Introduzione a TypeScript

Cos'è TypeScript?

Definizione

- TypeScript è un **superset tipizzato di JavaScript** sviluppato da Microsoft.
- Fornisce **controllo statico dei tipi**, strumenti avanzati per lo sviluppo e funzionalità OOP.
- Il codice TypeScript viene **compilato** in JavaScript per essere eseguito su browser o Node.js.

Differenze con JavaScript

Caratteristica	JavaScript	TypeScript
Tipizzazione	Dinamica	Statica
Supporto OOP	Limitato	Esteso
Debugging	Rischio runtime	Errori a compile time
Moduli & Namespace	Sì	Sì

Esempio di codice:

```
let message: string = "Ciao TypeScript!";
console.log(message);
```

Perché usare TypeScript?

Vantaggi principali

- **Miglioramento della qualità del codice**: errori rilevati in fase di compilazione.
- **IntelliSense avanzato**: suggerimenti automatici e completamento del codice.
- **Supporto ai moduli e alla programmazione orientata agli oggetti**.
- **Compatibilità con JavaScript**: puoi usare TypeScript in qualsiasi progetto JS esistente.

Installazione e configurazione

Installazione di TypeScript

```
npm install -g typescript
```

Verificare la versione:

```
tsc -v
```

Configurazione del progetto

Creare un file di configurazione:

```
tsc --init
```

Installazione e configurazione

Configurazione

Esempio di `tsconfig.json`:

```
{
  "compilerOptions": {
    "target": "ES6",
    "module": "CommonJS",
    "strict": true
  }
}
```

Compilazione di un file TypeScript:

```
tsc file.ts
```

I Generics in TypeScript

Cosa sono i Generics?

- I **Generics** sono una funzionalità di TypeScript che permette di creare **componenti riutilizzabili** in modo più flessibile.
- Consentono di scrivere funzioni, classi e interfacce che possono lavorare con diversi tipi di dati senza perdere il controllo sulla tipizzazione.

Perché si usano?

- Aumentano il **riutilizzo del codice** senza sacrificare la sicurezza dei tipi.
- Permettono di evitare l'uso di `any`, mantenendo il controllo sui tipi accettati.
- Migliorano la leggibilità e manutenibilità del codice.

Come si usano i generics?

Esempio di funzione generica:

```
function identity<T>(arg: T): T {
  return arg;
}
console.log(identity<string>("ciao")); // Output: "ciao"
console.log(identity<number>(42)); // Output: 42
```

Esempio di classe generica:

```
class Box<T> {
  private valore: T;
  constructor(valore: T) {
    this.valore = valore;
  }
  getValore(): T {
    return this.valore;
  }
}
```

I Decorator in TypeScript

Cosa sono i Decorator?

- I **Decorator** sono una funzionalità avanzata di TypeScript che permette di **modificare il comportamento di classi, metodi, proprietà e parametri**.
- Sono simili ai **decoratori di Python** o agli **annotation di Java**.

Perché si usano?

- Consentono di aggiungere **meta-programmazione** e funzionalità **cross-cutting** come logging, autorizzazioni e validazioni.
- Vengono usati nei framework TypeScript come **Angular** per estendere il comportamento di classi e componenti.

Come si usano i decoratori?

Per abilitare i decorator, bisogna attivare il supporto nel `tsconfig.json`:

```
{
  "experimentalDecorators": true,
  "emitDecoratorMetadata": true
}
```

Esempio di **decorator di metodo**:

```
function log(target: any, key: string, descriptor: PropertyDescriptor) {
  const originale = descriptor.value;
  descriptor.value = function (...args: any[]) {
    console.log(`Chiamato ${key} con argomenti:`, args);
    return originale.apply(this, args);
  };
}
class Utente {
  @log
  saluta(nome: string) {
    console.log(`Ciao, ${nome}!`);
  }
}
const u = new Utente();
u.saluta("Marco");
```

Esempio di **decorator di classe**:

```
function Entita(constructor: Function) {
  console.log(`Classe decorata: ${constructor.name}`);
}
@Entita
class Prodotto {
  constructor(public nome: string) {}
}
```

Conclusione

- **I Generics** migliorano il riutilizzo del codice mantenendo la sicurezza dei tipi.
- **I Decorator** permettono di estendere e modificare il comportamento delle classi in modo dichiarativo.
- Entrambe le funzionalità sono strumenti potenti per scrivere codice più modulare e scalabile.

Domande?

Grazie per l'attenzione!


Q&A Icon

Esempi pratici o chiarimenti?