Express 框架

服务器 Express框架

官方地址： https://www.expressjs.com.cn/

1. 框架介绍

部署服务器，托管前端的静态资源，原生方式比较麻烦

托管了指定目录下的静态资源 JavaScript 复制代码

```
// 1. 引入模块
const http = require('http');
const path = require('path');
const fs = require('fs');
const port = 8080;

// 2. 创建服务器对象模型
const app = http.createServer((req, res) => {

    // 2.1 获取纯粹的访问路径
    const { pathname } = new URL(req.url, `http://localhost:${port}`);
    if(pathname === '/')  pathname = 'index.html';

    // 2.2 拼接完整的绝对路径的文件地址
    const filename = path.join(__dirname, 'static', pathname);

    // 2.3 读取指定路径的内容
    fs.readFile(filename, (err, data) => {
        if(err) {
            res.setHeader('content-type', 'text/html;charset=utf8');
            res.end('您访问的页面不存在!')
            return;
        }

        // 2.4 读取成功，响应读取的内容
        res.end(data);
     })

})

// 3. 启动并监听服务器
app.listen(port, () =>{
  console.log(`server running at http://localhost:${port}`)
})
```

有了Express这个框架，在创建服务器，并且书写一些结构或者接口的时候，变得更加简单高效；

能够快速创建一个web的框架，是基于后端的角度来说的，所以并不是用于做前端的；

基于 Node.js 创建，也就是很多方式和方式都是来自于原生 Node 的封装，也可以共存；

就好像 jQuery 和 原生 js 之间的关系。

2. 安装和使用

在本地项目中安装和引入，即可快速的进行创建

首先需要在本地进行初始化一个模块配置文件 package.json

在本地项目中安装 Express　Bash　复制代码

```bash
# 指定一个目录
$ npm init -y


$ npm i express -S
# 或者
$ npm i express
```

引入express ，即可开始创建项目

使用 express 创建服务器　JavaScript　复制代码

```javascript
// 1. 引入模块
const express = require('express');

// 2. 创建服务器模块对象
const app = express();
// 2.1 声明指定的端口
const port = 8080;

// 3.0 监听访问的接口
app.get('/', (req, res) => {

    // 响应内容通过send方法返回，NodeJs 原生没有
    res.send('hello world.')
})

// 4. 启动并监听端口
app.listen(port, () => {
    console.log(`server running at http://localhost:${port}`);
})
```

安装 nodemon 的方式，启动服务器

服务器文件名称 和 package.json 的 main 字段一致

启动服务器指令 Bash 复制代码

```bash
$ npm i nodemon -g

# 在指定的目录，启动项目
$ nodemon
```

3. 发送文件

通过 send 方法可以发送处理之后的数据，通过sendFile方法可以发送指定文件

send 方法 和 sendFile 方法 JavaScript 复制代码

```javascript
// 1. 引入模块
const express = require('express');
// const path = require('path');

// 2. 创建服务器对象
const app = express();

// 2.1 监听根目录的请求
app.get('/', (req, res) => {
    // 直接的字符
    // res.send('ok!')
    // 数组或者对象
    // res.send([12, 45, 23, 55, 78]);
    // res.send({ a: 10, b: 20 });
    // 给JSON字符串 - 和上述效果一直，也就是默认会被解析
    // res.send(JSON.stringify({ a: 10, b: 20 }));
    // 给 buffer - 二进制会直接走下载方式
    // res.send(Buffer.from('hello'));

    // sendFile()   返回文件
    // res.sendFile(path.join(__dirname, 'static/index.html'))
    res.sendFile('./static/index.html', { root: __dirname });
})

// 3. 监听并启动服务器
app.listen(8080, () => {
    console.log('server running at http://localhost:8080');
})
```

4. 静态资源托管

express.static(指定托管的目录);

app.use() 使用中间件或者某个写好的模块;

静态资源托管的功能 JavaScript 复制代码

```
// 1. 引入模块
const express = require('express');

// 2. 创建服务器对象
const app = express();

// 2.1 托管静态资源目录
// express.static('指定托管的目录')
// 使用这个方式  app.use()
app.use(express.static('./static'));

// 3. 启动并监听
app.listen(8080, () => {
    console.log('server running at http://localhost:8080');
})
```

5. 使用模板的方式

5.1 ejs 模板

在项目内，下载ejs的模板， **不需要引入**

```
$ npm i ejs
```

JavaScript | 复制代码

```
// 2.1 设置使用ejs渲染的引擎方式
// 固定写法的参数1：  view engine
// 参数2：采用渲染模板的类似
app.set('view engine', 'ejs');
// 2.2 设置要托管或者解析的指定目录
// 固定的参数1：views
// 参数2: 指定的目录名称
app.set('views', './views')

// 2.3 设置单独监听的方式
app.get('/', (req, res) => {

    // 2.4 使用 render 方法替代了 send 方法
    // 合二为一，一是加载指定名称的模板文件，二是传入参数
    // 参数1：  指定渲染的模板名称
    // 参数2：  指定传入的对象，对象中的键，可以直接使用
    res.render('index', { title: '请问麻鹏辉在家嘛？ ' })
})
```

文件名称在 ./views/index.ejs ，先使用html，创建了页面之后再改成的 ejs

index.ejs | HTML | 复制代码

```
<h2>
    <!-- 固定的语法   传入过来的字段，可以直接使用 -->
    <%= title %>
</h2>
```

如果需要加载其他的资源文件，则还是需要开启静态资源托管

但是顺序一定显示 托管了静态资源文件，之后，再加载 ejs

JavaScript | 复制代码

```
// 静态资源托管所有的文件
app.use(express.static('./views'));
// 2.1 设置使用ejs渲染的引擎方式
// 固定写法的参数1： view engine
// 参数2：采用渲染模板的类似
app.set('view engine', 'ejs');
// 2.2 设置要托管或者解析的指定目录
// 固定的参数1：views
// 参数2: 指定的目录名称
app.set('views', './views')

// 2.3 设置单独监听的方式
app.get('/', (req, res) => {
    res.render('index', { title: '麻鹏辉去网吧了！ ' })
})
```

如果要请求多个文件

先声明监听的模板文件 JavaScript | 复制代码

```
// 2.3 设置单独监听的方式
app.get('/', (req, res) => {
    res.render('index', { title: '麻鹏辉去网吧了！ ' })
})

// 2.4 请求 demo.html
// app.get('/demo.html', (req, res) => {
app.get('/demo', (req, res) => {
    res.render('demo', { title: '加载成功' })
})
```

根据声明的路径，发起请求

HTML | 复制代码

```
<!-- <a href="./demo.html">跳转文件</a> -->
<a href="/demo">跳转文件</a>
```

创建 demo.ejs 文件

```HTML
<p>
    <%= title %>
</p>
```

5.2 ejs 模板的语法

分支语法

```HTML
<li>
    <span>商品优惠：</span>
    <em>
        <!-- 以三元表达式的方式直接输出 -->
        <%= goods.youhui ? goods.youhui : '暂无优惠' %>
    </em>
</li>

<% if(goods.gifts) { %>
    <li>
        <span>赠送礼品：</span>
        <em>
            <%= goods.gifts %>
        </em>
    </li>
<% } %>
```

循环语法

循环语法的结构 HTML 复制代码

```
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>用户名称</th>
            <th>年龄</th>
            <th>性别</th>
        </tr>
    </thead>
    <tbody>
        <% for(var i=0; i < arr.length; i++) { %>
            <tr>
                <td>
                    <%= arr[i].id %>
                </td>
                <td>
                    <%= arr[i].uname %>
                </td>
                <td>
                    <%= arr[i].age %>
                </td>
                <td>
                    <%= arr[i].gender %>
                </td>
            </tr>
            <% } %>
    </tbody>

    <tbody>
      <% arr.forEach(item=> { %>
          <tr>
              <td>
                  <%= item.id %>
              </td>
              <td>
                  <%= item.uname %>
              </td>
              <td>
                  <%= item.age %>
              </td>
              <td>
                  <%= item.gender %>
              </td>
          </tr>
          <% }) %>
```

```
    </tbody>
  </table>
```

5.3 art-template 模板

官网地址：https://aui.github.io/art-template/zh-cn/

下载使用： `$ npm i art-template express-art-template`

JavaScript | 复制代码

```
// 2.1 静态资源托管 - 与以下渲染的方式有冲突
// app.use(express.static('./public'));

// 2.2 设置渲染引擎的方式
app.engine('html', require('express-art-template'));
// 2.3 定义引擎的内容
app.set('view engine', 'html');
// 2.4 定义文件夹路径
app.set('views', './public');
```

传入数据并且渲染

JavaScript | 复制代码

```javascript
// 2.5 创建一个根目录的请求
app.get('/', (req, res) => {
    res.render('index', {
        title: '小东的歌很哇塞',
        goods: {
            goods_name: 'iPhone 14 Pro',
            goods_price: 9980,
            goods_num: 20,
            brands: 'Apple',
            youhui: undefined,
            gifts: null
        },
        arr: [
            { id: 3002, uname: '迪丽热巴123', age: 22, gender: '女' },
            { id: 3003, uname: '古力娜扎', age: 23, gender: '女' },
            { id: 3004, uname: '欧阳娜娜', age: 24, gender: '女' },
            { id: 3005, uname: '马尔扎哈', age: 25, gender: '男' },
            { id: 3005, uname: '易烊千玺', age: 19, gender: '男' },
        ]
    });
})
```

HTML | 复制代码

```html
<ul>
    <li>
        <span>商品品牌: </span>
        <em>{{ goods.brands }}</em>
    </li>
    <!-- 判断的语法 -->
    <li>
        <span>商品优惠: </span>
        <em>{{ goods.youhui ? goods.youhui : '暂无优惠' }} </em>
    </li>
    {{ if goods.gifts }}
    <li>
        <span>赠送礼品: </span>
        <em>{{ goods.gifts }}</em>
    </li>
    {{ /if }}
</ul>
<br><br><br>
<table>
    <thead>
        <tr>
            <th>ID</th>
            <th>用户名称</th>
            <th>年龄</th>
            <th>性别</th>
        </tr>
    </thead>
    <tbody>
        <!-- $index  内部使用的 索引  $value 内部
        {{ each arr item i }}
        <tr>
            <td> {{ item.id }} - {{ i }} </td>
            <td> {{ item.uname }} </td>
            <td> {{ item.age }} </td>
            <td> {{ item.gender }} </td>
        </tr>
        {{ /each }}
    </tbody>
</table>
```

同时保留静态资源托管的能力和模板的渲染

需要把托管文件的文件夹对应的文件名称，html 的后缀名 改为 htm

JavaScript | 复制代码

```
// 2.1 静态资源托管 - 与以下渲染的方式有冲突
app.use(express.static('./public'));

// 2.2 设置渲染引擎的方式
app.engine('htm', require('express-art-template'));
// 2.3 定义引擎的内容
app.set('view engine', 'htm');
// 2.4 定义文件夹路径
app.set('views', './public');
```

6. 路由 Router

路由器，通过一个入口，可以连接多个设备，进行对应的网络连接和数据传输。

nodejs 中服务器就是一台路由器，能监听到所有的客户发起的请求，然后处理对应的响应。

广义上来说，路由器就是一个服务器，但是客户请求的资源千奇百怪

服务器如果没有定义，那么无法做出响应

那么定义的这些请求路径，一般会有专门指定的方式请求，都是服务器端定的规矩

一个简单的路由模块 JavaScript | 复制代码

```
app.post('/user/add', (req, res) => {})
```

需要有专门的模块，来批量的定义这些路径，这种路径有一个专门的名词，叫做后端接口

后端接口： 请求方式、请求地址(路径)、请求参数(请求体)、其他的cookies等...

理论上来说，客户端请求，服务器端响应，但是如果服务器端并没有做出相应的业务，那么其实是无法响应的

如果需要服务器端有正常的响应，那么就需要服务器端对应的生产相关的路由模块，那么不同种类的路由模块集合在一起，就会形成对应的路由部分，后端主要的工作就是完成这个事儿。

并不是客户端可以随便的发起请求，服务器端能用的请求，都是后端整理好的，在开发中有专门的接口文档，前端所发起的请求都是由后端的接口文档定义的，不是随便发起的

6.1 定义路由的模块

创建了 router 文件夹，声明了路由模块的文件

users.js JavaScript 复制代码

```
/**
 * 此模块用于声明关于用户的接口
 * */
// 1. 引入模块
const express = require('express');

// 2. 创建路由对象
const users = express.Router();

// 2.1 请求添加用户的接口
users.get('/users/add', (req, res) => {
    res.send('添加用户成功！')
})

users.get('/users/del', (req, res) => {
    res.send('删除用户成功！')
})

users.get('/users/list', (req, res) => {
    res.send('用户列表')
})

users.get('/users/edit', (req, res) => {
    res.send('修改用户成功！')
})

// 3. 导出模块
module.exports = {
    users
};
```

主路由模块进行挂载和使用

JavaScript 复制代码

```
// 3. 接收处理用户的路由模块
const { users } = require('./router/users');
// 使用路由模块
app.use(goods);
```

6.2 get 请求和post请求的参数

get请求：

页面加载，url访问，link，script，a标签，img标签，音频视频加载....

post请求：

form表单中 ...

自由请求(随便发)：

ajax... (后期讲)

JavaScript | 复制代码

```
// 2.1 请求添加用户的接口
// http://localhost:9527/users/add?id=20&flag=true&list=desc
users.get('/users/add', (req, res) => {
  // 通过get请求发起的查询参数，直接由  req.query 获取
  console.log(req.query);
  res.send('添加用户成功！')
})

// 2.2 删除用户的接口 - restful 风格的api
// http://localhost:9527/users/del/55/asc
users.get('/users/del/:id/:list', (req, res) => {
  // 通过get请求发起的查询参数，直接由  req.query 获取
  console.log(req.params);
  res.send('删除用户成功！')
})

// 2.3 post 添加用户
users.post('/users/add', (req, res) => {
  // 如果是post请求，那么请求内容在 req.body 中
  console.log(req.body);
  res.send('ok');
})
```

发送post请求 HTML 复制代码

```html
<form action="/users/add" method="post">
    <p>
        用户名:
        <input type="text" name="username">
    </p>
    <p>
        密码:
        <input type="text" name="password">
    </p>
    <button>提交</button>
</form>
```

6.3 解析post的内容

默认情况下，post 请求的内容，携带的参数在请求体中，无法直接获取

创建一个自定义的中间件，可以接收到所有传输得的数据

app.js 中完成 JavaScript 复制代码

```javascript
// 定义了一个中间件 - 中间件就是一个函数
// 函数能够用于拦截所有的请求，处理完成之后再交由下一个中间件处理
app.use((req, res, next) => {
    // 监听前端发送的请求体，接收片段数据
    let str = '';
    req.on('data', chunk => {
        str += chunk
    })
    req.on('end', () => {
        console.log(str, 2222);
        // 拦截了所有的请求，获取到了请求体中的数据
        req.body = str;
        // 完成之后要放行
        next();
    })
});
```

JavaScript 复制代码

```
// 不需要下载，内置就有，用于转换字符的键值，变为对象的形式
const qs = require('qs');

// 3.3 添加用户的请求  post 方式
users.post('/add', (req, res) => {
    console.log(req.body, 111);
    console.log(qs.parse(req.body));
    res.send('请求完成!')
})
```

7. 下载项目生成器

使用项目生成器，可以快速的创建服务器端的项目

```
$ npm i express-generator -g
```

指定目录即可创建项目

```
$ express myapp
```

进入目录

```
$ cd myApp
```

```
$ npm i
```

```
$ nodemon
```

自带类似于 body-parser 的中间件，好处是可以直接获取post的请求体

body-parser JavaScript 复制代码

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

// 声明接收post请求体内容的片段
app.use(bodyParser.urlencoded({ extended: false }));
app.use(bodyParser.json());
```

express JavaScript 复制代码

```
app.use(express.json());
app.use(express.urlencoded({ extended: false }));
```

```
bin     ----        生成服务器和相关端口及错误处理的文件夹
    -   www         创建服务器的文件
node_modules        下载的模块包    - 基本不看
public  ----        前端工作目录
routes  ----        子路由模块

    -  index.js
    -  users.js
views   ----        模板加载的文件夹 - 基本不用
app.js  ----        主文件，项目启动文件
package.json  ----    包管理文件
```

8.0 中间件

中间件就是函数，函数包含了三个形参， req, res, next

中间件描述的就是请求到响应的不断循环操作间，每次请求到开始响应的这个过程，可以被拦截到处理完之后，通过next函数，放行到下一个中间件

中间件使用 app.use 的方式注册；

- Application-level middleware　　**应用级别的中间件**　自定义的拦截器，或者每一个接口
- Router-level middleware　　路由级别的中间件　app.use('/users', users);
- Error-handling middleware　　错误处理的中间件（用的不多，写的比较少）　=>　error
- Built-in middleware　　**内置的中间件** (唯一内置的中间件) => express.static()
- Third-party middleware　　第三方中间件 (需要下载的)　=>　body-parser

9.0 上传文件

上传文件必须要使用form表单，发送请求由于要携带文件，所以必须是post请求

请求头，需要发生变化

常见的 请求头 – 给 Content-Type

application/x-www-form-urlencoded

默认的，自动被解析为 name=zs&age=20&id=3

multipart/form-data

需要设置的，上传文件的时候使用，在表单中只需要设置enctype属性

application/json

请求体传入的内容是json格式

上传文件的结构 HTML 复制代码

```HTML
<div class="container">
    <!-- 请求方式是 POST  请求地址是后端写的接口  enctype 设置为上传文件的请求头 -->
    <form action="/uploads" method="POST" enctype="multipart/form-data">
        <p>
            图片说明：<input type="text" name="title">
        </p>
        <!-- 后端通过name能够知晓要获取什么内容 -->
        <p>
            上传文件：  <input type="file" name="demo">
        </p>
        <button>确定上传</button>
    </form>
</div>
```

使用第三方中间件，解决上传文件的需求

```
$ npm i multer
```

JavaScript | 复制代码

```javascript
const multer = require('multer');
const path = require('path');


// 声明一个上传对象的相关信息
const storage = multer.diskStorage({
    // 设置文件存储目录
    destination(req, file, cb) {
        // 指定目录存放上传的文件
        cb(null, './public/uploads');
    },
    // 设置文件上传的相关信息
    filename(req, file, cb) {
        // 从上传的文件中获取文件的后缀名
        let ext = path.extname(file.originalname);
        // 拼接一个随机的名称出来，避免重复
        cb(null, file.filename + '_' + Date.now() + ext);
    }
})

// 创建要上传的操作对象
const upload = multer({ storage });

// 2.2 上传文件的路由接口
// 参数2的位置上，写入要上传的文件字段，可以为多个
app.post('/uploads', upload.single('demo'), (req, res) => {

    // 根据上传的结果，返回具体的详细信息
    res.send('上传成功！')
})
```

方式2：

使用 formidable 中间件，也可以完成文件的上传操作

```
$ npm i formidable
```

JavaScript | 复制代码

```javascript
const formidable = require('formidable');

// 2.2 上传文件的路由接口
app.post('/uploads', (req, res) => {

    // 1. 创建表单对象
    const form = formidable({
        // 设置要保存的文件目录 - 最好不要用path拼
        uploadDir: './public/uploads',
        // 保留文件的后缀名
        keepExtensions: true
    })

    // 2. 解析表单
    // fields 字段：一般表单元素的信息
    // files  字段：上传的文件信息
    form.parse(req, (err, fields, files) => {
        if (err) throw err;

        // 响应上传成功之后的内容
        res.send('上传成功！')
    })

})
```