



進捗状況

Typescript と React コンポーネントの作成

B4 小林紹子

September 25, 2025

目次

1 Typescript

2 React 入門

3 React コンポーネントの作成

4 React Hook

- 状態のフック
- メモ化のフック

Typescript

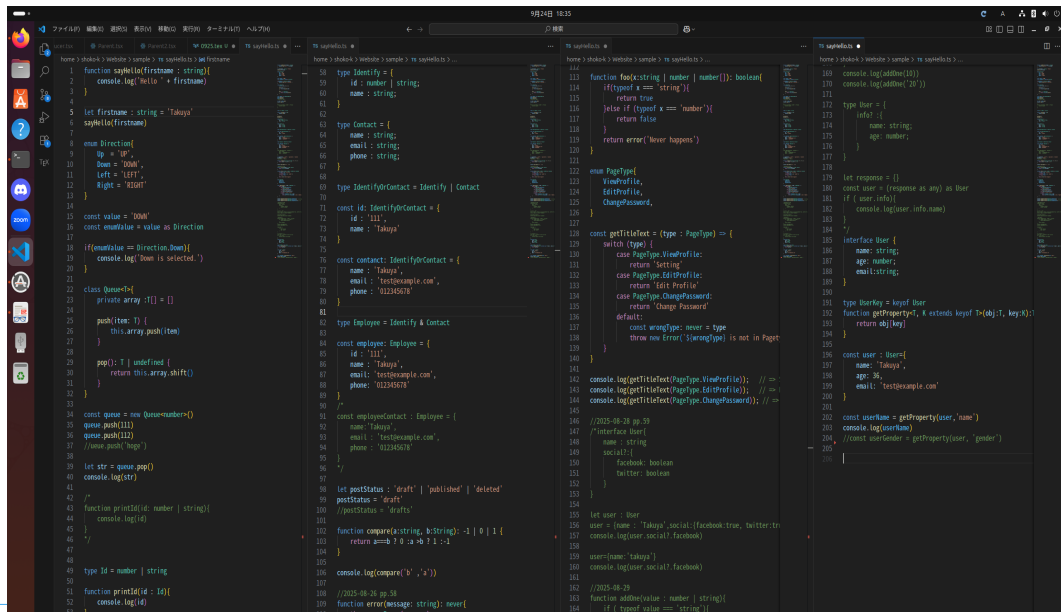
- **Typescript のメリット**

- Typescript は Javascript を拡張している上方互換言語.
- 以下の機能を追加している.
 - 型定義
 - インタフェースとクラス
 - null/undefined 安全など

- **Typescript のデメリット**

- 規模によってコンパイルに時間がかかる.

Typescript の自作サンプル一覧



```
1 function sayHello(firstname: string) {
2   console.log('hello ' + firstname)
3 }
4
5 let firstname: string = 'Takuya'
6 sayHello(firstname)
7
8 enum Direction {
9   Up = 'UP',
10  Down = 'DOWN',
11  Left = 'LEFT',
12  Right = 'RIGHT'
13 }
14
15 const value = 'DOWN'
16 const enumValue = value as Direction
17
18 if (enumValue == Direction.Down) {
19   console.log('Down is selected.')
20 }
21
22 class Queue<T> {
23   private array: T[] = []
24
25   push(item: T) {
26     this.array.push(item)
27   }
28
29   pop(): T | undefined {
30     return this.array.shift()
31   }
32 }
33
34 const queue = new Queue<number>()
35 queue.push(111)
36 queue.push(112)
37 //queue.push('hoge')
38
39 let str = queue.pop()
40 console.log(str)
41
42 /*
43 function printId(id: number | string) {
44   console.log(id)
45 }
46 */
47
48 type Id = number | string
49
50 function printId(id: Id) {
51   console.log(id)
52 }
53
54 type Identity = {
55   id: number | string;
56   name: string;
57 }
58
59 type Contact = {
60   name: string;
61   email: string;
62   phone: string;
63 }
64
65 type IdentifyOrContact = Identify | Contact
66
67 const id: IdentifyOrContact = {
68   id: '111',
69   name: 'Takuya'
70 }
71
72 const contact: IdentifyOrContact = {
73   name: 'Takuya',
74   email: 'test@example.com',
75   phone: '012345678'
76 }
77
78 type Employee = Identify & Contact
79
80 const employee: Employee = {
81   id: '111',
82   name: 'Takuya',
83   email: 'test@example.com',
84   phone: '012345678'
85 }
86
87 const employeeContact: Employee & Contact = {
88   name: 'Takuya',
89   email: 'test@example.com',
90   phone: '012345678',
91   id: '111'
92 }
93
94 //2025-08-28 pp.59
95 /*interface User {
96   name: string
97   age: number
98   email: string
99   social: {
100     facebook: boolean
101     twitter: boolean
102   }
103 }
104
105 let user: User
106 user = { name: 'Takuya', social: { facebook: true, twitter: true }
107 console.log(user.social?.facebook)
108
109 user = { name: 'Takuya' }
110 console.log(user.social?.facebook)
111
112 //2025-08-29
113 function addOne(value: number | string) {
114   if (typeof value === 'string') {
115     return value + 1
116   }
117   return value + 1
118 }
119
120 console.log(addOne(10))
121 console.log(addOne('20'))
122
123 type User = {
124   info: {
125     name: string;
126     age: number;
127   }
128 }
129
130 let response = {}
131 const user = (response as any) as User
132 if (user.info) {
133   console.log(user.info.name)
134 }
135
136 interface User {
137   name: string;
138   age: number;
139   email: string;
140 }
141
142 type UserKey = keyof User
143 function getProperty<T, K extends keyof T>(obj: T, key: K): T[K] {
144   return obj[key]
145 }
146
147 const user: User = {
148   name: 'Takuya',
149   age: 36,
150   email: 'test@example.com'
151 }
152
153 const userName = getProperty(user, 'name')
154 console.log(userName)
155
156 //const userGender = getProperty(user, 'gender')
157 console.log(userGender)
```

React の構成

React のプロジェクトは以下ようになっており, 最終的には結合・最小化された HTML/CSS/Javascript のコードとなって出力される.

```
shoko-k@shoko-k-MS-7E34:~/Website/react-sample$ tree -L 1
```

```
.
├── README.md
├── node_modules
├── package-lock.json
├── package.json
├── public
├── src
└── tsconfig.json
```

```
4 directories, 4 files
```

index.tsx をビルドした JavaScript が実行される

index.tsx には, JSX タグを用いて HTML タグやコンポーネントを扱う.

```
1 import React from 'react';
2 import ReactDOM from 'react-dom/client';
3 import './index.css';
4 import App from './App';
5 import reportWebVitals from './reportWebVitals';
6 import Hello from './components/Hello';
7 import Name from './components/Name';
8 import Message from './components/Message';
9 //import Parent from './components/ContainerSample';
10 import Parents from './components/ContainerSample2';
11 import Page from './components/ContextSample';
12 import Counter from './ReactSample/useState';
13 import Counter2 from './ReactSample/useReducer';
14 import { Parent2 } from './components/Parent2';
15
16 const root = ReactDOM.createRoot(document.getElementById('root') as HTMLElement);
17 root.render(
18   //以下はJSXタグ (JavaScriptやTypescript中にHTMLタグをそのまま書き込めるもの)
19   <React.StrictMode>
20     <App />
21     <Hello />
22     <Name />
23     <Message />
24     <Parent2 />
25     <Parents />
26     <Page />
27     <Counter initialValue={0} />
28     <Counter2 initialValue={0} />
29   </React.StrictMode>
30 );
31
```

App.tsx

それぞれの<App/>などは App.tsx などからインポートする。
App.tsx で App 関数を作成し, JSX タグを用いて HTML 要素を返す。

```
1  import React from 'react';
2  import logo from './logo.svg';
3  import './App.css';
4
5  function App() {
6    return (
7      <div className="App">
8        <header className="App-header">
9          <img src={logo} className="App-logo" alt="logo" />
10         <p>
11           Edit <code>src/App.tsx</code> and save to reload.
12         </p>
13         <a
14           className="App-link"
15           href="https://reactjs.org"
16           target="_blank"
17           rel="noopener noreferrer"
18         >
19           Learn React
20         </a>
21       </header>
22     </div>
23   );
24 }
25
26 export default App;
```

Web ページができるまでの大まかな流れ！

- 1 public/index.html が読み込まれて, ブラウザに描画される.
- 2 ブラウザが JavaScript のコードを取得し, React を使ったコードの実行を開始する.
- 3 render で与えられた App を root オブジェクト作成時に与えられた root という ID を持つ要素以下に描画する.

Web ページができるまでの大まかな流れ II

```
1 <!DOCTYPE html>
2 <html lang="en">
3   <head>
4     <meta charset="utf-8" />
5     <link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
6     <meta name="viewport" content="width=device-width, initial-scale=1" />
7     <meta name="theme-color" content="#000000" />
8     <meta
9       name="description"
10      content="Web site created using create-react-app"
11    />
12    <link rel="apple-touch-icon" href="%PUBLIC_URL%/logo192.png" />
13    <!--
14      manifest.json provides metadata used when your web app is installed on a
15      user's mobile device or desktop. See https://developers.google.com/web/fundamentals/web-app-manifest/
16    -->
17    <link rel="manifest" href="%PUBLIC_URL%/manifest.json" />
18    <!--
19      Notice the use of %PUBLIC_URL% in the tags above.
20      It will be replaced with the URL of the 'public' folder during the build.
21      Only files inside the 'public' folder can be referenced from the HTML.
22
23      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
24      work correctly both with client-side routing and a non-root public URL.
25      Learn how to configure a non-root public URL by running 'npm run build'.
26    -->
27    <title>React App</title>
28  </head>
29  <body>
30    <noscript>You need to enable JavaScript to run this app.</noscript>
31    <div id="root"></div>
32    <!--
33      This HTML file is a template.
34      If you open it directly in the browser, you will see an empty page.
35
36      You can add webfonts, meta tags, or analytics to this file.
37      The build step will place the bundled scripts into the <body> tag.
```

React コンポーネントとは

- 画面にどういう要素を表示するかを定義する設計図.
- 表示要素とその要素の挙動をセットにした部品.
- 何度も再利用できる部品のようなもの.

```
1 //Helloはclickするとアラートを出すテキストを返す
2 const Hello = () => {
3   //クリック時に呼ばれる関数
4   const onClick = () => {
5     //アラートを出す
6     alert('hello');
7   };
8   const text = 'Hello, React';
9
10  //テキストを子に持つdiv要素を返す
11  return (
12    //divのonClickにクリック時のコールバック関数を渡す
13    <div onClick={onClick}>{text}</div>
14  );
15 };
16
17 //外部からHelloを読み込めるようになる
18 export default Hello;
```

Figure: 例：Hello.tsx

テキスト入力

名前を入力するテキストボックスの作成.

```
1 import React from 'react';
2
3 const Name = () => {
4   const onChange = (e: React.ChangeEvent<HTMLInputElement>) => {
5     console.log(e.target.value);
6   };
7
8   return (
9     <div style={{ padding: '16px', backgroundColor: 'grey' }}>
10       <label htmlFor="name">名前</label>
11       <input id="name" className="input-name" type="text" onChange={onChange} />
12     </div>
13   );
14 };
15
16 export default Name;
```

Figure: Name.tsx

コンポーネントの組み合わせ

Text コンポーネントを利用して,message コンポーネントを作成する.

```
1  const Text = (props: { content: string }) => {
2    const { content } = props;
3    return <p className="text">{content}</p>;
4  };
5
6  const Message = (props: {}) => {
7    const content1 = 'This is parent component';
8    const content2 = 'Message uses Text component';
9
10   return (
11     <div>
12       /*contentというキーでコンポーネントにデータを渡す*/
13       <Text content={content1} />
14       <Text content={content2} />
15     </div>
16   );
17 };
18
19 export default Message;
```

Figure: Message.tsx

もっと自由な引数にする！

React.ReactElement で<p>この部分が背景色で囲まれます</p>が読み込まれる。

```
1  const Container = (props: { title: string; children: React.ReactElement }) => {
2    const { title, children } = props;
3
4    return (
5      <div style={{ background: 'red' }}>
6        <span>{title}</span>
7        <div>{children}</div>
8      </div>
9    );
10 };
11
12 const Parent = () => {
13   return (
14     <Container title="Hello">
15       <p>この部分が背景色で囲まれます</p>
16     </Container>
17   );
18 };
19 export default Parent;
```

Figure: ContainerSample.tsx

もっと自由な引数にする II

```
1 import React from 'react';
2
3 type ContainerProps = {
4   title: string;
5   children: React.ReactNode;
6 };
7
8 const Container = (props: ContainerProps): React.JSX.Element => {
9   const { title, children } = props;
10
11   return (
12     <div style={{ background: 'yellow' }}>
13       <span>{title}</span>
14       /*propsのchildrenを埋め込むと、このコンポーネントの開始タグと閉じタグで囲んだ要素を表示する*/
15       <div>{children}</div>
16     </div>
17   );
18 };
19
20 const Parents = (): React.JSX.Element => {
21   return (
22     //Containerを使用する際に、他の要素を囲って使用する
23     <Container title="Hello">
24       <p>この部分が背景色で囲まれる</p>
25     </Container>
26   );
27 };
28
29 export default Parents;
```

Figure: ContainerSample2.tsx

遠いところからもデータを受け取れるようにする！

- props: 親から子へ任意のデータを渡す.
- Context: 親以外からも必要なデータを参照する.

遠いところからもデータを受け取れるようにする II

```
1 import React from 'react';
2
3 //Titleを渡すためのContextを作成
4 const TitleContext = React.createContext('');
5
6 //Titleコンポーネントの中でContextの値を参照
7 const Title = () => {
8   //ConsumerでContextの値を参照
9   return (
10     <TitleContext.Consumer>
11       {(title) => {
12         return <h1>{title}</h1>;
13       }}
14     </TitleContext.Consumer>
15   );
16 };
17
18 const Header = () => {
19   return (
20     <div>
21       {/*HeaderからTitleには何もデータを渡さない*/}
22       <Title />
23     </div>
24   );
25 };
26
27 //Pageコンポーネントの中でContextに値を渡す
28 const Page = () => {
29   const title = 'React Book';
30   //Providerを使いContextに値をセット
31   //Provider以下のコンポーネントから値を参照できる
32   return (
33     <TitleContext.Provider value={title}>
34       <Header />
35     </TitleContext.Provider>
36   );
37 };
38
39 export default Page;
```


React Hook

- React Hook：状態管理などクラスを書かずに使えるようになる機能.
- 全部で 15 種類ある.
- コンポーネントのコードを綺麗に保ち, コードの再利用性を高める.

useState

const[状態, 更新関数]=useState(初期状態)

更新関数を呼び出すと状態が変化し、コンポーネントが再描画される.

```
1 import { useState } from 'react';
2
3 type CounterProps = {
4   initialValue: number;
5 };
6
7 //propsのinitialValueに初期値をセット
8 //<Counter initialValue= {0}/>
9 const Counter = (props: CounterProps) => {
10   const { initialValue } = props;
11   const [count, setCount] = useState(initialValue);
12
13   return (
14     <div>
15       <p>Count: {count}</p>
16       /*setCountを呼ぶことで状態を更新*/
17       <button onClick={() => setCount(count - 1)}>-</button>
18       <button onClick={() => setCount((prevCount) => prevCount + 1)}>+</button>
19     </div>
20   );
21 };
22
23 export default Counter;
```

Figure: カウンター

useReducer I

useState よりも多くの複雑な状態遷移を扱える

現在の状態と action を渡すと次の状態を返す reducer という関数を用いる。
ボタンが押されると dispatch 関数を使って action を発出する。

useReducer II

```
1 import { useReducer } from 'react';
2
3 type Action = 'DECREMENT' | 'INCREMENT' | 'DOUBLE' | 'RESET';
4
5 const reducer = (currentCount: number, action: Action) => {
6   switch (action) {
7     case 'INCREMENT':
8       return currentCount + 1;
9     case 'DECREMENT':
10      return currentCount - 1;
11    case 'DOUBLE':
12      return currentCount * 2;
13    case 'RESET':
14      return 0;
15    default:
16      return currentCount;
17  }
18 };
19
20 type CounterProps = {
21   initialValue: number;
22 };
23
24 const Counter2 = (props: CounterProps) => {
25   const { initialValue } = props;
26   const [count, dispatch] = useReducer(reducer, initialValue);
27
28   return (
29     <div>
30       <p>Count : {count}</p>
31       <button onClick={() => dispatch('DECREMENT')}>-</button>
32       <button onClick={() => dispatch('INCREMENT')}>+</button>
33       <button onClick={() => dispatch('DOUBLE')}>×2</button>
34       <button onClick={() => dispatch('RESET')}>Reset</button>
35     </div>
36   );
37 };
38
39 export default Counter2;
```

メモ化コンポーネント I

親コンポーネントが再描画されると子コンポーネントも自動的に描画される。
この再描画の伝搬を止めるのに利用される。
今回は memo 関数を用いている。

メモ化コンポーネント II

```
1 import React, { memo, useState } from 'react';
2
3 type FizzProps = {
4   isFizz: boolean;
5 };
6
7 //Fizzは通常の関数コンポーネント
8 //isFizzはtrueの場合はFizzと表示し、それ以外は何も表示しない
9 //isFizzの変化に関わらず、親が再描画されるとFizzも再描画される
10 const Fizz = (props: FizzProps) => {
11   const { isFizz } = props;
12   console.log('Fizzが再描画されました, isFizz = ${isFizz}');
13   return <span>{isFizz ? 'Fizz' : ''}</span>;
14 };
15
16 type BuzzProps = {
17   isBuzz: boolean;
18 };
19
20 //Buzzはメモ化した関数コンポーネント
21 //isBuzzはtrueの場合はBuzzと表示し、それ以外は何も表示しない
22 //親コンポーネントが再描画されても、isBuzzが変化しない限りはBuzzは再描画しない
23 const Buzz = memo<BuzzProps>((props) => {
24   const { isBuzz } = props;
25   console.log('Buzzが再描画されました, isBuzz=${isBuzz}');
26   return <span>{isBuzz ? 'Buzz' : ''}</span>;
27 });
28
29 //この形でexportしたときはimport {Parent} fromで読み込む
30 export const Parent = () => {
31   const [count, setCount] = useState(1);
32   const isFizz = count % 3 === 0;
33   const isBuzz = count % 5 === 0;
34
35   console.log('Parentが再描画されました, count = ${count}');
36   return (
37     <div>
38       <button onClick={() => setCount((c) => c + 1)}>+1</button>
39       <p>現在のカウント: ${count}</p>
40       <p>
41         <Fizz isFizz={isFizz} />
42         <Buzz isBuzz={isBuzz} />
43       </p>
44     </div>
45   );
46 };
```

① Download the React DevTools for a better development experience: <https://react.dev/Link/react-devtools>

Parentが再描画されました, count = 1
Fizzが再描画されました, isFizz = false
Buzzが再描画されました, isBuzz=false
Parentが再描画されました, count = 2
Fizzが再描画されました, isFizz = false
Parentが再描画されました, count = 3
Fizzが再描画されました, isFizz = true
Parentが再描画されました, count = 4
Fizzが再描画されました, isFizz = false
Parentが再描画されました, count = 5
Fizzが再描画されました, isFizz = false
Buzzが再描画されました, isBuzz=true
Parentが再描画されました, count = 6
Fizzが再描画されました, isFizz = true
Buzzが再描画されました, isBuzz=false
Parentが再描画されました, count = 7
Fizzが再描画されました, isFizz = false