

1つのtsxからVueとReact向けに ビルドする

STORES 株式会社 / ushironoko

話すこと

- tsx
- VueとReact
- ビルド

1つのtsxからVueとReact向けにビルドする

1つのtsxからVueとReact向けにビルドする

tsx

jsx を TypeScript で書いたもの。

jsx は HTML マークアップを JavaScript の関数の戻り値に定義できる。

これは

```
export default () => <Hello, World!/>;
```

こういう型になる

```
() => JSX.Element;
```

tsx

関数の中に書けるので、props で受けた値使うこともできる。

```
export default ({ text }: { text: string }) => <Hello, {text}!>;
```

```
({ text }: { text: string }) => JSX.Element;
```

tsx

tsxだけではコンポーネントに状態を持つことはできない。

状態を持つということは、副作用を持つということ。

```
const value = state("world!"); // stateを持つにはUIライブラリの力が必要  
export default () => <Hello, {value}!>;
```

副作用のある関数は、その作用をトリガーとして再レンダリングされなければならない。

stateの追跡やスケジューリング等、高度な機能が必要になるが、tsxはただのテンプレートなのでその機能は提供していない。

tsx

- tsxは、TypeScript の関数で定義できるテンプレート/コンポーネント
- 状態を持つには、UI ライブリ等の提供する state management runtime が必要

1つのtsxからVueとReact向けにビルドする

1つのtsxからVueとReact向けにビルドする

Reactにおけるtsx

React では `j(t)sx` を用いてマークアップをし、基本的にすべて関数コンポーネントになる。

関数コンポーネントに `state` を持たせるため、`useState` 等のフックが提供されている。

以下は「`setText`」を用いて `state` を更新すると、コンポーネントが再レンダリングされる。

```
import { useState } from "react";

const HelloWorld = () => {
  const [text, setText] = useState("Hello");

  return <button onClick={() => setText("こんにちは！")}>{text} World!</button>;
};

export default HelloWorld;
```

Reactにおけるtsx

React は独自のテンプレートをもたないが、独自の解釈でtsxを扱っている。

こうすることで、React18 における Form Action のような機能を実現している。

```
<form action={serverAction}>
```

通常 form の action 属性に関数を渡すことはできないが、React では可能。

TypeScript は React の jsx とそれ以外を区別しており、**tsc** でビルドする際の結果に影響する。

例えば **tsc --jsx react** でビルドすると、React の runtime で jsx を解釈するコードが生成される。

このコードは上記のコマンドでビルドすると

```
const HelloWorld = () => <div>Hello World!</div>;
```

こうなる

```
const HelloWorld = () => React.createElement("div", null, "Hello World!");
```

Reactにおけるtsx

tsc --jsx react-jsx

jsx runtimeをバンドルに埋め込み、依存を内部化する。

```
const HelloWorld = () => jsxRuntimeExports.jsx("div", null, "Hello World!");
```

tsc --jsx preserve --jsxFactory h

jsx コードをそのまま残し、指定された関数に置き換える。jsx の解釈を tsc が他のビルドツールに委ねた状態。

```
const HelloWorld = () => h("div", null, "Hello World!");
```

React では jsx 構文は最終的に createElement という関数に置き換わるが、

オプションを組み合わせることで、ビルド後の jsx 構文を誰に・どのように解釈させるか指示できる。

Vueにおけるtsx

Vue は独自のテンプレートを持っているが、tsx を用いることもでき、変換には Babel のプラグインを用いている。レンダー関数とJSX

Vue コンポーネントで tsx を記述した場合、それは VNode を生成するレンダー関数 `h` に置き換わる。様々な構文に対応しており、以下はその一部。

```
const App = () => <div>Hello World!</div>;  
  
import { defineComponent } from 'vue';  
  
const App = defineComponent({  
  setup() {  
    return () => (  
      <div>Hello World!</div>  
    );  
  },  
});  
  
h("div", null, "Hello World!"); // return VNode
```

さっき見ましたね？

```
h("div", null, "Hello World!"); // return VNode
```

j(t)sxの正体

- jsx は、それをビルドするツールによって異なる関数へ置き換えられる糖衣構文。
- React では React.createElement、Vue では h。
- tsc でも、そうでないツールでもビルドされうる。
- tsc は tsc でビルドしない時のためのオプションを提供している。

1つのtsxからVueとReact向けにビルドする

1つのtsxからVueとReact向けにビルドする

tsxからVueとReact向けにビルドする

- jsx は、それをビルドするツールによって異なる関数へ置き換えられる糖衣構文。
- React では React.createElement、Vue では h に変換される。
- state を持たせるにはランタイムが必要

tsxからVueとReact向けにビルドする

- jsx は、それをビルドするツールによって異なる関数へ置き換えられる糖衣構文。
- React では React.createElement、Vue では h に変換される。
- state を持たせるにはランタイムが必要

state を持たせると React か Vue のランタイムに依存してしまう。

言い換えれば、state を持たなければ、どちらでも使えるコンポーネントになる。

tsxからVueとReact向けにビルドする

関数型コンポーネントは、それ自身の状態を持たないコンポーネントの代替形態です。それらは純粋な関数のように動作します。`props` を受け取り、`vnode` を返します。コンポーネントのインスタンスを作成することなく（つまり、`this` はありません）、通常のコンポーネントのライフサイクルフックもなくレンダリングされます

関数型コンポーネント - <https://ja.vuejs.org#functional-components>

tsxからVueとReact向けにビルドする

関数型コンポーネントは、
純粋な関数のように動作します。
インスタンスを作成することなく（つまり、*this* はありません）、通常のコンポーネントのライフサイクルフックもなくレンダリングされます

コンポーネントの代替形態です。それらは
コンポーネントのイン

関数型コンポーネント - <https://ja.vuejs.org#functional-components>

この部分さえ守っていれば、関数コンポーネントとして認識される。

tsxからVueとReact向けにビルドする

このようなコンポーネント定義は、Vue でも React でも有効。

```
type Props = {
  liProps: { title: string; url: string }[];
  notificationEl: () => JSX.Element;
};

const Navigation = ({ liProps, notificationEl }: Props) => {
  return (
    <nav className=" ... ">
      <ul>
        {liProps.map(({ title, url }) => (
          <li className=" ... ">
            <a href={url}>{title}</a>
          </li>
        ))}
        <li>{notificationEl()}</li>
      </ul>
    </nav>
  );
};

export default Navigation;
```

tsxからVueとReact向けにビルドする

これをビルドすれば、Vue でも React でも使えるコンポーネントが手に入る。

ビルドツールはなんでも良いが、**unbuild** がおすすめ。

The screenshot shows the GitHub repository page for `unjs / unbuild`. The repository has 11 watchers, 90 forks, and 2.3k stars. It contains 3 branches and 93 tags. The main branch has the following commit history:

- renovate[bot]: chore(deps): update all non-major dependencies
- .github/workflows: chore(deps): update autofix-cl/action digest to ff86a55
- examples: refactor: add explicit return types (#412)
- src: fix(untyped): use schema module default export if is the o...
- test: chore: lint
- .editorconfig: initial commit
- .gitignore: feat: add --config to the CLI (#440)
- .prettierrc: chore: fix lint issue
- CHANGELOG.md: chore(release): v3.0.0-rc.11
- LICENSE: refactor: update repo
- README.md: docs: add more usage info (#401)
- eslint.config.mjs: chore: eslint ignore test/fixtures/dist
- package.json: chore(deps): update all non-major dependencies
- pnpm-lock.yaml: chore(deps): update all non-major dependencies
- renovate.json: refactor: update repo
- tsconfig.json: chore: stricter type checks (#413)
- vitest.config.ts: refactor: update repo

The repository is described as "A unified JavaScript build system" and includes links to Readme, MIT license, Activity, Custom properties, and Report repository. It has 15 releases, with the latest being v3.0.0-rc.11. The repository is used by 12.3k projects, including Next.js, Vite, and Tailwind CSS. There are 42 contributors, with 28 more listed.

tsxからVueとReact向けにビルドする

unjs/unbuild

1つの設定ファイルで複数のビルドを行うことができる。それぞれ別のエントリーポイントやビルダーを指定でき、オプションも分けられる。ビルダーのデフォルトは rollup。

rollup でコンポーネントをビルドする際に必要な設定はすでに unbuild に含まれている。

```
import { defineBuildConfig } from "unbuild";

export default defineBuildConfig([
  {
    // rollup で React 向けにビルドする
    entries: [
      { input: "Navigation.tsx", name: "Navigation.react", outDir: "dist/" },
    ],
  },
  {
    // rollup で Vue 向けにビルドする
    entries: [
      { input: "Navigation.tsx", name: "Navigation.vue", outDir: "dist/" },
    ],
  }
]);
```

しかし、これだけではまだ動かない。

tsxからVueとReact向けにビルドする

unjs/unbuild

- jsx は、それをビルドするツールによって異なる関数へ置き換えられる糖衣構文。
- React では React.createElement、Vue では h に変換される。👉
- state を持たせるにはランタイムが必要

tsxからVueとReact向けにビルドする

unjs/unbuild

デフォルトの設定では、jsx をどう解釈するのかは定義されていない。unbuild は内部的に esbuild を用いて jsx をコンパイルするため、esbuild オプションを渡せるようになっている。

esbuild のオプションに Vue 用の設定を渡すことで、Vue 用のコードとしてビルドすることができる。

```
entries: [
  {
    // ... 省略
    rollup: {
      esbuild: {
        tsconfigRaw: {
          compilerOptions: {
            jsx: "preserve", // jsx コードをそのまま残す
            jsxFactory: "h", // h 関数を使うことを指示
            },
            },
            },
            },
            },
            ],
            ]
```

tsxからVueとReact向けにビルドする

unjs/unbuild

React の場合は、esbuild にある jsx オプション (tscのオプションではない) に `automatic` を指定することでいい感じにビルドしてくれる。[auto-import-for-jsx | esbuild](#)

実際には `automatic` オプションを指定した場合、[`tsc --jsx react-jsx`](#) と同等の出力結果になる。

```
entries: [
  {
    // ... 省略
    rollup: {
      esbuild: {
        jsx: "automatic"
      },
    },
  },
],
]
```

しかし、まだこれだけでは動かない。

Vue固有の問題

unjs/unbuild

Vue向けビルドではjsxはh関数を使うように変換されるが、元コードにh関数をインポートするコードは含まれていないため、このままでは参照エラーになる。

```
// dist/Navigation.vue.mjs
const Navigation = ({ l, n }) => {
  // h は元のコードに含まれていない
  return h(
    "nav",
    { className: " ... " },
    h(
      "ul",
      { className: " ... " },
      l.map(** なんやかんや **),
      h("li", null, n())
    ),
  );
};

export { Navigation };
```

Vue固有の問題

unjs/unbuild

Q. jsxImportSource は？

A. esbuild が対応していない。

かつ、jsxImportSource は以下のようない export を要求しており、Vue は root から直接 h を import する形式なので使えない。

```
import { jsx, jsxs } from "xxx/jsx-runtime";
```

Vue はこう。

```
import { h } from "vue";
```

というわけで、自分でビルド結果に import 文を追加する。

Vue固有の問題

unjs/magicast

The screenshot shows the GitHub repository page for `unjs / magicast`. The repository is public and has 17 issues, 6 pull requests, 10 actions, 5 projects, and 2.3k stars. It features 9 branches and 20 tags. The main branch is selected. The repository was created by `antfu` and last updated 2 months ago. There are 159 commits in total. The repository description is: "Programmatically modify JavaScript and TypeScript source codes with a simplified, elegant and familiar syntax powered by recast and babel." The repository includes a README, MIT license, activity, custom properties, and 2.3k stars. It has 10 watchers, 39 forks, and 18 releases, with the latest being v0.3.5. Contributors include 21 individuals, and the repository uses JavaScript and TypeScript.

Issues 17 Pull requests 6 Actions Projects Security Insights

magicast Public Watch 10 Fork 39 Starred 2.3k

main 9 Branches 20 Tags Go to file Add file Code About

antfu chore: format ✓

.github/workflows chore(deps): update codecov/codecov-action action to v... 6 months ago

scripts chore: update deps 2 months ago

src chore: update deps 2 months ago

test chore: update deps 2 months ago

.editorconfig update project last year

.gitignore build: bundle recast (#B1) last year

.prettierrc update project last year

CHANGELOG.md chore(release): v0.3.5 2 months ago

LICENSE chore: add @antfu to the license last year

README.md chore: update deps 2 months ago

build.config.ts chore: update deps 2 months ago

eslint.config.js chore: update deps 2 months ago

helpers.d.ts chore: update tsconfig last year

package.json chore: format 2 months ago

pnpm-lock.yaml chore: update deps 2 months ago

pnpm-workspace.yaml chore: update deps 2 months ago

renovate.json update project last year

Programmatically modify JavaScript and TypeScript source codes with a simplified, elegant and familiar syntax powered by recast and babel.

Readme MIT license Activity Custom properties 2.3k stars 10 watching 39 forks Report repository

Releases 18 v0.3.5 (Latest) on Aug 27 + 17 releases

Contributors 21 + 7 contributors Languages

Vue固有の問題

unjs/magicast

ソースファイルを読み込んで AST にしたり、コードを文字列操作したりするためのライブラリ。

```
import { parseModule, loadFile, writeFile } from "magicast";
import { exit } from "node:process";
import path from "node:path";
import { fileURLToPath } from "node:url";

const MODULE_PATH = path.join(
  path.dirname(fileURLToPath(import.meta.url)),
  `./dist/Navigation.vue.mjs`,
);

const run = async () => {
  const mod = await loadFile(MODULE_PATH);
  // 元のコードの先頭に import 文を追加し、再度モジュールに変換する
  const result = parseModule(`import { h } from "vue";\n${mod.$code}`);
  return await writeFile(result, MODULE_PATH);
};

await run();
```

Vue固有の問題

ビルド結果を思い返すと、動かなさそうなやつがいる。

```
// dist/Navigation.vue.mjs
const Navigation = ({ l, n }) => {
  // h は元のコードに含まれていない
  return h(
    "nav",
    { className: " ... " }, // ←
    h(
      "ul",
      { className: " ... " }, // ←
      l.map(/* なんやかんや */),
      h("li", null, n())
    ),
  );
};

export { Navigation };
```

React と Vue では class/className の差分がある。元の定義は className にしているので、ここでも Vue 向けの変換を行う。

Vue固有の問題

className -> class への置換。

```
import { parseModule, loadFile, writeFile } from "magicast";
import { exit } from "node:process";
import path from "node:path";
import { fileURLToPath } from "node:url";

const MODULE_PATH = path.join(
  path.dirname(fileURLToPath(import.meta.url)),
  `./dist/Navigation.vue.mjs`,
);

const run = async () => {
  const mod = await loadFile(MODULE_PATH);
  const result = parseModule(mod.$code.replaceAll(" className:", " class:"));
  return await writeFile(result, MODULE_PATH);
};

await run();
```

ここまでやると動きます。

Vue固有の問題まとめ

- esbuild のオプションで tsconfig を拡張し、`jsx: "preserve"`、`jsxFactory: "h"` を設定する
- `h` が参照エラーにならないようにビルド結果に `import { h } from "vue";` を追加する
- 元コードが `className` を用いている場合は、`class` に変換する

tsxからVueとReact向けにビルドする

Q. slot はどうするの？

A. render prop で代用する

slot を React で同等の機能に変換するのはコストが高い。render prop ならランタイムを問わず動作するので、こちらに寄せる方が良い。

```
type Props = {
  notificationEl: () => JSX.Element;
};

const Navigation = ({ notificationEl }: Props) => {
  return (
    <nav className=" ... ">
      <ul>
        // ... 省略
        <li>{notificationEl()}</li> // ← これは React でも Vue でも有効
      </ul>
    </nav>
  );
};
```

tsxからVueとReact向けにビルドする

グローバルな JSX 名前空間は Vue3.4 から暗黙的に登録されなくなった。

コンポーネントを読み込むプロジェクト側で、別途 import する必要がある。

App.vue

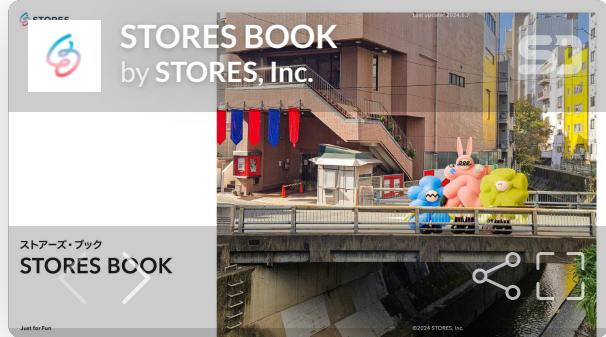
```
import "vue/jsx";

type Props = {
  notificationEl: () => JSX.Element; // 👈 これが有効になる
};

const Navigation = ({ notificationEl }: Props) => {
  return (
    <nav className=" ... ">
      <ul>
        // ... 省略
        <li>{notificationEl()}</li>
      </ul>
    </nav>
  );
};
```

おしまい

採用やってます👉



おまけ: 型を吐く

unbuild の rollup オプションは dts 出力するための拡張もできる。

そのままでは動かないため、React/Vue それぞれのビルト向けの tsconfig を作って渡す。esbuild に渡されるオプションとは別になるので注意（実態と違うが jsx: preserve にしないと通らない）。

tsconfig.react.json

```
{
  "compilerOptions": {
    /** 色々省略 */
    "jsx": "preserve",
    "jsxFactory": "createElement",
```

tsconfig.vue.json

```
{
  "compilerOptions": {
    /** 色々省略 */
    "jsx": "preserve",
    "jsxFactory": "h",
```

おまけ: 型を吐く

unbuild の設定に declaration: true を追加すると、型定義ファイルが出力される。

手元で試している感じでは、failOnWarn を false にしないとビルドが壊れるのって注意

unbuild.config.ts

```
import { defineBuildConfig } from "unbuild";

export default defineBuildConfig([
  // Build for React
  {
    entries: [
      // ...省略
    ],
    failOnWarn: false, // 🤦 これがないとビルドが壊れる
    declaration: true, // 🤦 React 向けの型定義ファイルが生成される
    rollup: {
      dts: {
        tsconfig: "tsconfig.react.json",
      },
      // ...省略
    },
  ],
])
```

本当におしまい