Express.js 快速参考

路由、中间件、请求、响应、常用模式

初始化

创建并启动服务器

```
const express = require("express");
const app = express();
app.listen(3000, () => console.log("Running on :3000"));
```

内置中间件

```
app.use(express.json());          // parse JSON bodies
app.use(express.urlencoded({ extended: true })); // form data
app.use(express.static("public")); // serve static files
```

路由

HTTP 方法

```
app.get("/users", (req, res) => res.json(users));
app.post("/users", (req, res) => res.status(201).json(req.body));
app.put("/users/:id", (req, res) => res.json(updated));
app.delete("/users/:id", (req, res) => res.sendStatus(204));
```

路由参数

```
app.get("/users/:id", (req, res) => {
  const { id } = req.params;
  res.json({ id });
});
```

查询字符串

```
// GET /search?q=express&page=2
app.get("/search", (req, res) => {
  const { q, page } = req.query;
  res.json({ q, page });
});
```

中间件

应用级中间件

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

路由级中间件

```
const auth = (req, res, next) => {
  if (!req.headers.authorization) return res.sendStatus(401);
  next();
};
app.get("/secret", auth, (req, res) => res.json({ ok: true }));
```

执行顺序

app.use(fn)	每次请求都执行（按注册顺序）
app.use(path, fn)	仅在匹配路径前缀时执行
next()	将控制权传给下一个中间件
next(err)	跳转到错误处理器

请求与响应

Request 对象

req.params	路由参数（**/users/:id**）
req.query	查询字符串（**?key=val**）
req.body	解析后的请求体（需解析器）
req.headers	请求头对象
req.method	HTTP 方法（GET、POST…）
req.path	URL 路径名
req.cookies	Cookie（需 cookie-parser）

Response 对象

res.json(obj)	发送 JSON 响应
res.send(body)	发送字符串/Buffer/对象
res.status(code)	设置 HTTP 状态码（可链式）
res.redirect(url)	302 重定向（可传状态码）
res.sendFile(path)	将文件作为响应发送
res.sendStatus(code)	发送状态码与默认文本
res.set(header, val)	设置响应头

错误处理

错误中间件

```
// Must have 4 parameters — Express recognizes it as error handler
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({ error: err.message });
});
```

错误处理器必须在所有 app.use() 和路由之后定义

异步错误

```
// Wrap async route handlers to catch rejections
const wrap = (fn) => (req, res, next) =>
  Promise.resolve(fn(req, res, next)).catch(next);

app.get("/data", wrap(async (req, res) => {
  const data = await fetchData();
  res.json(data);
}));
```

静态文件

托管静态目录

```
app.use(express.static("public"));
// serves public/style.css at /style.css

// With virtual path prefix
app.use("/assets", express.static("public"));
// serves public/style.css at /assets/style.css
```

选项

dotfiles	**'ignore'** | **'allow'** | **'deny'**
maxAge	Cache-Control max-age（毫秒）
index	索引文件名（默认：**index.html**）
fallthrough	404 时传给下一个中间件

模板

视图引擎配置

```
app.set("view engine", "ejs");
app.set("views", "./views");

app.get("/", (req, res) => {
  res.render("index", { title: "Home", items: [1, 2, 3] });
});
```

常用模板引擎

ejs	嵌入式 JS 模板（**<%= val %>**）
pug	缩进式（前身 Jade）
handlebars	Mustache 风格（**{{val}}**）

Router

模块化路由

```
// routes/users.js
const router = require("express").Router();
router.get("/", (req, res) => res.json(users));
router.get("/:id", (req, res) => res.json(user));
module.exports = router;
```

挂载 Router

```
const usersRouter = require("./routes/users");
app.use("/api/users", usersRouter);
// GET /api/users -> router's "/"
// GET /api/users/5 -> router's "/:id"
```

Router 方法

router.get/post/put/delete	HTTP 方法处理器
router.use(fn)	Router 级中间件
router.param(name, fn)	预处理路由参数
router.route(path)	在一个路径上链式注册方法

认证模式

JWT 中间件

```
const jwt = require("jsonwebtoken");
const auth = (req, res, next) => {
  const token = req.headers.authorization?.split(" ")[1];
  if (!token) return res.sendStatus(401);
  req.user = jwt.verify(token, process.env.SECRET);
  next();
};
```

受保护路由

```
app.get("/profile", auth, (req, res) => {
  res.json({ user: req.user });
});
app.use("/api/admin", auth, adminRouter);
```

常用模式

CORS

```
const cors = require("cors");
app.use(cors());                          // allow all origins
app.use(cors({ origin: "https://example.com" })); // restrict
```

环境与配置

```
const port = process.env.PORT || 3000;
app.listen(port);

// Access env in routes
if (app.get("env") === "production") {
  app.use(helmet());
}
```

常用 npm 包

cors	跨域资源共享
helmet	安全响应头
morgan	HTTP 请求日志
cookie-parser	解析 Cookie 头
dotenv	将 **.env** 加载到 **process.env**
multer	多部分表单数据（文件上传）