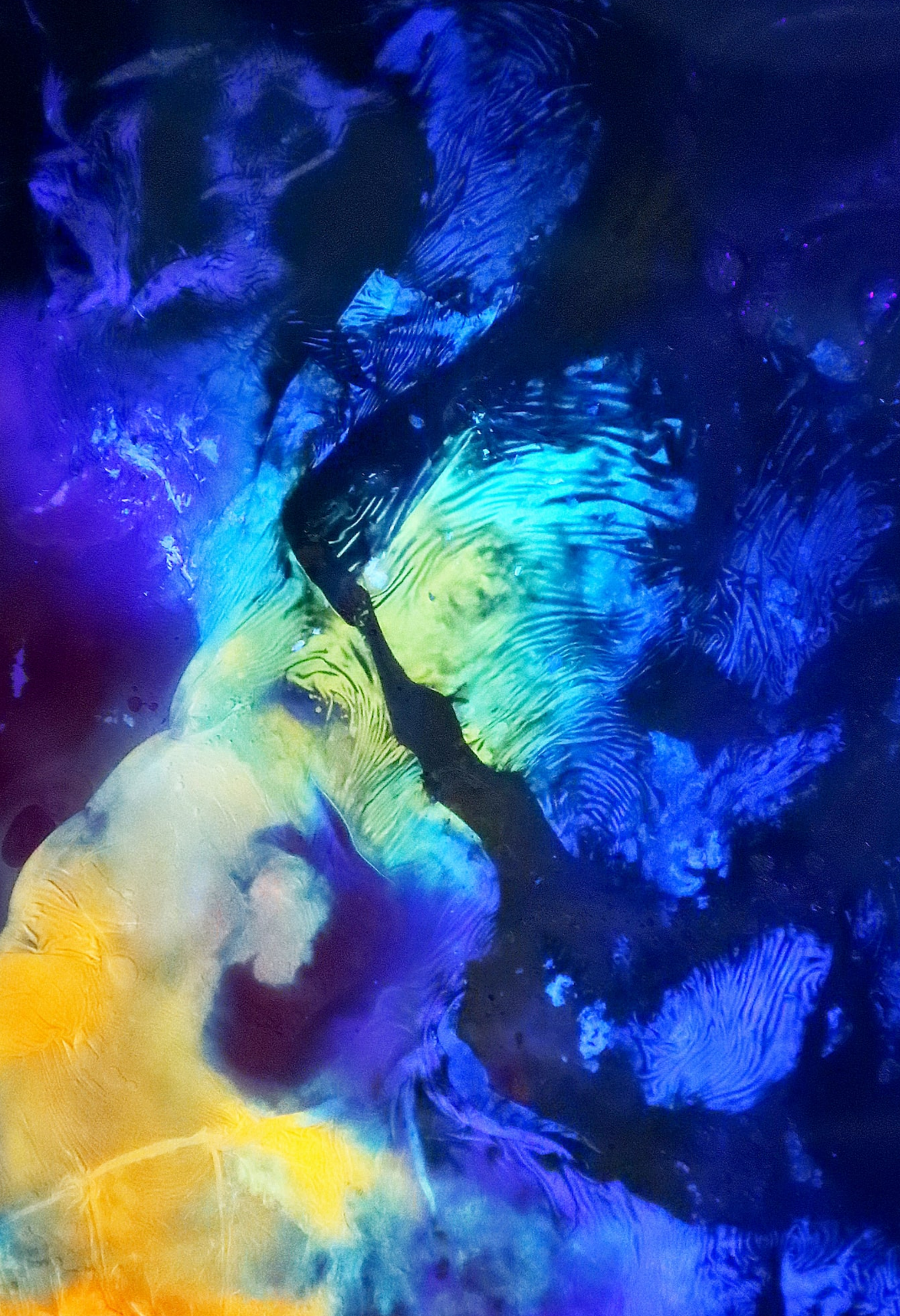




TypeScript 補足資料 React



React 環境構築

create-react-app



React開発環境を構築

```
npx create-react-app@5.0.1 react-  
typescript --template typescript
```

不要ファイルの削除

削除・・・App.test.tsx, logo.svg, index.css,
reportWebVitals.ts, setupTests.ts

App.tsx importロゴを削除

index.tsx webVitalを削除

App.css ファイル内を削除

ファイル・フォルダ構成

ファイルの拡張子 .tsx (.jsxではなく)

build ・ ・ npm run build実施しトランスパイル+バンドル

public ・ ・ HTMLファイルなど

src

App.css ・ ・ css

App.tsx ・ ・ コンポーネント

index.tsx ・ ・ エントリーポイント

react-app-env.d.ts ・ ・ 型定義ファイル

package.json(抜粋)

scripts

npm run start 簡易サーバー

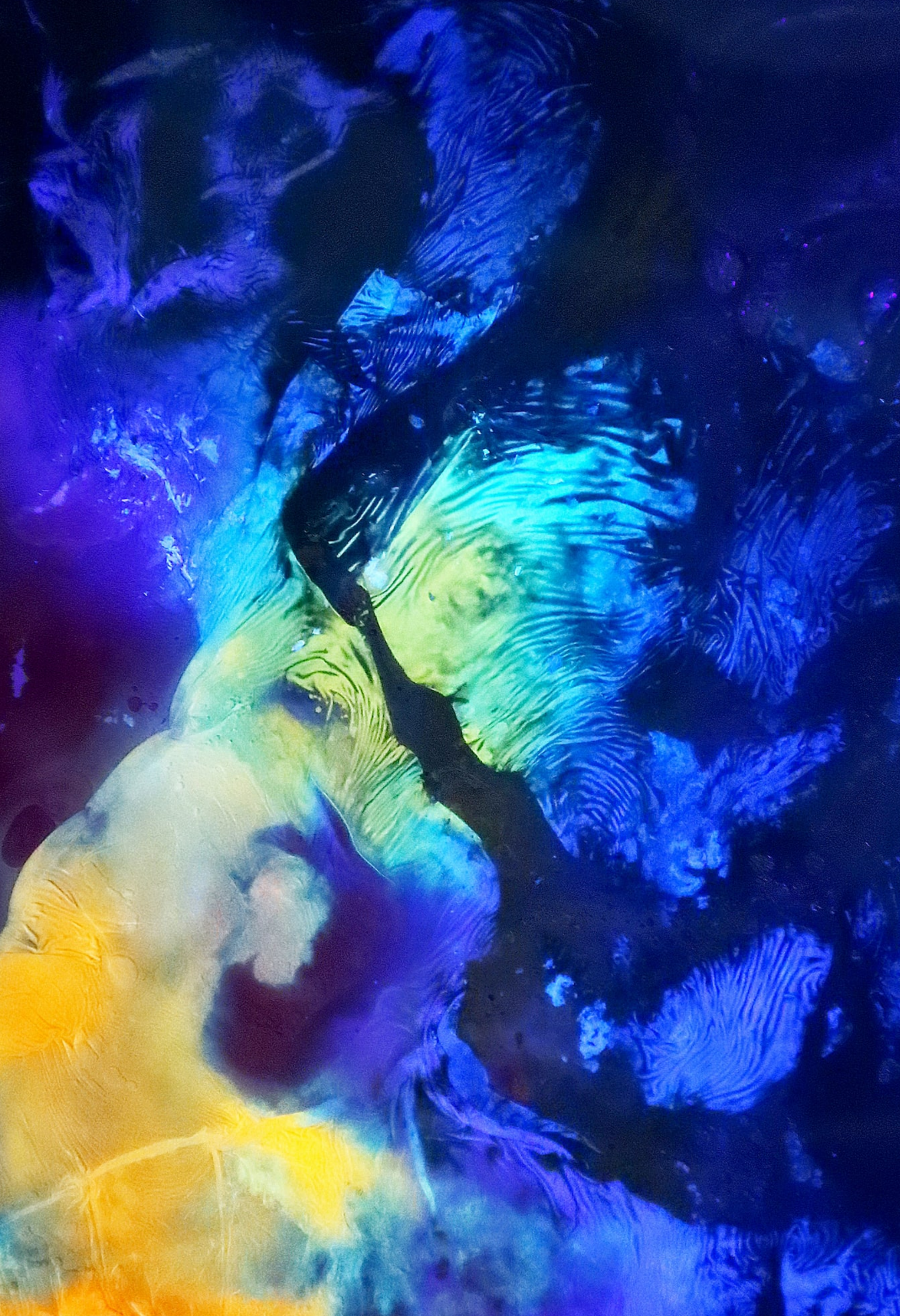
npm run build トランスパイル

npm run eject webpackなどの設定ファイルを外だし

devDependencies

"@types/react" 型定義ファイル

"@types/react-dom" 型定義ファイル



React with TypeScript

関数コンポーネントの型

- ・ クラスコンポーネント
 - ・ 関数コンポーネント ・ ・ React Hooksにより現在はこちらが主流
- 関数なので型が存在する (Function Component)

- ・ 型推論に任せる ・ ・ () => JSX.Element 型
- ・ 型を明示する ・ ・ React v18 以降

```
import React from 'react'
```

```
const App : React.FC = () => {}
```

分割代入を使うと

```
import { FC } from 'react'
```

```
const App : FC = () => {}
```


useState リアクティブな値

ReactHooksの一つ ジェネリクスで型定義

components/UseStateComponent.tsx

```
import { useState } from 'react'

const UseStateComponent = () => {
  const [ count, setCount ] = useState<number>(0)
  const handleClick = () => setCount( count + 1 )
  return (<><h2>useState</h2>
    <div>{ count }</div>
    <button onClick={handleClick}>+1 </button></>))
export default UseStateComponent
```


useState その2

typeで型作成しジェネリクスで設定する

components/UseStateComponent.tsx

```
import { useState } from 'react'
```

```
type Member = { id: number, name: string }
```

```
const UseStateComponent = () => {
```

```
  const [ member, setMember ] = useState<Member>({  
    id: 1, name: '三苦' })
```

略

```
  return (<><h2>useState</h2>
```

```
    <div>{ member.id} { member.name }</div>
```

略

イベント その1

クリック、inputフォームに入力 など

EventComponent.tsx

```
const EventComponent = () => {
```

```
  const handleChange = (e) => {
```

```
    console.log(e.target.value)
```

```
  }
```

```
  return (<
```

```
    <h2>Event</h2>
```

```
    <input type="text" onChange={handleChange} />
```

```
  </>)
```

```
}
```

```
const handleChange = (e) => {
```

```
  console.log(e.target.value)
```

```
}
```

```
return (<
```

```
  <h2>Event</h2>
```

```
  <input type="text" onChange={handleChange} />
```

(parameter) e: any

パラメーター 'e' の型は暗黙的に 'any' になります。 ts(7006)

[View Problem \(\F8\)](#) [Quick Fix... \(\%\)](#)

イベント その2

JSX側にコードを書いて

マウスカーソルを合わせると型推論が表示される

`<input type="text" onChange={(e)=>{}}>`

```
const handle (parameter) e:
  console.log React.ChangeEvent<HTMLInputElement>
}
return (<
<h2>Event</h2>
<input type="text" onChange={(e)=> {} } />
```

'e' が宣言されていますが、その値が読み取られることはありません。 ts(6133)

Quick Fix... (⌘.)

イベントその3

EventComponent.tsx

```
import React from 'react'
```

```
const EventComponent = () => {
```

```
  const handleChange = (e:
```

```
React.ChangeEvent<HTMLInputElement>) => {
```

```
    console.log(e.target.value)
```

```
  }
```

略

よく使うイベント関連の型

React.ChangeEvent<HTMLInputElement>

React.FormEvent<HTMLInputElement>

React.MouseEvent<HTMLButtonElement>

React.KeyboardEvent<HTMLParagraphElement>

などなど

[node_modules/@types/react/global.d.ts](#) も参照

Props 親->子 に渡す

propsはオブジェクト typeで設定するとわかりやすい

親 App.tsx

```
<PropsComponent name="浅野" />
```

子 PropsComponent.tsx

```
type Props = { name: string }
```

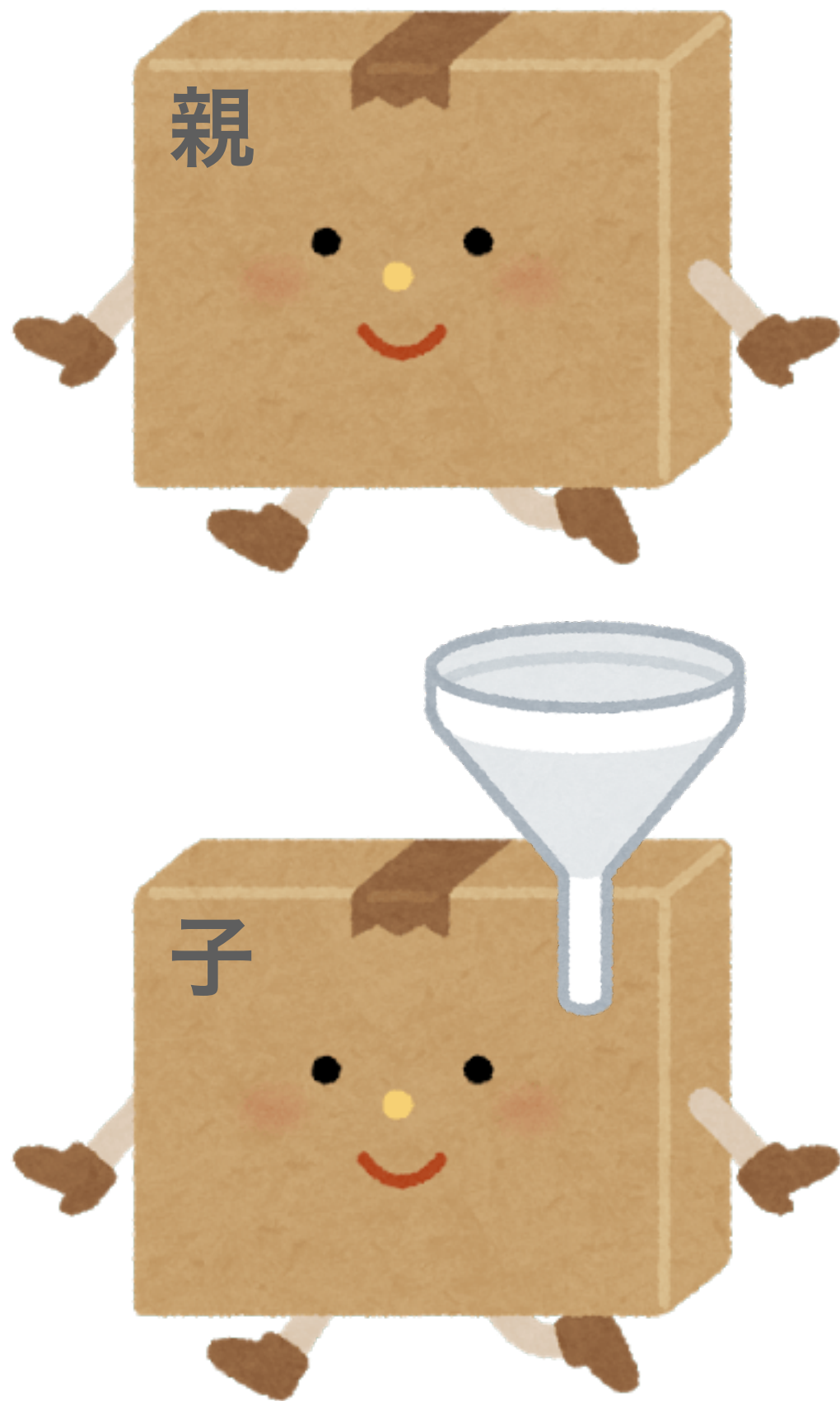
```
const PropsComponent = (props: Props) => {
```

```
  return (<
```

```
    <h2>Props</h2>
```

```
    { props.name }</>))
```

```
export default PropsComponent
```



Props 分割代入で受けとる

分割代入 {} で受け取る場合

親 App.tsx

```
<PropsComponent name="浅野" />
```

子 PropsComponent.tsx

```
type Props = { name: string }
```

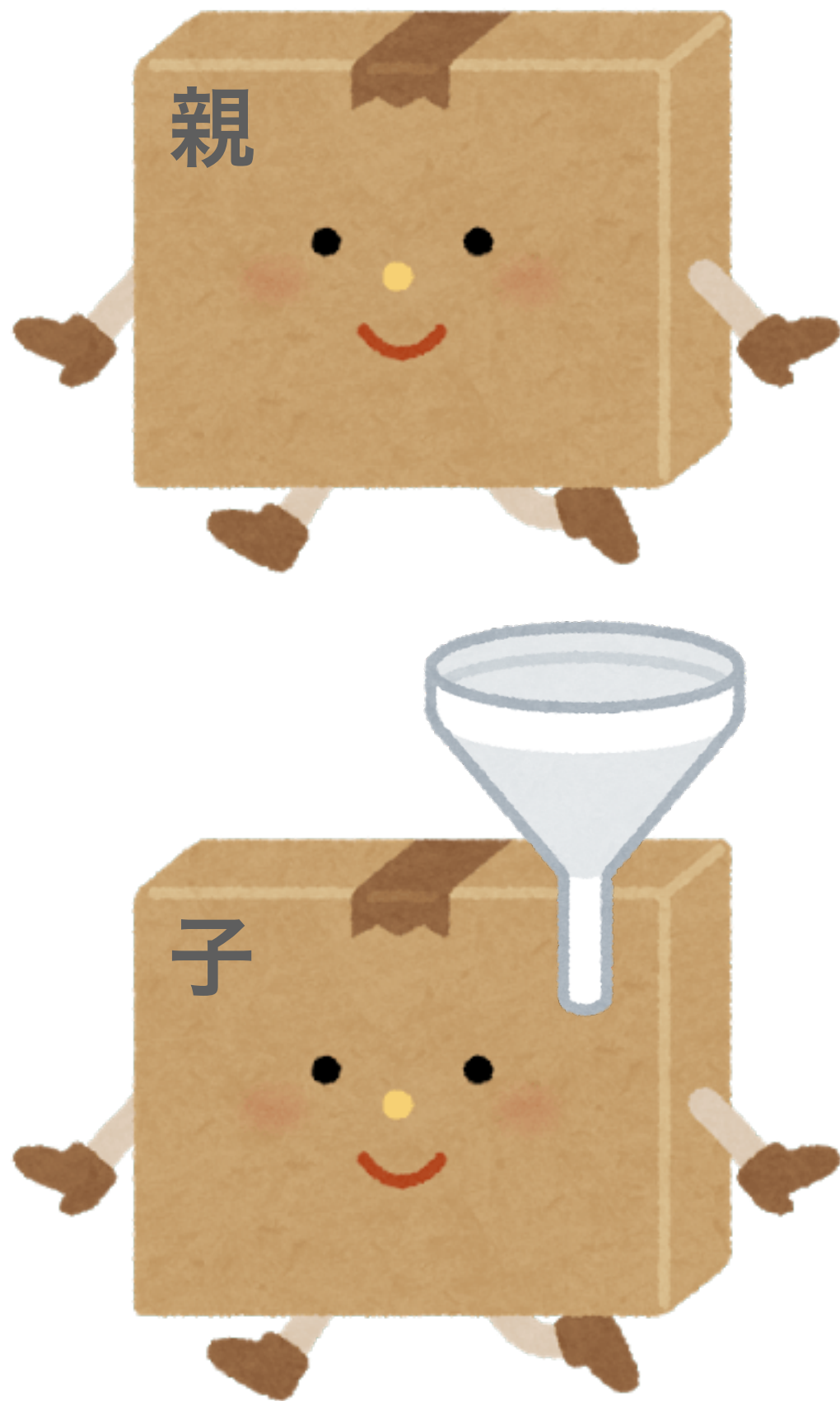
```
const PropsComponent = ({ name } : Props) => {
```

```
  return (<
```

```
    <h2>Props</h2>
```

```
    { name }</>))
```

```
export default PropsComponent
```



Props 関数型のパターン

関数の型つきで設定する方法もある

親 App.tsx

```
<FCPropsComponent name="伊東" />
```

子 FCPropsComponent.tsx

```
import { FC } from 'react'
```

```
type Props = { name: string }
```

```
const FCPropsComponent : FC<Props> = ({ name }) => {
```

```
  return (<
```

```
    <h2>FC_Props</h2>
```

```
    { name }</>))
```

```
export default FCPropsComponent
```

