得将请求处理程序从服务器传递到路由中，但感觉上这么做更离谱了，一路把这堆请求处理程序从我们的主文件传递到服务器中，再将之从服务器传递到路由。我们得JavaScript的对象仅仅是键/值对的集合，它又怎么会拥有方法呢？好吧，这里的值可以是字符串、数字或者……函数！

好了，最后再回到代码上来。现在我们已经确定将一系列请求处理程序通过一个对象来传递，并且需要使用松耦合的方式将这个对象注入到route()函数中。

我们先将这个对象引入到主文件index.js中：

var server = require("./server");  
var router = require("./router");  
**var requestHandlers = require("./requestHandlers");  
  
var handle = {}  
handle["/"] = requestHandlers.start;  
handle["/start"] = requestHandlers.start;  
handle["/upload"] = requestHandlers.upload;**  
server.start(router.route, **handle);**

虽然handle并不仅仅是一个“东西”（一些请求处理程序的集合），我还是建议以一个动词作为其命名，这样做可以让我们在路由中使用更流畅的表达式，稍后会有说明.

在完成了对象的定义后，我们把它作为额外的参数传递给服务器，为此将server.js修改如下：var http = require("http");  
var url = require("url");  
  
**function start(route, handle)** {  
  function onRequest(request, response) {  
    var pathname = url.parse(request.url).pathname;  
    console.log("Request for " + pathname + " received.");  
  
**route(handle, pathname);**  
  
    response.writeHead(200, {"Content-Type": "text/plain"});  
    response.write("Hello World");  
    response.end();  
  }  
  
  http.createServer(onRequest).listen(8888);  
  console.log("Server has started.");  
}  
  
exports.start = start;

这样我们就在start()函数里添加了handle参数，并且把handle对象作为第一个参数传递给了route()回调函数。

然后我们相应地在route.js文件中修改route()函数：

function route(handle, pathname) {  
  console.log("About to route a request for " + pathname);  
**if (typeof handle[pathname] === 'function')** {  
    handle[pathname]();  
  } else {  
    console.log("No request handler found for " + pathname);  
  }  
}  
  
exports.route = route;

<http://www.nodebeginner.org/index-zh-cn.html#whats-needed-to-route-requests>

https://github.com/manuelkiessling/nodebeginner.org/tree/master/code/application