Express

基于 Node.js 平台，快速、开放、极简的 Web 开发框架

开始了解 →

Express是什么？

官方描述：基于 Node.js 平台，快速、开放、极简的 Web 开发框架。

基于表单、模版渲染等驱动

金额

金额

部门

部长团

电子发票

浏览... 未选择文件。

提交 清空

功能相对受限，如较难实现主动推送消息、显示上传进度等复杂的功能； 界面和业务逻辑耦合性相对高

基于AJAX驱动

AJAX

Async JavaScript And XML

指不依赖于浏览器本身的标签、跳转、表单等机制，而是使用JavaScript提供的异步网络请求API，根据具体需要执行具体的网络请求，和自由的处理和显示返回结果。

主要特点：

- 前后端分离
- 前端：与渲染显示有关的代码，主要控制显示和一些与显示有关必要计算。通常以静态文件的形式提供给用户。
- 后端：存储主要的业务逻辑和业务数据。通常以API的形式组织。
- 前后端之间：通过网络请求（API调用），交换数据。
- 打个比方：前端相当于显示器、鼠标键盘；后端相当于CPU和硬盘。

HTTP请求：RESTful API

Representational State Transfer

- 请求：
  - URL: api.example.com/login
  - Method: **GET**、**POST**、PUT、DELETE、...
  - Headers: Authorization、Cookie、X-xxxxx、...；Content-Length、Content-Type、Origin、...
  - Body: 只有POST、PUT等请求才应该带请求Body。
- 响应：
  - Status Code：200 OK，404 Not Found，400 Bad Request，... https://developer.mozilla.org/zh-CN/docs/Web/HTTP/Status
  - Headers
  - Body：所有响应都可以（应当）带响应Body。

Postman

发送网络请求的工具，Web开发调试利器

从零开始

建立express的实例（服务器），并监听于指定的端口上：

```
const express = require('express')
const app = express() // 创建express的实例。该实例下定义了listen方法，即可在指定端口上建立http服务。
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Example app listening at http://localhost:${port}`)
})
```

然后执行`node app.js`

更好的开始

```
npx express-generator 项目名
cd 项目名
npm install
npm start
```

```
npx: 10 安装成功，用时 2.865 秒

  warning: the default view engine will not be jade in future releases
  warning: use `--view=jade' or `--help' for additional options


   create : board/
   create : board/public/
   create : board/public/javascripts/
   create : board/public/images/
   create : board/public/stylesheets/
   create : board/public/stylesheets/style.css
   create : board/routes/
   create : board/routes/index.js
   create : board/routes/users.js
   create : board/views/
   create : board/views/error.jade
   create : board/views/index.jade
```

中间件（请求处理函数）

```
app.get('/',
(req, res) => {
  res.send('Hello World!')
})
```

- 中间件的两种类型：
  - (req, res, next) 正常处理逻辑（参数<=3个）>
  - (err, req, res, next) 错误处理逻辑（参数>=4个）
- 中间件的注册：
  - app.use(中间件, 中间件, ...) // 全局生效
  - app.use(路径, 中间件, 中间件, ...) // 只在指定的请求路径下生效
- 多个中间件，按照注册的先后顺序被调用

Request对象

描述了与用户发来的请求有关的信息。

- query: 以键值对的形式存储query（URL中?之后的部分）
- headers: 待确认
- body: 存储请求的请求体（格式是什么？参见后文）
- protocol
- path
- cookies
- … https://www.expressjs.com.cn/4x/api.html#req

Response对象

- status(code): 设置返回码（若不设置，默认为200）
- send(xxx): 发送内容（字符串、字节数组Buffer等）
- json(object): 将指定对象序列化为json之后发送回去
- sendFile(filename): 将指定磁盘文件的内容发送回去
- end(): 结束请求。通常用于不希望发送任何内容的情况
- set(key， value): 设置header
- cookie(): 设置cookie
- … https://www.expressjs.com.cn/4x/api.html#res

报错模板分析

```
// catch 404 and forward to error handler
app.use(function(req, res, next) {
  next(createError(404));
});

// error handler
app.use(function(err, req, res, next) {
  // set locals, only providing error in development
  res.locals.message = err.message;
  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page
  res.status(err.status || 500);
  res.render('error'); // 渲染模板，本次课程不会涉及，感兴趣的同学可自行查阅学习
});
```

路由

你可能已经注意到了，我们刚才用express-generator创建的项目中存在这样的结构：

app.js

```
var indexRouter = require('./routes/index');
var usersRouter = require('./routes/users');

app.use('/', indexRouter);
app.use('/users', usersRouter);
```

routes/users.js

```
var router = express.Router();

router.get('/list', function(req, res, next) {
  res.send('user lists');
});

router.post('/login', function(req, res, next) {
  res.send('login');
});
```

body-parser

- 如果你只是建立一个express实例，那么，收到的所有请求的body都为undefined
- generator创建的代码中以注册中间件的形式为我们提供了常见的body-parser

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

其他

```
app.use(cookieParser()); // 自动帮我们处理cookie，否则cookie属性会是undefined
app.use(express.static(path.join(__dirname, 'public'))); // 使得放在public文件夹下的文件可以直接访问
```

一个简单的网站：公共展示板

- 需求分析：
  - 所有用户可以查看到所有的公告信息
  - 用户可以注册和登录
  - 已登录的用户可以发布新的公告，也可以编辑自己发布过的公告
  - 管理员可以编辑任何公告，和删除任何公告
- 代码可从 https://github.com/Starrah/2021summerexercise-board 获取 (MIT License)

用户注册、登录

components/users.ts

```
export let secret = "d41d8cd98f00b204e9800998ecf8427e"

export interface User {
    name: string
    password: string
    permission: string
}

export const users: Record<string, User> = {
    "root": {
        name: "root",
        password: "123456",
        permission: "admin",
    }
}
```

用户注册、登录

routes/users.ts

```
import { users, secret } from "../components/user"
import { Request, Response, Router } from "express"
import { sign } from "jsonwebtoken"

export const router = Router();

router.post('/logon', function (req: Request, res: Response) {
  let { name, password } = req.body
  if (users[name]) {
    res.status(400)
    res.json({ status: "fail", err: "user exists!" })
    return
  }
  users[name] = {
    name, password, permission: "normal"
  }
  res.json({ status: "success" })
});

router.post('/login', function (req: Request, res: Response) {
  let { name, password } = req.body
  if (!users[name]) {
```

公告信息-基础定义

components/posts.ts

```
export interface Post {
    id: number,
    username: string,
    timestamp: number,
    content: string
}

let count = 0

export function getAndIncreaseCount() {
    return count++
}

export const posts: Record<number, Post> = {}
```

公告信息-增删改

routes/posts.ts

```
import { Request, Response, Router } from "express"
import jwt from "express-jwt"
import moment from "moment";
import { posts, getAndIncreaseCount, Post } from "../components/posts"
import { secret, User, users } from "../components/users"


export const router = Router();


router.post('/', jwt(secret), function (req: any, res, next) {
  let post = {
    id: getAndIncreaseCount(),
    username: req.user.name,
    timestamp: Date.now(),
    content: req.body.content
  }
  posts[post.id] = post
  res.json({ status: "success", id: post.id })
});

router.put('/:id', jwt(secret), function (req: any, res, next) {
```

公告信息-获取

routes/posts.ts

```
function parsePost(post: Post) {
  return {
    username: post.username,
    time: moment(post.timestamp).format(),
    content: post.content
  }
}

router.get('/:id', function (req: any, res, next) {
  let id = req.param.id
  let post = posts[id]
  if (!post) {
    res.send(404)
    res.json({ status: "fail", err: "not found" })
  }
  let result = parsePost(post)
  res.send({ status: "success", post })
});

router.get('/', function (req: any, res, next) {
  let id = req.param.id
  let result = Object.values(posts).map(parsePost)
```

基于文件的动态式路由

express-router-dynamic

目录结构就是路由结构，文件的增删改动态应用到服务器中，真正所见即所得

```
import {dynamicRouter} from 'express-router-dynamic'

app.use("/", dynamicRouter({
  realPrefix: ["./routes"],
  libPrefix: [],
  autoIndex: ["index", "index.html", "index.js"],
  ignore: [
    '*.ts',
    '*.map',
    '*.json',
    /config/,
  ],
}))
```

基于文件的动态式路由

现实中使用的动态路由结构示例：

微信公众平台后端开发

express-wx

在express服务器的基础上，构建一个WXRouter对象：

```
var Express = require("express")
var Express_WX = require("express-wx")

// 构造Express的实例
var app = Express()

// 构造一个WXRouter的实例（使用express-wx包的WXRouter(config)函数，需要传入config）
var wxRouter = Express_WX.WXRouter({
  // config中只有appInfo是必填的，appInfo只有token字段是必填的。
  // 如果对更多的config配置感兴趣，请阅读WXRouterConfig接口上的注释（见bld/config.d.ts文件）
  appInfo: {
    token: "someToken" // 您需要保证在微信公众平台设置的Token和这里设置的Token一致
  },
  debugToken: "test"
})

// 把wxRouter对象注册到app上
app.use("/", wxRouter)

app.listen(8080) // 监听8080端口
```

微信公众平台后端开发

express-wx

同样支持动态路由加载。只需要在配置的handlersDir目录中放置请求处理的函数（中间件）即可。

被传入上述请求处理函数中的Request对象会比原生的express对象多出一个wx属性，用于直接获取到消息的内容、发送者等； Response对象会多出wx(WXMessage)、wxText(string)等方法，用于回复消息。

另外在WXRouter对象（通过req.wxRouter即可获得引用）上另有sendCustomMessage、sendTemplateMessage、signJSAPI等方法，可以发送客服消息、模版消息、微信JS-SDK签名等。

详见：https://github.com/Starrah/express-wx

微信公众平台后端开发

express-wx

```
var Express = require("express")
import {TextWXMessage, WXHandler} from "express-wx";

// 向WXHandler(fn)函数传入一个参数，参数的形式即为标准的express的(req, res, next)形式。
// 事实上您不使用WXHandler函数，而是直接提供一个普通函数也是可以的；但特别是当您使用TypeScript时还是强烈建议使用WXHandler，
// 因为这样可以获得更充分的类型提示。
export default WXHandler((req, res, next) => {
  // req是WXRequest类型的对象，它除了具有一般的express请求对象的接口以外，还额外有wxRouter属性（即当前WXRouter的实例，
  // 您可以从中获得公众号信息），
  // 和wx属性（类型为WXMessage，您可以从中获得用户发来的消息。）
  // WXMessage有很多种可能的类型、比如文本消息TextMessage类型；其上的text方法即是用户发来的消息文本。
  // 下面的语句把收到的消息打印出来。
  console.log("收到消息：" + (req.wx as TextWXMessage).text)
  //使用res.wx(WXMessage)方法可以回复消息给用户
  // 特殊的为了方便，回复文字内容时可以直接使用更简便的res.wxText(string)。
  // 下面的语句向用户回复消息"Hello World"。
  res.wxText("Hello World")
})
```

微信公众平台后端开发

express-wx

```
import {TextWXMessage, WXHandler} from "express-wx";
import * as moment from "moment"

// 这个处理逻辑用于在用户发送的文本带有“时间”两字时，向用户回复当前的时间。
var handler = WXHandler((req, res, next) => {
    if (req.wx instanceof TextWXMessage && req.wx.text.indexOf("时间") !== -1) {
        // 仅当收到的消息是文本消息且文本内容中含有“时间”二字时
        // 返回当前日期和时间的格式化字符串，使用moment库
        res.wxText(moment(new Date()).format('YYYY-MM-DD HH:mm:ss'))
    }
    else next() // 否则，则本函数不处理此消息，调用next函数交给后续的人处理。
})
// 设置一下优先级
// 因为我们有另一个回复一切请求的hellowrorld.js，如果helloworld比这个函数先被调用就无法实现回复当前时间的功能了。
// 一个handler在没有特殊设置的情况下默认优先级为0，因此我们要把上面这个handler的优先级设置为一个大于0的数
handler.priority = 10

export default handler
```

Websocket

express-ws

在app.js中，通过用express-ws对象调用app，将ws方法注入app和router类型中：

```
import * as ExpressWS from 'express-ws'
ExpressWS(app)
```

路由函数：

```
router.ws("/", async (ws, req, next) => {
  ws.on('message', (msg)=>{
    ws.send(`server received ${msg}`)
  })
  while (ws.readyState == ws.CONNECTING || ws.readyState == ws.OPEN) {
    if (ws.readyState == ws.OPEN) ws.send(new Date().toISOString())
    await delay(2000)
  }
})
```

测试API（公开开放使用）： wss://api.starrah.cn/wstest

CDN