TypeScript & React 入門

モダンウェブサイト開発への道

小林 紹子

July 31, 2025

はじめに - ウェブ開発の役割

- ウェブサイト開発、大きく 2 つの領域。
  - フロントエンド:
    - ユーザーが使う画面（UI）。
    - ブラウザで動き、操作に反応。
    - React、TypeScript が活躍。
  - バックエンド:
    - サーバー側の処理、データ管理。
    - データベース連携、API 提供。
    - Node.js、Python などが活躍。
- 今回はフロントエンド開発の強力な味方、TypeScript と React に焦点を当てる。

TypeScript とは

- **JavaScript のスーパーセット。**
  - JavaScript の機能を全て含む上位互換言語。
  - 既存の JavaScript コードはそのまま TypeScript としても有効。
- **静的型付けの導入。**
  - 変数や関数の型を定義。
  - コード実行前のエラー検出が可能。
- **主な機能:**
  - **型注釈**: 変数に型を明示。
  - **インターフェース**: オブジェクト構造の定義。
  - **型推論**: 型の自動判別。
- **Node.js との関連:**
  - Node.js でも TypeScript を利用し、サーバーサイドの堅牢性を強化。

TypeScript のメリット

- 堅牢性:
  - 型によるバグ早期発見。
  - 大規模アプリでも安定した開発。
- 開発生産性:
  - エディタの強力な補完、エラーチェック。
  - 安全なコード改修（リファクタリング）。
- 保守性:
  - コードの意図が明確で理解しやすい。
  - 大規模プロジェクト、複数人開発向き。
- 可読性:
  - 型情報によるコードの明確化。
- **JavaScript 開発全般への恩恵**:
  - フロントエンド、Node.js などのバックエンド開発でもメリット享受。

React とは

- **UI 構築のための JavaScript ライブラリ。**
  - Facebook（現 Meta）が開発。
  - ウェブページの見た目（UI）作成に特化。
- **コンポーネント指向。**
  - UI を「再利用可能な独立した部品（コンポーネント）」に分割。
  - 部品を組み合わせ、複雑な UI を効率的に構築。
- **仮想 DOM (Virtual DOM)。**
  - 実際の DOM の軽量なコピー。
  - UI 変更時、仮想 DOM で差分計算、必要な部分だけ更新。
  - 高速な UI 描画を実現。
- **宣言的 UI。**
  - UI が「どうあるべきか」を記述。
  - コードが直感的で理解しやすい。

Reactの主な概念

- コンポーネント:
  - UIの最小単位。関数コンポーネントが主流。
  - 例: ボタン、ヘッダー、商品カードなど。

```
function MyButton()  return <button>クリック</button>;
```

- **JSX (JavaScript XML)**:
  - JavaScript中にHTMLのような構文を書く拡張機能。
  - React要素を作成。

```
const element = <h1>Hello, React!</h1>;
```

- **State (状態)**:
  - コンポーネントが持つ、変化するデータ。
  - 例: カウンターの値、フォーム入力値。
- **Props (プロパティ)**:
  - 親から子へデータを渡す仕組み。
  - 例: ボタンのテキスト、画像のURL。

Reactのメリット

- 開発効率:
  - コンポーネント再利用で開発時間短縮。
  - 大規模 UI でも効率的な開発。
- 保守性:
  - コンポーネントごとの責務が明確で、変更が容易。
  - 変更の影響範囲が限定され、バグのリスクを低減。
- パフォーマンス:
  - 仮想 DOM による効率的な UI 更新で、ユーザー体験向上。
  - スムーズなアニメーション、高速ページ遷移。
- 宣言的 UI:
  - コードが直感的で理解しやすい。
  - 複雑な UI ロジックも簡潔に表現。
- 優れたユーザー体験:
  - インタラクティブで滑らかな UI 提供。

TypeScript と React の組み合わせ - 最強のフロントエンド

- 型安全な **React** アプリケーション開発:
  - React の Props や State に TypeScript の型定義を適用。
  - コンポーネント間のデータ受け渡しにおける型エラーを開発段階で防止。
  - 例:

    ```
    interface UserProps  name: string; age: number;  const User
    React.FC<UserProps> = ( name, age ) =>  return <p>name (age
    歳)</p>; ;
    ```
- 大規模フロントエンド開発の生産性・信頼性向上:
  - 複雑な UI を持つアプリでも、TypeScript の型システムが品質を担保。
- 開発ツール連携強化:
  - VS Code などで強力な補完、リアルタイムエラーチェック。

Node.js との関係、それぞれの強み

- **Node.js**:
  - 役割: サーバーサイド処理、API 構築、データベース連携。
  - 強み:
    - JavaScript でサーバーサイド記述、**フルスタック開発**を容易に。
    - 非同期 I/O に強く、リアルタイムアプリなどに適応。
  - **TypeScript との組み合わせ**: Node.js アプリでも TypeScript 導入で、バックエンドも型安全に開発。
- **TypeScript & React**:
  - 役割: 高度な**ユーザーインターフェース**構築、ユーザー体験向上。
  - 強み:
    - 宣言的 UI、コンポーネントによる再利用性、仮想 DOM による効率的な UI 更新。
    - TypeScript による型安全開発がフロントエンド開発の複雑さを大幅軽減。
- 結論:
  - 競合ではなく、ウェブサイト全体を構築するための**異なる役割を持つ、補完しあうツール**。
  - Node.js で API を構築し、TypeScript と React でその API を利用するフロントエンド開発が現代の一般的パターン。

まとめ - 次のステップへ

- **TypeScript** と **React** は、現代ウェブ開発において堅牢でインタラクティブなフロントエンド構築に不可欠なツール。
- Node.js などバックエンド技術との組み合わせで、より強力で高機能なウェブアプリ構築が可能。
- 次のステップ:
  - 公式ドキュメントの参照:
    - React: `https://react.dev/`
    - TypeScript: `https://www.typescriptlang.org/`
  - 簡単な Todo リストアプリなど、実際に手を動かすこと。