Stream

Http协议讲解

HTTP协议解释

Node对http协议的抽象。抽象出了四个类

---

http.ClientRequst(客户端请求)

http.get(url,callback)-request的快捷方法

http.request(options,callback)

---

http.Server

http.createServer可以创建 一个 http.Server实例。

listen([port][, hostname][, backlog][, callback])

request事件

---

http.ServerResponse

statusCode(状态码操作)

设置状态码
    状态代码有三位数字组成,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、说明:

    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,eg:输入了错误的 URL
    500 Internal Server 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

setHeader(响应头操作)

使用方法:
setHeader(name, value)

常用头列表:

Expires

set-cookie

Cache-Control

Last-Modified

Location
    Location 响应报头域用于重定向接受者到一个新的位置。Location 响应报头域常用在更换域名的   (需要结合响应状态码使用 302)

response.write(chunk[, encoding][, callback])--(响应内容操作)

如果这个方法调用在setHeader之前，那么就会设置默认header，并且发送chunk。
这个方法可以多次调用。

response.end([data][, encoding][, callback])(响应内容操作)

这个方法调用后，就会结束本次响应。
和response.write的区别是
response.end(data)功能等于 response.write(data),response.end();

http.IncomingMessage

htpp.Server对象创建的对象。

headers

请求响应头的kye-value对

```
// Prints something like:
//
// { 'user-agent': 'curl/7.22.0',
//   host: '127.0.0.1:8000',
//   accept: '*/*' }
console.log(request.headers);
```

method

只读，获取请求的方法类型：'GET', 'DELETE'.

statusCode

响应的状态码

url

```
请求的url的字符串
例如：请求是：
GET /status?name=ryan HTTP/1.1\r\n
Accept: text/plain\r\n
\r\n

那么响应式：

'/status?name=ryan'
```

有事件

- data
- end

常用方法

- http.createServer(callback(req,res))-创建服务器

DNS

- lookup

Path

```
处理文件和目录的工具方法。
```

path.dirname(path)

```
path.dirname(path)
    返回路径的目录

    path.dirname('/foo/bar/baz/asdf/quux')
    // returns '/foo/bar/baz/asdf'
```

path.extname(path)

```
返回路径的后缀
path.extname('index.html')
// returns '.html'

path.extname('index.coffee.md')
// returns '.md'
```

path.parse

```
path.parse('/home/user/dir/file.txt')
// returns
// {
//    root : "/",
//    dir : "/home/user/dir",
//    base : "file.txt",
//    ext : ".txt",
//    name : "file"
// }
```

Query Strings

```
解析和处理查询字符串的
const querystring = require('querystring');
```

querystring.parse(str[, sep[, eq[, options]]])

For example, the query string 'foo=bar&abc=xyz&abc=123' is parsed into:

```
{
  foo: 'bar',
  abc: ['xyz', '123']
}
```

URL

URL字符串和URL对象

```
url字符串是:
'http://user:pass@host.com:8080/p/a/t/h?query=string#hash'

解析后是:

┌──────────────────────────────────────────────────────────────────────────────
──┐
|                                    href
|
├──────────┬┬────────────┬──────────────────┬──────────────────────────────┬─────
──┤
| protocol ||   auth     |      host        |           path               | hash
|
|          ||            ├──────────┬──────┼──────────┬───────────────────┤
|
|          ||            | hostname | port | pathname |     search        |
|
|          ||            |          |      |          ├─┬─────────────────┤
|
|          ||            |          |      |          | |    query        |
|
"  http:   // user:pass @ host.com : 8080   /p/a/t/h  ?  query=string
#hash "
|          ||            |          |      |          | |                 |
|
└──────────┴┴────────────┴──────────┴──────┴──────────┴ ┴─────────────────┴─────
──┘
```

URL的定义和

```
$ node
> require('url').parse('/status?name=ryan')
{
  href: '/status?name=ryan',
  search: '?name=ryan',
  query: 'name=ryan',
  pathname: '/status'
}
```

- urlObj
- parse

Utilities

提示一下废除和新特性。

网络相关信息

Cookie

Cookie 指某些网站为了辨别用户身份、进行session跟踪而储存在用户本地终端上的数据

Cookie是由服务器端生成，发送给User-Agent（一般是浏览器），浏览器会将Cookie的key/value保存到某个目录下的文本文件内，下次请求同一网站时就发送该Cookie给服务器（前提是浏览器设置为启用cookie）。Cookie名称和值可以由服务器端开发自己定义

session和cookie的区别

1．由于HTTP协议是无状态的协议，所以服务端需要记录用户的状态时，就需要用某种机制来识具体的用户，这个机制就是Session.典型的场景比如购物车，当你点击下单按钮时，由于HTTP协议无状态，所以并不知道是哪个用户操作的，所以服务端要为特定的用户创建了特定的Session，用用于标识这个用户，并且跟踪用户，这样才知道购物车里面有几本书。这个Session是保存在服务端的，有一个唯一标识。在服务端保存Session的方法很多，内存、数据库、文件都有。集群的时候也要考虑Session的转移，在大型的网站，一般会有专门的Session服务器集群，用来保存用户会话，这个时候 Session 信息都是放在内存的，使用一些缓存服务比如Memcached之类的来放 Session。

2．思考一下服务端如何识别特定的客户？这个时候Cookie就登场了。每次HTTP请求的时候，客户端都会发送相应的Cookie信息到服务端。实际上大多数的应用都是用 Cookie 来实现Session跟踪的，第一次创建Session的时候，服务端会在HTTP协议中告诉客户端，需要在 Cookie 里面记录一个 Session ID，以后每次请求把这个会话ID发送到服务器，我就知道你是谁了。有人问，如果客户端的浏览器禁用了 Cookie 怎么办？一般这种情况下，会使用一种叫做URL重写的技术来进行会话跟踪，即每次HTTP交互，URL后面都会被附加上一个诸如 sid=xxxxx 这样的参数，服务端据此来识别用户。

3．Cookie其实还可以用在一些方便用户的场景下，设想你某次登陆过一个网站，下次登录的时候不想再次输入账号了，怎么办？这个信息可以写到Cookie里面，访问网站的时候，网站页面的脚本可以读取这个信息，就自动帮你把用户名给填了，能够方便一下用户。这也是Cookie名称的由来，给用户的一点甜头。
所以，总结一下：
Session是在服务端保存的一个数据结构，用来跟踪用户的状态，这个数据可以保存在集群、数据库、文件中；
Cookie是客户端保存用户信息的一种机制，用来记录用户的一些信息，也是实现Session的一种方式。

缓存机制

Cache-Control与Expires

Cache-Control与Expires的作用一致，都是指明当前资源的有效期，控制浏览器是否直接从浏览器缓存　取数据还是重新发请求到服务器取数据。只不过Cache-Control的选择更多，设置更细致，如果同时设置的　话，其优先级高于Expires。

Last-Modified/ETag与Cache-Control/Expires

配置Last-Modified/ETag的情况下，浏览器再次访问统一URI的资源，还是会发送请求到服务器询问文件　是否已经修改，如果没有，服务器会只发送一个304回给浏览器，告诉浏览器直接从自己本地的缓存取数据；　如果修改过那就整个数据重新发给浏览器

缓存机制列表