

Express.js

Node Web 框架

版权声明

本内容版权属杭州饥人谷教育科技有限公司（简称饥人谷）所有。

任何媒体、网站或个人未经本网协议授权不得转载、链接、转贴，或以其他方式复制、发布和发表。

已获得饥人谷授权的媒体、网站或个人在使用时须注明「资料来源：饥人谷」。

对于违反者，饥人谷将依法追究 responsibility。

联系方式

如果你想要购买本课程

请微信联系 [xiedaimala02](#) 或 [xiedaimala03](#)

如果你发现有人盗用本课程

请微信联系 [xiedaimala02](#) 或 [xiedaimala03](#)

HTTP 协议

万维网的基石

curl -v

- **curl -s -v 网址**

- ✓ -s 是 silent, 用于隐藏进度条
- ✓ -v 是 verbose, 用于打印全部 header
- ✓ * 开头的是注释
- ✓ > 开头的是 HTTP 请求
- ✓ < 开头的是 HTTP 响应

- **举例**

- ✓ curl -s -v <http://xiedaimala.com>
- ✓ 得到 301 和 Location, 于是重新请求 (-L 自动重定向)
- ✓ curl -s -o nul -v <https://xiedaimala.com>
- ✓ -o nul 是为了隐藏 HTML 文本, 内容太多不方便演示
- ✓ Linux 或 mac 要将 nul 改成 /dev/null

请求和响应

- 请求

- ✓ `POST /xxx HTTP/1.1`
- ✓ `Host: xiedaimala.com`
- ✓ `User-Agent: curl/7.61.1`
- ✓ `Accept: */*`

- ✓ `{"username":"frank"}`

- ✓ 分为四部分

- ✓ 一、请求行

- ✓ 二、请求头

- ✓ 三、回车

- ✓ 四、请求体/消息体

- 响应

- ✓ `HTTP/1.1 301 Moved Permanently`
- ✓ `Content-Type: text/html`
- ✓ `Content-Length: 193`
- ✓ `Location: https://xiedaimala.com/`

- ✓ `<!DOCTYPE html>`

- ✓ `<html>`

- ✓ `<head>...</head>`

- ✓ `<body>...</body>`

- ✓ `</html>`

- ✓ 分为四部分

- ✓ 一、状态行

- ✓ 二、响应头

- ✓ 三、回车

- ✓ 四、响应体/消息体

请求和响应

- 请求

- ✓ 如果请求体的内容为 JSON
- ✓ 那么请求头就要有 Content-Type: application/json
- ✓ 这一规范可以[在 MDN 查看](#)

- 响应

- ✓ 如果响应体的内容为 JSON
- ✓ 那么响应头就要有 Content-Type: application/json

- HTTP 的复杂性

- ✓ HTTP 复杂就复杂在它有很多请求头和响应头
- ✓ 每个请求头或响应头功能各不相同，我们遇到再作了解

Web 框架

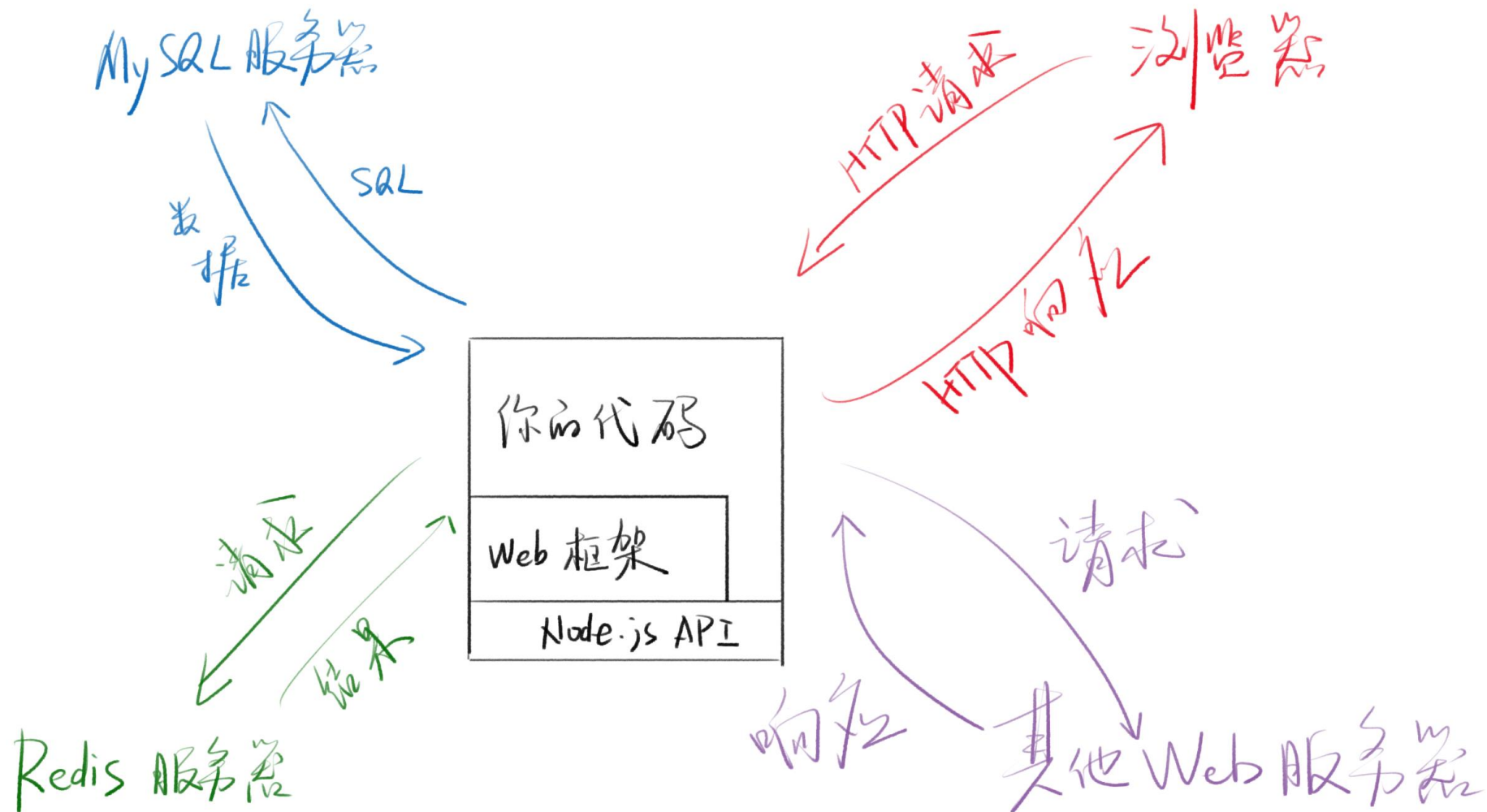
- 功能

- ✓ 更方便地处理 HTTP 请求与响应
- ✓ 更方便地链接数据库、Redis
- ✓ 更方便的路由
- ✓ 其他：HTML 模板

- 理念

- ✓ Web 框架的主流思路都是 MVC
- ✓ Model 处理数据相关逻辑
- ✓ View 处理视图相关逻辑，前后分离之后，View 不重要
- ✓ Controller 负责其他逻辑

架构示意



处理 HTTP 请求与响应

- 最简单的封装

- ✓ 将请求封装为 `['get', '/xxx'], {请求头}, '请求体'`
- ✓ 将响应封装为 `[status, {响应头}, '响应体']`

- Node.js 的封装

- ✓ 封装在 http 模块中了
- ✓ 使用 `request` (`IncomingMessage` 的实例) 读取请求
- ✓ 使用 `response` (`ServerResponse` 的实例) 设置响应

- Express 的封装

- ✓ 封装级别高一点点，只需理解 Express 的编程模型即可
- ✓ [中文文档](#)在此

express-demo-1

- 创建项目

- ✓ 创建目录，用 WebStorm 或者 VSCode 打开
- ✓ yarn init -y; git init;
- ✓ 添加 .gitignore 文件
- ✓ 提交到 git，推送至 GitHub

- 开始学习 express

- ✓ CRM 学习法：Copy - Run - Modify

Hello World

- 安装 `express`

- ✓ `yarn add express`
- ✓ 或者 `npm i express`
- ✓ 上面两个命令二选一，不要混用

- 创建 `app.js`

- ✓ 内容 Copy 自文档
- ✓ 然后 Run 一下 `app.js`，命令为 `node app.js`
- ✓ 打开 <http://localhost:3000> 预览
- ✓ 最好 Modify 几处代码，比如改内容、路径和端口

用 CRM 学到了什么

- `app = express()`
 - ✓ 这个 app 应该是核心
 - ✓ `app.get('/xxx', fn)` 用于对 GET /xxx 请求做出相应
 - ✓ `app.listen(3000, fn)` 开启端口监听

使用 TypeScript

- 准备工作

- ✓ yarn global add typescript ts-node 全局安装工具
- ✓ yarn add @types/express 安装类型支持
- ✓ tsc --init
- ✓ 修改 tsconfig 的 target 和 noImplicitAny
- ✓ 将 require 改为 import

- 运行

- ✓ ts-node app2.ts

app 的类型

- 使用 IDE 查看类型

- ✓ 用 VSCode 或 WebStorm 可以查看 app 对象的类型
- ✓ 类型为 Express 接口
- ✓ Express extends Application
- ✓ Application extends EventEmitter, IRouter, ...
- ✓ 其中 IRouter 包含了 get/post/put 等方法
- ✓ 有了 TypeScript, 都不用看文档了

- 题外话

- ✓ 当前项目是一个配置的不错的学习环境
- ✓ 建议将其单独上传到 [express-starter-1](#)
- ✓ 方便之后的项目直接 git clone

脚手架

一键搞定项目目录

express-generator

- 安装

- ✓ yarn global add express-generator

- 使用

- ✓ express -h 查看帮助
- ✓ express --view=ejs . 注意有一个点
- ✓ 这句话用于创建文件，点表示当前目录
- ✓ 由于它会覆盖文件，所以要重新安装 @types/express

- CRM 学习法

- ✓ yarn install; yarn start
- ✓ 分析 app.js，主要 API 为 app.set 和 app.use
- ✓ app.set 用于改配置，app.use 用于使用中间件
- ✓ 记得提交可运行的代码，防止后面改出问题

改为 TypeScript

• 改写

- ✓ 把 app.js 复制到 app.ts
- ✓ yarn add @types/node --dev 这样才能使用 require
- ✓ 你也可以用 import 代替 require
- ✓ 你也可以用 const 代替 var
- ✓ 需 RequestHandler 和 ErrorRequestHandler 断言
- ✓ 将 bin/www 中的入口改为 app.ts
- ✓ 添加 start:ts 脚本，将 node 改为 ts-node

• 疑问

- ✓ 为什么 ts-node 可以运行 JS
- ✓ 答：本来就可以呀，只是添加了对 TS 的支持
- ✓ 注意不要在生产环境这样用

总结

- 运行 Hello World

- ✓ 了解 app.set 和 app.use
- ✓ 改为 TypeScript, 熟悉了一下 TypeScript

- 使用脚手架

- ✓ express --view=ejs .
- ✓ 了解 express 的目录结构, 了解 MVC
- ✓ 改为 TypeScript, 理解了 TS 的断言
- ✓ 目前为止所有代码: [链接](#)

- 接下来

- ✓ 深入理解 express 的编程模型
- ✓ 从 app.use 开始

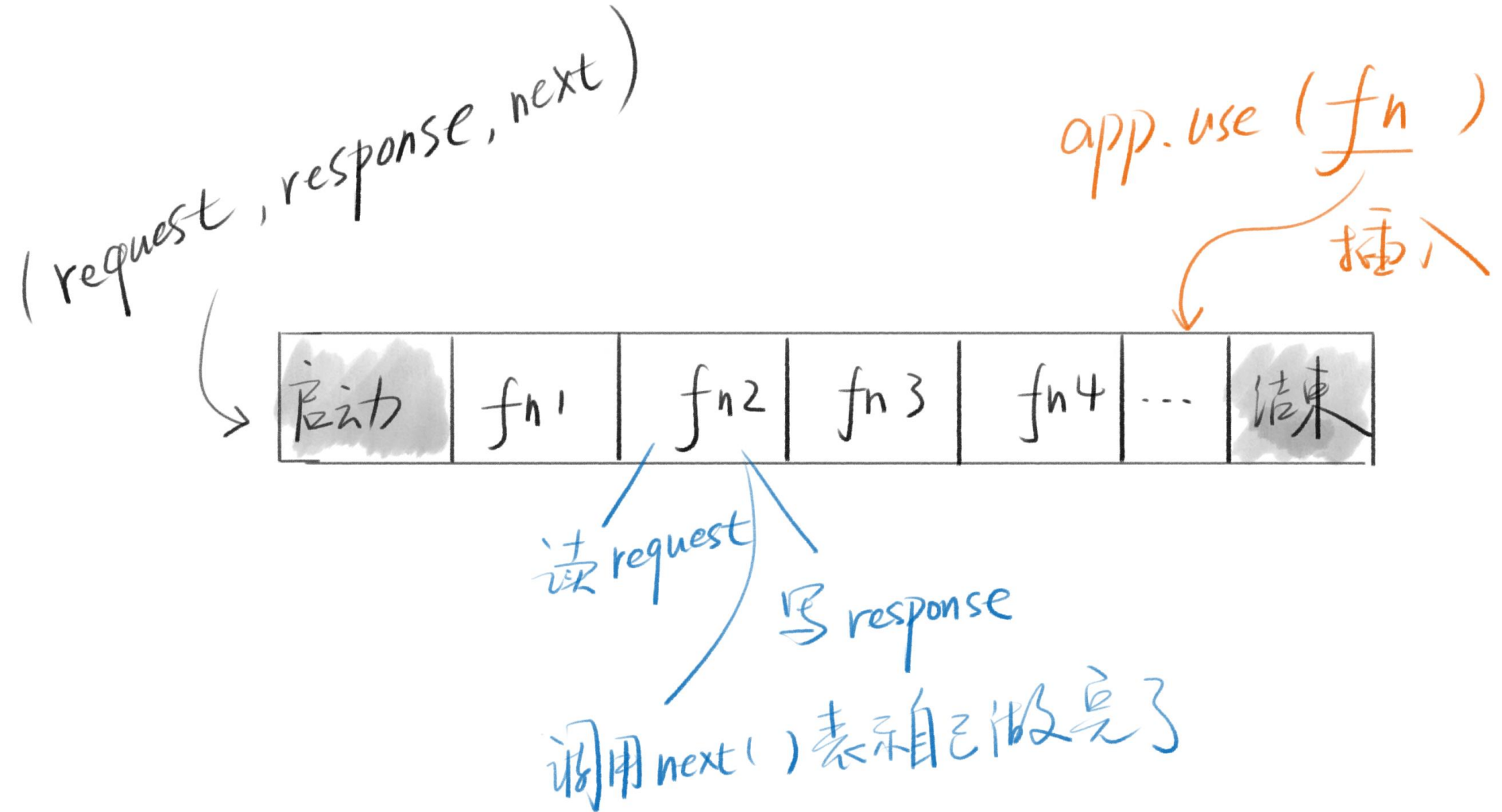
app.use()

use 是使用，请问使用什么

理解 app.use

- 创建新目录 `express-demo-2`
 - ✓ 使用 `express-starter-1`
 - ✓ 尝试使用 `req.url` 和 `res.send`
 - ✓ 多次使用会怎样？会报错
 - ✓ 改成 `res.write`，还记得流吗。
 - ✓ 为什么不会关闭呢？加上 `res.end()` 试试
 - ✓ `next` 什么时候可以省略（自行测试看看）

express 的编程模型



中间件

fn 就是中间件，因为它是
被插入到启动和结束中间的物件

优点

- 模块化

- ✓ 这种模型使得每个功能都能通过一个函数实现
- ✓ 然后通过 `app.use` 将这个函数整合起来
- ✓ 如果把函数放到文件或 npm 里，就实现了模块化

- 以 `express-demo-1` 举例

- ✓ `app.use(logger('dev'));`
- ✓ `logger('dev')` 会返回一个函数
- ✓ 这个函数会在每次请求到达的时候，打印出信息
- ✓ 我们根本就不用去了解它是如何做到的
- ✓ 我们也可以很快做出一个类似的模块

路由

- 使用 `app.use` 如何实现路由

```
app.use((req, res, next) => {  
  if (req.path === '/xxx' && req.method === 'get') {  
    res.write('home');  
  }  
  next();  
});
```

- 更方便的写法

- ✓ `app.use('/xxx', fn)`
- ✓ `app.get('/xxx', fn)`
- ✓ `app.post('/xxx', fn)`
- ✓ `app.route('/xxx').all(f1).get(f2).post(f3)`
- ✓ 这些都是 API 糖

错误处理

- `next()` 能传参数吗?

- ✓ 你可以看文档，也可以看 TypeScript 定义
- ✓ 推荐后者

- `next(error)`

- ✓ 会直接进入 errorHandler，不执行后面的中间件
- ✓ errorHandler 的默认实现见[文档](#)

- 如何自定义 errorHandler

- ✓ 还是看文档，文档说一般在最后定义
- ✓ `app.use((err, req, res, next) => {})`
- ✓ 可以定义多个这样的中间件

next('route')

- 这是一个特殊参数

- ✓ 见[源码 lib/router/route.js](#) 中 next 函数的定义
- ✓ 我本人是没用过这个参数啦
- ✓ 大家可以看看文档的例子，用到的时候再细说
- ✓ 这里主要是教大家如何看源码

恭喜

express.js 的核心就是这些

再见

下节课了解 `express.js` 其他功能