第5章 Node.js Web服务器开发

Web服务器一般指网站服务器，是驻留于因特网上某种类型计算机的程序。Web服务器的基本功能就是提供Web信息浏览服务。如果读者有过制作静态的HTML网站的经历，大概会知道Web服务器提供静态文件服务的功能，将静态文件部署到服务器指定的根目录下，就可以通过访问服务器的IP地址和端口号来访问这个静态的HTML文件，然后服务器将文件返回给浏览器。

Node也具备开发服务器的能力，但是这和传统意义上的服务器略有不同，可以使用Node提供的模块自己手动编写一个服务器应用。不过，使用Node编写一个服务器应用非常简单，只需要几行代码就可以了，而且对自己写的服务器程序有足够强的控制力。

扫码观看

5.1 使用Node.js搭建Web服务器

5.1.1 http模块

Node.js提供了http模块，主要用于搭建HTTP服务器端和客户端，使用HTTP服务器或客户端功能必须调用http模块，示例代码如下。

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

1. 使用Node创建Web服务器

本节使用http模块搭建一个最基本的HTTP服务器架构，在硬盘上先创建一个server.js文件，在文件中编写创建服务器的代码，示例代码如下。

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


//创建服务器
http.createServer( function (request, response) {
   //解析请求，包括文件名
   var pathname = url.parse(request.url).pathname;

   //输出请求的文件名
   console.log("Request for " + pathname + " received.");
```

```
    //从文件系统中读取请求的文件内容
    fs.readFile(pathname.substr(1), function (err, data) {
        if (err){
            console.log(err);
            //HTTP 状态码: 404 : NOT FOUND
            //Content Type: text/html
            response.writeHead(404, {'Content-Type': 'text/html'});
    }else{
            //HTTP 状态码: 200 : OK
            //Content Type: text/html
            response.writeHead(200, {'Content-Type': 'text/html'});

            //响应文件内容
            response.write(data.toString());
        }
        //发送响应数据
        response.end();
     });
}).listen(8080);

//控制台会输出以下信息
console.log('Server running at http://127.0.0.1:8080/');
```

接下来,在该目录下再创建一个 index.html 文件,示例代码如下。

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>我的第一个标题</h1>
        <p>我的第一个段落。</p>
    </body>
</html>
```

在当前目录下打开命令行窗口,运行 server.js 文件,在命令行窗口中执行如下命令。

```
$ node server.js
Server running at http://127.0.0.1:8080/
```

命令运行成功后,打开浏览器访问 http://localhost:8080/index.html,显示效果如图 5.1 所示。

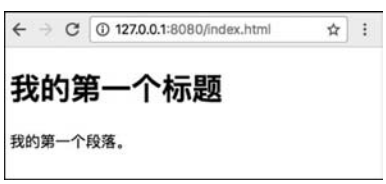

图 5.1 访问 index.html

2. 使用 Node 创建 Web 客户端

使用 Node 创建 Web 客户端需要引入 http 模块,创建 client.js 文件,示例代码如下。

```
var http = require('http');
//用于请求的选项
var options = {
    host: 'localhost',
    port: '8080',
    path: '/index.html'
};

//处理响应的回调函数
var callback = function(response){
    //不断更新数据
    var body = '';
    response.on('data', function(data) {
            body += data;
    });

    response.on('end', function() {
            //数据接收完成
            console.log(body);
    });
}
//向服务器端发送请求
var req = http.request(options, callback);
req.end();
```

在当前目录下打开命令行窗口，运行 client.js 文件，控制台中会输出如下的内容。

```
$ node client.js
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
    </head>
    <body>
        <h1>我的第一个标题</h1>
        <p>我的第一个段落。</p>
    </body>
</html>
```

5.1.2 事件驱动编程

Node 的核心理念是事件驱动编程，对于 Node.js 的开发人员来说，必须掌握这些事件，以及知道如何响应这些事件。实际上，如果学过 HTML，就会明白事件的概念，例如，在页面上添加一个按钮，然后绑定一个单击事件。服务器端的事件驱动和这个按钮的单击事件的道理是一样的。在 5.1.1 节的代码示例中，事件是隐含的，HTTP 请求就是要处理的事

件。http.createServer() 方法将函数作为一个参数，每次有 HTTP 请求发送过来就会调用该函数。

5.1.3 路由

路由就是 URL 到服务器端函数的一种映射，这个定义还是比较抽象的。可以举个生活中的例子，去电影院看电影，都会提前买好电影票，每张电影票都会有指定的座位，观众只需要根据电影票上的座位，找到自己的位置就可以了。把观众看作客户端的每次请求，然后 URL 就是电影票，服务器端定义的路由就是观影大厅的座椅，浏览器的请求（观众）按照电影票上的座位号（URL）去找到自己的位置（服务器端路由函数）对号入座。

在服务器端定义一个路由时，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。因此，需要查看 HTTP 请求，从中提取出请求的 URL 以及 GET/POST 参数。

服务器端所需要的所有的参数都在 request 对象中，该对象作为 onRequest()回调函数的第一个参数传递，但是解析这些数据还需要其他的 Node.js 模块，如 url 和 querystring 模块。querystring 模块还可以用来解析 POST 请求体中的参数。

创建 server.js 文件，在文件中编写服务器应用代码，为 onRequest()函数加上一些逻辑，用来找出浏览器请求的 URL 路径。示例代码如下。

```
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.");
}

start()
```

在当前目录下运行 server.js 文件，执行如下命令。

```
node server
```

服务器启动成功后在浏览器中访问 http://localhost:8888/，效果如图 5.2 所示。

访问成功后再看控制台的输出内容，获取到了 URL 中 pathname 的内容，效果如图 5.3 所示。

图 5.2　通过路由访问

图 5.3　控制台输入 pathname 的内容

5.1.4　静态资源服务

还可以通过路由的方式，访问服务器端的静态资源文件，例如，一个 HTML 网页文件或一张图片，因为在访问这些文件时，文件内容不会发生任何变化，所以被称为“静态资源文件”。

Node 服务器对外提供静态资源文件的访问时，需要先使用 Node 读取到指定内容的内容，然后将这些内容发送给浏览器。所以，要先在项目中创建一个名为 public 的目录，用于存放这些静态文件。在这个目录下创建一些 HTML 文件，例如 home.html、about.html、notfound.html，再创建一个 img 子目录，在该子目录下存放一个 logo.jpg 图片。当这些文件都准备完毕后，就可以编写服务器端的路由配置代码了。示例代码如下。

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

function serveStaticFile(res, path, contentType, responseCode){
    if (!responseCode) responseCode = 200;
    fs.readFile(__dirname + path, function(err, data) {
                if (err) {
        res.writeHead(500, {
                'Content-Type': 'text/plain'
        });
        res.end('500 - Internal Error');
    } else {
        res.writeHead(responseCode, {
                'Content-Type': contentType
```

```
            });
            res.end(data);
        }
    });
}

//规范化 URL,去掉查询字符串、可选的反斜杠,并把它变成小写
http.createServer(function(req, res) {
    var path = req.url.replace(/\/?(?:\?.*)?$/, '').toLowerCase();
    switch (path) {
        case '':
            serveStaticFile(res, '/public/home.html', 'text/html');
            break;
        case '/about':
            serveStaticFile(res, '/public/about.html', 'text/html');
            break;
        case '/img/logo.jpg':
            serveStaticFile(res, '/public/img/logo.jpg', 'image/jpeg');
            break;
        default:
            serveStaticFile(res, '/public/404.html', 'text/html', 404);
            break;
    }
}).listen(3000);

console.log('Server started on localhost:3000');
```

在上面的代码中,创建了一个辅助函数 serveStaticFile,它完成了大部分工作。fs.readFile 是读取文件的异步方法。这个函数有同步版本 fs.readFileSync,但这种异步思考问题的方式,在 Node 开发中还是很重要的。函数 fs.readFile 读取指定文件中的内容,当读取完文件后执行回调函数,如果文件不存在,或者读取文件时遇到许可权限方面的问题,会设定 err 变量,并且会返回一个 HTTP 500 的状态码表明服务器错误。如果文件读取成功,文件会带着特定的响应码和内容类型发给客户端。

扫码观看

5.2 请求与响应对象

5.2.1 URL 的组成部分

URL(Uniform Resource Locator,统一资源定位符)是计算机 Web 网络相关的术语,就是俗称的网址。每个网页都有属于自己的 URL 地址,而且所有地址都是具有唯一性。

HTTP URL 是以 http:// 和 https:// 开头的,完整的 URL 地址书写格式如下。

```
http://localhost:3000/about?id=1&kw=hello
```

一个完整的 URL 主要是由以下几个部分组成的。

1. 协议

协议规定了如何传输请求，主要是处理 http 和 https，其他常见的协议还有 file 和 ftp。

2. 主机名

主机名是服务器的标识，运行在本地的计算机或者是本地网络的服务器可以使用一个单词(localhost)或一串数字(IP 地址)来表示。在因特网环境下，主机名通常是以一个顶级域名结尾，如 .com 或 .net。每个以域名表示的主机名，都可以设置多个子域名，俗称二级域名，其中 www 最为常见。

3. 端口

每一台服务器都有一系列端口号，一些端口号比较特殊，如 80 和 443 端口。如果省略端口值，那么默认 80 端口负责 HTTP 传输，443 端口负责 HTTPS 传输。如果不使用 80 和 443 端口，就需要一个大于 1023 的端口号。

4. 路径

URL 中影响应用程序的第一个组成部分通常是路径，路径是应用中的页面或其他资源的唯一标识。

5. 查询字符串

查询字符串是一种键值对集合，是可选的，它以问号(?)开头，键值对则以与号(&)分隔开。所有的名称和值都必须使用 URL 编码，JavaScript 提供了一个嵌入式的函数 encodeURIComponent 来处理。

6. 信息片段

信息片段被严格限制在浏览器中使用，不会传递到服务器，用它控制单页应用或 Ajax 富应用越来越普遍。最初，信息片段只是用来让浏览器展现文档中通过锚点标记指定的部分。

5.2.2 HTTP 请求方法

HTTP 是超文本传输协议，其定义了客户端和服务器端之间文本传输的规范。根据 HTTP 标准，HTTP 可以使用多种请求方法，在 HTTP 1.0 版本中定义了三种请求方法，分别是 GET、POST、HEAD 方法。到了 HTTP 1.1 版本中新增了六种请求方法，分别是 OPTIONS、PUT、PATCH、DELETE、TRACE 和 CONNECT 方法。

1. GET 请求

请求指定的页面信息，并返回实体主体。

2. POST 请求

向指定资源提交数据进行处理请求(例如提交表单或者上传文件)。数据被包含在请求体中。POST 请求可能会导致新的资源的建立和(或)已有资源的修改。

3. HEAD 请求

类似于 GET 请求，只不过返回的响应中没有具体的内容，用于获取报头。

4. PUT 请求

从客户端向服务器传送的数据取代指定的文档内容。

5. DELETE 请求

请求服务器删除指定的页面。

6. CONNECT 请求

HTTP 1.1 协议中预留给能够将连接改为管道方式的代理服务器。

7. OPTIONS 请求

允许客户端查看服务器的性能。

8. TRACE 请求

回显服务器收到的请求，主要用于测试或诊断。

9. PATCH 请求

是对 PUT 方法的补充，用来对已知资源进行局部更新。

5.2.3 请求报头

在浏览网页时，发送到服务器的并不只是 URL，当访问一个网站时，浏览器会发送很多数据信息，这些信息包含用户的设备信息，如浏览器、操作系统、硬件设备等，还包含一些其他信息。所有这些用户信息都将会作为请求报头发送给服务器。在服务器端也可以查看浏览器发送过来的这些信息，以 Express 为例，可以在路由的函数中获取到这些信息。示例代码如下。

```
app.get('/headers', function(req, res) {
    res.set('Content-Type', 'text/plain');
    var s = '';
    for (var name in req.headers) s += name + ': ' + req.headers[name] + '\n';
    res.send(s);
});
```

5.2.4 响应报头

浏览器以请求报头的形式发送用户信息到服务器，当服务器响应时，同样会回传一些浏览器没有必要渲染和显示的信息，通常是元数据和服务器信息。内容类型头信息，用来告诉浏览器正在被传输的内容类型，浏览器都根据内容类型来做进一步处理。除了内容类型之外，报头还会指出响应信息是否被压缩，以及使用的是哪种编码。响应报头还可以包含关于浏览器对资源缓存时长的提示，这对于优化网站是非常重要的。响应报头还经常会包含一些关于服务器的信息，一般会指出服务器的类型，有时甚至会包含操作系统的详细信息。

向浏览器返回服务器信息存在着一定的风险，会给黑客留下可乘之机，从而使站点陷入危险。对安全要求比较高的服务器需要忽略这些信息，甚至提供虚假的信息。以 Express 为例，如果要禁用 X-Powered-By 头信息，可以使用下面的代码。

```
app.disable('x-powered-by');
```

在浏览器开发者工具中可以找到响应头信息。例如，在 Chrome 浏览器中查看响应报头信息可以在开发者工具中的 Network 栏目中查看。

5.2.5 请求体

除了请求报头外，请求还需要一个主体，一般 GET 请求没有主体内容，但是 POST 请

求是有的。POST 请求主体最常见的媒体类型是 application/x-www-form-urlendcoded，是键值对集合的简单编码，用 & 符号分隔。如果 POST 请求需要支持文件上传，则媒体类型是 multipart/form-data，它是一种更为复杂的格式。最后是 Ajax 请求，它可以使用 application/json。

5.2.6 参数

对于任何一个请求，参数可以来自查询字符串、请求的 Cookies、请求体或指定的路由参数。在 Node 应用中，请求对象的参数方法会重写所有的参数。

5.2.7 请求对象

请求对象通常会被传递到回调方法中，对于请求对象的形参，会被命名为 req 或 request。请求对象的生命周期始于 Node 的一个核心对象 http.IncomingMessage 的实例。在 Express 中添加了一些新的功能，请求对象的属性和方法除了 Node 提供的 req.headers 和 req.url 之外，所有的方法都是由 Express 提供的。

req.params：包含命名过的路由参数。

req.param(name)：返回命名的路由参数，或者 GET 请求或 POST 请求参数。

req.query：包含以键值对存放的查询字符串参数(通常称为 GET 请求参数)。

req.body：包含 POST 请求参数。

req.route：关于当前匹配路由的信息，主要用于路由调试。

req.cookies：包含从客户端传递过来的 Cookies 值。

req.singnedCookies：用法与 req.cookies 相同。

req.headers：从客户端接收到的请求报头。

req.accepts([types])：用来确定客户端是否接受一个或一组指定的类型。

req.ip：客户端的 IP 地址。

req.path：请求路径(不包含协议、主机、端口或查询字符串)。

req.host：用来返回客户端所报告的主机名。

req.xhr：如果请求由 Ajax 发起将会返回 true。

req.protocol：用于标识请求的协议(HTTP 或 HTTPS)。

req.secure：如果连接是安全的，将返回 true。等同于 req.protocol===='https'。

req.url：返回了路径和查询字符串(它们不包含协议、主机或端口)。

req.originalUrl：返回了路径和查询字符串，但会保留原始请求和查询字符串。

req.acceptedLanguages：用来返回客户端首选的一组语言。

5.2.8 响应对象

响应对象通常会被传递到回调函数中，作为回调函数的形参，会被命名为 res、resp 或 response。响应对象的生命周期始于 Node 核心对象 http.ServerResponse 的实例。Express 中添加了一些附加功能，响应对象的属性和方法都是由 Express 提供的。

res.status(code)：用于设置 HTTP 状态代码。Express 默认为 200。

res.set(name,value)：用于设置响应头。

res.cookie(name,value,[options])：用于设置客户端 Cookies 值。

res.clearCookie(name,[options])：用于清除客户端 Cookies 值。

res.redirect([status],url)：重定向浏览器。默认重定向代码是 302。

res.send([status],body)：用于向客户端发送响应及可选的状态码。

res.json([status],json)：向客户端发送 JSON 以及可选的状态码。

res.jsonp([status],json)：向客户端发送 JSONP 及可选的状态码。

res.type(type)：用于设置 Content-Type 头信息。

res.format(object)：该方法允许根据接收请求报头发送不同的内容。

res.attachment([filename])：将响应报头 Content-Disposition 设置为 attachment。

res.download(path,[filename],[callback])：会将响应报头 Content-Disposition 设为 attachment,可以指定要下载的文件。

res.sendFile(path,[option],[callback])：根据路径读取指定文件并将内容发送到客户端。

res.links(links)：用于设置链接响应报头。

res.locals：包含用于渲染视图的默认上下文。

res.render(view,[locals],callback)：使用配置的模板引擎渲染视图,默认响应代码为 200。