第9章 Node.js基础模块二

课程提要

- 什么是网络
- http协议
- http模块
- path模块
- url模块
- NPM

9.1 什么是网络

1.网络

网络是由节点和连线构成，表示诸多对象及其相互联系。在数学上，网络是一种图，一般认为专指加权图。网络除了数学定义外，还有具体的物理含义，即网络是从某种相同类型的实际问题中抽象出来的模型。在计算机领域中，网络是信息传输、接收、共享的虚拟平台，通过它把各个点、面、体的信息联系到一起，从而实现这些资源的共享。网络是人类发展史来最重要的发明，提高了科技和人类社会的发展。

2.通信

通信，指人与人或人与自然之间通过某种行为或媒介进行的信息交流与传递，从广义上指需要信息的双方或多方在不违背各自意愿的情况下采用任意方法，任意媒质，将信息从某一方准确安全地传送到另一方。

3.网络通信

网络是用物理链路将各个孤立的工作站或主机相连在一起，组成数据链路，从而达到资源共享和通信的目的。通信是人与人之间通过某种媒体进行的信息交流与传递。网络通信是通过网络将各个孤立的设备进行连接，通过信息交换实现人与人，人与计算机，计算机与计算机之间的通信。

网络通信中最重要的就是网络通信协议。当今网络协议有很多，局域网中最常用的有三个网络协议：MICROSOFT的NETBEUI、NOVELL的IPX/SPX和TCP/IP协议。应根据需要来选择合适的网络协议。

9.2 http协议

超文本传输协议（HTTP，HyperText Transfer Protocol)是互联网上应用最为广泛的一种网络协议。所有的WWW文件都必须遵守这个标准。设计HTTP最初的目的是为了提供一种发布和接收HTML页面的方法。1960年美国人Ted Nelson构思了一种通过计算机处理文本信息的方法，并称之为超文本（hypertext）,这成为了HTTP超文本传输协议标准架构的发展根基。Ted Nelson组织协调万维网协会（World Wide Web Consortium）和互联网工程工作小组（Internet Engineering Task Force ）共同合作研究，最终发布了一系列的RFC，其中著名的RFC 2616定义了HTTP 1.1。

HTTPS（全称：Hyper Text Transfer Protocoover Secure Socket Layer），是以安全为目标的HTTP通道，简单讲是HTTP的安全版。即HTTP下加入SSL层，HTTPS的安全基础是SSL，因此加密的详细内容就需要SSL。 它是一个URI scheme（抽象标识符体系），句法类同http:体系。用于安全的HTTP数据传输。https:URL表明它使用了HTTP，但HTTPS存在不同于HTTP的默认端口及一个加密/身份验证层（在HTTP与TCP之间）。这个系统的最初研发由网景公司(Netscape)进行，并内置于其浏览器Netscape Navigator中，提供了身份验证与加密通讯方法。现在它被广泛用于万维网上安全敏感的通讯，例如交易支付方面。

9.2.1 HTTP协议原理

一次HTTP操作称为一个事务，其工作过程可分为四步：

1. 首先客户机与服务器需要建立连接。只要单击某个超级链接，HTTP的工作就开始了。
2. 建立连接后，客户机发送一个请求给服务器，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是MIME信息包括请求修饰符、客户机信息和可能的内容。
3. 服务器接到请求后，给予相应的响应信息，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是MIME信息包括服务器信息、实体信息和可能的内容。
4. 客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接。

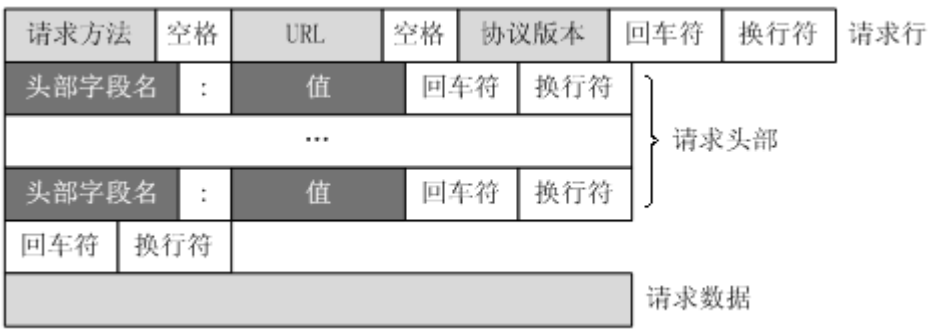
如果在以上过程中的某一步出现错误，那么产生错误的信息将返回到客户端，由显示屏输出。对于用户来说，这些过程是由HTTP自己完成的，用户只要用鼠标点击，等待信息显示就可以了。

许多HTTP通讯是由一个用户代理初始化的并且包括一个申请在源服务器上资源的请求。最简单的情况可能是在用户代理和服务器之间通过一个单独的连接来完成。在Internet上，HTTP通讯通常发生在TCP/IP连接之上。缺省端口是TCP 80，但其它的端口也是可用的。但这并不预示着HTTP协议在Internet或其它网络的其它协议之上才能完成。HTTP只预示着一个可靠的传输。

这个过程就好像我们打电话订货一样，我们可以打电话给商家，告诉他我们需要什么规格的商品，然后商家再告诉我们什么商品有货，什么商品缺货。这些，我们是通过电话线用电话联系（HTTP是通过TCP/IP），当然我们也可以通过传真，只要商家那边也有传真。

9.2.2 请求报文

一个HTTP请求报文由请求行（request line）、请求头（header）、空行、请求数据4个部分组成。



9.2.2.1 请求行（重点：请求方法GET、POST、PUT、DELETE)

请求行即请求报文的起始行，由请求方法字段、URL字段和HTTP协议版本字段3个字段组成。

请求方法描述了服务器应该执行的操作。

HTTP协议的请求方法有GET、POST、HEAD、PUT、DELETE、OPTIONS、TRACE、CONNECT。GET和POST是最常见的HTTP方法，除此以外还包括DELETE、HEAD、OPTIONS、PUT、TRACE。不过，当前的大多数浏览器只支持GET和POST，Spring 3.0提供了一个HiddenHttpMethodFilter，允许你通过"_method"的表单参数指定这些特殊的HTTP方法（实际上还是通过POST提交表单）。服务端配置了HiddenHttpMethodFilter后，Spring会根据_method参数指定的值模拟出相应的HTTP方法，这样，就可以使用这些HTTP方法对处理方法进行映射了。

URL字段描述了要对哪个资源执行这个方法；

HTTP协议版本字段用来告知服务器客户端使用的是哪种HTTP版本。

示例：

```
POST /index.htmHTTP/1.1
```

9.2.2.2 请求头（重点）

请求头由关键字/值对组成，每行一对，关键字和值用英文冒号":"分隔。请求头通知服务器有关于客户端请求的信息，典型的请求头有：

- User-Agent：产生请求的浏览器类型。
- Accept：客户端可识别的内容类型列表。
- Host：请求的主机名，允许多个域名同处一个IP地址，即虚拟主机。

示例：

```
Host: localhost
User-Agent: Mozilla/5.0 (Windows NT 5.1; rv:10.0.2) Gecko/20100101 Firefox/10.0.2
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-cn,zh;q=0.5
Accept-Encoding: gzip, deflate
Connection: keep-alive
Referer: <a target=_blank href="http://localhost/" style="color: rgb(51, 102, 153);
text-decoration: none;">http://localhost/</a>
Content-Length: 25
Content-Type: application/x-www-form-urlencoded
```

9.2.2.3 空行

空行的作用是通过一个空行，告诉服务器请求头部到此为止。

9.2.2.4 请求数据

请求数据不在GET方法中使用，而是在POST方法中使用。POST方法适用于需要客户填写表单的场合。与请求数据相关的最常使用的请求头是Content-Type和Content-Length。

9.2.2 响应报文

HTTP响应报文由三个部分组成，分别是：状态行、消息报头、响应正文。

在响应中唯一真正的区别于请求报文在于第一行中用状态信息代替了请求信息。状态行（status line）通过提供一个状态码来说明所请求的资源情况。

状态行格式如下： HTTP-VersioStatus-Code Reason-Phrase CRLF

其中，HTTP-Version表示服务器HTTP协议的版本；Status-Code表示服务器发回的响应状态代码；Reason-Phrase表示状态代码的文本描述。状态代码由三位数字组成，第一个数字定义了响应的类别，且有五种可能取值。

```
1xx：指示信息--表示请求已接收，继续处理。
2xx：成功--表示请求已被成功接收、理解、接受。
3xx：重定向--要完成请求必须进行更进一步的操作。
4xx：客户端错误--请求有语法错误或请求无法实现。
5xx：服务器端错误--服务器未能实现合法的请求。
```

常见状态代码、状态描述的说明如下（重点状态代码为200,404,403,500）：

```
200 OK：客户端请求成功。
400 Bad Request：客户端请求有语法错误，不能被服务器所理解。
401 Unauthorized：请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用。
403 Forbidden：服务器收到请求，但是拒绝提供服务。
404 Not Found：请求资源不存在，举个例子：输入了错误的URL。
500 InternaServer Error：服务器发生不可预期的错误。
503 Server Unavailable：服务器当前不能处理客户端的请求，一段时间后可能恢复正常。
```

示例：

```
HTTP/1.1 200 OK Date: Sat, 31 Dec 2005 23:59:59 GMT Content-Type:
text/html;charset=ISO-8859-1 Content-Length: 122
<html>
    <head>
        <title>Wrox Homepage</title>
    </head>
    <body>
        <!-- body goes here -->
    </body>
</html>
```

9.3 http模块

9.3.1 http模块介绍

我们知道传统的HTTP服务器会由Aphche、Nginx、IIS之类的软件来担任，但是nodejs并不需要，nodejs提供了http模块，自身就可以用来构建服务器，而且http模块是由C++实现的，性能可靠。

http模块中封装了高效的http服务器和http客户端。http.server是一个基于事件的HTTP服务器，内部是由c++实现的，接口由JavaScript封装。http.request是一个HTTP客户端工具，用于用户向服务器发送数据。

9.3.2 http服务端

http.server是一个基于事件的HTTP服务器，所有的请求都被封装到独立的事件当中，我们只需要对他的事件编写相应的行数就可以实现HTTP服务器的所有功能，它继承自EventEmitter，提供了以下的事件：

request：当客户端请求到来的时候，该事件被触发，提供两个参数request和response，分别是http.ServerRequest和http.ServerResponse表示请求和响应的信息。

connection：当TCP建立连接的时候，该事件被触发，提供了一个参数socket，为net.socket的实例(底层协议对象)

close：当服务器关闭的时候会被触发

除此之外还有checkContinue、upgrade、clientError等事件。

最常用的还是request事件，http也给这个事件提供了一个捷径：http.createServer([requestListener]) 。

示例：

```
var http=require("http");
http.createServer(function(req,res){
    res.writeHead(200,{
        "content-type":"text/plain"
    });

    res.write("hello nodejs");
    res.end();
}).listen(3000);
```

打开浏览器，输入localhost:3000我们就可以看到屏幕上的hello nodejs了，这表明这个最简单的nodejs服务器已经搭建成功了。以上代码是通过直接创建一个http.Server对象，然后为其添加request事件监听，其实也就说createServer方法其实本质上也是为http.Server对象添加了一个request事件监听。

9.3.2.1 http.ServerRequset请求信息

我们都知道HTTP请求分为两部分：请求头和请求数据，如果请求的内容少的话就直接在请求头协议完成之后立即读取，请求数据如果相对较长一点，就需要一定的时间传输。因此提供了三个事件用于控制请求数据传输：

(1)data：当请求体数据到来时，该事件被触发，该事件一共一个参数chunk，表示接受到的数据。

(2)end：当请求体数据传输完成时，该事件被触发，此后将不会再有数据到来。

(3)close：用户当前请求结束时，该事件被触发，不同于end，如果用户强制终止了传输，也会触发close 。

ServerRequest的属性：

名称	含义
complete	客户端请求是否已经发送完成
httpVersion	HTTP协议版本，通常是1.0或1.1
method	HTTP请求方法，如：GET,POST
url	原始的请求路径
headers	HTTP请求头
trailers	HTTP请求尾(不常见)
connection	当前HTTP连接套接字，为net.Socket的实例
socket	connection属性的别名
client	client属性的别名

9.3.2.2 http.ServerResponse返回客户端信息

http.ServerResponse决定了用户最终能到的结果，它是由http.Server的request事件发送的，作为第二个参数传递。主要有三个函数：

- response.writeHead(statusCode,[headers])：向请求的客户端发送响应头。
- statusCode是HTTP的状态码，如200为成功，404未找到等；
- headers是一个类似关联数组的对象，表示响应头的每个属性；
- response.write(data,[encoding]) 向请求客户端发送相应内容：data是buffer或字符串，encoding为编码 。
- response.end([data],[encoding]) 结束响应，告知用户所有发送已经完成，当所有要返回的内容发送完毕，该函数必须被调用一次，如果不调用，客户端永远处于等待状态。

9.3.3 http客户端

http模块提供了两个函数http.request和http.get，功能是作为客户端向服务器端发送请求。

9.3.3.1 http.request

http.request(options,callback)用于发起http请求，接收两个参数，options是一个类似关联数组的对象，里面包含一些请求的参数，callback表示请求后的回调。

options常用的参数如下：

名称	含义
host	请求网站的域名或IP地址
port	请求网站的端口，默认是80
methods	请求方法，默认是GET
path	请求的相对于根的路径，默认是”/”。QueryString含在其中，例如/search?query=helios
headers	一个关联数组对象，为请求头的内容

示例：

```
var http=require('http');
var options={
    hostname:"www.baidu.com",
    port:80,
    methods:"GET",
    path:'/'
}

var req=http.request(options,function(res){
   console.log('STAUS: '+res.statusCode);
   console.log('HRADERS: '+JSON.stringify(res.headers));
   res.setEncoding("utf-8");
   res.on('data',function(chunk){
       console.log(chunk);
   })
})

req.end();
```

需要记住的是：如果我们使用http.request方法时没有调用end方法，服务器将不会收到信息。

9.3.3.2 http.get

http模块还提供了http.get(options,callback)，用来更简单的处理GET方式的请求，它是http.request的简化版本，唯一的区别在于http.get自动将请求方法设为GET请求，同时不需要手动调用req.end(); 这个就好像jQuery中的$.ajax和$.GET的区别。

示例：

```
var http=require('http');
var options={
    hostname:"www.baidu.com",
}

var req=http.get(options,function(res){
   console.log('STAUS: '+res.statusCode);
   console.log('HRADERS: '+JSON.stringify(res.headers));
   res.setEncoding("utf-8");
   res.on('data',function(chunk){
       console.log(chunk);
   })
})
```

9.3.4 http.ServerRequset和http.request区别

http.ServerRequest表示客户端请求的信息，是服务端request事件的一个参数。http.request()是客户端用于发起http请求的一个函数。

9.4 path模块

9.4.1 path模块介绍

path模块是一个专门处理路径问题的模块。通过它，我们可以轻松的获得一个文件、文件夹的绝对路径、相对路径、文件名称、文件扩展名等信息。

9.4.2 path方法

9.4.2.1 normalize

无论是相同系统环境还是不同系统环境，我们编写的路径格式通常是有区别的，通过 path.normalize 方法，我们可以将其转换成该系统环境中大家都认可的一种统一格式。

示例：

```
var path = require("path");
var myPath = path.normalize('/foo/bar//baz/asdf/quux/..');
console.log(myPath);
```

9.4.2.2 join

path.join方法可以将两个或者多个路径拼接成一个绝对路径，并能够识别当中的绝对路径及相对路径。

注意：

a) 传入的参数是字符串的路径片段，可以是一个，也可以是多个。

b) 返回的是一个拼接好的路径，但是根据平台的不同，他会对路径进行不同的规范化，举个例子，Unix系统是”/“，Windows系统是”“，那么你在两个系统下看到的返回结果就不一样。

c) 如果返回的路径字符串长度为零，那么他会返回一个'.'，代表当前的文件夹。

d) 如果传入的参数中有不是字符串的，那就直接会报错。

示例：

```
var path = require("path");
var path1 = 'path1',
    path2 = 'path2/pp',
    path3 = '/path3';
var myPath = path.join(path1, path2, path3);

console.log(myPath);
```

9.4.2.3 resolve

path.resolve 方法会把一个路径或路径片段的序列解析为一个绝对路径。

示例：

```
var path = require("path");
var myPath = path.resolve('path1', 'path2', 'a/bc/');
console.log(myPath);
```

9.4.2.4 parse

path.parse方法返回一个对象，对象的属性表示 path 的元素。

返回的对象有以下属性：

属性	解释
root	代表根目录
dir	代表文件所在的文件夹
base	代表整一个文件
name	代表文件名
ext	代表文件的后缀名

示例：

```
var path = require('path');
var obj = path.parse('/Users/laihuamin/Documents/richEditor/editor/src/task.js');
console.log(obj);
```

9.5 url模块

URL模块用于解析和处理URL字符串，提供了三个方法：

- parse：通过此方法，我们可以轻松获取URL中详细的某一部分数据（例如原始url）
- format：与parse方法相反，通过此方法我们可将url对象轻松转换成url字符串
- resolve：通过此方法，我们可以轻松获得当前目录的绝对路径。

9.5.1 parse方法

parse方法将URL解析成以下几部分：

- href：原始url
- protocal：url协议
- host：主机
- host中又包含以下信息：
- auth：用户认证
- port：端口
- hostname：主机名
- pathname：跟在host之后的整个文件路径
- search：url中HTTP GET信息，包含了?
- query：跟search类似，不包含?
- hash：片段部分，也就是URL#之后的部分

语法：

```
parse(urlStr[, parseQueryString][, slashesDenoteHost])
```

注释：

- urlStr：需要处理的url字符串；
- parseQueryString： 是否将查询参数也解析成对象。为true时将使用查询模块分析查询字符串，默认为false。
- slashesDenoteHost: 解析主机处理，双斜线表示主机。默认为false，//foo/bar 形式的字符串将被解释成 { pathname: '//foo/bar' }；如果设置成true，//foo/bar 形式的字符串将被解释成 { host: foo', pathname: /bar' }

示例：

```
var ur= require("url");
var myurl="http://www.nodejs.org/some/url/?with=query¶m=that#about";
var parsedUrl=url.parse(myurl);
```

结果如下：

```
Ur{
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'www.nodejs.org',
  port: null,
  hostname: 'www.nodejs.org',
  hash: '#about',
  search: '?with=query¶m=that',
  query: 'with=query¶m=that',
  pathname: '/some/url/',
  path: '/some/url/?with=query¶m=that',
  href:'http://www.nodejs.org/some/url/?with=query¶m=that#about'
}
```

9.5.2 format方法

format方法就是parse的相反过程，把对象转换成url字符串。

示例：

```
var ur= require("url");
var testObj1 = {
  protocol: 'http:',
  slashes: true,
  auth: null,
  host: 'www.nodejs.org',
  port: null,
  hostname: 'www.nodejs.org',
  hash: '#about',
  search: '?with=query¶m=that',
  query: 'with=query¶m=that',
  pathname: '/some/url/',
  path: '/some/url/?with=query¶m=that',
  href: 'http://www.nodejs.org/some/url/?with=query¶m=that#about'
```

```
}

var rsUr= url.format(testObj1);
```

9.5.3 resolve方法

resolve方法返回从根目录指定到当前目录的绝对路径url。返回结果去除参数和锚点，返回结果为标准url路径格式。

语法：

```
url.resolve(from, to)
```

示例：

```
var url=require('url');
var url1 = url.resolve('/one/two/three', 'four');        // '/one/two/four'
console.log(url1);
var url2 = url.resolve('http://example.com/', '/one');    // 'http://example.com/one'
console.log(url2);
var url3 = url.resolve('http://example.com/one', '/two'); // 'http://example.com/two'
console.log(url3);
```

9.6 NPM

9.6.1 NPM包管理器介绍

NPM（node package manager），通常称为node包管理器。顾名思义，它的主要功能就是管理node包，包括：安装、卸载、更新、查看、搜索、发布等。

NPM的背后，是基于couchdb的一个数据库，详细记录了每个包的信息，包括作者、版本、依赖、授权信息等。它的一个很重要的作用就是：将开发者从繁琐的包管理工作（版本、依赖等）中解放出来，更加专注于功能的开发。

NPM为我们打开了连接整个 JavaScript 天才世界的一扇大门。它是世界上最大的软件注册表，每星期大约有 30 亿次的下载量，包含超过 600000 个 包（package） （即，代码模块）。来自各大洲的开源软件开发者使用 npm 互相分享和借鉴。

NPM是随同NodeJS一起安装的包管理工具，能解决NodeJS代码部署上的很多问题，常见的使用场景有以下几种：

- 允许用户从NPM服务器下载别人编写的第三方包到本地使用。
- 允许用户从NPM服务器下载并安装别人编写的命令行程序到本地使用。
- 允许用户将自己编写的包或命令行程序上传到NPM服务器供别人使用。

NPM 由三个独立的部分组成：

- 网站 -- 是开发者查找包（package）、设置参数以及管理 npm 使用体验的主要途径。
- 注册表（registry）-- 是一个巨大的数据库，保存了每个包（package）的信息。
- 命令行工具 (CLI) -- CLI 通过命令行或终端运行。开发者通过 CLI 与 npm 打交道。

npm 已经集成到 nodejs 上了，安装 nodejs 之后自动带有 npm 功能，因此，不再需要安装 npm。我们可以通过输入 "npm -v" 来测试是否成功安装，出现版本提示表示安装成功。

9.6.2 NPM包安装模式

NPM的包安装分为本地安装（local）、全局安装（global）两种。

两者区别：

本地安装：package会被下载到当前所在目录，也只能在当前目录下使用。

全局安装：package会被下载到到特定的系统目录下，安装的package能够在所有目录下使用。

语法：

```
本地安装：npm install<Module Name>
全局安装：npm install<Module Name> -g
```

示例：

```
npm install grunt-cli          //  本地安装grunt命令行工具
```

运行以上命令，就会在当前目录下安装grunt-cli。安装结束后，当前目录下会多出一个node_modules目录，grunt-cli就安装在里面。

上面已经安装了grunt-cli，然后我们跑到其他目录下面运行如下命令：grunt

控制台提示grunt命令不存在，为什么呢？因为上面只是进行了本地安装，grunt命令只能在对应安装目录下使用。

如果需要在所有目录下使用，采用全局安装。

示例：

```
npm install grunt-cli -g
```

运行以上代码后，在所有目录下都可以使用grunt命令了。

9.6.3 NPM包管理

npm的包管理命令是使用频率最高的，所以也是我们需要牢牢记住并熟练使用的。主要包括：安装、卸载、更新、查看、搜索、发布等。

```
安装最新版本模块：npm install<Module Name>
安装某个版本的模块：npm install<Module Name>@"version"  //  version为版本号
卸载模块：npm uninstal<Module Name>;
卸载某个版本的模块：npm uninstall<Module Name>@"version"  //  version为版本号
查看当前目录安装了哪些模块：npm ls
查看特定某个模块的安装目录、版本：npm ls <Module Name>
查看特定某个模块的详细信息：npm info <Module Name>
更新模块：npm update <Module Name>
搜索模块：npm search <Module Name>
```

9.7 课程总结

- 什么是网络
- http协议
- http模块
- path模块
- url模块
- NPM

9.8 实训

1.使用http模块搭建一个web服务。需求如下：

1）通过http模块的createServer方法来创建服务，并根据请求报文(req)来获取客户端发送过来的信息做出响应；

2）响应报文(res)能够将我们想传递给客户端的信息发送出去；

3）通过listen方法来监听一个web服务的地址和端口(这个地址和端口是提供给客户端访问或者请求数据用的)。

2.使用path模块来获得一个文件“path.js”的绝对路径。

3.通过url模块获得一个url(http://127.0.0.1/index.html?name=Harrison&id=8)请求参数的 id值。