link: null
title: 珠峰架构师成长计划
description: Express 是一个小巧且灵活的 Node.js Web应用框架，它有一套健壮的特性，可用于开发单页、多页和混合Web应用。
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=117 sentences=300, words=1920

1. Express 介绍

Express 是一个小巧且灵活的 Node.js Web应用框架，它有一套健壮的特性，可用于开发单页、多页和混合Web应用。

2. Express 的应用

npm安装

```
npm install express
```

创建http服务

```

var express = require('express');

var app = express();

app.listen(3000);
```

根据请求路径来处理客户端发出的GET请求

- 第一个参数path为请求的路径
- 第二个参数为处理请求的回调函数

```
app.get(path,function(req, res));
```

get方法使用:

```
const express = require('express');
const app = express();
app.get('/hello',function(req,res){
    res.end('hello');
});
app.get('/world',function(req,res){
    res.end('world');
});
app.get('*',function(req,res){
    res.setHeader('Content-Type','text/plain;charset=utf8');
    res.end('Not Found');
});
app.listen(3000);
```

get方法实现:

```
let url = require('url');
let express = function () {
    let app = function (req, res) {
        let {pathname} = url.parse(req.url, true);
        let method = req.method.toLowerCase();
        for (let i = 0; i < app.routes.length; i++) {
            let {path, method, handler} = app.routes[i];
            if ((path == pathname || path == "*") && method == req.method.toLowerCase()) {
                return handler(req, res);
            }
        }
        res.end(`CANNOT ${req.method} ${req.url}`);
    }
    app.routes = [];
    app.listen = function (port) {
        require('http').createServer(app).listen(port);
    }
    app.get = function (path, handler) {
        app.routes.push({
            path, handler, method: 'get'
        });
    }
    return app;
}

module.exports = express;
```

根据请求路径来处理客户端发出的POST请求

- 第一个参数path为请求的路径
- 第二个参数为处理请求的回调函数

```
app.post(path,function(req,res));
```

post方法的使用:

```

var express = require('./express');

var app = express();

app.post('/hello', function (req,res) {
   res.end('hello');
});
app.post('*', function (req,res) {
    res.end('post没找到');
});
app.listen(3000);
```

通过linux命令发送post请求

```
curl -X POST http://localhost:3000/hello
```

post的实现：

增加所有请求的方法

```
http.METHODS.forEach(function(method){
        app[method] = function (path, handler) {
            app.routes.push({
                path, handler, method
            });
        }
});
```

监听所有的请求方法，可以匹配所有的HTTP动词。根据请求路径来处理客户端发出的所有请求

- 第一个参数path为请求的路径
- 第二个参数为处理请求的回调函数

```
app.all(path,function(req, res));
```

all的方法使用：

```
const express = require('express');
const app = express();
app.all('/world',function(req,res){
    res.end('all world');
});
app.listen(3000);
```

```
app.all = function (path, handler) {
        app.routes.push({
            path, handler, method: 'all'
        });
    }
```

中间件就是处理HTTP请求的函数，用来完成各种特定的任务，比如检查用户是否登录、检测用户是否有权限访问等，它的特点是：

- 一个中间件处理完请求和响应可以把相应数据再传递给下一个中间件
- 回调函数的next参数,表示接受其他中间件的调用，函数体中的next(),表示将请求数据继续传递
- 可以根据路径来区分返回执行不同的中间件

中间件的使用方法：

增加中间件

```
var express = require('express');
var app = express();
app.use(function (req,res,next) {
    console.log('全部匹配');
    next();
});
app.use('/water', function (req,res,next) {
    console.log('只匹配/water');
    next();
});
app.get('/water', function (req,res) {
    res.end('water');
});
app.listen(3000);
```

use方法的实现：在路由数组中增加中间件

```
app.use = function (path,fn) {
    if(typeof fn !='function'){
        fn = path;
        path = '/';
    }
    app.routes.push({method:'middle',path:path,fn:fn});
}
```

```
let url = require('url');
let http = require('http');
let express = function () {
    let app = function (req, res) {
        let {pathname} = url.parse(req.url, true);
        let method = req.method.toLowerCase();
        let index = 0;

        function next(err) {
            if (index >= app.routes.length) {
                return res.end(`CANNOT ${method} ${pathname}`);
            }
            let route = app.routes[index++];
            if (route.method == 'middle') {
                if (route.path == '/' || pathname.startsWith(route.path + '/') || route.path == pathname) {
                    route.handler(req, res, next);
                } else {
                    next();
                }
            } else {
                if ((route.path == pathname || route.path == "*") && (route.method == req.method.toLowerCase()) || method == 'all') {
                    return route.handler(req, res);
                } else {
                    next();
                }
            }
        }

        next();
    }
    app.routes = [];
    app.listen = function (port) {
        http.createServer(app).listen(port);
    }
    http.METHODS.forEach(function (method) {
        method = method.toLowerCase();
        app[method] = function (path, handler) {
            app.routes.push({
                path, handler, method
            });
        }
    });
    app.all = function (path, handler) {
        app.routes.push({
            path, handler, method: 'all'
        });
    }
    app.use = function (path, handler) {
        if (typeof handler != 'function') {
            handler = path;
            path = "/";
        }
        app.routes.push({
            method: 'middle',
            path,
            handler
        });
    }

    return app;
}

module.exports = express;
```

错误中间件：next中可以传递错误，默认执行错误中间件

```
var express = require('express');
var app = express();
app.use(function (req,res,next) {
    console.log('过滤石头');
    next('stone is too big');
});
app.use('/water', function (req,res,next) {
    console.log('过滤沙子');
    next();
});
app.get('/water', function (req,res) {
    res.end('water');
});
app.use(function (err,req,res,next) {
    console.log(err);
    res.end(err);
});
app.listen(3000);
```

错误中间件的实现：对错误中间件进行处理

```
let url = require('url');
let http = require('http');
let express = function () {
    let app = function (req, res) {
        let {pathname} = url.parse(req.url, true);
        let method = req.method.toLowerCase();
        let index = 0;

        function next(err) {
            if (index >= app.routes.length) {
                return res.end(`CANNOT ${method} ${pathname}`);
            }
            let route = app.routes[index++];
            if (err) {
                if (route.method == 'middle' && route.handler.length == 4) {
                    route.handler(req, res, next)
                }
            }
            if (route.method == 'middle') {
                if (route.path == '/' || pathname.startsWith(route.path + '/') || route.path == pathname) {
                    route.handler(req, res, next);
                } else {
                    next();
                }
            } else {
                if ((route.path == pathname || route.path == "*") && (route.method == req.method.toLowerCase()) || method == 'all') {
                    return route.handler(req, res);
                } else {
                    next();
                }
            }
        }

        next();
    }
    app.routes = [];
    app.listen = function (port) {
        http.createServer(app).listen(port);
    }
    http.METHODS.forEach(function (method) {
        method = method.toLowerCase();
        app[method] = function (path, handler) {
            app.routes.push({
                path, handler, method
            });
        }
    });
    app.all = function (path, handler) {
        app.routes.push({
            path, handler, method: 'all'
        });
    }
    app.use = function (path, handler) {
        if (typeof handler != 'function') {
            handler = path;
            path = "/";
        }
        app.routes.push({
            method: 'middle',
            path,
            handler
        });
    }

    return app;
}

module.exports = express;
```

- `req.hostname` 返回请求头里取的主机名
- `req.path` 返回请求的URL的路径名
- `req.query` 查询字符串

```

app.get('/',function(req,res){
    res.write(JSON.stringify(req.query))
    res.end(req.path+" "+req.path);
});
```

具体实现：对请求增加方法

```
  req.path = pathname;
  req.query = query;
```

req.params 匹配到的所有路径参数组成的对象

```
app.get('/school/:name/:age', function (req,res) {
    console.log(req.params);
    res.end('water');
});
```

params实现：增加params属性

```
let url = require('url');
let http = require('http');
let express = function () {
    let app = function (req, res) {
        let {pathname, query} = url.parse(req.url, true);
        let method = req.method.toLowerCase();
        let index = 0;
        req.path = pathname;
        req.query = query;

        function next(err) {
            if (index >= app.routes.length) {
                return res.end(`CANNOT ${method} ${pathname}`);
            }
            let route = app.routes[index++];

            if (err) {
                if (route.method == 'middle' && route.handler.length == 4) {
                    route.handler(req, res, next)
                } else {
                    next();
                }
            } else {

                if (route.method == 'middle') {
                    if (route.path == '/' || pathname.startsWith(route.path + '/') || route.path == pathname) {
                        route.handler(req, res, next);
                    } else {
                        next();
                    }
                } else {
                    if (route.paramNames) {
                        let matchers = pathname.match(new RegExp(route.path));
                        if (matchers) {
                            let params = {};
                            for(let i=0;i1];
                            }
                            req.params = params;
                            route.handler(req,res);
                        }else{
                            next();
                        }
                    } else {
                        if ((route.path == pathname || route.path == "*") && (route.method == req.method.toLowerCase()) || method == 'all') {
                            return route.handler(req, res);
                        } else {
                            next();
                        }
                    }
                }
            }
        }

        next();
    }
    app.routes = [];
    app.listen = function (port) {
        http.createServer(app).listen(port);
    }
    http.METHODS.forEach(function (method) {
        method = method.toLowerCase();
        app[method] = function (path, handler) {
            const layer = {path, handler, method};
            if (path.includes(':')) {
                let paramNames = [];
                layer.path = path.replace(/:([^\/]+)/g, function () {
                    paramNames.push(arguments[1]);
                    return '([^\/]+)';
                });
                layer.paramNames = paramNames;
            }
            app.routes.push(layer);
        }
    });
    app.all = function (path, handler) {
        app.routes.push({
            path, handler, method: 'all'
        });
    }
    app.use = function (path, handler) {
        if (typeof handler != 'function') {
            handler = path;
            path = "/";
        }
        app.routes.push({
            method: 'middle',
            path,
            handler
        });
    }
    return app;
}

module.exports = express;
```

参数为要响应的内容,可以智能处理不同类型的数据,在输出响应时会自动进行一些设置，比如HEAD信息、HTTP缓存支持等等

```
res.send([body]);
```

当参数是一个字符串时，这个方法会设置Content-type为text/html

```
app.get('/', function (req,res) {
    res.send('hello world');
});
```

当参数是一个Array或者Object，这个方法返回json格式

```
app.get('/json', function (req,res) {
     res.send({obj:1});
});
app.get('/arr', function (req,res) {
 res.send([1,2,3]);
});
```

当参数是一个number类型，这个方法返回对应的状态码短语

```
app.get('/status', function (req,res) {
    res.send(404);

});
```

send方法的实现：自定义send方法

```
   res.send = function(msg){
            let type = typeof msg;
            if(type == 'object'){
                res.setHeader('Content-Type','application/json');
                msg = JSON.stringify(msg);
            }else if(type == 'number'){
                res.setHeader('Content-Type','application/plain');
                res.status(msg);
                res.end(http.STATUS_CODES[msg]);
            }else{
                res.setHeader('Content-Type','application/html');
                res.end(msg);
            }
        }
```

3. 模板的应用

npm安装ejs

```
 npm install ejs
```

使用ejs模版

```
var express = require('express');
var path = require('path');
var app = express();
app.set('view engine','ejs');
app.set('views',path.join(__dirname,'views'));
app.listen(3000);
```

配置成html格式

```
app.set('view engine','html')
app.set('views',path.join(__dirname,'views'));
app.engine('html',require('ejs').__express);
```

- 第一个参数 要渲染的模板
- 第二个参数 渲染所需要的数据

```
app.get('/', function (req,res) {
    res.render('hello',{title:'hello'},function(err,data){});
});
```

读取模版渲染

```
res.render = function (name, data) {
    var viewEngine = engine.viewEngineList[engine.viewType];
    if (viewEngine) {
        viewEngine(path.join(engine.viewsPath, name + '.' + engine.viewType), data, function (err, data) {
            if (err) {
                res.status(500).sendHeader().send('view engine failure' + err);
            } else {
                res.status(200).contentType('text/html').sendHeader().send(data);
            }
        });
    } else {
        res.status(500).sendHeader().send('view engine failure');
    }
}
```

4. 静态文件服务器

如果要在网页中加载静态文件（css、js、img），就需要另外指定一个存放静态文件的目录，当浏览器发出非HTML文件请求时，服务器端就会到这个目录下去寻找相关文件

```
var express = require('express');
var app = express();
var path = require('path');
app.use(express.static(path.join(__dirname,'public')));
app.listen(3000);
```

配置静态服务器

```
express.static = function (p) {
    return function (req, res, next) {
        var staticPath = path.join(p, req.path);
        var exists = fs.existsSync(staticPath);
        if (exists) {
            res.sendFile(staticPath);
        } else {
            next();
        }
    }
};
```

5. 重定向

redirect方法允许网址的重定向，跳转到指定的url并且可以指定status，默认为302方式。

- 参数1 状态码(可选)
- 参数2 跳转的路径

```
res.redirect([status], url);
```

使用重定向

```
app.get('/', function (req,res) {
    res.redirect('http://www.baidu.com')
});
```

302重定向

```
res.redirect = function (url) {
    res.status(302);
    res.headers('Location', url || '/');
    res.sendHeader();
    res.end();
};
```

6. 接收 post 响应体

安装body-parser

```
 npm install body-parser
```

接收请求体中的数据

```
app.get('/login', function (req,res) {
    res.sendFile('./login.html',{root:__dirname})
});
app.post('/user', function (req,res) {
    console.log(req.body);
    res.send(req.body);
});
app.listen(3000);
```

实现bodyParser

```
function bodyParser () {
    return function (req,res,next) {
        var result = '';
        req.on('data', function (data) {
            result+=data;
        });
        req.on('end', function () {
            try{
                req.body = JSON.parse(result);
            }catch(e){
                req.body = require('querystring').parse(result);
            }
            next();
        })
    }
};
```

源代码地址