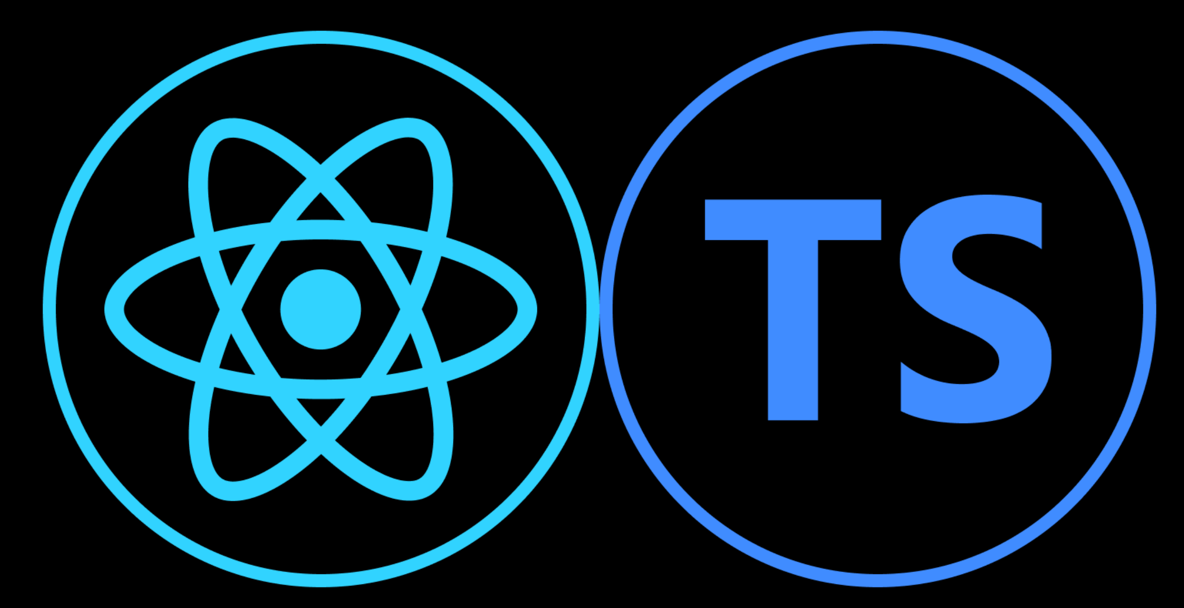


TypeScript マスター with React

三好アキ

TypeScript × React に徹底フォーカス

ー目次ー

はじめに

P.4 ― 本書のねらい
P.4 ― 本書の対象読者
P.5 ― 本書の構成
P.6 ― コードのダウンロードと本書で使うツール
P.6 ― ターミナルの使い方
P.7 ― Node.js
P.7 ― npm
P.8 ― VS Code
P.8 ― エラーが発生した場合の対処方法

第1章　TypeScriptについて知ろう

P.9 ― この章で学ぶこと
P.9 ― この章ですること
P.10 ― データとTypeScript
P.12 ― TypeScriptをむずかしいと感じる理由
P.13 ― TypeScriptの追加機能
P.16 ― Reactの予備知識（コンポーネントとfunctionの記法）

第2章　TypeScriptでReactアプリを開発しよう　その1

P.21 ― この章で学ぶこと
P.22 ― この章ですること
P.23 ― 開発ツールのインストール（React + Vite + TypeScript）
P.27 ― ReactコンポーネントをTypeScriptで書く
P.40 ― アプリの構成を確認する
P.45 ― イベントとfunctionに型をつける　その1
P.53 ― イベントとfunctionに型をつける　その2
P.61 ― 複雑な構造のデータに型をつける
P.70 ― フォームの入力文字を消す
P.72 ― 画像とCSSを追加する
P.74 ― ローディングを追加する
P.76 ― childrenに型をつける
P.84 ― アプリをオンラインで公開する（デプロイ）

第3章　TypeScriptだけを使ってみよう

P.85 ― この章で学ぶこと
P.85 ― この章ですること

P.86 — TypeScriptの開発セットアップ
P.91 — 配列の型定義
P.92 — typeとinterface
P.94 — 入れ子になった型定義（lookup型）

第4章　TypeScriptでReactアプリを開発しよう　その2

P.95 — この章で学ぶこと
P.95 — この章ですること
P.96 — アプリの構造の確認
P.101 — ツールの準備とクリーンアップ
P.102 — ページとルーティングの準備
P.107 — ユーザー登録機能を開発する
P.119 — ログイン機能を開発する
P.127 — 複数のデータを受け取るコンポーネントを開発する
P.144 — 柔軟な型を定義してコンポーネントを汎用化する
P.150 — 外部データを取得するコンポーネントを開発する
P.158 — ローディングとボタンを開発する
P.165 — 複数の型定義をひとつのコンポーネントで使う方法
P.178 — CSSと画像を追加する
P.181 — アプリをオンラインで公開する（デプロイ）

第5章　TypeScriptをもっと使ってみよう

P.182 — この章で学ぶこと
P.182 — この章ですること
P.183 — セットアップ
P.184 — 型の確認方法
P.185 — CSSProperties
P.188 — ComponentProps
P.191 — 引数とreturnがあるfunctionの型定義
P.193 — 型アサーション（as）
P.195 — as const
P.199 — ユーティリティタイプ（Type Utility）
P.203 — anyとunknownの違い
P.205 — ジェネリクス
P.218 — インデックス・シグネチャ（Index Signature）
P.220 — enum
P.221 — あとがき
P.223 — 著者について

はじめに

本書のねらい

2020年ごろまで、TypeScriptはオプション的なスキル、いわば「知っていたらベター」という扱いでしたが、近年ではフロントエンド・エンジニアならば知っておかなければならない必須スキルとなりました。これはReact開発において特に顕著で、企業案件や商業目的のReactアプリはTypeScriptで書くのが今はデフォルトになっています。

TypeScriptだけ、あるいはReactだけにフォーカスした書籍はいくつもありますが、本書では特に「TypeScriptをReactで使う」という点に重点を置いています。TypeScriptの機能の中には、「Reactでよく使うもの」と「Reactではほとんど使わないもの」があるからです。

これまでJavaScriptでしかReactを書いたことのなかったビギナーの方でも、TypeScriptの基礎から発展的内容まで幅広くカバーしている本書を使えば、自信をもってTypeScriptでReactアプリを開発できるようになれるでしょう。

本書の対象読者

本書を読む上でTypeScriptの知識や経験は必要ありません。Reactの深い知識や高度なスキルも必要ありません。しかし、Reactアプリを少なくとも数回は作ったことのある人が本書の対象としている読者です。具体的な知識としては以下のものになります。

- Reactアプリの構造
- state
- props
- イベント
- APIからのデータ取得
- レンダリング
- コンポーネント／function

最後の「コンポーネント／function」は非常に重要なので、次章で簡単に復習します。上記のReactの知識に自信のない人や、React自体の理解をもっと深めたい人は、JavaScriptの基礎の基礎からReactの発展的内容までカバーした拙著「Reactマスター Zero To Hero」を参考にしてください。

本書の構成

第1章では、TypeScriptが使われる理由や特徴、そしてビギナーがTypeScriptをむずかしく感じる理由を紹介します。章の後半は、Reactコンポーネントとfunctionの記法についての復習です。

第2章ではシンプルなReactアプリをTypeScriptで開発します。

第3章はコラム的な短い章です。TypeScriptだけにフォーカスして、TypeScriptのベーシックな事項を紹介します。

第4章では、2つ目のReact + TypeScriptアプリを開発します。難易度は第2章よりも上がっており、やや高度なTypeScriptのテクニックについても解説しています。

最後の第5章では、それまでにカバーできなかった事項を紹介します。

本書のメインは第2章と第4章です。この2つの章では実際にアプリ開発をしながら、React + TypeScriptの基礎から発展的内容までを解説していきます。

なお、TypeScriptだけを扱う章（第3章）が、React + TypeScriptの2つを扱う章（第2章）よりも後ろに置かれていることには理由があります。本書を読む読者の目標は、「TypeScriptだけを使って開発したい」ではなく「TypeScriptとReactを使って開発したい」という点にあると思います。「React + TypeScript」という目標地点の高さを先に確認しておけば、それが思っていたほどは高くはないことに気がつけ、その道のりの途中にある「TypeScript」にも心に余裕を持った状態でアプローチできるでしょう。

TypeScriptの追加機能

先ほど「JavaScript + 追加機能 = TypeScript」と書きましたが、この追加機能の中でもっとも特徴的なものが「型（かた／Type）」と呼ばれるものです。

「型」という言葉は馴染みがなく、むずかしく聞こえるかもしれません。しかし、これは単純にデータの「種類」と「形」のことです。実例を見れば、どれだけ単純で簡単なものであるのかわかります。

次のようなデータがあったとします。

```
1, 2, 3, 4, 5
```

見ての通り、このデータの種類は「数字」、つまり「number」です。次の例を見てみましょう。

```
"三島", "谷崎", "川端"
```

このデータの種類は「文字」です。ダブルクオテーション（`" "`）、あるいはシングルクオテーション（`' '`）で挟まれたものをプログラミング用語では「文字列」と呼ぶので、これは「string」というデータの種類になります。データの「種類」は次の7つしかないので、覚えることに困難はないでしょう。

```
string、number、boolean、null、undefined、bigint、symbol
```

以上がデータの「種類」です。次はデータの「形」を見てみましょう。

```
{
    name: "メガネ",
    color: "赤色",
    price: "5500円",
}
```

このように `{` と `}` で挟まれたデータはオブジェクトと呼ぶので、このデータの「形」は「オブジェクト」になります。次のようなシンプルなものもオブジェクトです。

```
{
    season: "春",
}
```

Reactの予備知識（コンポーネントとfunctionの記法）

次章以降を進める上で確認しておきたいReactの知識として、functionの記法、イベントの記法、そして分割代入についての説明をします。

functionの記法

Reactコンポーネントの実態はfunctionです。そしてfunctionの書き方にはいくつかの種類があります。functionの基本的な構造は、次のようになっています。

```
function 名前(){

    実行したい操作

    return 操作後のデータ
}
```

これは `const` を使って次のようにも書けます。「名前」の位置が変わっていることに注意してください。

```
const 名前 = function(){

    実行したい操作

    return 操作後のデータ
}
```

アローfunction（アロー関数）を使うと次のようにも書けます。 `function` という文字が `=>` に変わり、場所が右に移動しています。Reactのコードでよく見かけ、そして次章以降の開発にも関係するのがこの記法です。

```
const 名前 = () => {

    実行したい操作

    return 操作後のデータ
}
```

なおfunctionには「無名function」と呼ばれるものもあり、これは文字通り「名前」のないfunctionです。

第2章　TypeScriptでReactアプリを開発しよう　その1

この章で学ぶこと

React + Vite + TypeScriptのセットアップ
型定義の書き方
useStateの型
propsの型
分割代入したpropsの型の記法
イベントの型
イベントの型の記法
functionの型の構造
非同期処理の型
型を確認する方法
型推論
any型
複雑な構造をしたデータの型
Layoutコンポーネント
ローディングの設定
childrenの型
JSX.Element
React.ReactNode
React.ReactElement
デプロイ（Netlify）

開発ツールのインストール（React + Vite + TypeScript）

従来のReact開発では「create-react-app」というツールを使うのが一般的でした。しかし2023年春以降メンテナンスは停止されており、React公式サイトでも使用が推奨されていないので、本書では「create-react-app」に代わって人気を集める「React + Vite」を使います。まずインストールをしましょう。

ターミナル上で、「React + Vite」をインストールしたいフォルダに移動します。ここでは「ダウンロード」フォルダにいるものとします。次のコマンドをターミナルに打って「Enter」で実行してください。

```
npm create vite@latest
```

いくつか質問が出るので回答していきます。もしここで次の質問が出たら、「Yes」を表す「y」を押して先に進んでください。

```
Need to install the following packages:
    create-vite@5.2.1
Ok to proceed? (y)
```

最初の質問はこのアプリの名前で、ここで入力したものがフォルダの名前にも使われます。

```
? Project name: › vite-project
```

名前は好きなもので構いませんが、ここでは「meal-finder-ts」と打って「Enter」を押しましょう。次の質問が出ます。

```
✔ Project name: … meal-finder-ts
? Select a framework: › - Use arrow-keys. Return to submit.
    Vanilla
    Vue
❯   React
    Preact
    Lit
    Svelte
    Solid
    Qwik
    Others
```

Reactを使った開発をしたいので、キーボードの矢印キーで「React」を選んで「Enter」で決定しましょう。次は、開発言語にJavaScriptとTypeScriptのどちらを使うかを問う質問が出ます。

ここで、functionの扱っているデータは2つあるのがわかります。ひとつは `()` 内の「渡すデータ」、もうひとつは操作後の「returnするデータ」です。TypeScriptではこれら2つのデータの型も書く必要があります。書く場所を確認しましょう。

```
const 名前 = (データ: 渡すデータの型): returnするデータの型 => {

    実行したい操作

    return 操作後のデータ
}
```

これをHeaderコンポーネントに当てはめて考えると、次のようになります。

```
const Header = (props: propsの型情報): returnの型情報 => {
    return (
      ...
```

では、Reactコンポーネントでreturnされるものとは何でしょうか。 `return` のあとを見ると `(` があり、 `<header>` などのHTMLタグが見えます。しかし、これは実はHTMLではなく、JSXというHTMLを真似たタグ要素（Element）です。なので、このHeaderコンポーネントでreturnされているものはJSX要素です。これは次のように書きます。

```
const Header = ({ text, time, color }: HeaderProps): JSX.Element => {
    return (
      ...
```

ここでは `JSX.Element` の代わりに `React.ReactNode` も使えます。

```
const Header = ({ text, time, color }: HeaderProps): React.ReactNode => {
    return (
      ...
```

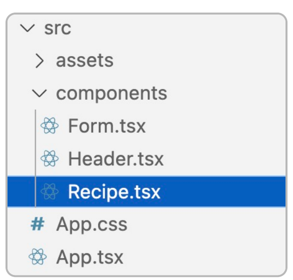
`React.ReactNode` よりも定義がやや厳しいのが `JSX.Element` ですが、この2つの違いは重要ではありません。一般的にreturnの型情報は書かないことが多いからです。いま書いたコードを消して、 `=>` の上にマウスを置くと次のように表示されます。これはTypeScriptがreturnの型情報を推測してくれているからです。

```
function({ text, time, color }: HeaderProps): JSX.Element
=> {
```

TypeScriptを使っていると、私たちがあえて型を書かなくても推測してくれている場面が多くあります。これは「型推論」と呼ばれます。推測してくれている型情報はあえて書く必要はありません。なお

複雑な構造のデータに型をつける

`components` フォルダに `Recipe.tsx` を作り、次のコードを打ってください。




```
// Recipe.tsx

const Recipe = () => {
    return (
        <>
            レシピ
        </>
    )
}

export default Recipe
```

`App.tsx` で読み込みます。

```
// App.tsx

import { useState } from "react"
import Header from "./components/Header"
import Form from "./components/Form"
import Recipe from "./components/Recipe"   // 追加

const App = () => {

    ...

            <Form setMealName={setMealName}
                  getMealData={getMealData}
            />
            <Recipe/>    // 追加
        </div>
    )
...
```

第3章　TypeScriptだけを使ってみよう

この章で学ぶこと

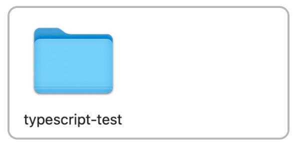
TypeScriptの開発環境セットアップ
TypeScriptのコンパイル方法
TypeScriptとJavaScriptのコード比較
tsconfig.jsonの設定
interfaceとtypeの違い
配列の型定義
インターセクション型
lookup型

この章ですること

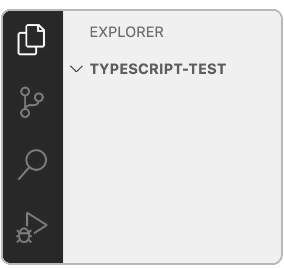
この章ではTypeScriptだけにフォーカスして、TypeScriptのセットアップと使い方を紹介します。短い章なので、次章の2つ目のアプリ制作に進む前に軽く読み進めてください。

TypeScriptの開発セットアップ

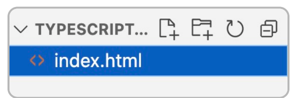
フォルダ「typescript-test」を作ってください。名前は好きなもので構いません。



VS Codeを開き、上部メニューバー「File」→「Open...」から選択するか、もしくは直接ドラッグ & ドロップで「typescript-test」フォルダを開きます。この時点では中身は何もありません。



ブラウザで表示を確認したいので、ファイル `index.html` を作り、次のコードを書いてください。




```
// index.html

<!doctype html>
<html>
    <head>
        <title>TypeScriptテスト</title>
    </head>
    <body>
        <h1>こんにちは</h1>
        <script src="./main.js"></script>
    </body>
</html>
```

ごく普通のHTMLファイルで、 `</script>` タグで `main.js` というJavaScriptファイルを読み込んでいます。このHTMLファイルをブラウザにドラッグ＆ドロップすると「こんにちは」と表示されます。

第4章　TypeScriptでReactアプリを開発しよう　その2

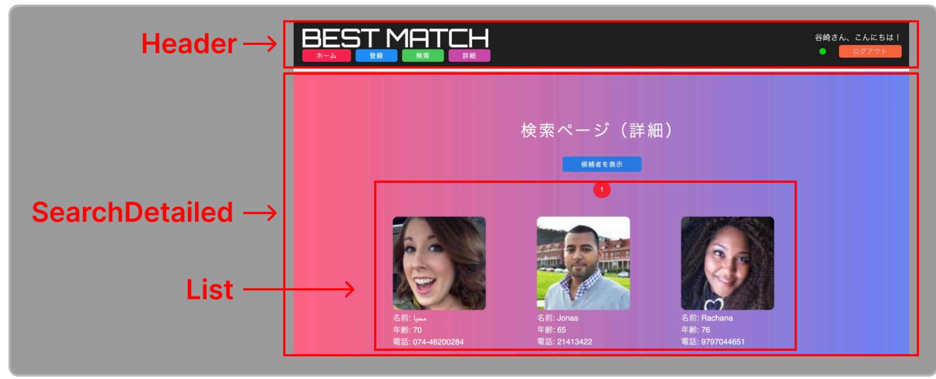
この章で学ぶこと

タプル型
ユニオン型
リテラル型
型引数
ジェネリクス
ジェネリクスの記法（Arrow function）
型ガード
ナローイング
ページ設定
useStateでオブジェクトを扱う方法
Local Storage
イベント・オブジェクトの型
ログイン状態の維持の仕組み
ログイン状態によって表示を変える
useEffectの働き
非Nullアサーション
Reactコンポーネントの汎用化
柔軟な型を定義する方法
非同期処理
型定義を共有する方法

この章ですること

本章では、2つ目のアプリを作りながらReact + TypeScriptの解説をしていきます。アプリ自体はシンプルですが、扱う内容の難易度は第2章よりもやや上がっています。

詳細ページは、検索ページとほぼ同じでHeader、SearchDetailed、そしてListコンポーネントです。



なおどちらのページでも、Listが表示される前にローディングのアニメーションが出るので、Loadingコンポーネントが使われているのもわかります。

すべてのページで表示されているHeaderコンポーネントは、ヘッダー画像と5つのボタンから構成されています。

このアプリでは、いたるところで似たような見た目のボタンが使われているので、Buttonコンポーネントが必要なのもわかります。唯一すこし変わった形をしているのは、検索ページと詳細検索ページにある数の表示される赤丸ボタンです。

このアプリで使われているコンポーネントをまとめると次のようになります。

```
// ページ
Home.tsx
Register.tsx
Search.tsx
SearchDetailed.tsx

// その他
Header.tsx
List.tsx
Button.tsx
Loading.tsx
```

以上、アプリの仕組みと必要なコンポーネントがわかったので、開発を始めていきましょう。

柔軟な型を定義してコンポーネントを汎用化する

完成見本を見ると、ヘッダー以外にもすべてのページでボタンが使われているのがわかります。まずはHomeコンポーネントです。次のようにコードを修正してください。

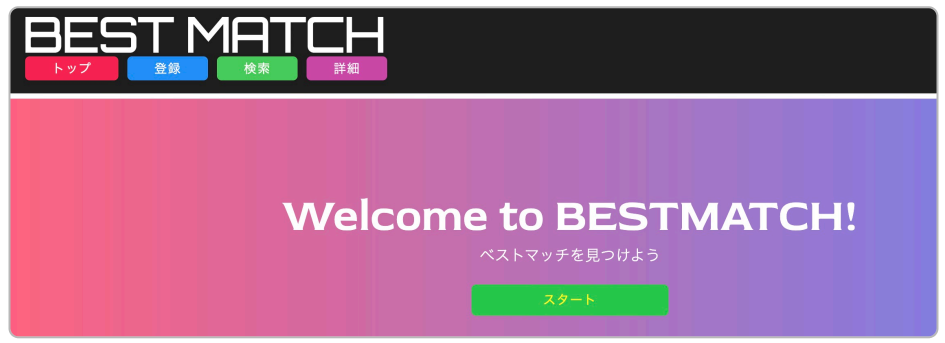
```
// Home.tsx

import { Link } from "react-router-dom"
import Button from "../components/Button"

const Home = () => {
    return (
        <div className="home">
            <h1>Welcome to BESTMATCH!</h1>
            <p>ベストマッチを見つけよう</p>
            <Link to="/register">
                <Button text="スタート"
                        buttonColor="#2dc84d"
                />
            </Link>
        </div>
    )
}

export default Home
```

ここで完成見本を見ると、HomeコンポーネントのボタンはHeaderコンポーネントのボタンよりも横幅があるのがわかります。さらに文字の色も黄色です。



2つメッセージがありますが、注目すべきは2つ目です。

```
Property 'firstName' does not exist on type 'CandidatePropsD'」

（firstNameは、CandidatePropsDに存在していません）
```

これはその通りで、`firstName`は検索ページの30件のデータを格納している`candidates`にしかない項目なので、写真付きの詳細検索ページの`candidatesD`には存在していません。つまりこのメッセージでTypeScriptは、「`CandidateProps[]`と`CandidatePropsD[]`、どちらの型情報を使えばいいのか判断ができない」と言っているのです。

もしこれが、次のようにシンプルなプリミティブ型なら、TypeScriptが正しく推測してくれる場合があります。

```
type ListProps = {
    candidates: string | number;
}
```

しかし複数の項目を含んだ型定義の場合は、開発者がTypeScriptに教える必要があるのです。ここで解決方法は次の3つがあります。

1：型アサーション（as）
2：型ガード
3：ジェネリクス

第5章　TypeScriptをもっと使ってみよう

この章で学ぶこと

型の確認方法
CSSProperties
ComponentProps
ComponentPropsWithoutRef
restオペレーター
functionの型定義
型アサーション（as）
as const
anyとunknown
Type Utility
Pick
Omit
Partial
Required
Record
ジェネリクス
インデックス・シグネチャ
enum

この章ですること

本章では、これまでに触れる機会のなかったReact + TypeScriptの便利な型や、柔軟な型定義を可能にするTypeScriptの記法について解説します。

ComponentProps

`App.tsx` のコードを、 `<Button/>` タグに `style` オブジェクトを渡す前の状態に戻します。

```
// App.tsx

import Button from "./Button"

const App = () => {
    return (
        <>
            <h1>こんにちは</h1>
            <Button/>
        </>
    )
}

export default App
```

Buttonコンポーネントに `<button>` タグで使う属性を渡します。

```
// App.tsx

import Button from "./Button"

const App = () => {
    return (
        <>
            <h1>こんにちは</h1>
                    // ⬇追加
            <Button disabled={true}
                    type="submit"
                    name="button"
                    value="abc"
            />
                    // ⬆追加
        </>
    )
}

export default App
```

`as const` は配列にも使えます。次のようなコードがあったとき、 `array[4]` という存在しないデータを指定してもエラーは出ません。

```
// main.ts

const array = ["りんご", "ぶどう", "みかん"]

array[0]   // りんご
array[2]   // みかん
array[4]   // エラーなし
```

しかし `as const` を付けると、エラーが出ます。

```
// main.ts

const array = ["りんご", "ぶどう", "みかん"] as const

array[0]   // りんご
array[2]   // みかん
array[4]   // エラー
```

ジェネリクス

前章でも登場したジェネリクスは、「とりあえずの仮置きとして置かれた、実質的な中身のない型情報」のことです。「型はあとで決める」という柔軟さと、複数のオブジェクトやfunctionに対して使い回すことのできる汎用性を得ることができます。しかしこのような説明だけでは理解しにくいので、コードで確認しましょう。

本章ではわかりやすいように、ジェネリクスを「型定義に対して使うもの」と「functionに対して使うもの」に分け、最後に両方を合わせたケースを紹介します。

型定義のジェネリクス

次のコードを書いてください。

```
// main.ts

const userOne: any = {
    data: "太郎",
    color: "赤"
}
```

`any` の代わりに、しっかりした型定義を用意しましょう。

```
// main.ts

// ⬇追加
type User = {
    data: string;
    color: string;
}
// ⬆追加

const userOne: User = {    // 修正
    data: "太郎",
    color: "赤"
}
```