Express框架

目录 Contents

1. Express框架简介及初体验

1.1 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。

1. Express框架简介及初体验

1.2 Express框架特性

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

1. Express框架简介及初体验

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. Express框架简介及初体验

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. Express框架简介及初体验

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。

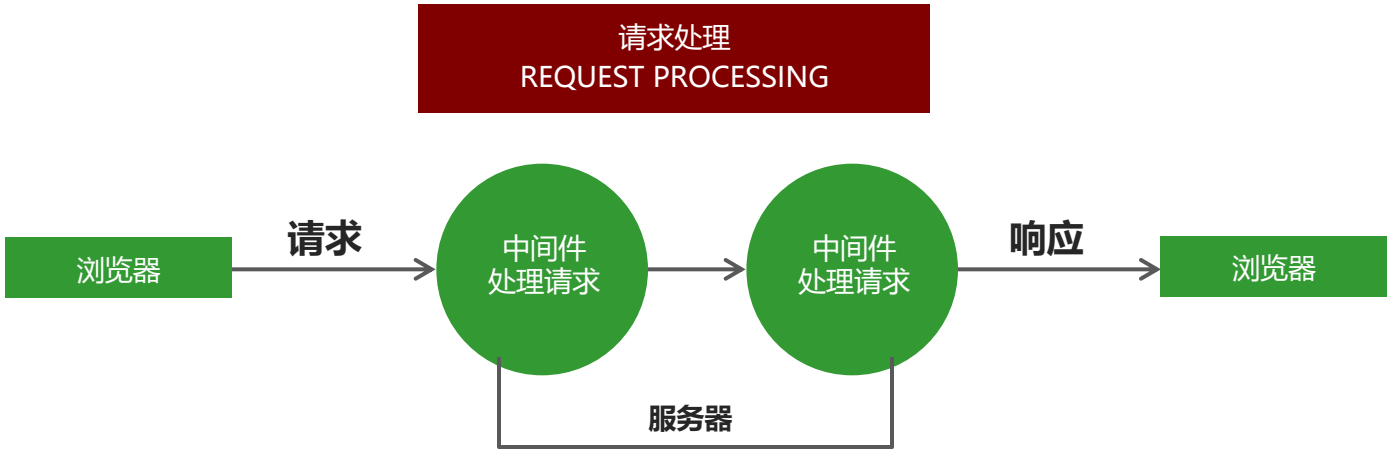
```
// 引入Express框架
const express = require('express');
// 使用框架创建web服务器
const app = express();
// 当客户端以get方式访问/路由时
app.get('/', (req, res) => {
    // 对客户端做出响应 send方法会根据内容的类型自动设置请求头
    res.send('Hello Express'); // <h2>Hello Express</h2> {say: 'hello'}
});
// 程序监听3000端口
app.listen(3000);
```

目录 Contents

2. 中间件

2.1 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。



2. 中间件

2.1 什么是中间件

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
```

2. 中间件

2.1 什么是中间件

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);
});
```

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();
});
```

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

2. 中间件

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。

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

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。

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

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);
    }
});
```

目录 Contents

2. Express请求处理

2.1 构建模块化路由

```
const express = require('express')
// 创建路由对象
const home = express.Router();
// 将路由和请求路径进行匹配
app.use('/home', home);
// 在home路由下继续创建路由
home.get('/index', () => {
    // /home/index
    res.send('欢迎来到博客展示页面');
});
```

2. Express请求处理

2.1 构建模块化路由

```
// 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);
app.use('/admin', admin);
```

2. Express请求处理

2.3 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"}
});
```

2. Express请求处理

2.4 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。

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

2. Express请求处理

2.5 Express路由参数

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

```
localhost:3000/find/123
```

2.6 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

目录 Contents

3. express-art-template模板引擎

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。

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

3. express-art-template模板引擎

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。

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