目录

[1. Express框架简介及初体验 2](#_Toc2947)

[1.1 Express框架是什么 2](#_Toc21525)

[1.2 Express框架特性 2](#_Toc12623)

[1.3 原生Node.js与Express框架对比之路由 2](#_Toc5505)

[1.4 原生Node.js与Express框架对比之获取请求参数 3](#_Toc21310)

[1.5 Express初体验 4](#_Toc314)

[2. 中间件 5](#_Toc3865)

[2.1 什么是中间件 5](#_Toc30133)

[2.2 app.use中间件用法 6](#_Toc18982)

[2.3 中间件应用 6](#_Toc18743)

[2.4 错误处理中间件 6](#_Toc22342)

[2.3中间件应用 6](#_Toc18951)

[2.4 错误处理中间件 6](#_Toc7907)

[2.5 捕获错误 7](#_Toc19747)

[3. Express请求处理 8](#_Toc24413)

[3.1构建模块化路由 8](#_Toc10097)

[3.2 GET参数的获取 8](#_Toc4573)

[3.3 POST参数的获取 9](#_Toc3539)

[3.4 Express路由参数 10](#_Toc7858)

[3.5 静态资源的处理 10](#_Toc27307)

[4. express-art-template模板引擎 11](#_Toc486)

[4.1 模板引擎 11](#_Toc7408)

[4.2 app.locals 对象 11](#_Toc11918)

[5. 实例 12](#_Toc692)

[5.1 framework/框架入门.js 12](#_Toc6065)

[5.2 framework/中间件.js 12](#_Toc24348)

[5.3 framework/app.use.js 13](#_Toc1442)

[5.4 framework/中间件应用.js 14](#_Toc11247)

[5.5 framework/错误处理中间件.js 15](#_Toc31455)

[5.6 framework/异步函数错误的捕获.js 16](#_Toc23092)

[5.7 framework/构建模块化路由的基础代码.js 16](#_Toc25268)

[5.8 framework/构建模块化路由.js 17](#_Toc29929)

[5.9 framework/如何获取get请求参数.js 17](#_Toc7577)

[5.10 framework/如何获取post请求参数.js 18](#_Toc7758)

[5.11 framework/app.use方法.js 19](#_Toc8658)

[5.12 framework/路由参数.js 20](#_Toc25744)

[5.13 framework/静态资源访问功能.js 20](#_Toc23014)

[5.14 framework/模板引擎.js 20](#_Toc4323)

[5.15 framework/app.locals对象.js 22](#_Toc30687)

# Express框架简介及初体验

## Express框架是什么

Express是一个基于Node平台的web应用开发框架，它提供了一系列的强大特性，帮助你创建各种Web应用。我们可以使用 npm install express 命令进行下载。

## Express框架特性

1. 提供了方便简洁的路由定义方式
2. 对获取HTTP请求参数进行了简化处理
3. 对模板引擎支持程度高，方便渲染动态HTML页面
4. 提供了中间件机制有效控制HTTP请求
5. 拥有大量第三方中间件对功能进行扩展

## 原生Node.js与Express框架对比之路由

1. app**.**on**(**'request'**,** **(**req**,** res**)** **=>** **{**
2. // 获取客户端的请求路径
3. **let** **{** pathname **}** **=** url**.**parse**(**req**.**url**);**
4. // 对请求路径进行判断 不同的路径地址响应不同的内容
5. **if** **(**pathname **==** '/' **||** pathname **==** 'index'**)** **{**
6. res**.**end**(**'欢迎来到首页'**);**
7. **} else** **if** **(**pathname **==** '/list'**)** **{**
8. res**.**end**(**'欢迎来到列表页页'**);**
9. **} else** **if** **(**pathname **==** '/about'**)** **{**
10. res**.**end**(**'欢迎来到关于我们页面'**)**
11. **} else** **{**
12. res**.**end**(**'抱歉, 您访问的页面出游了'**);**
13. **}**
14. **});**
15. // 当客户端以get方式访问/时
16. app**.**get**(**'/'**,** **(**req**,** res**)** **=>** **{**
17. // 对客户端做出响应
18. res**.**send**(**'Hello Express'**);**
19. **});**
20. // 当客户端以post方式访问/add路由时
21. app**.**post**(**'/add'**,** **(**req**,** res**)** **=>** **{**
22. res**.**send**(**'使用post方式请求了/add路由'**);**
23. **});**

## 原生Node.js与Express框架对比之获取请求参数

1. app**.**on**(**'request'**,** **(**req**,** res**)** **=>** **{**
2. // 获取GET参数
3. **let {**query**}** **=** url**.**parse**(**req**.**url**,** **true);**
4. // 获取POST参数
5. **let** postData **=** ''**;**
6. req**.**on**(**'data'**,** **(**chunk**)** **=>** {
7. postData **+=** chunk**;**
8. }**);**
9. req**.**on**(**'end'**,** **()** **=>** {
10. console**.**log**(**querystring**.**parse**(**postData**)**
11. }**));**
12. **});**
13. app**.**get**(**'/'**,** **(**req**,** res**)** **=>** **{**
14. // 获取GET参数
15. console**.**log**(**req**.**query**);**
16. **});**
17. app**.**post**(**'/'**,** **(**req**,** res**)** **=>** **{**
18. // 获取POST参数
19. console**.**log**(**req**.**body**);**
20. **})**

## 1.5 Express初体验

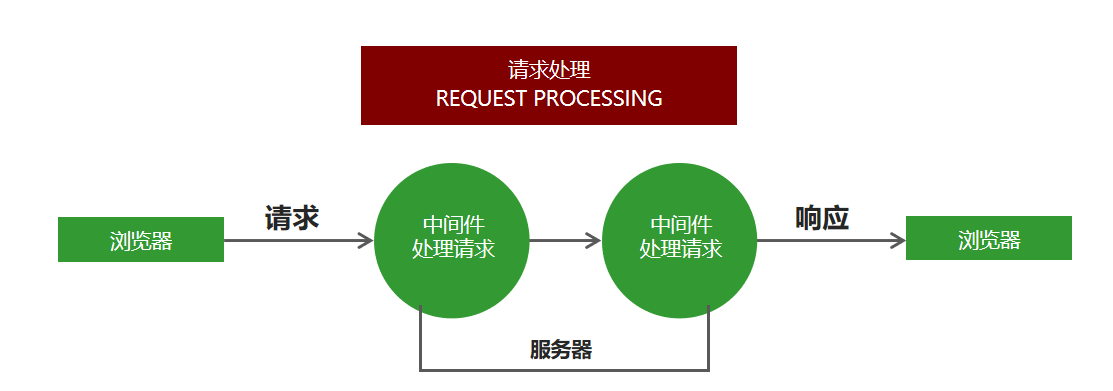
使用Express框架创建web服务器及其简单，调用express模块返回的函数即可。

1. // 引入Express框架
2. **const** express **=** require**(**'express'**);**
3. // 使用框架创建web服务器
4. **const** app **=** express**();**
5. // 当客户端以get方式访问/路由时
6. app**.**get**(**'/'**,** **(**req**,** res**)** **=>** **{**
7. // 对客户端做出响应 send方法会根据内容的类型自动设置请求头
8. res**.**send**(**'Hello Express'**);** // <h2>Hello Express</h2> {say: 'hello'}
9. **});**
10. // 程序监听3000端口
11. app**.**listen**(**3000**);**

# 中间件

## 什么是中间件

中间件就是一堆方法，可以接收客户端发来的请求、可以对请求做出响应，也可以将请求继续交给下一个中间件继续处理。

中间件主要由两部分构成，中间件方法以及请求处理函数。

中间件方法由Express提供，负责拦截请求，请求处理函数由开发人员提供，负责处理请求。

1. app**.**get**(**'请求路径'**,** '处理函数'**)** // 接收并处理get请求
2. app**.**post**(**'请求路径'**,** '处理函数'**)** // 接收并处理post请求

可以针对同一个请求设置多个中间件，对同一个请求进行多次处理。

默认情况下，请求从上到下依次匹配中间件，一旦匹配成功，终止匹配。

可以调用next方法将请求的控制权交给下一个中间件，直到遇到结束请求的中间件。

1. app**.**get**(**'/request'**,** **(**req**,** res**,** next**)** **=>** **{**
2. req**.**name **=** "张三"**;**
3. next**();**
4. **});**
5. app**.**get**(**'/request'**,** **(**req**,** res**)** **=>** **{**
6. res**.**send**(**req**.**name**);**
7. **});**

## app.use中间件用法

app.use 匹配所有的请求方式，可以直接传入请求处理函数，代表接收所有的请求。

1. app**.**use**((**req**,** res**,** next**)** **=>** **{**
2. console**.**log**(**req**.**url**);**
3. next**();**
4. **});**

## 中间件应用

app.use 第一个参数也可以传入请求地址，代表不论什么请求方式，只要是这个请求地址就接收这个请求。

1. app**.**use**(**'/admin'**,** **(**req**,** res**,** next**)** **=>** **{**
2. console**.**log**(**req**.**url**);**
3. next**();**
4. **});**

## 错误处理中间件

## 2.3中间件应用

1. 路由保护，客户端在访问需要登录的页面时，可以先使用中间件判断用户登录状态，用户如果未登录，则拦截请求，直接响应，禁止用户进入需要登录的页面。
2. 网站维护公告，在所有路由的最上面定义接收所有请求的中间件，直接为客户端做出响应，网站正在维护中。
3. 自定义404页面

## 2.4 错误处理中间件

在程序执行的过程中，不可避免的会出现一些无法预料的错误，比如文件读取失败，数据库连接失败。

错误处理中间件是一个集中处理错误的地方。

1. app**.**use**((**err**,** req**,** res**,** next**)** **=>** **{**
2. res**.status(**500**).**send**(**'服务器发生未知错误'**);**
3. **})**

当程序出现错误时，调用next()方法，并且将错误信息通过参数的形式传递给next()方法，即可触发错误处理中间件。

1. app**.**get**(**"/"**,** **(**req**,** res**,** next**)** **=>** **{**
2. fs**.**readFile**(**"/file-does-not-exist"**,** **(**err**,** data**)** **=>** **{**
3. **if** **(**err**)** **{**
4. next**(**err**);**
5. **}**
6. **});**
7. **});**

## 捕获错误

在node.js中，异步API的错误信息都是通过回调函数获取的，支持Promise对象的异步API发生错误可以通过catch方法捕获。

异步函数执行如果发生错误要如何捕获错误呢？

try catch 可以捕获异步函数以及其他同步代码在执行过程中发生的错误，但是不能其他类型的API发生的错误。

1. app**.**get**(**"/"**,** **async** **(**req**,** res**,** next**)** **=>** **{**
2. **try** **{**
3. **await** User**.**find**({**name**:** '张三'**})**
4. **}catch(**ex**)** **{**
5. next**(**ex**);**
6. **}**
7. **});**

# Express请求处理

## 3.1构建模块化路由

1. // admin.js
2. **const** admin **=** express**.**Router**();**
3. admin**.**get**(**'/index'**,** **()** **=>** **{**
4. res**.**send**(**'欢迎来到博客管理页面'**);**
5. **});**
6. module**.**exports **=** admin**;**
7. // home.js
8. **const** home **=** express**.**Router**();**
9. home**.**get**(**'/index'**,** **()** **=>** **{**
10. res**.**send**(**'欢迎来到博客展示页面'**);**
11. **});**
12. module**.**exports **=** home**;**
13. // app.js
14. **const** home **=** require**(**'./route/home.js'**);**
15. **const** admin **=** require**(**'./route/admin.js'**);**
16. app**.**use**(**'/home'**,** home**);**
17. app**.**use**(**'/admin'**,** admin**);**

## 3.2 GET参数的获取

Express框架中使用req.query即可获取GET参数，框架内部会将GET参数转换为对象并返回。

1. // 接收地址栏中问号后面的参数
2. // 例如: <http://localhost:3000/?name=zhangsan&age=30>
3. app**.**get**(**'/'**,** **(**req**,** res**)** **=>** **{**
4. console**.**log**(**req**.**query**);** // {"name": "zhangsan", "age": "30"}
5. **});**

## 3.3 POST参数的获取

Express中接收post请求参数需要借助第三方包 body-parser。

1. // 引入body-parser模块
2. **const** bodyParser **=** require**(**'body-parser'**);**
3. // 配置body-parser模块
4. app**.**use**(**bodyParser**.**urlencoded**({** extended**:** **false** **}));**
5. // 接收请求
6. app**.**post**(**'/add'**,** **(**req**,** res**)** **=>** **{**
7. // 接收请求参数
8. console**.**log**(**req**.**body**);**
9. **})**

## 3.4 Express路由参数

1. app**.**get**(**'/find/:id'**,** **(**req**,** res**)** **=>** **{**
2. console**.**log**(**req**.**params**);** // {id: 123}
3. **});**
4. localhost**:**3000**/**find**/**123

## 3.5 静态资源的处理

通过Express内置的express.static可以方便地托管静态文件，例如img、CSS、JavaScript 文件等。

1. app**.**use**(**express**.static(**'public'**));**

现在，public 目录下面的文件就可以访问了。

http://localhost:3000/images/kitten.jpg

http://localhost:3000/css/style.css

http://localhost:3000/js/app.js

http://localhost:3000/images/bg.png

http://localhost:3000/hello.html

# express-art-template模板引擎

## 4.1 模板引擎

为了使art-template模板引擎能够更好的和Express框架配合，模板引擎官方在原art-template模板引擎的基础上封装了express-art-template。

使用npm install art-template express-art-template命令进行安装。

1. // 当渲染后缀为art的模板时 使用express-art-template
2. app**.**engine**(**'art'**,** require**(**'express-art-template'**));**
3. // 设置模板存放目录
4. app**.**set**(**'views'**,** path**.**join**(**\_\_dirname**,** 'views'**));**
5. // 渲染模板时不写后缀 默认拼接art后缀
6. app**.**set**(**'view engine'**,** 'art'**);**
7. app**.**get**(**'/'**,** **(**req**,** res**)** **=>** **{**
8. // 渲染模板
9. res**.**render**(**'index'**);**
10. **});**

## 4.2 app.locals 对象

将变量设置到app.locals对象下面，这个数据在所有的模板中都可以获取到。

1. app**.**locals**.**users **=** **[{**
2. name**:** '张三'**,**
3. age**:** 20
4. **},{**
5. name**:** '李四'**,**
6. age**:** 20
7. **}]**

# 实例

## 5.1 framework/框架入门.js

// 引入express框架

const express = require('express');

// 创建网站服务器

const app = express();

app.get('/' , (req, res) => {

    // send()

    // 1. send方法内部会检测响应内容的类型

    // 2. send方法会自动设置http状态码

    // 3. send方法会帮我们自动设置响应的内容类型及编码

    res.send('Hello. Express');

})

app.get('/list', (req, res) => {

    res.send({name: '张三', age: 20})

})

// 监听端口

app.listen(3000);

console.log('网站服务器启动成功');

## 5.2 framework/中间件.js

// 引入express框架

const express = require('express');

// 创建网站服务器

const app = express();

app.get('/request', (req, res, next) => {

    req.name = "张三";

    next();

})

app.get('/request', (req, res) => {

    res.send(req.name)

})

// 监听端口

app.listen(3000);

console.log('网站服务器启动成功');

## 5.3 framework/app.use.js

// 引入express框架

const express = require('express');

// 创建网站服务器

const app = express();

// 接收所有请求的中间件

app.use((req, res, next) => {

    console.log('请求走了app.use中间件');

    next()

})

// 当客户端访问/request请求的时候走当前中间件

app.use('/request', (req, res, next) => {

    console.log('请求走了app.use / request中间件')

    next()

})

app.get('/list', (req, res) => {

    res.send('/list')

})

app.get('/request', (req, res, next) => {

    req.name = "张三";

    next();

})

app.get('/request', (req, res) => {

    res.send(req.name)

})

// 监听端口

app.listen(3000);

console.log('网站服务器启动成功');

## 5.4 framework/中间件应用.js

// 引入express框架

const express = require('express');

// 创建网站服务器

const app = express();

// 网站公告

// app.use((req, res, next) => {

//  res.send('当前网站正在维护...')

// })

app.use('/admin', (req, res, next) => {

    // 用户没有登录

    let isLogin = true;

    // 如果用户登录

    if (isLogin) {

        // 让请求继续向下执行

        next()

    } else {

        // 如果用户没有登录 直接对客户端做出响应

        res.send('您还没有登录 不能访问/admin这个页面')

    }

})

app.get('/admin', (req, res) => {

    res.send('您已经登录 可以访问当前页面')

})

app.use((req, res, next) => {

    // 为客户端响应404状态码以及提示信息

    res.status(404).send('当前访问的页面是不存在的')

})

// 监听端口

app.listen(3000);

console.log('网站服务器启动成功');

## 5.5 framework/错误处理中间件.js

// 引入express框架

const express = require('express');

const fs = require('fs');

// 创建网站服务器

const app = express();

app.get('/index', (req, res, next) => {

    // throw new Error('程序发生了未知错误')

    fs.readFile('./01.js', 'utf8', (err, result) => {

        if (err != null) {

            next(err)

        }else {

            res.send(result)

        }

    })

    // res.send('程序正常执行')

})

// 错误处理中间

app.use((err, req, res, next) => {

    res.status(500).send(err.message);

})

// 监听端口

app.listen(3000);

console.log('网站服务器启动成功');

## 5.6 framework/异步函数错误的捕获.js

// 引入express框架

const express = require('express');

const fs = require('fs');

const promisify = require('util').promisify;

const readFile = promisify(fs.readFile);

// 创建网站服务器

const app = express();

app.get('/index', async (req, res, next) => {

    try {

        await readFile('./aaa.js')

    }catch (ex) {

        next(ex);

    }

})

// 错误处理中间

app.use((err, req, res, next) => {

    res.status(500).send(err.message);

})

// 监听端口

app.listen(3000);

console.log('网站服务器启动成功');

## 5.7 framework/构建模块化路由的基础代码.js

// 引入express框架

const express = require('express');

// 创建网站服务器

const app = express();

// 创建路由对象

const home = express.Router();

// 为路由对象匹配请求路径

app.use('/home', home);

// 创建二级路由

home.get('/index', (req, res) => {

    res.send('欢迎来到博客首页页面')

})

// 端口监听

app.listen(3000);

## 5.8 framework/构建模块化路由.js

// 引入express框架

const express = require('express');

// 创建网站服务器

const app = express();

const home = require('./route/home');

const admin = require('./route/admin');

app.use('/home', home);

app.use('/admin', admin);

// 端口监听

app.listen(3000);

Framework/route/home.js

const express = require('express');

const home = express.Router();

home.get('/index', (req, res) => {

    res.send('欢迎来到博客首页页面')

});

module.exports = home;

Framework/route/admin.js

const express = require('express');

const admin = express.Router();

admin.get('/index', (req, res) => {

    res.send('欢迎来到博客管理页面')

});

module.exports = admin;

## 5.9 framework/如何获取get请求参数.js

// 引入express框架

const express = require('express');

// 创建网站服务器

const app = express();

app.get('/index', (req, res) => {

    // 获取get请求参数

    res.send(req.query)

})

// 端口监听

app.listen(3000);

## 5.10 framework/如何获取post请求参数.js

Framework/post.html

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>Document</title>

</head>

<body>

    <form action="http://localhost:3000/add" method="post">

        <input type="text" name="username">

        <input type="password" name="password">

        <input type="submit" name="">

    </form>

</body>

</html>

// 引入express框架

const express = require('express');

const bodyParser = require('body-parser');

// 创建网站服务器

const app = express();

// 拦截所有请求

// extended: false 方法内部使用querystring模块处理请求参数的格式

// extended: true 方法内部使用第三方模块qs处理请求参数的格式

app.use(bodyParser.urlencoded({extended: false}))

app.post('/add', (req, res) => {

    // 接收post请求参数

    res.send(req.body)

})

// 端口监听

app.listen(3000);

## 5.11 framework/app.use方法.js

// 引入express框架

const express = require('express');

const bodyParser = require('body-parser');

// 创建网站服务器

const app = express();

app.use(fn ({a: 2}))

function fn (obj) {

    return function (req, res, next) {

        if (obj.a == 1) {

            console.log(req.url)

        }else {

            console.log(req.method)

        }

        next()

    }

}

app.get('/', (req, res) => {

    // 接收post请求参数

    res.send('ok')

})

// 端口监听

app.listen(3000);

## 5.12 framework/路由参数.js

// 引入express框架

const express = require('express');

const bodyParser = require('body-parser');

// 创建网站服务器

const app = express();

app.get('/index/:id/:name/:age', (req, res) => {

    // 接收post请求参数

    res.send(req.params)

})

// 端口监听

app.listen(3000);

## 5.13 framework/静态资源访问功能.js

const express = require('express');

const path = require('path');

const app = express();

// 实现静态资源访问功能

app.use('/static',express.static(path.join(\_\_dirname, 'public')))

// 端口监听

app.listen(3000);

## 5.14 framework/模板引擎.js

Framework/views/index.art

{{ msg }}

<ul>

{{each users}}

<li>

{{$value.name}}

{{$value.age}}

</li>

{{/each}}

</ul>

Framework/views/list.art

{{msg}}

<ul>

{{each users}}

<li>

{{$value.name}}

{{$value.age}}

</li>

{{/each}}

</ul>

const express = require('express');

const path = require('path');

const app = express();

// 1.告诉express框架使用什么模板引擎渲染什么后缀的模板文件

//  1.模板后缀

//  2.使用的模板疫情

app.engine('art', require('express-art-template'))

// 2.告诉express框架模板存放的位置是什么

app.set('views', path.join(\_\_dirname, 'views'))

// 3.告诉express框架模板的默认后缀是什么

app.set('view engine', 'art');

app.get('/index', (req, res) => {

    // 1. 拼接模板路径

    // 2. 拼接模板后缀

    // 3. 哪一个模板和哪一个数据进行拼接

    // 4. 将拼接结果响应给了客户端

    res.render('index', {

        msg: 'message'

    })

});

app.get('/list', (req, res) => {

    res.render('list', {

        msg: 'list page'

    })

})

// 端口监听

app.listen(3000);

## 5.15 framework/app.locals对象.js

const express = require('express');

const path = require('path');

const app = express();

// 模板配置

app.engine('art', require('express-art-template'))

app.set('views', path.join(\_\_dirname, 'views'))

app.set('view engine', 'art');

app.locals.users = [{

    name: 'zhangsan',

    age: 20

},{

    name: '李四',

    age: 30

}]

app.get('/index', (req, res) => {

    res.render('index', {

        msg: '首页'

    })

});

app.get('/list', (req, res) => {

    res.render('list', {

        msg: '列表页'

    });

})

// 端口监听

app.listen(3000);