静的ファイル（outディレクトリ）保存ガイド

> 📋 **ステータス**: アクティブ（静的ファイル保存ガイド）
> 📅 **作成日**: 2025-01-XX
> 👤 **用途**: Windows環境でビルドした際の静的ファイル（outディレクトリ）の保存方法

概要

Windows環境で`npm run build`を実行すると、Next.jsが静的ファイルを`out`ディレクトリにエクスポートします。この`out`ディレクトリ全体がTauriアプリのバンドルに含まれ、アプリケーションのフロントエンドとして機能します。

重要: `out`ディレクトリは**全体を保存**する必要があります。個別のファイルを選ぶのではなく、ディレクトリ全体をそのまま保存してください。

---

outディレクトリの構造

`npm run build`を実行すると、以下のような構造で`out`ディレクトリが生成されます：

```
out/
├── index.html                    # エントリーポイント（必須）
├── _next/                        # Next.jsの静的アセット（必須）
│   ├── static/
│   │   ├── chunks/               # JavaScriptチャンク
│   │   ├── css/                  # CSSファイル
│   │   └── media/                # メディアファイル
│   └── ...
├── organization/                 # ページディレクトリ
│   ├── index.html                # 組織ページ
│   └── detail/
│       ├── index.html            # 組織詳細ページ
│       └── meeting/
│           └── index.html        # 会議ページ
├── companies/
│   └── index.html                # 企業ページ
├── knowledge-graph/
│   └── index.html                # 知識グラフページ
├── settings/
│   └── index.html                # 設定ページ
└── ... (その他のページ)
```

保存すべきファイル

✅ 保存するもの：outディレクトリ全体

`out`ディレクトリ全体をそのまま保存してください。

理由：

1. **すべてのファイルが相互依存している**
   - `index.html`は`_next/static/`内のJavaScriptとCSSを参照
   - 各ページのHTMLも`_next/static/`内のリソースを参照
   - 一部のファイルだけを保存すると、アプリが正常に動作しません
2. **Tauriが`out`ディレクトリ全体をバンドルに含める**
   - `tauri.conf.json`の`bundle.resources`に`"../out"`が指定されている
   - ディレクトリ全体がアプリのリソースとして含まれる
3. **相対パスで参照されている**
   - すべてのファイルは相対パスで相互参照している
   - ディレクトリ構造を維持する必要がある

❌ 保存しないもの

以下のファイルは**不要**です（`out`ディレクトリには含まれません）：

- `.next/` - Next.jsのビルドキャッシュ（開発用）
- `node_modules/` - 依存関係（ビルド時に使用されるが、`out`には含まれない）
- ソースファイル（`.tsx`, `.ts`など）- ビルド済みの静的ファイルのみが`out`に含まれる

保存方法

方法1: ディレクトリ全体をZIP化（推奨）

Windows環境で`out`ディレクトリをZIP化：

```
# PowerShellで実行
Compress-Archive -Path "out" -DestinationPath "out-static-files.zip"
```

または、エクスプローラーで`out`フォルダを右クリック → 「送る」→ 「圧縮（zip形式）フォルダー」

方法2: ディレクトリ全体をコピー

```
# PowerShellで実行
Copy-Item -Path "out" -Destination "C:\Backup\out" -Recurse
```

方法3: バージョン管理システムにコミット（開発用）

開発環境でバージョン管理する場合：

```
# .gitignoreにout/が含まれている場合は、一時的に除外を解除
git add -f out/
git commit -m "Add static files for Windows build"
```

注意: 通常は`out/`を`.gitignore`に含めるため、この方法は推奨されません。

確認方法

1. outディレクトリの存在確認

```
# PowerShellで実行
Test-Path "out"
# 結果: True が返れば存在する

# ディレクトリの内容を確認
Get-ChildItem -Path "out" -Recurse | Measure-Object
# ファイル数が表示される
```

2. 必須ファイルの確認

以下のファイルが存在することを確認：

```
# index.htmlの存在確認
Test-Path "out\index.html"

# _next/static/ディレクトリの存在確認
Test-Path "out\_next\static"

# 主要なページの存在確認
Test-Path "out\organization\index.html"
Test-Path "out\companies\index.html"
Test-Path "out\knowledge-graph\index.html"
```

3. ファイルサイズの確認

```
# outディレクトリ全体のサイズを確認
$size = (Get-ChildItem -Path "out" -Recurse | Measure-Object -Property
Length -Sum).Sum
Write-Host "outディレクトリのサイズ: $([math]::Round($size / 1MB, 2)) MB"
```

期待されるサイズ: 通常、数MBから数十MB程度（アプリケーションの規模による）

Tauriビルド時の動作

ビルドプロセス

1. **`npm run build`を実行**
   - Next.jsが`out`ディレクトリに静的ファイルを生成
   - すべてのページがHTMLとして出力される
2. **`npm run tauri:build`を実行**
   - Tauriが`tauri.conf.json`の`frontendDist: "../out"`を参照
   - `bundle.resources`に`"../out"`が指定されているため、`out`ディレクトリ全体がアプリバンドルに含まれる
3. **アプリバンドルの生成**
   - `src-tauri/target/release/bundle/msi/`にWindowsインストーラーが生成される
   - `out`ディレクトリの内容がアプリのリソースとして含まれる

確認方法

ビルド後、生成されたMSIファイルのサイズを確認：

```
# MSIファイルのサイズを確認
Get-ChildItem -Path "src-tauri\target\release\bundle\msi\*.msi" |
Select-Object Name, @{Name="Size(MB)";Expression=
{[math]::Round($_.Length / 1MB, 2)}}
```

期待されるサイズ: `out`ディレクトリのサイズ + Rustバイナリのサイズ（通常、合計で50-200MB程度）

トラブルシューティング

問題1: outディレクトリが空または存在しない

原因: `npm run build`が正常に完了していない

解決方法:

```
# ビルドを再実行
npm run build

# エラーメッセージを確認
# エラーが表示される場合は、エラー内容を確認して修正
```

問題2: アプリが起動しない、または白い画面が表示される

原因: outディレクトリの内容が不完全、またはパス参照の問題

解決方法:

1. outディレクトリの内容を確認

```
Get-ChildItem -Path "out" -Recurse | Select-Object FullName
```

2. index.htmlが存在するか確認

```
Test-Path "out\index.html"
```

3. _next/static/ディレクトリが存在するか確認

```
Test-Path "out\_next\static"
```

4. ビルドを再実行

```
# クリーンビルド
Remove-Item -Path "out" -Recurse -Force
npm run build
```

問題3: 画像やCSSが読み込まれない

原因: パス参照の問題、またはoutディレクトリの構造が壊れている

解決方法:

1. next.config.jsの設定を確認

```
// next.config.js
{
  output: 'export',
  distDir: 'out',
  images: {
    unoptimized: true,  // 必須
  },
  trailingSlash: true,
}
```

2. outディレクトリの構造を確認

- `_next/static/`ディレクトリが存在するか
- CSSファイルやJavaScriptファイルが存在するか

3. ビルドを再実行

```
Remove-Item -Path "out" -Recurse -Force
npm run build
```

ベストプラクティス

1. ビルドのタイミング

- **開発中**: `out`ディレクトリは通常`.gitignore`に含める（ビルド成果物のため）
- **デプロイ時**: Windows環境で`npm run build`を実行して`out`ディレクトリを生成
- **バックアップ**: 必要に応じて`out`ディレクトリをZIP化してバックアップ

2. バージョン管理

- `out`ディレクトリは通常バージョン管理に含めない（`.gitignore`に含める）
- 理由: ビルド成果物のため、ソースコードから再生成可能

3. 配布時の確認

- MSIインストーラーを作成する前に、`out`ディレクトリの内容を確認
- アプリを起動して、すべてのページが正常に表示されることを確認

まとめ

保存すべきもの

✅ **`out`ディレクトリ全体**

保存方法

1. Windows環境で`npm run build`を実行
2. 生成された`out`ディレクトリ全体をZIP化またはコピー
3. 必要に応じてバックアップとして保存

重要なポイント

- **個別のファイルを選ぶのではなく、ディレクトリ全体を保存**
- **ディレクトリ構造を維持する**
- **すべてのファイルが相互依存しているため、一部だけを保存すると動作しない**

関連ドキュメント

- Windowsデプロイ用パッケージ作成ガイド
- ビルド・デプロイガイド
- React/Next.js設定

最終更新: 2025-01-XX