http协议

要理解Web服务器程序的工作原理，首先，我们要对HTTP协议有基本的了解。如果你对HTTP协议不太熟悉，先看一看HTTP协议简介（https://www.liaoxuefeng.com/wiki/1016959663602400/1017804782304672）。

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。

```
'use strict';

// 导入http模块:
var http = require('http');

// 创建http server，并传入回调函数:
var server = http.createServer(function (request, response) {
    // 回调函数接收request和response对象,
    // 获得HTTP请求的method和url:
    console.log(request.method + ': ' + request.url);
    // 将HTTP响应200写入response, 同时设置Content-Type: text/html:
    response.writeHead(200, {'Content-Type': 'text/html'});
    // 将HTTP响应的HTML内容写入response:
    response.end('<h1>Hello world!</h1>');
});

// 让服务器监听8080端口:
server.listen(8080);
console.log('Server is running at http://127.0.0.1:8080/');
```

文件服务器

process.argv

返回当前命令行指令参数 ，但不包括node特殊(node-specific) 的命令行选项（参数）。常规第一个元素会是 'node'， 第二个元素将是 .Js 文件的名称。接下来的元素依次是命令行传入的参数。

```
'use strict';

var
    fs = require('fs'),
    url = require('url'),
    path = require('path'),
    http = require('http');

// 从命令行参数获取root目录，默认是当前目录:
var root = path.resolve(process.argv[2] || '.');

console.log('Static root dir: ' + root);

// 创建服务器:
var server = http.createServer(function (request, response) {
    // 获得URL的path，类似 '/css/bootstrap.css':
    var pathname = url.parse(request.url).pathname;
    // 获得对应的本地文件路径，类似 '/srv/www/css/bootstrap.css':
    var filepath = path.join(root, pathname);
    // 获取文件状态:
    fs.stat(filepath, function (err, stats) {
        if (!err && stats.isFile()) {
            // 没有出错并且文件存在:
            console.log('200 ' + request.url);
            // 发送200响应:
            response.writeHead(200);
            // 将文件流导向response:
            fs.createReadStream(filepath).pipe(response);
        } else {
            // 出错了或者文件不存在
```

```
            console.log('404 ' + request.url);
            // 发送404响应:
            response.writeHead(404);
            response.end('404 Not Found');
        }
    });
});

server.listen(8080);
console.log('Server is running at http://127.0.0.1:8080/');
```