http

> 本文对Node的内置模块http进行介绍，包括该模块的基本情况和简单使用。

1.0 模块简介

http是Node的内置核心模块，包含了对HTTP处理的封装。

在Nodejs文件中可以直接在代码里通过 `var http = require("http")` 的方式来进行加载，该模块主要用来处理客户端HHTP请求以及服务器端的响应。在传统的HTTP服务器可能会使用 `Apache` 、 `Nginx` 或 `IIS` 之类的服务器端软件来处理，但在Node中并不需要这么复杂，我们使用它内置的http模块就可以非常方便的来构建服务器而且稳定可靠(**Node中的HTTP服务器继承自TCP服务器的 `net` 模块，它能够与多个客户端保持连接，因为其采用事件驱动的形式而并不会为每个连接都创建额外的线程，这保证了服务器的低内存占用率以实现高并发**)。

我们可以非常方便的使用http模块来创建服务器或者是发起客户端网络请求。下面给代码示例：

创建Node服务器

```
//备注：文件名为server.js
//001 引入Node内置的http模块
var http = require("http");

//002 创建http服务器
var httpServer = http.createServer(function(request,response){

    //设置响应头信息
    response.writeHead(200,{
        "Content-type":"text/plain;charset=utf-8",
    })

    //设置具体的响应信息
    response.write("Hi! Nice to meet u ...\n\n");
    response.write("这是响应的信息01---\n");
    response.write("这是响应的信息02---\n");
    response.write("这是响应的信息03---\n");

    //响应结束(end)
    response.end("这是响应的信息04---end");
})

//003 开启服务监听
httpServer.listen(3000,"127.0.0.1",function(){
    console.log("开启服务监听：3000端口");
})
```

运行这段代码(在命令中通过node server.js运行)，终端打印 开启服务监听：3000端口 信息。
在浏览器中访问http://127.0.0.1:3000/页面将显示下面的内容：

```HTML
Hi! Nice to meet u ...

这是响应的信息01---
这是响应的信息02---
这是响应的信息03---
这是响应的信息04---end
```

发起HTTP网络请求

```
//001 导入http模块
var http = require("http");

//002 声明变量(组织数据)
var responseData = "";
var options = {
    "host":"127.0.0.1", //请求的主机地址
    "port":"3000",      //请求的端口号
    "method":"get"      //请求的方法
}
//003 创建并发起Http网络Get请求
http.request(options,function(response){

    //事件监听：接收服务器端返回的数据(响应数据)
    response.on("data",function(data){
        responseData += data;
    })

    //事件监听：如果接收完成那么就打印服务器返回的所有数据
    response.on("end",function(){
        console.log("服务器端响应完成，接收到的数据：");
        console.log(responseData);
    })
}).end();
```

上面的代码通过http模块中的 http.request 方法创建并发起一个网络请求，并监听服务器的响应，当接收完服务器返回的响应数据之后打印并显示，给出执行情况。

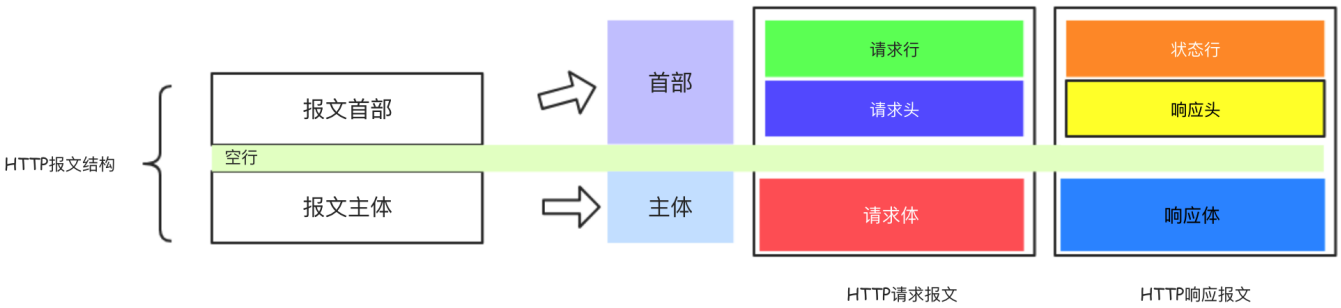
```HTML
wendingding:node wendingding$ node request.js
服务器端响应完成，接收到的数据：
Hi! Nice to meet u ...

这是响应的信息01---
这是响应的信息02---
这是响应的信息03---
这是响应的信息04---end
```

2.0 HTTP报文

HTTP全称 `HyperText Transfer Protocol` ，即超文本传协议，属于应用层协议构建于TCP协议之上。

HTTP协议规定了客户端和服务器端之间应该如何进行通信。在 请求-响应模型 中，请求是客户端向服务器端索要数据或服务的过程，响应是服务器端把数据返回给客户端(为客户端提供服务)的过程，我们把它们在通信过程中的消息内容称为HTTP报文，下面简单介绍HTTP报文的结构(也可以参考这篇文章)。



HTTP请求报文结构

- 请求行 请求的方法和协议等信息
- 请求头 客户端以及请求本身的描述信息
- 请求体 提交给服务器端的参数(GET请求没有请求体信息)

HTTP响应报文结构

- 状态行 请求的状态码
- 响应头 服务器端以及对响应本身的描述信息
- 响应体 服务器返回给客户端的具体数据(`JSON/XML/Other`)。

为了方便理解，这里我们使用命令行工具中的curl来发起网络请求并打印报文详情。

```bash
wendingding:node wendingding$ curl -v 127.0.0.1:3000
* Rebuilt URL to: 127.0.0.1:3000/
*   Trying 127.0.0.1...
* Connected to 127.0.0.1 (127.0.0.1) port 3000 (#0)
> GET / HTTP/1.1
> Host: 127.0.0.1:3000
> User-Agent: curl/7.49.1
> Accept: */*
>
< HTTP/1.1 200 OK
```

```
< Content-type: text/plain;charset=utf-8
< Date: Thu, 29 Nov 2018 03:05:54 GMT
< Connection: keep-alive
< Transfer-Encoding: chunked
<
* Connection #0 to host 127.0.0.1 left intact
Hi ! Nice to meet u ~
```

备注 HTTP协议采用的是请求-响应模式，基本上以一问一答的方式来实现服务，需要注意虽然HTTP服务基于TCP会话实现但其本身却没有会话的特点且HTTP协议传递的消息都是明文的。

3.0 服务端核心方法

createServer方法

作用 创建HTTP服务器。
语法 `var server = http.createServer([ RequsetListener ])`
参数 RequsetListener　　可选的函数类型 | 用于指定当接收到客户端请求时执行的回调函数。
展开
声明 `function RequsetListener(request,response){//...函数体}`
形参

- request　　**http.IncomingMessage**对象 | 包含客户端请求信息。
- response　　**http.ServerResponse**对象　| 包含服务器响应相关的信息和方法。

第一个参数 → request的核心成员

`request.method`	请求方法。
`request.url`	请求的路径。
`request.headers`	请求头信息(对象)。
`request.rawHeaders`	接收到的原始请求头信息。
`request.httpVersion`	请求使用的HTTP协议版本。

第二个参数 → response的核心成员

`response.finished`	响应是否已完成(默认 `false`)。
`response.statusCode`	隐式响应头返回的状态码。
`response.statusMessage`	隐式响应头返回的状态信息。
`response.getHeaders()`	获取所有响应头信息(浅拷贝)。
`response.getHeader(name)`	读取指定的响应头信息。
`response.getHeaderNames()`	获取响应头信息字段数组。
`response.removeHeader(name)`	删除指定的响应头信息。

`response.setHeader(name, value)`	设置响应头信息同 `writeHead` 。
`response.setTimeout(msecs, [callback])`	设置 socket 的超时时间。
`response.write(chunk,[encoding],[callback])`	设置响应体数据。
`response.end([data],[encoding],[callback])`	设置响应体数据(结束)。
`response.writeHead(statusCode,[msg],[headers])`	设置响应头信息，优先级更高。

在 `createServer` 回调函数中两个参数分别是请求对象和响应对象，其中请求对象封装了对TCP连接的读操作，而响应对象则封装了对底层连接的写操作。这里做深入的展开：

当接收到客户端发起的网络请求后，HTTP请求报文的头部将通过模块内的 `http_parser` 进行解析，在解析的过程中，请求行(第一行: `GET / HTTP/1.1`)被分解为 `method(GET)、url(/)、httpVersion(1.1)` 属性，而请求头中的信息被保存到 `headers` 属性。

如果客户端请求中存在请求体(参数)，那么可以通过 `url` 模块的parse方法来解析路径获取参数。

```
//001 引入Node内置的http模块
var url = require("url");
var http = require("http");

//002 创建http服务器
var httpServer = http.createServer(function(request,response){

    //使用url模块把请求路径解析为对象
    var urlObj = url.parse(request.url,true);

    //打印请求对象中的核心属性
    console.log("method " + request.method);
    console.log("url " + request.url);
    console.log("query ",urlObj.query);
    console.log("httpVersion " + request.httpVersion);
    console.log("headers ",request.headers);

    //获取客户端提交的参数(请求体信息)
     //设置响应头信息
     response.writeHead(200,{
        "Content-type":"text/plain;",
    })

    //设置响应信息
    response.end("Hi! Nice to meet u ~");

}).listen(3000,"127.0.0.1",function(){
    console.log("开启服务监听: 3000端口");
})
```

在命令行窗口中通过 `node` 命令来执行，下面列出打印结果。

```
wendingding:node wendingding$ node server.js
开启服务监听: 3000端口
method GET
url     /?username=wendingding&password=123
```

```
query  { username: 'wendingding', password: '123' }
httpVersion 1.1

headers
{ 'host': '127.0.0.1:3000',
  'connection': 'keep-alive',
  'cache-control': 'max-age=0',
  'upgrade-insecure-requests': '1',
  'user-agent': 'Mozilla/5.0 AppleWebKit/537.36 Chrome/70.0.3538.102 Safari/537.36',
  'accept': 'text/html,application/xhtml+xml,application/xml;q=0.9,image/apng,*/*;q=0.8',
  'accept-encoding': 'gzip, deflate, br',
  'accept-language': 'zh-CN,zh;q=0.9,en;q=0.8',
  'cookie': 'io=6oIwtImAumUxvtIvAACD'
}
```

我们可以通过响应对象（ `response` ）来设置响应头信息以及构建响应体。

响应对象的 `setHeader` 方法和 `writeHead` 方法都能够设置响应头信息，它们的区别在于**只有当调用writeHead方法后，通过setHeader设置(可以调用N次)的信息才会被写入到连接(响应头)中**。

响应对象的 `write` 方法和 `end` 方法均能够用来构建响应体信息，它们的区别在于 `end` 方法执行的时候会先调用内部的 `write` 方法来发送数据，然后发送信号告知服务器本次响应结束，响应结束后，HTTP服务器可能会将当前连接直接用于后面的请求或者是关闭网络连接。

注意点 设置响应头信息需要在write和end方法前，响应结束后应该调用end方法结束请求，否则客户端将一直处于等待状态。

listen方法

作用 开启服务器监听。
语法 `http.createServer().listen(port,[host],[backlog],[callBack])`
参数

- port　　指定需要监听的端口号。
- host　　指定需要监听的地址，省略表示监听所有的客户端连接。
- backlog　　指定允许客户端连接的最大数量，默认511。
- callBack　　指定 `listening` 事件触发的回调函数(没有任何参数)。

writeHead方法

作用 设置响应头信息。
语法 `response.writeHead(statusCode,[msg],[headers])`
参数

- statusCode　　响应状态吗，譬如200。
- msg　　响应状态信息，譬如 `Not found` 。
- headers　　具体的响应头信息(以`key:value` 组织成对象)。

4.0 客户端核心方法

HTTP客户端的处理方式同服务器端的处理方式几乎一致，不同在于服务器端主要设置响应头和构建响应体信息，而客户端主要设置请求信息(请求头和请求体)，它本身其实就是服务器端服务模型的另一部分。我们可以使用 `request` 方法来发起一个网络请求，或者也可以直接使用 `get` 方法来快速的发起一个get请求，其结构同Ajax异步发送网络请求基本一致。

request方法

作用 创建并发送网络请求。
语法 `http.request(url,[ options ],[ callBack ]) | http.request(options,[ callBack ])`
参数

- options　　请求的配置对象。
- callBack　　获取服务器端响应时执行的函数，参数为响应对象。

> **Options主要配置项**
> ❏ host　　服务器的域名或IP地址，默认为 `localhost` 。
> ❏ hostname　　服务器的名称。
> ❏ port　　服务器端口，默认为 `80` 。
> ❏ method　　请求方法，默认为 `GET` 。
> ❏ path　　请求路径，默认为 `/` 。
> ❏ agent　　用于指定HTTP代理。
> ❏ headers　　用于指定客户端的请求头信息。

```
//001 导入http模块
var http = require("http");

//002 创建并发起Http网络Get请求
var httpRequest = http.request({
    "host":"127.0.0.1", //请求的主机地址
    "port":"3000",      //请求的端口号
},function(res){

    //获取响应对象中的信息
    console.log("statusCode ",res.statusCode);
    console.log("响应头信息 ",res.headers);
    res.on("data",function(data){
        console.log("响应体数据 ==> ",data.toString("utf8"));
    })
})

//004 结束请求
httpRequest.end();
```

在命令行工具中通过 `node` 命令来发起网络请求，并打印服务器返回的响应信息。

BASH

```
wendingding:node wendingding$ node request.js
statusCode  200
响应头信息  { 'content-type': 'text/plain;',
  date: 'Thu, 29 Nov 2018 07:53:36 GMT',
```

```
  connection: 'close',
  'transfer-encoding': 'chunked' }
响应体数据 ==>  Hi! Nice to meet u ~
```

5.0 事件

为了方便应用层的使用，HTTP服务器和客户端都抽象了一些事件，这些事件都能够使用 `on` 方法来进行监听，不同的事件对应请求或响应的不同阶段。

HTTP服务事件

connection	当客户端和服务器建立连接的时候触发。
request	在请求发送到服务器端并解析出请求头后触发。
close	当调用close方法停止接受新连接已有连接都断开的时候触发。
connect	当客户端发起CONNECT请求(代理)的时候触发。
timeout	当服务器超时的时候触发(可以通过 `server.setTimeOut` 来设置)。

HTTP请求事件

timeout	当客户端请求超时的触发。
abort	当请求已被客户端终止时触发。
response	当接收到服务器响应的时候触发。
socket	当底层连接池中建立的连接分配给当前请丢对象时触发。
connect	当客户端发起CONNECT请求时，如果服务器端返回200则触发。

http模块中事件的监听和触发比较恶心，这里简单在下面列出具体的情况。

HTML

```
如果请求成功，则以下事件会被依次触发：
    ① 'socket' 事件。
    ② 'response' 事件。
        [1] res 对象的 'data' 事件（多次，若响应体为空，则不触发）。
        [2] res 对象的 'end' 事件。
    ③ 'close' 事件。

如果连接出错，则以下事件会被依次触发：
    ① 'socket' 事件。
    ② 'error'  事件。
    ③ 'close'  事件。

如果连接成功之前调用 req.abort()，则以下事件会被依次触发：
    ① 'socket' 事件。
        (此时调用 req.abort())
    ② 'abort' 事件。
    ③ 'close' 事件。
    ④ 'error' 事件并带上错误信息 'Error: socket hang up' 和错误码 'ECONNRESET'。
```

如果响应接收到之后调用 req.abort()，则以下事件会被依次触发：

```
① 'socket' 事件。
② 'response' 事件。
    [1] res 对象的 'data' 事件（多次）。
    (此时调用 req.abort())
③ 'abort' 事件。
④ 'close' 事件。
    res 对象的 'aborted' 事件。
    res 对象的 'end' 事件。
    res 对象的 'close' 事件。
```