Bien sûr ! Voici une documentation structurée et détaillée sur l'exclusion de fichiers de la transpilation en TypeScript, basée sur les concepts de votre carte mentale.

Exclusion de Fichiers de la Transpilation en TypeScript

Introduction

La transpilation est le processus de conversion du code source d'un langage (par exemple, TypeScript) vers un autre (par exemple, JavaScript) afin qu'il puisse être exécuté par un environnement cible (navigateur, Node.js). Dans les projets TypeScript, il est souvent nécessaire d'exclure certains fichiers ou répertoires de ce processus pour plusieurs raisons :

- **Performance :** Réduire le nombre de fichiers à traiter peut accélérer la transpilation et la vérification des types.
- **Pertinence :** Certains fichiers (tests, scripts de build, documentation, fichiers temporaires, dépendances tierces) n'ont pas vocation à être transpilés ou distribués avec le code de production.
- **Éviter les erreurs :** Les bibliothèques tierces (node_modules) peuvent contenir des avertissements ou des erreurs de type qui ne sont pas pertinentes pour votre propre code.
  La configuration des exclusions se fait principalement via le fichier tsconfig.json à la racine de votre projet TypeScript.

1. Concepts Fondamentaux : tsconfig.json

Le fichier tsconfig.json est le fichier de configuration principal pour un projet TypeScript. Il spécifie les options de compilation, les fichiers à inclure et à exclure, et d'autres paramètres du projet.
Les propriétés clés pour la gestion des inclusions/exclusions sont :

- **include** : Un tableau de motifs de chemins de fichiers glob qui spécifie les fichiers à inclure dans le programme TypeScript. Si cette option est omise, le compilateur inclura tous les fichiers .ts, .tsx et .d.ts dans le répertoire courant et ses sous-répertoires, à l'exception de ceux spécifiés par exclude.
- **exclude** : Un tableau de motifs de chemins de fichiers glob qui spécifie les fichiers et les répertoires à exclure du programme. Ces fichiers sont ignorés par le compilateur et ne sont pas transpilés.

- **compilerOptions** : Contient toutes les options de configuration du compilateur, y compris skipLibCheck.

Motifs Glob (Glob Patterns)

Les motifs glob sont des chaînes de caractères spéciales utilisées pour correspondre à des noms de fichiers ou de chemins :

- * : Correspond à zéro ou plusieurs caractères (sauf un séparateur de répertoire).
- ** : Correspond à zéro ou plusieurs répertoires dans un chemin.
- ? : Correspond à un seul caractère.

2. Stratégies d'Exclusion

2.1. Exclure un Fichier Cible Spécifique

Pour exclure un ou plusieurs fichiers spécifiques qui ne doivent pas être transpilés :

Utilisation de l'option exclude :

Ajoutez le chemin relatif du fichier dans le tableau exclude.

Exemple :

Si vous avez un fichier src/tests/util.test.ts que vous ne voulez pas transpiler :

JSON

```
// tsconfig.json
{
  "compilerOptions": {
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "module": "commonjs",
    "target": "es2018",
    "esModuleInterop": true
  },
  "include": [
    "src/**/*" // Inclut tous les fichiers sous 'src' par défaut
  ],
  "exclude": [
    "src/tests/util.test.ts", // Exclut spécifiquement ce fichier
    "src/data/temp.ts"      // Vous pouvez ajouter plusieurs fichiers
  ]
}
```

2.2. Exclure Tous les Fichiers avec une Extension Spécifique (.dev.ts)

Pour exclure tous les fichiers qui suivent un certain modèle de nommage, comme les fichiers de développement (.dev.ts), vous pouvez utiliser des motifs glob.

Utilisation de l'option exclude avec un motif glob :

Le motif **/*.dev.ts correspond à tous les fichiers se terminant par .dev.ts dans n'importe quel sous-répertoire.

Exemple :

Pour ignorer tous les fichiers de configuration ou de test qui se terminent par .dev.ts :

JSON
```
// tsconfig.json
{
  "compilerOptions": {
    // ...
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    "**/*.dev.ts" // Exclut tous les fichiers se terminant par .dev.ts, quel que soit leur emplacement
  ]
}
```

2.3. Exclure les Fichiers d'un Répertoire Spécifique

Vous pouvez exclure des répertoires entiers, ce qui est très utile pour les répertoires de tests, de build, ou d'outils.

Utilisation de l'option exclude :
Exemple :

Pour exclure l'intégralité d'un dossier tests ou scripts :

JSON
```
// tsconfig.json
{
  "compilerOptions": {
    // ...
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    "src/tests",     // Exclut le répertoire 'src/tests' et tous ses contenus
    "scripts",       // Exclut le répertoire 'scripts' à la racine du projet
    "dist"           // Il est courant d'exclure le répertoire de sortie
  ]
```

```
}
```

2.4. Exclure les Fichiers de node_modules

Le répertoire node_modules contient toutes les dépendances de votre projet. Il est **crucial** de ne pas transpiler ces fichiers, car ils sont déjà compilés et la vérification de type peut prendre énormément de temps et générer de nombreuses erreurs non pertinentes.

Comportement par défaut :

Par défaut, TypeScript exclut le répertoire node_modules. Vous n'avez donc généralement pas besoin de l'ajouter explicitement à exclude à moins que vous n'utilisiez une option include très générique (par exemple, ["**/*"]) qui l'engloberait.

Exclusion explicite (si nécessaire) :

JSON

```
// tsconfig.json
{
  "compilerOptions": {
    // ...
  },
  "exclude": [
    "node_modules" // Exclut explicitement le répertoire node_modules
  ]
}
```

Option skipLibCheck (Recommandé pour node_modules) :

Plutôt que d'exclure entièrement node_modules (ce qui pourrait empêcher l'accès aux définitions de types de bibliothèques), l'option skipLibCheck est souvent la meilleure solution. Elle indique au compilateur TypeScript de ne pas effectuer de vérification de type sur les fichiers de déclaration (.d.ts) des bibliothèques (node_modules). Cela réduit considérablement le temps de compilation et supprime les erreurs de type provenant de code tiers que vous ne contrôlez pas.

Exemple d'utilisation de skipLibCheck :

JSON

```
// tsconfig.json
{
  "compilerOptions": {
    "outDir": "./dist",
    "rootDir": "./src",
    "strict": true,
    "module": "commonjs",
    "target": "es2018",
    "esModuleInterop": true,
    "skipLibCheck": true // Permet d'ignorer les erreurs de type dans les fichiers de librairies
```

```
  },
  "include": [
    "src/**/*"
  ],
  "exclude": [
    // Pas besoin d'exclure node_modules ici si skipLibCheck est true et votre 'include' ne l'englobe pas par erreur
  ]
}
```

3. Interaction entre include et exclude

Il est important de comprendre comment include et exclude interagissent :

1. Le compilateur commence par prendre tous les fichiers .ts, .tsx, et .d.ts du répertoire racine du projet et de ses sous-répertoires (si include est omis).
2. Si include est spécifié, seuls les fichiers correspondant aux motifs de include sont considérés.
3. Ensuite, les fichiers et répertoires correspondant aux motifs de exclude sont retirés de cette liste.
4. Cependant, si un fichier *exclu* est explicitement référencé (par un import ou un /// <reference path="..." />) par un fichier qui est *inclus*, il sera tout de même inclus dans la compilation. exclude ne fait qu'empêcher les fichiers d'être automatiquement découverts.

4. Bonnes Pratiques

- **Soyez spécifique avec include :** Définissez clairement les répertoires de votre code source avec include (par exemple, "src/**/*"). Cela limite le champ de recherche du compilateur.
- **Utilisez exclude pour les répertoires non pertinents :** Excluez les dossiers comme dist, build, temp, tests (si vous ne voulez pas les transpiler pour la production).
- **Activez skipLibCheck :** Pour les projets avec de nombreuses dépendances, skipLibCheck: true est presque toujours une bonne idée pour éviter les problèmes de typage des bibliothèques tierces.
- **Vérifiez votre configuration :** Après avoir modifié tsconfig.json, recompilez votre projet pour vous assurer que les exclusions fonctionnent comme prévu.

J'espère que cette documentation structurée vous est claire et utile !