

# FASTAPI クイックリファレンス

パスオペレーション、バリデーション、依存性、認証、テスト

## セットアップ

### 最小アプリ

```
from fastapi import FastAPI
app = FastAPI()

@app.get("/")
async def root():
    return {"message": "Hello, World!"}
```

### アプリの起動

```
pip install "fastapi[standard]"
fastapi dev main.py # 自動リロード付き開発モード
fastapi run main.py # 本番環境
```

### 主な特徴

**非同期ネイティブ** async/await と ASGI (Uvicorn)  
**自動ドキュメント** Swagger UI は `/docs`、ReDoc は `/redoc`  
**型バリデーション** リクエスト/レスポンスに Pydantic モデル  
**OpenAPI** OpenAPI スキーマの自動生成  
**依存性注入** 組み込みの DI システム

## パスオペレーション

### HTTP メソッド

```
@app.get("/items")
@app.post("/items")
@app.put("/items/{item_id}")
@app.patch("/items/{item_id}")
@app.delete("/items/{item_id}")
```

### パスパラメーター

```
@app.get("/users/{user_id}")
async def get_user(user_id: int):
    return {"user_id": user_id}
```

```
# Enum 制約
from enum import Enum
class Color(str, Enum):
    red = "red"
    blue = "blue"
```

### ステータスコードとタグ

```
from fastapi import status

@app.post("/items", status_code=status.HTTP_201_CREATED,
         tags=["items"])
async def create_item(item: Item):
    return item
```

## リクエストボディ

### Pydantic モデル

```
from pydantic import BaseModel, Field

class Item(BaseModel):
    name: str
    price: float = Field(gt=0, description="Must be positive")
    tags: list[str] = []
```

### ネストモデル

```
class Address(BaseModel):
    street: str
    city: str
    zip_code: str
```

```
class User(BaseModel):
    name: str
    address: Address
```

### エンドポイントでの使用

```
@app.post("/items")
async def create_item(item: Item):
    return {"name": item.name, "price": item.price}
```

### バリデーション機能

**Field(gt=0)** 0 より大きい  
**Field(min\_length=1)** 最小文字列長  
**Field(max\_length=100)** 最大文字列長  
**Field(pattern='^[a-z]+\$')** 正規表現パターン一致  
**Field(default=None)** デフォルト値付きのオプション  
**EmailStr** メールバリデーション (pydantic[email])

## クエリパラメーター

### 基本クエリパラメーター

```
@app.get("/items")
async def list_items(skip: int = 0, limit: int = 10):
    return items[skip: skip + limit]
# GET /items?skip=0&limit=20
```

### クエリバリデーション

```
from fastapi import Query

@app.get("/search")
async def search(
    q: str = Query(min_length=3, max_length=50),
    page: int = Query(default=1, ge=1),
):
    return {"q": q, "page": page}
```

### オプションと必須

```
async def read_items(
    q: str | None = None, # オプション
    name: str = ... # 必須 (省略記号)
    tags: list[str] = Query(default=[]),
):
    return {"q": q, "name": name}
```

### ヘッダーと Cookie

```
from fastapi import Header, Cookie

async def read(
    user_agent: str | None = Header(default=None),
    session_id: str | None = Cookie(default=None),
):
    return {"ua": user_agent}
```

## レスポンスモデル

### レスポンスモデル

```
class ItemOut(BaseModel):
    name: str
    price: float

@app.get("/items/{id}", response_model=ItemOut)
async def get_item(id: int):
    return items[id] # 余分なフィールドをフィルタリング
```

### 複数のレスポンス型

```
from fastapi.responses import JSONResponse, HTMLResponse
```

```
@app.get("/html", response_class=HTMLResponse)
async def get_html():
    return "<h1>Hello</h1>"
```

### レスポンスモデルオプション

**response\_model** 出力フィルタリング用 Pydantic モデル  
**response\_model\_exclude\_unset** 明示的に設定されていないフィールドを省略  
**response\_model\_include** 特定フィールドのホワイトリスト  
**response\_model\_exclude** 特定フィールドのブラックリスト

## エラーレスポンス

```
from fastapi import HTTPException

@app.get("/items/{id}")
async def get_item(id: int):
    if id not in items:
        raise HTTPException(status_code=404, detail="Not found")
    return items[id]
```

## 依存性

### 関数依存性

```
from fastapi import Depends
```

```
async def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

### エンドポイントでの使用

```
@app.get("/users")
async def list_users(db: Session = Depends(get_db)):
    return db.query(User).all()
```

### クラスベース依存性

```
class Pagination:
    def __init__(self, skip: int = 0, limit: int = 10):
        self.skip = skip
        self.limit = limit
```

```
@app.get("/items")
async def list_items(pg: Pagination = Depends()):
    return items[pg.skip : pg.skip + pg.limit]
```

### 依存性のスコープ

#### Depends (func)

エンドポイントごとの依存性  
**app = FastAPI(dependencies=[...])** 全ルートへのグローバル依存性  
**APIRouter(dependencies=[...])** ルーターレベルの依存性

#### yield

セットアップ/ティアダウン (DB セッション、ロックなど)

## 認証

### OAuth2 パスワードベアラー

```
from fastapi.security import OAuth2PasswordBearer

oauth2_scheme = OAuth2PasswordBearer(tokenUrl="token")

@app.get("/users/me")
async def read_me(token: str = Depends(oauth2_scheme)):
    user = decode_token(token)
    return user
```

### JWT トークンフロー

```
from jose import jwt
SECRET = "your-secret-key"

def create_token(data: dict):
    return jwt.encode(data, SECRET, algorithm="HS256")

def decode_token(token: str):
    return jwt.decode(token, SECRET, algorithms=["HS256"])
```

### トークンエンドポイント

```
from fastapi.security import OAuth2PasswordRequestForm

@app.post("/token")
async def login(form: OAuth2PasswordRequestForm = Depends()):
    user = authenticate(form.username, form.password)
    if not user:
        raise HTTPException(status_code=401)
    return {"access_token": create_token({"sub": user.id})}
```

### セキュリティスキーム

**OAuth2PasswordBearer** フォームログイン経由のベアラートークン  
**HTTPBasic** Basic ユーザー名/パスワード認証  
**APIKeyHeader** ヘッダーの API キー  
**APIKeyCookie** Cookie の API キー

## バックグラウンドタスク

## シンプルなバックグラウンドタスク

```
from fastapi import BackgroundTasks

def send_email(to: str, body: str):
    # レスポンス後に実行される低遅延処理
    email_client.send(to, body)

@app.post("/notify")
async def notify(bg: BackgroundTasks):
    bg.add_task(send_email, "user@example.com", "Hello!")
    return {"status": "queued"}
```

### バックグラウンド付き依存性

```
async def log_request(bg: BackgroundTasks):
    bg.add_task(write_log, "request received")

@app.get("/items", dependencies=[Depends(log_request)])
async def list_items():
    return items
```

### バックグラウンド vs ワーカー

**BackgroundTasks** レスポンス後の軽量タスク (メール、ログ)  
**Celery / ARQ** 別ワーカーが必要な重い処理  
**asyncio.create\_task** ファイアアンドフォゲットの非同期コルーチン

## ミドルウェア

### カスタムミドルウェア

```
import time
from starlette.middleware.base import BaseHTTPMiddleware

class TimingMiddleware(BaseHTTPMiddleware):
    async def dispatch(self, request, call_next):
        start = time.time()
        response = await call_next(request)
        duration = time.time() - start
        response.headers["X-Process-Time"] = str(duration)
        return response
```

### ミドルウェアの追加

```
app.add_middleware(TimingMiddleware)
```

### CORS

```
from fastapi.middleware.cors import CORSMiddleware

app.add_middleware(
    CORSMiddleware,
    allow_origins=["https://example.com"],
    allow_methods=["*"],
    allow_headers=["*"],
)
```

### 組み込みミドルウェア

**CORSMiddleware** クロスオリジンリソース共有  
**TrustedHostMiddleware** 許可されたホスト名を制限  
**GZipMiddleware** Gzip レスポンス圧縮  
**HTTPSRedirectMiddleware** HTTP を HTTPS にリダイレクト

## テスト

### テストクライアント

```
from fastapi.testclient import TestClient

client = TestClient(app)

def test_read_root():
    resp = client.get("/")
    assert resp.status_code == 200
    assert resp.json() == {"message": "Hello, World!"}
```

### POST のテスト

```
def test_create_item():
    resp = client.post("/items", json={
        "name": "Widget",
        "price": 9.99,
    })
    assert resp.status_code == 201
    assert resp.json()["name"] == "Widget"
```

### 依存性のオーバーライド

```
async def mock_db():
    return FakeDB()

app.dependency_overrides[get_db] = mock_db

def test_with_mock_db():
    resp = client.get("/users")
    assert resp.status_code == 200
```

### 非同期テスト

```
import pytest
from httpx import AsyncClient, ASGITransport

@pytest.mark.anyio
async def test_async():
    transport = ASGITransport(app=app)
    async with AsyncClient(transport=transport) as ac:
        resp = await ac.get("/")
        assert resp.status_code == 200
```