Express

> Express是一个基于Node平台的web应用开发框架，提供一系列强大特性，帮助你创建各种 Web应用。

安装

使用`npm install express`进行本地安装。

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
4. 拥有大量第三方中间件对功能进行扩展

启动

```
// 引入Express框架
const express = require('express');
// 使用框架创建web服务器
const app = express();
// 程序监听3000端口
app.listen(3000);
// 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'}
});
```

路由

Express框架对路由功能进行了封装，提供了get、post等方法，使得定义路由非常简便。

```
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');
});
```

中间件

> 中间件本质上就是函数，可访问请求对象和响应对象，可对请求进行拦截处理，处理后再将控制权向下传递，也可终止请求，向客户端做出响应。

需求场景

打印访问日志、网站维护公告、登录拦截

使用方法

使用`app.use()`方法定义中间件。

- 该方法可以传递一个函数作为参数，表示任何任何请求都会经过该中间件，都会执行该参数内部的代码。

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

  - 中间件函数有三个参数，分别为请求对象req、响应对象res、释放控制权方法next。
  - 中间件函数中的代码执行完成之后需要调用next()方法，才能开始执行下一个中间件，否则请求将挂起。

- 该方法的第一个参数也可以是请求路径，表示只有该请求路径才会经过该中间件。

```
app.use('/user', (req, res, next) => {
   console.log(req.method);
   next();
});
```

模块化路由

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');
});
```

```
// 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);
```

静态资源

通过 Express 内置的 `express.static` 可以方便地托管静态文件，例如图片、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
```

参数处理

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

Express中允许开发人员定义路由参数，定义方式如下：

```
// 以get方式访问/list路由时携带id参数
app.get('/list/:id', (req, res) => {
    console.log(req.params) // {"id":"12"}
});
// 客户端请求时地址栏应该写成这样
// http://localhost:3000/list/12
```

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

```
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));

req.body // post请求参数
```

模板引擎

Express框架没有内置模板引擎，但是对其他模板引擎提供了良好的支持，比如art-template。

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除了express-art-template。

安装

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

使用

```
var express = require('express');
var app = express();
// 告诉Express框架 使用art-template模板引擎渲染HTML模板
app.engine('html', require('express-art-template'));
// 当render方法渲染模板时，如果模板后缀被省略，则将html作为默认的文件后缀
app.set('view engine', 'html');
// 当render方法渲染模板时，如果模板所在路径被省略，则在内部会自动将以下路径进行拼接
// 设置模板所在目录
app.set('views', path.join(__dirname, 'views'));
app.get('/', function (req, res) {
    res.render('index.art', {
        user: {
            name: 'aui',
            tags: ['art', 'template', 'nodejs']
        }
    });
});
```

404页面

```
// 处理用户访问路径出错的情况
app.get('*', (req, res) => res.render('error.html'));
// 此中间件一定要写在程序的最后 其他路径都匹配不成功时再匹配404页面
```

错误处理

Express默认会把程序错误信息输出到页面中，在开发阶段有助于开发人员观察错误信息，但是在项目发布以后是不希望用户看到这些错误信息的，所以可以使用Express提供的错误处理中间件处理错误信息。

```
app.use((err, req, res, next)=>{
    // 在控制台中输出错误信息
    console.error(err.stack);
    // 对客户端做出响应
    res.status(500).send('程序出错');
});
```

app.locals

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
}]
```

```
// index.art 模板文件
<ul>
  {{each users}}
  <li>
        {{$value.name}}
        {{$value.age}}
  </li>
  {{/each}}
</ul>
```