TypeScript Guide

👨🏽‍🦳 par Célian Lebacle

TypeScript Documentation

1. Introduction à TypeScript

TypeScript est un superset de JavaScript qui ajoute un système de typage statique optionnel ainsi que des fonctionnalités avancées (interfaces, classes, generics, etc.).
Tout code JavaScript est aussi du TypeScript valide, mais l'ajout de types permet de détecter des erreurs lors de la compilation, d'améliorer la maintenabilité du code et d'offrir une meilleure expérience de développement (complétion, refactorisation, documentation automatique). Votre code TypeScript sera compilé en JavaScript, de ce fait le typage offert par TypeScript sert simplement à la lecture du code et sa simplification.

Quels sont ses avantages par rapport à JavaScript ?

- **Détection précoce des erreurs :** Le typage statique permet de repérer les incohérences avant l'exécution.
- **Maintenabilité :** Des types explicites rendent le code plus compréhensible et facilitent la collaboration.
- **Productivité :** Les IDE modernes offrent une autocomplétion et des outils de refactorisation basés sur les annotations de types.
- **Adoption progressive :** Vous pouvez migrer progressivement un projet JavaScript vers TypeScript.

👉 Consultez la **documentation officielle**.

2. Installation et Configuration

Installation

Initialisez un projet NodeJS :

```
npm init
```

Installez TypeScript globalement via npm :

```
npm install -g typescript
```

Vérifiez l'installation :

```
tsc --version
```

Initialisation d’un Projet

Générez un fichier de configuration de base avec :

```
tsc --init
```

Exemple de fichier `tsconfig.json` minimal :

```
{
  "compilerOptions": {
    "outDir": "dist",
    "target": "ES6",
    "noEmitOnError": true,
    "strict": true,
    "declaration": true
  },
  "files": ["src/app.ts"],
  "include": ["src"]
}
```

Ce fichier configure le compilateur pour générer du JavaScript ES6 et activer le mode strict pour renforcer les vérifications de types.

👉 **Documentation : TSConfig**

3. Les Types de Base

TypeScript étend JavaScript avec des types primitifs et d'autres types spéciaux :

Types Primitifs

- **string, number, boolean :**

```
let message: string = "Bonjour";

let age: number = 30;

let actif: boolean = true;
```

- **null et undefined :**
  En mode strict, ces types doivent être explicitement autorisés dans une union pour être assignés à une variable d'un autre type.
- **Tableaux :**
  Deux syntaxes équivalentes :

```
let nombres: number[] = [1, 2, 3]; let fruits: Array<string> = ["pomme", "banane"];
```

- **Tuples :**
  Des tableaux à taille fixe avec des types différents :

```
let personne: [string, number] = ["Alice", 30];
```

- **any :**
  Désactive la vérification de type (utilisez-le avec parcimonie).

```
let variable: any = "texte"; variable = 42;
```

- **unknown :**
  Similaire à `any` mais exige un contrôle de type avant utilisation.

```
let data: unknown = "Hello"; if (typeof data === "string") { console.log(data.toUpperCase()); }
```

- **void :**
  Utilisé pour indiquer qu'une fonction ne renvoie rien.

```
function log(message: string): void { console.log(message); }
```

- **never :**
  Représente une valeur qui n'arrive jamais (par exemple, une fonction qui lance une exception).

```
function erreur(message: string): never { throw new Error(message); }
```

👉 **Documentation : Basic Types**

4. Interfaces et Types Personnalisés

TypeScript vous permet de définir des types personnalisés grâce aux **interfaces** et aux **alias de types**.

Interfaces

Les interfaces décrivent la forme d'un objet :

```
interface Personne {
  nom: string;
  age: number;
  ville?: string; // Le '?' offre la possibilité de rendre une propriété optionnelle
}

function saluer(p: Personne) {
  console.log(`Bonjour ${p.nom}, ${p.age} ans`);
}
```

Les interfaces peuvent inclure des méthodes et être étendues :

```
interface Forme {
  surface(): number;
}
```

Alias de Types

Les alias (déclarés avec `type`) permettent de nommer n'importe quel type, y compris des unions ou des objets :

```
type ID = string | number;
type Point = { x: number; y: number };
```

Comparaison :

- **Interfaces :** Idéales pour décrire la structure d'un objet, extensibles via `extends`.
- **Alias de Types :** Plus flexibles (peuvent représenter des unions, intersections, etc.).

👉 **Documentation :**

- **Interfaces**
- **Type Aliases**

5. Classes et Héritage

Déclaration d’une Classe

```
class Point {
  x: number;
  y: number;

  constructor(x: number, y: number) {
    this.x = x;
    this.y = y;
  }

  deplacer(dx: number, dy: number): void {
    this.x += dx;
    this.y += dy;
  }
}
```

Héritage

```
class Point3D extends Point {
  z: number;

  constructor(x: number, y: number, z: number) {
    super(x, y);
    this.z = z;
  }

  deplacer(dx: number, dy: number, dz: number): void {
    super.deplacer(dx, dy);
    this.z += dz;
  }
}
```

Modificateurs d’accès

- `public` (par défaut) : accessible partout.
- `private` : accessible uniquement dans la classe.
- `protected` : accessible dans la classe et ses sous-classes.

```
class Compte {
  private solde: number = 0;
  public deposer(montant: number): void {
    this.solde += montant;
  }
  public getSolde(): number {
    return this.solde;
  }
}
```

👉 **Documentation : Classes**

6. Les Generics

Les generics permettent de créer des composants réutilisables et flexibles.

Fonction Générique

```
function identite<T>(valeur: T): T {
  return valeur;
}

let sortie1 = identite("TypeScript"); // Infère string
let sortie2 = identite<number>(123); // Explicite number
```

Classe Générique

```
class Boite<T> {
  contenu: T;
  constructor(contenu: T) {
    this.contenu = contenu;
  }
  ouvrir(): T {
    return this.contenu;
  }
}

const b1 = new Boite<string>("bonjour");
const b2 = new Boite<number>(42);
```

Contraintes sur les Generics

```
function plusLong<T extends { length: number }>(a: T, b: T): T {
  return a.length >= b.length ? a : b;
}

console.log(plusLong("bonjour", "salut"));
```

👉 **Documentation : Generics**

7. Enums et Tuples

Enums

Les enums définissent un ensemble de valeurs nommées.

```
enum Couleur {
  Rouge,
  Vert,
  Bleu,
}
let peinture: Couleur = Couleur.Vert;
console.log(peinture); // Affiche 1
console.log(Couleur[2]); // Affiche "Bleu"
```

Exemple d’enum de chaînes :

```
enum Role {
  Admin = "ADMIN",
  User = "USER",
  Visiteur = "VIS",
}
```

Tuples

Les tuples sont des tableaux à taille fixe avec des types prédéfinis.

```
let personne: [string, number];
personne = ["Alice", 30];
```

👉 **Documentation :**

- **Enums**
- **Tuples**

8. Narrowing et Type Guards

Le narrowing est le processus d'affinement des types en fonction du flux de contrôle.

Exemple avec `typeof`

```
function traiter(x: string | number) {
  if (typeof x === "string") {
    console.log(x.toUpperCase());
  } else {
    console.log(x.toFixed(2));
  }
}
```

Exemple avec `instanceof`

```
if (date instanceof Date) {
  console.log(date.toUTCString());
}
```

Exemple avec l'opérateur `in`

```
interface Poisson {
  nager(): void;
}
interface Oiseau {
  voler(): void;
}

function move(animal: Poisson | Oiseau) {
  if ("nager" in animal) {
    animal.nager();
  } else {
    animal.voler();
  }
}
```

Garde de type personnalisée

```
function estPoisson(animal: Poisson | Oiseau): animal is Poisson {
  return (animal as Poisson).nager !== undefined;
}

function moveAnimal(animal: Poisson | Oiseau) {
  if (estPoisson(animal)) {
    animal.nager();
  } else {
    animal.voler();
  }
}
```

👉 **Documentation :**

- **Narrowing**
- **Type Guards**

9. Modules et Espaces de Noms

Modules (import/export)

Créez des modules pour organiser votre code. Par exemple :

math.ts

```
export function addition(a: number, b: number): number {
  return a + b;
}
export const PI = 3.14;
```

app.ts

```
import { addition, PI } from "./math";
console.log(addition(10, 5)); // Affiche 15
```

Espaces de Noms

Les namespaces regroupent du code dans un espace de noms global.

```
namespace Geometrie {
  export function aireCarre(cote: number): number {
    return cote * cote;
  }
}
console.log(Geometrie.aireCarre(5)); // Affiche 25
```

👉 **Documentation : Namespaces and Modules**

10. Fichiers de Déclaration (.d.ts)

Les fichiers de déclaration fournissent des informations de type pour du code JavaScript ou des bibliothèques externes.

Exemple pour une bibliothèque globale

```
// math-ext.d.ts
declare namespace MathExt {
  function randomInt(min: number, max: number): number;
}
```

Exemple pour un module npm non typé

```
// super-lib.d.ts
declare module "super-lib" {
  export function superify(x: string): string;
  export const version: string;
}
```

👉 **Documentation : Declaration Files**

11. Compilation et Configuration Avancée

Options de Configuration

Un fichier `tsconfig.json` avancé peut ressembler à :

```
{
  "compilerOptions": {
    "target": "ES2018",
    "module": "commonjs",
    "outDir": "./dist",
    "strict": true,
    "sourceMap": true,
    "declaration": true,
    "esModuleInterop": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src"]
}
```

Compilation

Compilez votre projet avec :

```
tsc
```

Ou utilisez le mode surveillance :

```
tsc --watch
```

👉 **Documentation : TSConfig Reference**

12. Outils et Écosystème TypeScript

TypeScript s'intègre dans un riche écosystème :

- **Build Tools & Bundlers :** Webpack, Rollup, Parcel, Vite (avec `ts-loader` ou Babel).
- **IDE et Éditeurs :** Visual Studio Code, WebStorm, etc.
- **Linting et Formatage :** ESLint avec `@typescript-eslint/parser` et Prettier.
- **Frameworks :** Angular (utilise TS par défaut), React (fichiers `.tsx`), Vue 3.
- **Tests :** Jest (avec `ts-jest`), Mocha.
- **Interopérabilité :** Babel pour transpiler TS, génération automatique de types via les packages `@types`.

👉 **Ressources supplémentaires :**

- **TypeScript Playground**

13. Ressources Complémentaires

- **Documentation Officielle : TypeScript Docs**
- **Livres et Guides :**
  - *TypeScript Deep Dive* – Livre gratuit de Basarat Ali Syed.
  - *Effective TypeScript* par Dan Vanderkam.
  - *Programming TypeScript* par Boris Cherny.
- **Tutoriels Vidéo :**
  - **Tutoriel TypeScript par Grafikart** (en français)
  - **Learn TypeScript – Full Course for Beginners** (freeCodeCamp)
- **Articles et Blogs :**
  - Le blog officiel de TypeScript sur DevBlog de Microsoft(**https://devblogs.microsoft.com/typescript/**)
- **Frameworks :**
  - React + Vite.
  - VueJS.
  - NestJS
  - NextJS
  - Angular