### 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) => {...} )

1. 解析请求

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

1. 根据路径读取本地文件内容

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

1. 错误响应操作

  res.writeHead(404, {'Content-Type': 'text/html'});

            res.end('<h1>服务器未访问<h1>');

1. 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

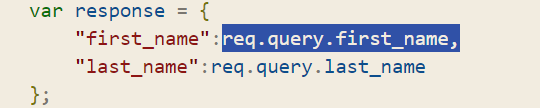
$ cnpm install multer --save 用于处理 enctype="multipart/form-data"（设置表单的MIME编码）的表单数据

#### 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

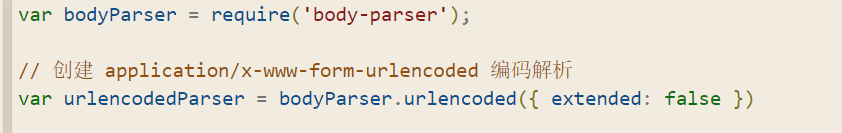
Response 对象 - response 对象表示 HTTP 响应，即在接收到请求时向客户端发送的 HTTP 响应数据。

#### get: 处理get路由里的参数

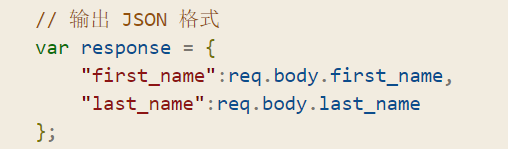


#### 处理post 路由里的参数

注意设置

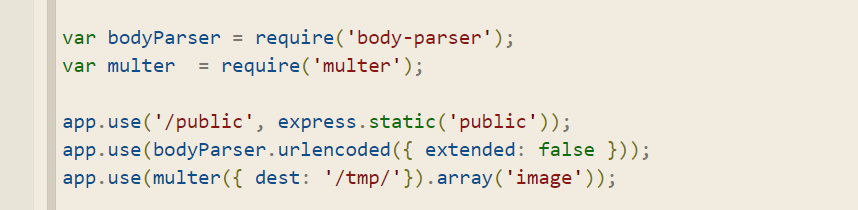


不然req.body 会是 undefined



#### 处理表单上传文件

注意设置



不然 req.files是undefined



#### 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

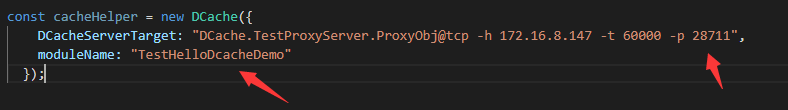
app.listen(8081, () => {

   console.log( `express app started at http://${host}:${port}`);

});

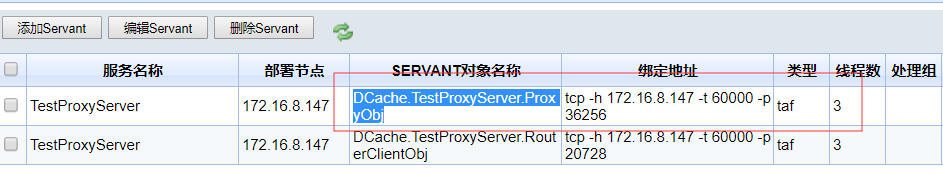
### Decache使用

#### 配置

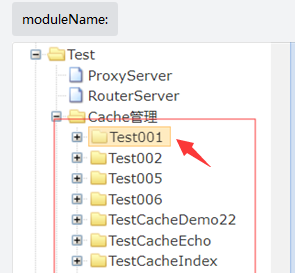


分别对应

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