Express Web框架

① Express 介绍

- Express 是一个第三方模块
- Express 是一个基于 Node.js 平台，快速、开放、极简的 **web 开发框架**。
- 丰富的 API 支持，强大而灵活的**中间件**特性
- Express 不对 Node.js 已有的特性进行二次抽象，只是在它之上扩展了 Web 应用所需的基本功能
- Express 官网
- Express 中文文档（非官方）
- Express GitHub仓库

特点

- Web 应用程序
  - Express 是一个保持最小规模的灵活的 Node.js Web 应用程序开发框架，为 Web 和移动应用程序提供一组强大的功能。
- API
  - 使用您所选择的各种 HTTP 实用工具和中间件，快速方便地创建强大的 API。
- 性能
  - Express 提供精简的基本 Web 应用程序功能，而不会隐藏您了解和青睐的 Node.js 功能。
- 框架
  - 许多流行的开发框架 都基于 Express 构建。

② 起步

2.1 安装

> 参考文档：http://expressjs.com/en/starter/installing.html

> 参考文档(中文)：https://www.expressjs.com.cn/starter/installing.html

```
# 创建并切换到 myapp 目录
mkdir myapp
cd myapp

# 初始化 package.json 文件
npm init -y

# 安装 express 到项目中
npm install express --save
```

2.2 Hello World

> 参考文档：http://expressjs.com/en/starter/hello-world.html

> 参考文档(中文):https://www.expressjs.com.cn/starter/hello-world.html

- 在当前myapp目录下创建一个app.js文件，代码如下：

```
// 0. 加载 Express
const express = require('express')

// 1. 调用 express() 得到一个 app
//    类似于 http.createServer()
const app = express()

// 2. 设置请求对应的处理函数
//    当客户端以 GET 方法请求 / 的时候就会调用第二个参数：请求处理函数
app.get('/', (req, res) => {
  // send方法内部调用 response.end()
  // 并且内部已经设置了 Content-Type和其它必要的响应头
  res.send('hello world')
})

// 3. 监听端口号，启动 Web 服务
app.listen(3000, () => console.log('app listening on port 3000!'))
```

- 启动服务：

```
$ node app.js
```

- 打开浏览器输入网址：http://127.0.0.1:3000 测试访问。

2.3 基本路由

> 参考文档：http://expressjs.com/en/starter/basic-routing.html

> 参考文档(中文)：https://www.expressjs.com.cn/starter/basic-routing.html

- 路由就像开车从北京-上海，出发之前可以查询走的路线，最终选择的那条路线就相当于**路由**
- 路由（**Routing**）是由一个 **URL**（或者叫路径标识）和一个特定的 **HTTP 方法**（GET、POST 等）组成的，涉及到应用如何处理响应客户端请求。
- 每一个路由都可以有一个或者多个处理器函数，当匹配到路由时，这些函数将被执行。
- 路由的定义的结构如下：

```
app.METHOD(PATH, HANDLER)
```

其中：

- `app` 是 express 实例
- `METHOD` 是一个 HTTP 请求方法
- `PATH` 是服务端路径（定位标识,请求url地址一部分）
- `HANDLER` 是当路由匹配到时需要执行的处理函数

下面是一些基本示例。

- 路径
  - http://127.0.0.1:3000/xxxx
  - app.get('路径')
  - 路径：域名后面的path
- 处理 get 请求

```
// 当你以 GET 方法请求 / 的时候，执行对应的处理函数
app.get('/', function (req, res) {
  res.send('Hello World!')
})
```

- 处理 post 请求

```
// 当你以 POST 方法请求 / 的时候，指定对应的处理函数
app.post('/', function (req, res) {
  res.send('Got a POST request')
})
```

路由的参考文档 routing guide.

2.4 处理静态资源

> 参考文档：http://expressjs.com/en/starter/static-files.html

> 参考文档(中文)：https://www.expressjs.com.cn/4x/api.html#express.static

- 目录结构

```
.
├── node_modules  npm安装的第三方包目录，使用 npm 装包会自动创建
├── public / statics页面需要使用的静态资源
|   ├── css
|   ├── js
|   ├── images
|   └── ...
├── views 所有页面（只存储 html 文件，模板）
|   ├── publish.html
|   └── index.html
├── app.js 服务端程序入口文件，执行该文件会启动我们的 web 服务器
```

- express 中提供了方便的处理静态资源的方式

```
// 开放 public 目录中的资源
// 不需要访问前缀
app.use(express.static('public'))
// http://127.0.0.1:3000/css/index.css
// http://127.0.0.1:3000/images/lj.jpg
```

```
// http://127.0.0.1:3000/images/timg.jpg

// 开放 files 目录资源，同上
app.use(express.static('files'))

// 限制访问前缀
app.use('/public', express.static('public'))
// http://127.0.0.1:3000/public/css/index.css

// 开放 public 目录资源，限制访问前缀
app.use('/static', express.static('public'))

// 开放 public 目录，限制访问前缀
// path.join(__dirname, 'public') 会得到一个绝对路径
app.use('/public', express.static(path.join(__dirname, 'public')))
```

注意： `express.static()` 使用相对路径的时候，相对于**工作目录**(执行 node 程序的目录)，推荐此处使用**绝对路径**。

③ 在express中使用art-template模板引擎

- 我们可以使用模板引擎处理服务端渲染，但是 Express 为了保持其极简灵活的特性并没有提供类似的功能。
- 同样的，Express 也是开放的，它支持开发人员根据自己的需求将模板引擎和 Express 结合实现**服务端渲染**的能力。

art-template 模板引擎

> 参考文档：
>
> - art-template 官方文档
> - express-art-template 官方文档

- art-template是一个简单且超快速的模板引擎
- 特征：
  - 性能接近JavaScript渲染限制
  - 调试友好。语法错误或运行时错误将准确定位在模板的哪一行。支持在模板文件中设置断点（Webpack Loader）
  - 支持Express，Koa，Webpack
  - 支持模板继承和子模板
  - 浏览器版本只有6KB

安装：

```
npm install --save art-template
npm install --save express-art-template
```

art模板在Express框架中的配置

- 在线文档：https://aui.github.io/art-template/express/
- 实例参考如下实例代码：app.js

```
// 0. 加载 Express
const express = require('express')
const path = require('path')

// 1. 调用 express() 得到一个 app
//    类似于 http.createServer()
const app = express()

// view engine setup
app.engine('html', require('express-art-template'));
//app.set('view', {  //此配置报错需要注释掉
//    debug: process.env.NODE_ENV !== 'production'
//});
app.set('views', path.join(__dirname, 'views')); //设置视图渲染存储目录
app.set('view engine', 'html'); //设置模板文件后缀名

// 2. 设置请求对应的处理函数
//    当客户端以 GET 方法请求 / 的时候就会调用第二个参数：请求处理函数
app.get('/', (req, res) => {
    // send方法内部调用 response.end()
    // 并且内部已经设置了 Content-Type和其它必要的响应头
    //res.send('hello world')
    res.render('index.html', {
        title:'欢迎使用Express框架与art-template模板',
        link:'<a href="https://www.expressjs.com.cn">Express中文网</a>',
        num: [10,20,30,40],
        dlist:[
          {name:'张无忌',age:20,sex:1},
          {name:'赵敏',age:18,sex:0},
          {name:'张三丰',age:60,sex:1}
        ]
    });
})

// 3. 监听端口号，启动 Web 服务
app.listen(3000, () => console.log('app listening on port 3000!'));
```

- art-template模板语法分两种：标准语法(standard syntax)和原始语法(original syntax)
- 在线文档：https://aui.github.io/art-template/docs/syntax.html
- 在项目目录下创建views目录，并在里面新建index.html模板文件，代码如下：

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
```

```
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Document</title>
</head>
<body>
    <h1>{{title}}</h1>
    <p>网址(默认html标签被转义)：{{link}} </p>
    <p>网址(使用@原始输出)：{{@link}} </p>
    <p>数组的输出：{{num[0]}}、{{num[1]}}、{{num[2]}}、{{num[3]}}</p>
    <p>数值计算：{{num[0]*2}} {{num[1]+num[2]}} {{num[3]/2}}</p>

    <h4>遍历输出</h4>
    <ul>
        {{each dlist}}
        <li>{{$index}} => {{$value.name}} :
            {{$value.age}} : {{$value.sex==1?'男':'女'}} :
            {{if $value.sex==1}}男{{else}}女{{/if}}</li>
        {{/each}}
    </ul>
</body>
</html>
```

输出结果：

```
欢迎使用Express框架与art-template模板
网址(默认html标签被转义)：<a href="https://www.expressjs.com.cn">Express中文网</a>

网址(使用@原始输出)：Express中文网

数组的输出：10、20、30、40

数值计算：20 50 20

遍历输出
    .0 => 张无忌 : 20 : 男 :  男
    .1 => 赵敏 : 18 : 女 :  女
    .2 => 张三丰 : 60 : 男 :  男
```

④ Express 应用程序生成器

- 通过应用生成器工具 express-generator 可以快速创建一个应用的骨架。

4.1 使用Express应用程序生成器来搭建项目

- 你可以通过 npx （`包含在 Node.js 8.2.0 及更高版本中`）命令来运行 Express 应用程序生成器。

```
$ mkdir mydemo

$ cd mydemo

$ npx express-generator
```

- 注意：对于较老的 `Node` 版本，请通过 npm 将 Express 应用程序生成器安装到全局环境中并执行即可,具体详见
- 然后安装所有依赖包：

```
$ npm install
```

- 最后通过如下命令启动此应用

```
$ npm start
```

- 然后在浏览器中打开 http://localhost:3000/ 网址就可以看到这个应用了。
- 通过生成器创建的应用一般都有如下目录结构：

```
mydemo
├── app.js
├── bin
│   └── www
├── package.json
├── public
│   ├── images
│   ├── javascripts
│   └── stylesheets
│       └── style.css
├── routes
│   ├── index.js
│   └── users.js
└── views
    ├── error.pug
    ├── index.pug
    └── layout.pug
```

4.2 将art-template模板添加到此项目

- 安装art-template模板

```
npm install --save art-template
npm install --save express-art-template
```

- 修改app.js文件配置art-template模板

```
...

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

...

//倒数第3行，将res.render('error.pug'); 中的文件后缀名改为html
res.render('error.html');
```

- 修改routes/index.js 子路由文件:

```
res.render('index.pug', { title: 'Express' });
改为
res.render('index.html', { title: 'Express' });
```

- 进入views目录将里面的三个文件的后缀名pug改为html（layout.html、error.html、index.html）
- 修改views/layout.html文件内容如下：

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Express框架--{{title}}</title>
  <link rel="stylesheet" href="/stylesheets/style.css">
</head>
<body>
  {{block 'content'}} ... {{/block}}
</body>
</html>
```

- 修改views/error.html文件内容如下：

```
{{extend './layout.html'}}

{{block 'content'}}
    <h1>{{message}}</h1>
    <h2>{{error.status}}</h2>
    <pre>{{error.stack}}</pre>
{{/block}}
```

- 修改views/index.html文件内容如下：

```
{{extend './layout.html'}}

{{block 'content'}}
    <h1>{{title}}</h1>
    <p>welcome to {{title}}</p>
{{/block}}
```

- 重启服务运行 `nmp start` 然后在浏览器中打开 http://localhost:3000/ 网址就可以看到这个应用了。

⑤ 学生信息管理CURD案例

5.1 将实现准备好的静态页面放置到页面中：

- 浏览学生信息

- 添加学生信息表单页面

- 编辑学生信息表单页

5.2 编写学生信息操作模块：

- 定义数据文件（在项目根目录下创建db.json文件）：


```
{"students":[{"id":1,"name":"张三","sex":"1","age":"20","classid":"web220"},{"id":2,"name":"李四","sex":"0","age":"25","classid":"web221"},{"name":"王五","sex":"1","age":"20","classid":"web102","id":3},{"name":"赵六","sex":"0","age":"20","classid":"web221","id":5},{"name":"田七","sex":"1","age":"22","classid":"web102","id":6}]}
```


- 编写上面db.json数据文件的操作模块
- 在项目根目录下创建model目录，并在里面创建student.js文件，代码如下：

```
//数据信息操作的model层：完成对db.json信息的访问处理
const fs = require("fs");
const dbfile = "./db.json";

/**
 * 获取所有学员信息
 */
exports.findAll = (callback)=>{
  fs.readFile(dbfile,"utf-8",(err,data)=>{
    if(err){
      callback(err,null);
    }else{
      callback(null,JSON.parse(data).students);
    }
  });
}

/**
```

```
 * 执行添加学员信息方法
 */
exports.save = (student,callback)=>{
  fs.readFile(dbfile,"utf-8",(err,data)=>{
    if(err){
      return callback(err);
    }
    let stulist = JSON.parse(data).students;
    //为即将添加的学生信息放置一个id号（获取最多id+1）
    student.id = stulist[stulist.length-1].id + 1;
    //将当前学生信息添加到学生信息列表中
    stulist.push(student);
    //将学生信息的json格式转成字串
    let studata = JSON.stringify({"students":stulist});

    //写回文件中
    fs.writeFile(dbfile,studata,(err)=>{
      callback(err);
    });
  });
}


/**
 * 根据 id 获取学生信息对象
 * @param  {Number}   id       学生 id
 * @param  {Function} callback 回调函数
 */
exports.findById = (id, callback) => {
  fs.readFile(dbfile, 'utf8', (err, data) => {
    if (err) {
      return callback(err);
    }
    var stulist = JSON.parse(data).students;
    //从学员信息列表中获取出指定id的单条信息
    var ret = stulist.find((item) => {
      return item.id === parseInt(id);
    })
    callback(null, ret);
  })
}

/**
 * 修改学生
 */
exports.updateById = (student, callback) => {
  fs.readFile(dbfile, 'utf8', (err, data) => {
    if (err) {
      return callback(err);
```

```
    }
    var stulist = JSON.parse(data).students;

    // 注意：这里记得把 id 统一转换为数字类型
    student.id = parseInt(student.id);

    // 你要修改谁，就需要把谁找出来
    // EcmaScript 6 中的一个数组方法：find
    // 需要接收一个函数作为参数
    // 当某个遍历项符合 item.id === student.id 条件的时候，find 会终止遍历，同时返回遍历项
    var stu = stulist.find( (item) => {
      return item.id === student.id;
    })

    // 这种方式你就写死了，有 100 个难道就写 100 次吗？
    // stu.name = student.name
    // stu.age = student.age

    // 遍历拷贝对象
    for (var key in student) {
      stu[key] = student[key];
    }

    // 把对象数据转换为字符串
    var studata = JSON.stringify({students:stulist})

    // 把字符串保存到文件中
    fs.writeFile(dbfile, studata, (err) => {
       callback(err);
    })
  })
}

/**
 * 删除学生
 */
exports.deleteById = (id, callback) => {
  fs.readFile(dbfile, 'utf8', (err, data) => {
    if (err) {
      return callback(err);
    }
    var stulist = JSON.parse(data).students;

    // findIndex 方法专门用来根据条件查找元素的下标
    var deleteId = stulist.findIndex((item) => {
      return item.id === parseInt(id);
    })
```

```
    // 根据下标从数组中删除对应的学生对象
    stulist.splice(deleteId, 1);

    // 把对象数据转换为字符串
    var studata = JSON.stringify({students:stulist})

    // 把字符串保存到文件中
    fs.writeFile(dbfile, studata,(err) => {
      callback(err);
    })
  })
}
```

- 测试：

5.3 编写学生信息管理路由文件

- 在routes路由目录下创建stu.js路由文件

```
//学生信息管理路由文件
var express = require('express');
const Student = require("../model/student"); //导入自定义模块（数据信息操作）

var router = express.Router();

/*
 * 浏览学生信息
 */
router.get('/', (req, res, next) => {
  //res.render("stu/index");
  Student.findAll((err,students)=>{
    //判断数据查询是否成功!
    if(err){
      //响应错误信息
      return res.status(500).send("服务器端错误! ");
    }
    //console.log(students);
    //将获取的结果students以stulist名放入到模板中，并加载模板输出
    res.render("stu/index.html",{"stulist":students});
  });
});

/*
 * 加载添加学生信息表单
 */
router.get('/add', (req, res, next) => {
  //res.send('加载添加学生信息表单');
  res.render("stu/add.html");
});
```

```
/*
 * 执行学生信息添加
 */
router.post('/add', (req, res, next) => {
  //res.send('执行学生信息添加');
  //console.log(req.body);
  Student.save(req.body,(err)=>{
    //判断数据查询是否成功!
    if(err){
      //响应错误信息
      return res.status(500).send("服务器端错误（添加失败）！");
    }
    res.redirect("/stu");//重定向到浏览学员信息页
  });
});

/*
 * 加载学生信息编辑表单
 */
router.get('/edit', (req, res, next) => {
  Student.findById(req.query.id,(err,stu) => {
    if(err){
      //响应错误信息
      return res.status(500).send("服务器端错误（获取信息失败）！");
    }
    res.render("stu/edit.html",{"stu":stu});
  });
});

/*
 * 执行学生信息编辑
 */
router.post('/edit', (req, res, next) => {
  Student.updateById(req.body,(err)=>{
    //判断数据查询是否成功!
    if(err){
      //响应错误信息
      return res.status(500).send("服务器端错误（修改失败）！");
    }
    res.redirect("/stu");//重定向到浏览学员信息页
  });
});

/*
 * 执行删除学生信息
 */
router.get('/delete', (req, res, next) => {
  //console.log(req.query.id);//获取要删除信息的id号
```

```
  Student.deleteById(req.query.id,(err)=>{
     //判断数据查询是否成功!
    if(err){
      //响应错误信息
      return res.status(500).send("服务器端错误（删除失败）！");
    }
    res.redirect("/stu");//重定向到浏览学员信息页
  });

  //res.send('执行删除学生信息');
});

module.exports = router;
```

- 在项目app.js主文件中导入stu.js路由文件

```
...
var stuRouter = require('./routes/stu');
...

...
app.use('/stu', stuRouter);
...
```

5.4 修改模板实现数据的输出

- 编辑views/stu/index.html

```
<!DOCTYPE html>
<html lang="zh-CN">
  <head>
    <meta charset="utf-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <!-- 上述3个meta标签*必须*放在最前面，任何其他内容都*必须*跟随其后！ -->
    <title>学生信息管理</title>
    <!-- Bootstrap -->
    <link href="/css/bootstrap.min.css" rel="stylesheet">
    <style>
      body{background-color:#eee;}
    </style>
  </head>
  <body>
    <!-- 页面容器开始 -->
    <div class="container">
      <div class="row">
```

```
        <!--页面左侧主体开始-->
        <div class="col-md-12 text-center">
            <h2>学生信息管理</h2>
            <br/>
            <a type="button" href="/stu/add" class="btn btn-primary btn-sm">添
加信息</a>
            <a type="button" href="/" class="btn btn-primary btn-sm">返回首
页</a>
            <br/><br/>
            <table class="table table-hover text-center">
              <tr>
                <th class="text-center">id号</th>
                <th class="text-center">姓名</th>
                <th class="text-center">性别</th>
                <th class="text-center">年龄</th>
                <th class="text-center">班级</th>
                <th class="text-center">编辑</th>
              </tr>
              {{each stulist}}
              <tr>
                <td>{{$value.id}}</td>
                <td>{{$value.name}}</td>
                <td>{{$value.sex==1?"男":"女"}}</td>
                <td>{{$value.age}}</td>
                <td>{{$value.classid}}</td>
                <td>
                  <a type="button" href="/stu/edit?id={{$value.id}}"
class="btn btn-info btn-xs">编辑</a>
                  <a type="button" href="/stu/delete?id={{$value.id}}"
class="btn btn-danger btn-xs">删除</a>
                </td>
              </tr>
              {{/each}}
            </table>
        </div>
        <!--页面左侧主体开始-->
      </div>
    </div>
    <!-- 页面容器结束 -->
    <script src="/js/jquery.min.js"></script>
    <script src="/js/bootstrap.min.js"></script>
  </body>
</html>
```

- 其他详见课堂项目案例代码。