Node.js GET/POST 请求

获取 GET 请求的内容

GET 请求直接被嵌入在路径中，URL 是完整的请求路径，包括了?后面的部分，node.js 中 url 模块中的 parse 函数提供解析后面的内容作为 GET 请求的参数功能

通过 url.parse 将 req.url 解析成 url 对象，url.parse 为 true 会将 Url 对象 query 解析为{key:value}

```
res.end(util.inspect(url.parse(req.url, true)));
```

获取 URL 的参数

```
var params = url.parse(req.url, true).query;
```

获取 POST 请求内容

POST 请求的内容全部的都在请求体中,node.js 默认是不会解析请求体的，当你需要的时候，需要手动来做。

```
// 定义一个变量,存储 post 传入的参数
    let post = '';
    // 监听 data 事件，获取参数
    req.on('data', chunk => {
        post += chunk;
    });
    // req 监听 end 事件，解析 post 提交的参数，返回给客户端
    req.on('end', () => {
        console.log(post);
        // 解析 post 返回{key: value}
        post = querystring.parse(post);
        console.log(post);
        // 将对象转换成字符串
        console.log(util.inspect(post));
        res.end(util.inspect(post));
    })
```

Node.js 创建 web 服务器流程

1. 创建服务器

```
http.createServer((req, res) => {...} )
```

2. 解析请求

```
var pathname = url.parse(req.url).pathname;
```

3. 根据路径读取本地文件内容

```
fs.readFile(pathname.substr(1), (err, data) => { ...})
```

4. 错误响应操作

```
  res.writeHead(404, {'Content-Type': 'text/html'});
            res.end('<h1>服务器未访问<h1>');
```

5. 正确响应操作

```
 res.writeHead(200, {'Content-Type': 'text/html; charset=utf-8'});
            res.write(data.toString());
            res.end();
```

Node.js Express 框架

Express 特性

Express 框架核心特性：

1.可以设置中间件来响应 HTTP 请求。

2.定义了路由表用于执行不同的 HTTP 请求动作。

3.可以通过向模板传递参数来动态渲染 HTML 页面。

安装 Express

```
$ cnpm install express --save

$ cnpm install body-parser --save 用于处理 JSON, Raw, Text 和 URL 编码的数据

$ cnpm install cookie-parser --save 解析 Cookie 的工具
```

```
$ cnpm install multer --save 用于处理 enctype="multipart/form-data"（设置表单的 MIME 编码）的表单数据
```

请求和响应

Request 对象 - request 对象表示 HTTP 请求，包含了请求查询字符串，参数，内容，HTTP 头部等属性

Response 对象 - response 对象表示 HTTP 响应，即在接收到请求时向客户端发送的 HTTP 响应数据。

get: 处理 get 路由里的参数

```
var response = {
    "first_name":req.query.first_name,
    "last_name":req.query.last_name
};
```

处理 post 路由里的参数

注意设置

```
var bodyParser = require('body-parser');

// 创建 application/x-www-form-urlencoded 编码解析
var urlencodedParser = bodyParser.urlencoded({ extended: false })
```

不然 req.body 会是 undefined

```
// 输出 JSON 格式
var response = {
    "first_name":req.body.first_name,
    "last_name":req.body.last_name
};
```

处理表单上传文件

注意设置

```
var bodyParser = require('body-parser');
var multer  = require('multer');

app.use('/public', express.static('public'));
app.use(bodyParser.urlencoded({ extended: false }));
app.use(multer({ dest: '/tmp/'}).array('image'));
```

不然 req.files 是 undefined

```
   console.log(req.files[0]);  // 上传的文件信息

   var des_file = __dirname + "/" + req.files[0].originalname;
   fs.readFile( req.files[0].path, function (err, data) {
        fs.writeFile(des_file, data, function (err) {
         if( err ){
              console.log( err );
         }else{
               response = {
                   message:'File uploaded successfully',
                   filename:req.files[0].originalname
               };
          }
          console.log( response );
          res.end( JSON.stringify( response ) );
       });
   });
})
```

Express DEMO

```
var express = require('express');
var app = express();
var fs = require("fs");
// post 请求需要引入的模块
var bodyParser = require('body-parser');
// 上传文件需要引入的模块
var multer  = require('multer');

// 创建 application/x-www-form-urlencoded 编码解析
// var urlencodedParser = bodyParser.urlencoded({ extended: false })
app.use(bodyParser.urlencoded({ extended: false }));
```

```
// 使用上传文件模块
app.use(multer({ dest: '/tmp/'}).array('image'));
```

```
const port = process.env.HTTP_PORT || 3000;
const host = process.env.HTTP_IP || '0.0.0.0';
```

```
// 设置静态文件
app.use('/public', express.static('public'));
```

```
// 设置主页路由
app.get('/index.html', (req, res) => {
    res.sendFile(__dirname + '/' + 'index.html');
})
// 设置表单 get 请求路由
app.get('/process_get', (req, res) => {
    // 获取参数
    var response = {    //get 请求通过 query 获取提交的数据
        "first_name": req.query.first_name,
        "last_name": req.query.last_name
    };
    res.end(JSON.stringify(response));
});
// 设置表单 post 请求路由
app.post('/process_post', function (req, res) {

    // 输出 JSON 格式
    var response = {
        "first_name":req.body.first_name,
        "last_name":req.body.last_name
    };
    console.log(response);
    res.end(JSON.stringify(response));
 })
// 文件上传路由
app.post('/file_upload', function (req, res) {

    console.log(req.files[0]);  // 上传的文件信息

    var des_file = __dirname + "/" + req.files[0].originalname;
    fs.readFile( req.files[0].path, function (err, data) {
         fs.writeFile(des_file, data, function (err) {
          if( err ){
```

```
                console.log( err );
            }else{
                  response = {
                      message:'File uploaded successfully',
                      filename:req.files[0].originalname
                  };
             }
             console.log( response );
             res.end( JSON.stringify( response ) );
          });
    });
 })
// 设置监听端口
app.listen(8081, () => {
   console.log( `express app started at http://${host}:${port}` );
});
```

Decache 使用

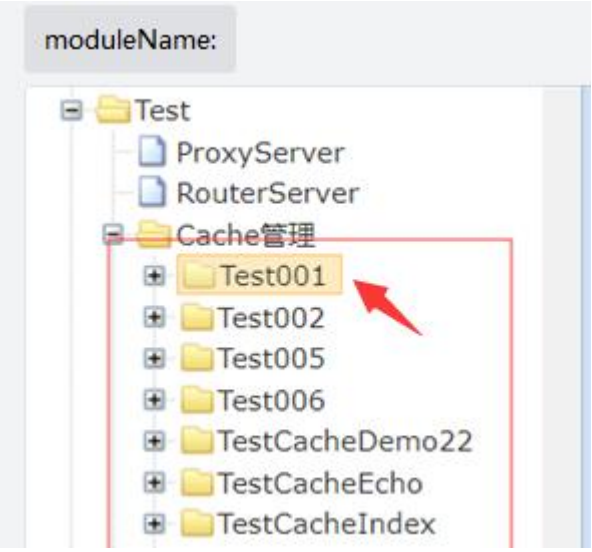
配置

```
const cacheHelper = new DCache({
    DCacheServerTarget: "DCache.TestProxyServer.ProxyObj@tcp -h 172.16.8.147 -t 60000 -p 28711",
    moduleName: "TestHelloDcacheDemo"
});
```

分别对应

proxy 的

moduleName：



DCache api

getString 根据 key 查询 value

```
        @param key 的值
```

getStringWithVer 根据 key 查询对应的结构

```
      @param keyItem, key 的值
ret.data = { value: 'fixed_value', ver: 2 }
```

getStringBatch 根据 key 查询对应的结构

```
       @param keyItem, key 的值
 ret.data.vtValue.value =
    [
  {
    keyItem: 'a',
    value: '1',
    ret: 0,
    ver: 2,
```

```
    _classname: 'DCache.SKeyValue'
  }]
```

setString 设置 key-value

```
      @param keyItem, key 的值
@param value, 要设置的数据
```

setStringBatch 批量设置 key-value

```
     @param keyItem, key 的值
@@param value, 查询结构，返回数据
```

setStringEx 设置 key-value

```
     @param keyItem, key 的值
@param value, 要设置的数据
@param ver，数据版本（1）
@param dirty，是否脏数据
@param expireTimeSecond，设置数据过期的绝对时间，以秒为单位。
CacheServer 将根据这个时间自动淘汰过期数据。如果数据没有过期的概念，
请将此参数设为 0
```

delString 删除 key-value

```
     @param keyItem, key 的值
```