### HTTP服务器

要开发HTTP服务器程序，从头处理TCP连接，解析HTTP是不现实的。这些工作实际上已经由Node.js自带的http模块完成了。应用程序并不直接和HTTP协议打交道，而是操作http模块提供的request和response对象。

request对象封装了HTTP请求，我们调用request对象的属性和方法就可以拿到所有HTTP请求的信息；

response对象封装了HTTP响应，我们操作response对象的方法，就可以把HTTP响应返回给浏览器。

用Node.js实现一个HTTP服务器程序非常简单。我们来实现一个最简单的Web程序hello.js，它对于所有请求，都返回Hello world!：

### 文件服务器

让我们继续扩展一下上面的Web程序。我们可以设定一个目录，然后让Web程序变成一个文件服务器。要实现这一点，我们只需要解析request.url中的路径，然后在本地找到对应的文件，把文件内容发送出去就可以了。

解析URL需要用到Node.js提供的url模块，它使用起来非常简单，通过parse()将一个字符串解析为一个Url对象：

'use strict';

**var** url = **require**('url');

console.log(url.parse('http://user:pass@host.com:8080/path/to/file?query=string#hash'));

结果如下：

Url {

protocol: 'http:',

slashes: true,

auth: 'user:pass',

host: 'host.com:8080',

port: '8080',

hostname: 'host.com',

hash: '#hash',

search: '?query=string',

query: 'query=string',

pathname: '/path/to/file',

path: '/path/to/file?query=string',

href: 'http://user:pass@host.com:8080/path/to/file?query=string#hash' }

处理本地文件目录需要使用Node.js提供的path模块，它可以方便地构造目录：

'use strict';

**var** path = **require**('path');

*// 解析当前目录:***var** workDir = path.resolve('.'); *// '*H:/yuanjiaCN.github.io/nodeTest*'*

*// 组合完整的文件路径:当前目录+'pub'+'index.html':***var** filePath = path.join(workDir, 'pub', 'index.html');*// '*H:/yuanjiaCN.github.io/nodeTest*/index.html'*

使用path模块可以正确处理操作系统相关的文件路径。在Windows系统下，返回的路径类似于H:/yuanjiaCN.github.io/nodeTest/index.html，这样，我们就不关心怎么拼接路径了。

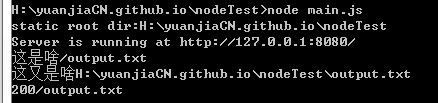
最后，我们实现一个文件服务器main.js：

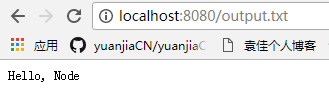
var fs = require("fs");  
var url = require("url");  
var path = require("path");  
var http = require("http");  
  
var root = path.resolve(process.argv[2] || ".");  
console.log("static root dir:" + root);  
  
var server = http.createServer(function (request, response) {  
 var pathname = url.parse(request.url).pathname;  
 console.log("这是啥"+ pathname);  
 var filepath = path.join(root, pathname);  
 console.log("这又是啥"+ filepath);  
 fs.stat(filepath, function (err, stats) {  
 if(!err && stats.isFile()){  
 console.log("200" + request.url);  
 response.writeHead(200);  
 fs.createReadStream(filepath).pipe(response);  
 } else {  
 console.log("404" + request.url);  
 response.writeHead(404);  
 response.end("404 Not Found");  
 }  
 });  
});  
server.listen(8080);  
console.log("Server is running at http://127.0.0.1:8080/")

没有必要手动读取文件内容。由于response对象本身是一个Writable Stream，直接用pipe()方法就实现了自动读取文件内容并输出到HTTP响应。

在命令行运行node main.js /path/to/dir，把/path/to/dir改成你本地的一个有效的目录，然后在浏览器中输入http://localhost:8080/index.html：

注意！index.html可以替换成其他文件，这里我用的是output.txt





这样就很清楚了，pathname = url.parse(request.url).pathname;

pathname是文件名。

filepath = path.join(root, pathname);这是整个目录+文件名

另外！关于命令行输出，只要不关闭命令行，每刷新一次网页，

这是啥/ ……

这又是啥……

200/output.txt

之类的就会再出现一次

包括刚才寝室的网不稳定，一度出现了几句

这是啥/ ……

这又是啥……

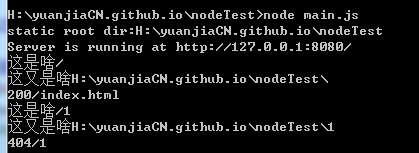
404/

试了一下，发现如果只输入http://localhost:8080/就会报错，那么我来增强一下它的性能。

如果发现我输入的是目录，就自动找一下目录下的index.html文件，并显示它。

var fs = require("fs");  
var url = require("url");  
var path = require("path");  
var http = require("http");  
  
var root = path.resolve(process.argv[2] || ".");  
console.log("static root dir:" + root);  
  
var server = http.createServer(function (request, response) {  
 var pathname = url.parse(request.url).pathname;  
 console.log("这是啥"+ pathname);  
 var filepath = path.join(root, pathname);  
 console.log("这又是啥"+ filepath);  
  
 fs.stat(filepath, function (err, stats) {  
 if(!err && stats.isFile()){  
 console.log("200" + request.url);  
 response.writeHead(200);  
 fs.createReadStream(filepath).pipe(response);  
 } else if(!err && stats.isDirectory) {  
 console.log("200" + request.url + "index.html");  
 filepath = path.join(root, "index.html");  
 response.writeHead(200);  
 fs.createReadStream(filepath).pipe(response);  
 } else {  
 console.log("404" + request.url);  
 response.writeHead(404);  
 response.end("404 Not Found");  
 }  
 });  
});  
server.listen(8080);  
console.log("Server is running at http://127.0.0.1:8080/")

写的不是很优雅。。见谅



第一个测试直接输入http://localhost:8080/

事实上获取的是http://localhost:8080/index.html

第二个测试我输入了http://localhost:8080/1