ミドルウェア

FastAPI アプリケーションにミドルウェアを追加できます。

「ミドルウェア」は、すべての**リクエスト**に対して、それがあらゆる特定の*path operation*によって処理される前に機能する関数です。また、すべての**レスポンス**に対して、それを返す前に機能します。

- ミドルウェアはアプリケーションに届いたそれぞれの**リクエスト**を受け取ります。
- その後、その**リクエスト**に対して何かを実行したり、必要なコードを実行したりできます。
- 次に、アプリケーションの残りの部分に**リクエスト**を渡して (*path operation* によって) 処理させます。
- 次に、ミドルウェアはアプリケーション (の *path operation*) によって生成された**レスポンス**を受け取ります。
- その**レスポンス**に対して何かを実行したり、必要なコードを実行したりできます。
- そして、**レスポンス**を返します。

!!! note "技術詳細" `yield` を使った依存関係をもつ場合は、終了コードはミドルウェアの *後に* 実行されます。

バックグラウンドタスク (後述) がある場合は、それらは全てのミドルウェアの *後に* 実行されます。

ミドルウェアの作成

ミドルウェアを作成するには、関数の上部でデコレータ `@app.middleware("http")` を使用します。

ミドルウェア関数は以下を受け取ります:

- `request`。
- パラメータとして `request` を受け取る関数 `call_next`。
    - この関数は、対応する*path operation*に `request` を渡します。
    - 次に、対応する*path operation*によって生成された `response` を返します。
- その後、`response` を返す前にさらに `response` を変更することもできます。

```
{!../../../docs_src/middleware/tutorial001.py!}
```

!!! tip "豆知識" 'X-'プレフィックスを使用してカスタムの独自ヘッダーを追加できます。

ただし、ブラウザのクライアントに表示させたいカスタムヘッダーがある場合は、StarletteのCORSドキュメントに記載されているパラメータ `expose_headers` を使用して、それらをCORS設定に追加する必要があります ([CORS (オリジン間リソース共有)](cors.md) {.internal-link target=_blank})

!!! note "技術詳細" `from starlette.requests import Request` を使用することもできます。

FastAPIは、開発者の便利のためにこれを提供していますが、Starletteから直接きています。

`response` の前後

path operation が `request` を受け取る前に、`request` とともに実行されるコードを追加できます。

また `response` が生成された後、それを返す前にも追加できます。

例えば、リクエストの処理とレスポンスの生成にかかった秒数を含むカスタムヘッダー `X-Process-Time` を追加できます:

```
{!../../../docs_src/middleware/tutorial001.py!}
```

その他のミドルウェア

他のミドルウェアの詳細については、高度なユーザーガイド: 高度なミドルウェア{.internal-link target=_blank}を参照してください。

次のセクションでは、ミドルウェアを使用して CORS を処理する方法について説明します。