Express.js クイックリファレンス

ルーティング、ミドルウェア、リクエスト、レスポンス、パターン

セットアップ

サーバーの作成と起動

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

組み込みミドルウェア

```
app.use(express.json());          // JSON ボディをパース
app.use(express.urlencoded({ extended: true })); // フォームデータ
app.use(express.static("public")); // 静的ファイルを提供
```

ルーティング

HTTP メソッド

```
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));
```

ルートパラメーター

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

クエリ文字列

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

ミドルウェア

アプリケーションレベル

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

ルートレベル

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

実行順序

app.use(fn)	すべてのリクエストで実行（順番に）
app.use(path, fn)	一致するパスプレフィックスのみで実行
next()	次のミドルウェアに制御を渡す
next(err)	エラーハンドラーにスキップ

リクエストとレスポンス

リクエストオブジェクト

req.params	ルートパラメーター（**/users/:id**）
req.query	クエリ文字列（**?key=val**）
req.body	パース済みリクエストボディ（パーサー必要）
req.headers	リクエストヘッダーオブジェクト
req.method	HTTP メソッド（GET、POST など）
req.path	URL パス名
req.cookies	Cookie（cookie-parser 必要）

レスポンスオブジェクト

res.json(obj)	JSON レスポンスを送信
res.send(body)	文字列/Buffer/オブジェクトを送信
res.status(code)	HTTP ステータスを設定（チェーン可）
res.redirect(url)	302 リダイレクト（ステータスも指定可）
res.sendFile(path)	ファイルをレスポンスとして送信
res.sendStatus(code)	デフォルトテキスト付きでステータスを送信
res.set(header, val)	レスポンスヘッダーを設定

エラーハンドリング

エラーミドルウェア

```
// 4つのパラメーターが必要 — Express がエラーハンドラーと認識
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(err.status || 500).json({ error: err.message });
});
```

他のすべての app.use() とルートの後に定義する

非同期エラー

```
// 非同期ルートハンドラーを包んで拒否をキャッチ
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);
}));
```

静的ファイル

静的ディレクトリを提供

```
app.use(express.static("public"));
// public/style.css を /style.css で提供

// 仮想パスプレフィックス付き
app.use("/assets", express.static("public"));
// public/style.css を /assets/style.css で提供
```

オプション

dotfiles	**'ignore'** | **'allow'** | **'deny'**
maxAge	Cache-Control max-age（ミリ秒）
index	インデックスファイル名（デフォルト: **index.html**）
fallthrough	404 時に次のミドルウェアに渡す

テンプレート

ビューエンジンのセットアップ

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

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

よく使うエンジン

ejs	埋め込み JS テンプレート（**<%= val %>**）
pug	インデントベース（旧 Jade）
handlebars	Mustache スタイル（**{{val}}**）

ルーター

モジュール化ルート

```
// 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;
```

ルーターをマウント

```
const usersRouter = require("./routes/users");
app.use("/api/users", usersRouter);
// GET /api/users -> ルーターの "/"
// GET /api/users/5 -> ルーターの "/:id"
```

ルーターメソッド

router.get/post/put/delete	HTTP メソッドハンドラー
router.use(fn)	ルーターレベルのミドルウェア
router.param(name, fn)	ルートパラメーターの前処理
router.route(path)	1 つのパスでメソッドをチェーン

認証パターン

JWT ミドルウェア

```
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();
};
```

保護されたルート

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

よくあるパターン

CORS

```
const cors = require("cors");
app.use(cors());                          // すべてのオリジンを許可
app.use(cors({ origin: "https://example.com" })); // 制限
```

環境設定

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

// ルートで環境変数にアクセス
if (app.get("env") === "production") {
  app.use(helmet());
}
```

便利な npm パッケージ

cors	クロスオリジンリソース共有
helmet	セキュリティヘッダー
morgan	HTTP リクエストロガー
cookie-parser	Cookie ヘッダーをパース
dotenv	**.env** を **process.env** に読み込む
multer	マルチパートフォームデータ（ファイルアップロード）