ChatGPT

TypeScript における依存境界の型解決の挙動

package.json の `types` / `exports` と型解決

- **公式ページ**: Declaration Files – Publishing (TypeScript Handbook)
  見出し: Including declarations in your npm package
  引用: “Set the `types` property to point to your bundled declaration file.” [1]
  つなぎ文: 公式ドキュメントでは、パッケージに `main` JS がある場合、`package.json` の `types` フィールドで型定義ファイルへのパスを明示的に指定する必要があると説明しています [1]。これによりTypeScriptは型定義を正しく発見し、解決できます。

- **公式ページ**: Modules – Reference (TypeScript Handbook)
  見出し: Directory modules
  引用: “Directory modules may also contain a package.json file, where resolution of the "main" and "types" fields are supported, and take precedence over index.js lookups.” [2]
  つなぎ文: Handbook のモジュール解説では、ディレクトリ・モジュール（フォルダ）内に `package.json` があれば、その中の `"types"` フィールドを使って型定義を解決し、`index.js` より優先することが明記されています [2]。つまり、`package.json` の `types` が指定されていれば、そちらが型解決の基準になります。

`compilerOptions.types` / `typeRoots` の挙動

- **公式ページ**: TSConfig Reference – types (TypeScript Docs)
  見出し: types
  引用: “If `types` is specified, only packages listed will be included in the global scope.” [3]
  つなぎ文: `tsconfig.json` の `types` オプションを設定すると、そのリストに挙げた型パッケージのみが含まれると公式に説明されています [3]。したがって `types` を指定した場合、その他の `@types` は探索対象から外れます。

- **公式ページ**: TSConfig Reference – typeRoots (TypeScript Docs)
  見出し: typeRoots
  引用: “If `typeRoots` is specified, only packages under `typeRoots` will be included.” [4]
  つなぎ文: また、`typeRoots` を設定すると、そのフォルダ内の型パッケージのみが探索対象になると説明されています [4]。したがって `types` オプションを併用すると、`types` で指定しない限り `typeRoots` 内の型も読み込まれない（無視される）仕様です。

`moduleResolution` (Node16/NodeNext/Bundler) の影響

- **公式ページ**: TSConfig Reference – moduleResolution (TypeScript Docs)
  見出し: moduleResolution
  引用: “`'bundler'` for use with bundlers. Like `node16` and `nodenext`, this mode supports package.json `"imports"` and `"exports"`, but unlike the Node.js resolution modes, `bundler` never requires file extensions on relative paths in imports.” [5]
  つなぎ文: `moduleResolution` オプションでは、モードごとに解決アルゴリズムが異なると説明されています。例えば `bundler` モードではパッケージの `"imports"` や `"exports"` をサポートし、相対パスの拡張子を省略しても解決できることが明記されています [5]。

- **公式ページ**: Modules – Reference (TypeScript Handbook)
  見出し: package.json "exports"
  引用: “When `moduleResolution` is set to `node16`, `nodenext`, or `bundler`… TypeScript follows Node.js’s package.json `"exports"` spec when resolving from a package directory…” [6]
  つなぎ文: さらに、`moduleResolution` を `node16` や `nodenext` にすると、Node.js の `"exports"` フィールド仕様に従ってモジュール解決が行われるとあります [6]。これにより、ESM と CJS の両方式でルールに合ったファイルが選ばれます。

TS7016/TS2307 など「型が見つからない」系エラー

- **公式ページ**: Modules – Reference (TypeScript Handbook)
  見出し: package.json "exports" (解決例より)
  引用: “…If no types are found, a second pass through all `node_modules` resolves to … which counts as a successful resolution, but one that does not provide types, leading to any-typed imports…” [7]
  つなぎ文: 公式にはこれらエラーコードの説明はありませんでしたが、モジュール解決の例から、型定義が見つからない場合は最終的にJSファイルで解決されてしまい、型がなく `any` 扱いになることが示されています [7]。この「型情報なし」の状態が TS7016/TS2307 エラーの原因となります。

参考文献: TypeScript 公式ドキュメント（ハンドブック、TSConfig リファレンス等） [1] [2] [3] [4] [5] [6] [7]。

---

[1] TypeScript: Documentation - Publishing
https://www.typescriptlang.org/docs/handbook/declaration-files/publishing.html

[2] [6] [7] TypeScript: Documentation - Modules - Reference
https://www.typescriptlang.org/docs/handbook/modules/reference.html

[3] TypeScript: TSConfig Option: types
https://www.typescriptlang.org/tsconfig/types

[4] TypeScript: TSConfig Option: typeRoots
https://www.typescriptlang.org/tsconfig/typeRoots.html

[5] TypeScript: TSConfig Option: moduleResolution
https://www.typescriptlang.org/tsconfig/moduleResolution.html