LUOJIN 2020-05-19 **Express框架**

Content

目标

- 能够使用Express创建web服务器
- 能够使用Express处理请求参数
- 能够使用Express处理静态资源
- 能够使用中间件处理请求
- 能够在Express中集成art-template模板引擎

1. Express框架简介及初体验

1.1 Express框架是什么？

1）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。

2）这个是nodejs的第三方模块-使用 npm install express 命令进行下载。

1.2 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

1.3 原生Node.js与Express框架对比之路由

```
app.on('request', (req, res) => {
    // 获取客户端的请求路径
    let { pathname } = url.parse(req.url);
    // 对请求路径进行判断 不同的路径地址响应不同的内容
    if (pathname == '/' || pathname == 'index') {
       res.end('欢迎来到首页');
    } else if (pathname == '/list') {
       res.end('欢迎来到列表页页');
    } else if (pathname == '/about') {
       res.end('欢迎来到关于我们页面')
    } else {
       res.end('抱歉, 您访问的页面出游了');
    }
});
```

```
// 当客户端以get方式访问/时
app.get('/', (req, res) => {
    // 对客户端做出响应
    res.send('Hello Express');
});

// 当客户端以post方式访问/add路由时
app.post('/add', (req, res) => {
   res.send('使用post方式请求了/add路由');
});
```

1.4 原生Node.js与Express框架对比之获取请求参数

```
app.on('request', (req, res) => {
    // 获取GET参数
    let {query} = url.parse(req.url, true);
    // 获取POST参数
    let postData = '';
    req.on('data', (chunk) => {
        postData += chunk;
    });
    req.on('end', () => {
        console.log(querystring.parse(postData)
    }));
});
```

```
app.get('/', (req, res) => {
    // 获取GET参数
    console.log(req.query);
});

app.post('/', (req, res) => {
    // 获取POST参数
    console.log(req.body);
})
```

1.5 Express初体验

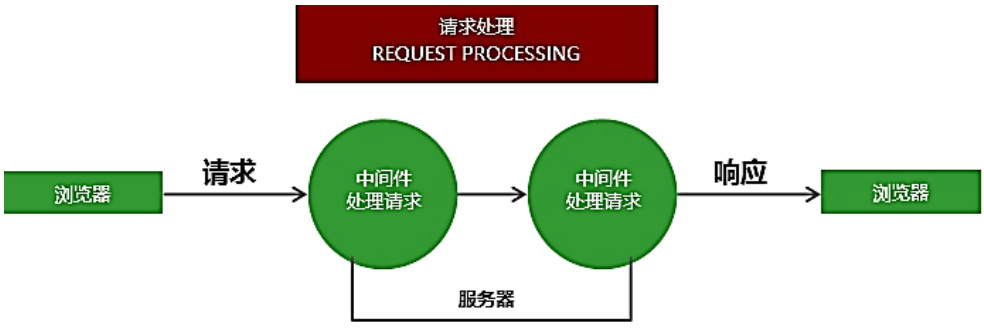
使用Express框架创建web服务器及其简单，调用express模块返回的函数即可。

```
// 引入Express框架
const express = require('express');
// 使用框架创建web服务器
const app = express();
// 当客户端以get方式访问/路由时
app.get('/', (req, res) => {
    // 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
    res.send('Hello Express'); // <h2>Hello Express</h2> {say: 'hello'}
});
// 程序监听3000端口
app.listen(3000);
```

2. 中间件

2.1 什么是中间件

1）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。



2）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。

· app.get('请求路径', '处理函数') // 接收并处理get请求

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

3）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

```
app.get('/request', (req, res, next) => {
    req.name = "张三";
    next(); });
```

```
app.get('/request', (req, res) => {
    res.send(req.name);
})
```

```
const express = require('express')

// 创建网站服务器
const app = express();
app.get('/request', (req, res, next) => {
    req.name = 'luojin'
    next() // 当前执行完成后，还会向下匹配中间件
})
app.get('/request', (req, res) => {
    res.send(req.name)
})

app.listen(3000);
console.log("网站服务器启动成功")
```

```
const express = require('express')

// 创建网站服务器
const app = express();
app.get('/', (req, res) => {
    // send () 方法，不再是end
    // 1，内部检测相应内容的类型
    // 2，会自动设置http状态码
    // 3，会帮助我们自动设置响应的内容

    res.send('hello express')

})
app.get('/list', (req, res) => {
    res.send({
        name: 'luojin',
        age: 20
    })

})

app.listen(3000);
console.log("网站服务器启动成功")
```

```
const express = require('express')
const app = express();// 创建网站服务器

app.use((req, res, next) => {// 接收所有请求的中间件
    console.log('请求走了app.use中间件')
    next()
})
// 当用户端访问/request的请求时候走当前中间件
app.use('/request',(req, res, next) => {
    console.log('请求走了app.use中间件 /request')
    next()
})

app.use('/list',(req, res) => {
    res.send('/list')  //请求走了app.use中间件
})

app.get('/request', (req, res, next) => {
    req.name = 'luojin'
    next()
})
app.get('/request', (req, res) => {
    res.send(req.name)
})

app.listen(3000);
console.log("网站服务器启动成功")
```

2.2 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

  app.use((req, res, next) => { console.log(req.url); next(); });

  中间件是有顺序的

- 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。

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

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

```
app.use((req, res, next) => { // 网站公告
    res.send("当前网站正在维护.....")
})
```

```
const express = require('express')
const app = express();// 创建网站服务器

app.use('/admin',(req, res, next) => {
    let isLogin = true;
    if(isLogin){
        next()  // 如果用户登陆，请求继续向下执行
    }else{
        res.send('您还没有登陆，不能访问当前页面')
    }
})

app.get('/admin', (req, res) => {
    res.send('您已经登陆，可以访问当前页面了')
})

app.listen(3000);
console.log("网站服务器启动成功")

app.use((req,res,next) => {// 这个在网站的最后边,在listen前边
    res.status(404).send('当前访问的页面不存在的')
})
```

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。

想要出错以后，还能继续运行，需要捕获错误，处理

只能捕获到同步代码错误，异步需要手动触发，调用next(0方法

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

当异步程序出现错误时，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

```
app.get("/", (req, res, next) => {
        fs.readFile("/file-does-not-exist", (err, data) => {
        if (err) {
                next(err);
        }
    });
});
```

```
const express = require('express')
const fs = require('fs')
const app = express();// 创建网站服务器

// 普通路由中间件
app.use('/index',(req, res,next) => {
    //throw new Error('程序发生了未知错误')
    fs.readFile('./demo.txt', 'utf8',(err,result)=>{
        if(err!=null){
            next(err)
//ENOENT: no such file or directory, open 'C:\Users\luojin\Desktop\bmcode\node\framework\demo.txt'
        }else{
            res.send(result)
        }
    })
    //res.send('程序正常执行')
})

// 错误处理中间件
app.use((err, req, res, next) => {
    res.status(500).send(err.message)
})

app.listen(3000);
console.log("网站服务器启动成功")
```

2.5 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

```
app.get("/", async (req, res, next) => {
        try {
                await User.find({name: '张三'})
        }catch(ex) {
                next(ex);  // 调用next() 触发错误处理中间件
        }
});
```

3. Express请求处理

3.1 构建模块化路由

在一般情况下，路由的数量是非常多的，如果将所有的方在同一文件中，非常可怕？

所以提供了模块化，进行分类，方便管理。

例如：博客网站 - 用户看的，文章列表，详情页面。管理员看的文章发布等。

const express = require('express') // 引入express框架，返回express方法，在他下课调用其他方法

const home = **express.Router();** // 创建路由对象

app.use('/home', home); // 将路由和请求路径进行匹配

home.get('/index', () => { // 在home路由下继续创建路由 /home/index 二级路由

res.send('欢迎来到博客展示页面'); });

```
const express = require('express')// 引入express 框架
const app = express()// 创建网站服务器
const home = express.Router()// 创建路由对象
app.use('/home', home)// 为路由对象匹配请求路径
home.get('/index',(req,res)=>{// 创建二级路由
    res.send('欢迎来到博客首页')
})
app.listen(3000)//端口监听
```

3.2 构建模块化路由

```
const express = require('express')
const admin = express.Router()
admin.get('/index', (req,res)=>{
    res.send('欢迎来到我的管理页面')
})
module.exports = admin;
```

```
const express = require('express')
const home = express.Router()
home.get('/index', (req,res)=>{
    res.send('欢迎来到我的首页页面')
})
module.exports = home;
```

```
//引入express 框架
const express = require('express')
const app = express()

// 导入其他两个路由模块
const home = require('./route/home.js')
const admin = require('./route/admin.js')

//路径匹配
app.use('/home',home)
app.use('/admin',admin)

app.listen(3000)
```

// home.js
const **home** = express.Router();
home.get('**/index**', () => {
res.send('欢迎来到博客展示页面');
});
module.exports = home;
	// admin.js
	const **admin** = express.Router();
	admin.get**('/index',** () => {
	res.send('欢迎来到博客管理页面');
	});
	module.exports = admin;

// app.js
const home = require('**./route/home.js');**
const admin = require('**./route/admin.js');**
app.use('/home', home); // 利用中间件进行路由匹配
app.use('/admin', admin);	

3.3 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。

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

3.4 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**。

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()

//下载body-parser第三方模块
//拦截所有请求
//extend: false 方法内部使用querystring模块处理请求参数的格式
//extend: true 方法内部使用第三方模块 qs 处理请求参数格式
app.use(bodyParser.urlencoded({extend: false}))
app.post('/add',(req,res) => {
    res.send(req.body) //req.body获取post参数
})

app.listen(3000)
```

```
<form action="http://localhost:3000/add" method="post">
    <input type="text" name ="username">
    <input type="password" name="password">
    <input type="submit">
</form>
```

html文件

```
const express = require('express')
const bodyParser = require('body-parser')
const app = express()

app.use(fn({a:1}))

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) => {
    res.send('okay')
})

app.listen(3000)

// app.use()需要传一个请求处理函数进去，现在为什么传方法的调用？
// 方法的调用也返回一个函数
```

```
const bodyParser = require('body-parser'); // 引入body-parser模块
app.use(bodyParser.urlencoded({ extended: false })); //配置body-parser模块
app.post('/add', (req, res) => { // 接收请求
        console.log(req.body); // 接收请求参数
})
```

3.5 Express路由参数

更容易看出传了那些参数

```
app.get('/find/:id', (req, res) => {
    console.log(req.params); // {id: 123} req.params 获取参数
});
localhost:3000/find/123
```

3.6 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。

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

现在，public 目录下面的文件就可以通过以下方式访问了。

① http://localhost:3000/images/kitten.jpg

② http://localhost:3000/css/style.css

③ http://localhost:3000/js/app.js

④ http://localhost:3000/images/bg.png

⑤ http://localhost:3000/hello.html

```
const express = require('express')
const app = express()
const path = require('path')

//实现静态资源访问功能
//  - 利用app.use()这个中间件拦截所有请求，把请求交给express.static取处理
//  - '/static' 写得话就是这个下访问，所有的都可以访问
app.use('/static',express.static(path.join(__dirname,'public')))
app.listen(3000)
```

4. express-art-template模板引擎

4.1 模板引擎

I. 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**。

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

```
// 当渲染后缀为art的模板时 使用express-art-template
app.engine('art', require('express-art-template')); // engine 告诉使用的啥模板
app.set('views', path.join(__dirname, 'views')); // 设置模板存放目录
app.set('view engine', 'art'); // 渲染模板时不写后缀，默认拼接art后缀=设置模板后缀
app.get('/', (req, res) => { // 渲染模板
    res.render('index');
});
```

```
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
//   - 参数1： 模板后缀
//   - 参数2： 使用的模板引擎
app.engine('art', require('express-art-template'))

// 2, 告诉express框架模板存放的目录是什么
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')

// 4, 渲染模板
//   - 拼接了模板路径，模板后缀
//   - 哪一个模板和哪一个数据进行拼接
//   - 将拼接结果响应给了客户端
app.get('/index', (req, res) => {
    res.render('index', {
        msg: 'message'
    })
})

app.get('/list',(req,res) => {
    res.render('list',{
        msg: 'list page'
    })
})

app.listen(3000)
```

4.2 app.locals 对象

不同的页面中，总会有公共数据，代码中如何查询公共数据呢?

- 在不同页面路由中都去查询这个相同的数据，render将数据填充到模板中，麻烦
- 只写一次，让所有能用到都而已来拿到这个数据呢?

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。

app.locals.users = [{ name: '张三', age: 20 },{ name: '李四', age: 20 }]

```
const express = require('express')
const path = require('path')
const app = express()

app.engine('art', require('express-art-template'))
app.set('views', path.join(__dirname,'views'))
app.set('view engine', 'art')

app.locals.users = [{
    name: 'luojin',
    age: 20
},
{
    name: 'hahhaa',
    age: 30
}]

app.get('/index', (req, res) => {
    // 不希望在这里查询一下
    res.render('index', {
        msg: '首页'
    })
})

app.get('/list',(req,res) => {
    // 不希望在这里再查询一下
    res.render('list',{
        msg: '列表页'
    })
})

app.listen(3000)
```

index.art

```
{{msg}}

<ul>
    {{each users}}
    <li>
        {{$value.name}}
        {{$value.age}}
    </li>
    {{/each}}
</ul>
```