React （v19）

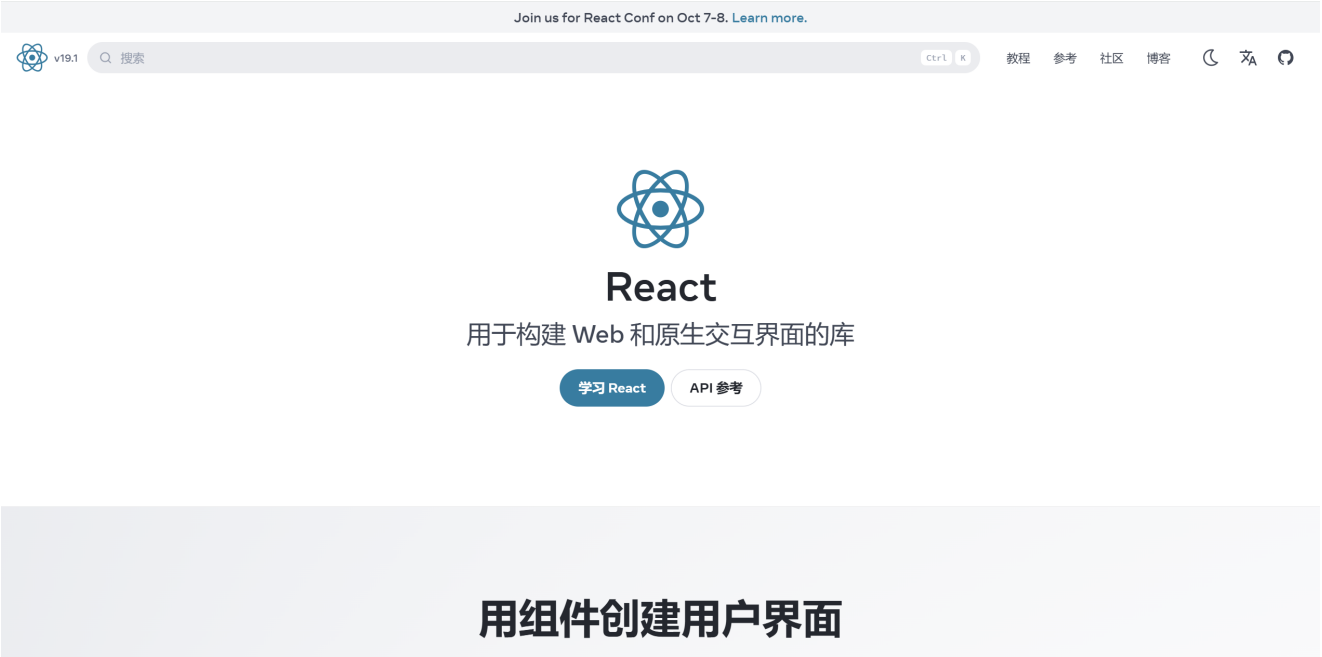
作者： -- 天气预报

日期： -- 2025-08-25

01. 官网

> https://zh-hans.react.dev/

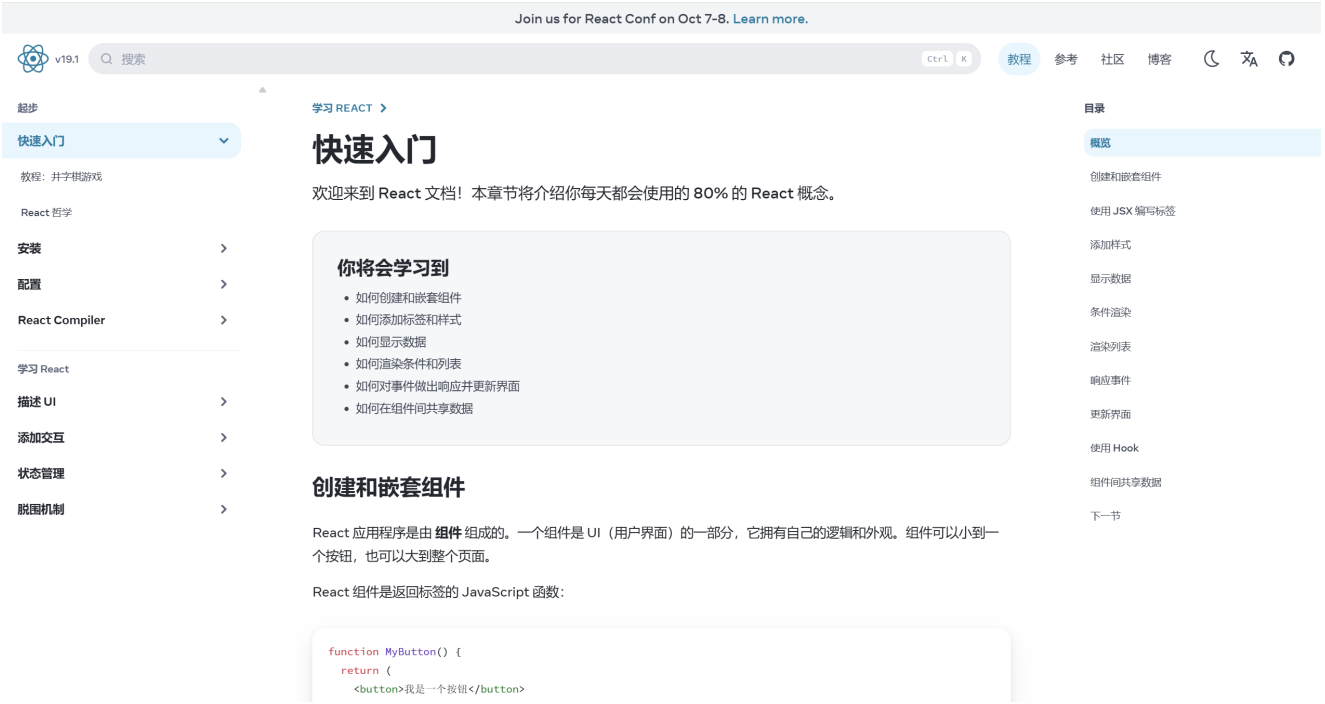
- UI



1.1 入门

> https://zh-hans.react.dev/learn

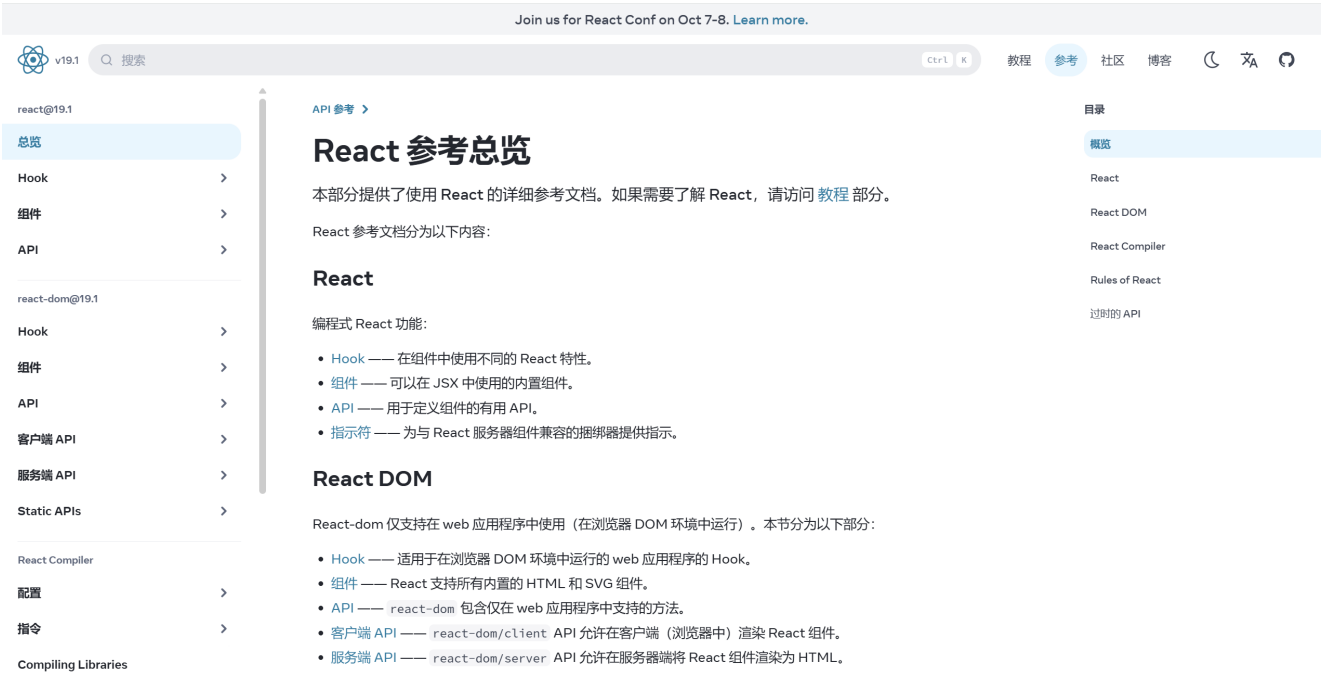
- UI



1.2 参考手册

> https://zh-hans.react.dev/reference/react

- UI



1.3 环境

> react & react-dom | vite | @vitejs/plugin-react

1.3.1 手动安装

> 环境依赖 | vite 配置文件 | main.jsx | App.jsx | index.html 单文件模版

- npm

```
npm init --yes

npm install react react-dom

npm install vite @vitejs/plugin-react -D
```

- **package.json**

```
{
    "name": "react",
    "version": "1.0.0",

    "scripts": {
        "dev": "vite",
        "build": "vite build"
    },

    "keywords": [],
    "author": "",
    "license": "ISC",
    "description": "",

    "dependencies": {
        "react": "^19.1.1",
        "react-dom": "^19.1.1"
    },

    "devDependencies": {
        "@vitejs/plugin-react": "^5.0.1",
        "vite": "^7.1.3"
    },

    "type": "module"
}
```

- **vite.config.js**

```
import { defineConfig } from "vite";
import react from "@vitejs/plugin-react";

export default defineConfig(
    {
        plugins: [
            react(),
        ]
    }
);
```

- **src/App.jsx**

```jsx
export function App() {
    return (
        <div>
            <h1>react</h1>
        </div>
    )
}
```

- **src/main.jsx**

```jsx
import { App } from "./App";

import { createRoot } from "react-dom/client";

const root = document.getElementById("root");

createRoot(root).render(<App/>);
```

- **index.html**

```html
<!doctype html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <title>React</title>
    </head>
    <body>
        <div id="root"></div>
        <script type="module" src="./src/main.jsx"></script>
    </body>
</html>
```

- **http://localhost:5173/**

```
npm run dev
```

1.3.2 模版

```
=== npm create vite@latest
```

- **console**

```
>> Project name: »                    >>> 输入项目名称

>> Vanilla
>> Vue
>> React                              >>> 选择此项
>> Preact
>> Lit
>> Svelte
>> Solid
>> Qwik
>> Others


>>>> TypeScript
>>>> TypeScript + SWC
>>>> JavaScript
>>>> JavaScript + SWC                  >>> 选择此项
>>>> Remix
```

- **npm**

```
>> cd project-name
    >> npm install
    >> npm run dev
```

02. 样式

> 推荐 行内 结合 Tailwind 方案

2.1 行内

> <element style={ stylesObject }>

- **src/App.jsx**

```
export function App() {
    return (
        <div style={ container }>
            <h1 style={ title }>react</h1>
        </div>
    )
}

const container = {
    padding: "64px",
    backgroundColor: "oklch(64.5% 0.246 16.439)",
}

const title = {
    fontSize: "64px",
```

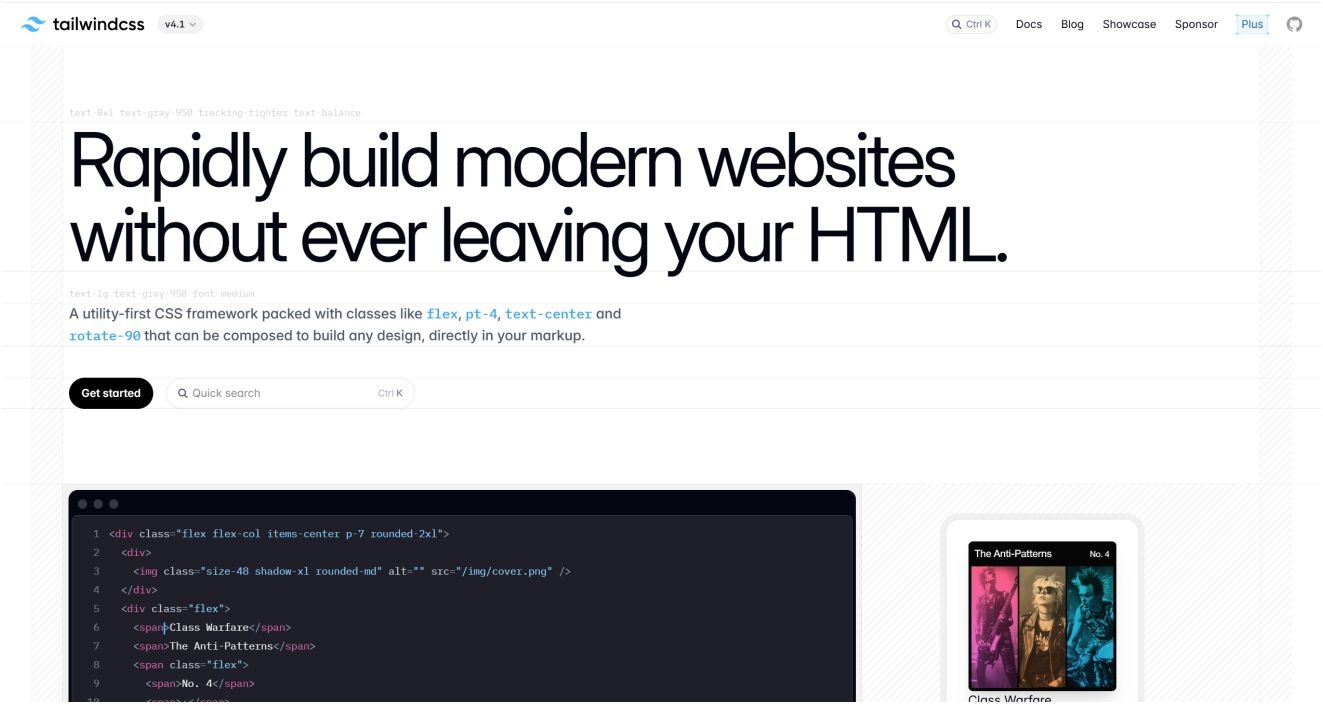
```
    textAlign: "center",
    color: "white",
}
```

2.2 Tailwind

> https://tailwindcss.com/

2.2.1 官网

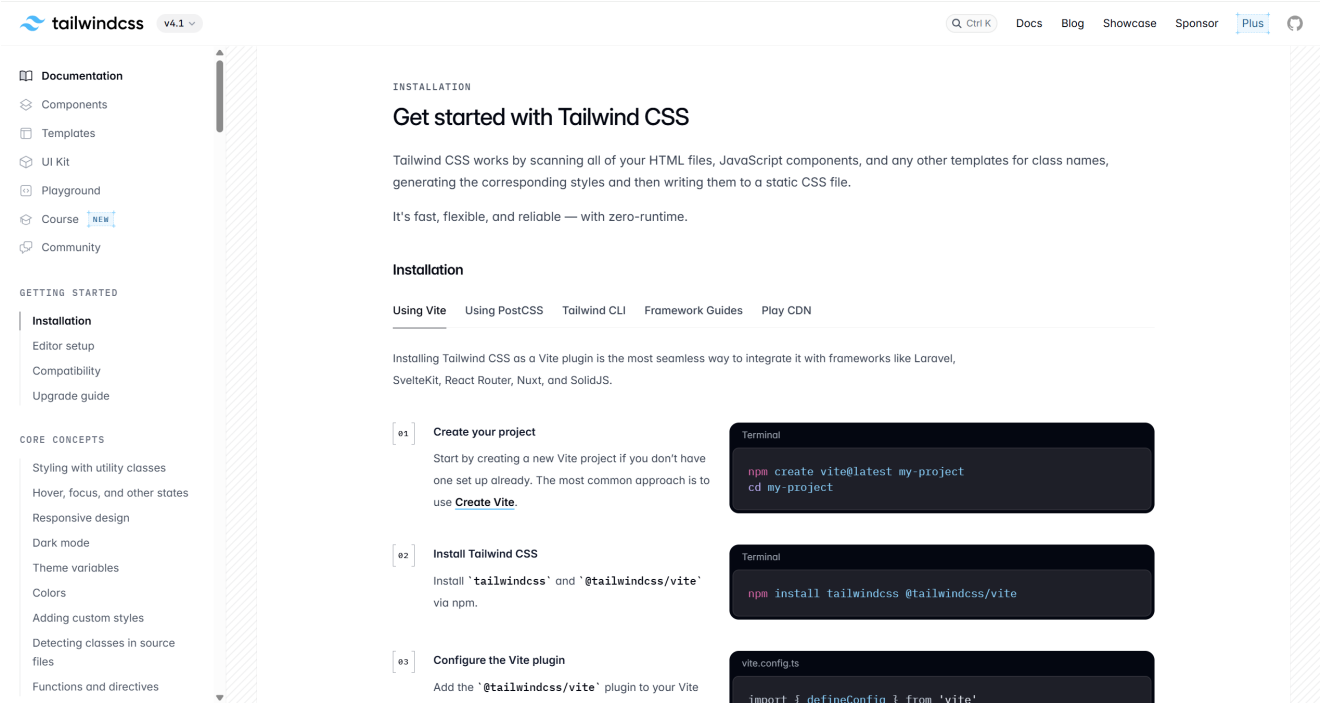
- UI



2.2.2 手册

> https://tailwindcss.com/docs/installation/using-vite

- UI



2.2.3 环境

> npm install tailwindcss @tailwindcss/vite -D

- package.json

```
{
    "name": "react",
    "version": "1.0.0",
    "scripts": {
        "dev": "vite",
        "build": "vite build"
    },
    "keywords": [],
```

```
    "author": "",
    "license": "ISC",
    "description": "",
    "dependencies": {
        "react": "^19.1.1",
        "react-dom": "^19.1.1"
    },
    "devDependencies": {
        "@tailwindcss/vite": "^4.1.12",
        "@vitejs/plugin-react": "^5.0.1",
        "tailwindcss": "^4.1.12",
        "vite": "^7.1.3"
    },
    "type": "module"
}
```

- **vite.config.js**

```
import { defineConfig } from "vite";

import react from "@vitejs/plugin-react";
import tailwindcss from "@tailwindcss/vite";

export default defineConfig(
    {
        plugins: [
            react(), tailwindcss(),
        ]
    }
);
```

- **src/style.css**

```
@import "tailwindcss";
```

- **src/main.jsx**

```
import "./style.css";

import { App } from "./App";

import { createRoot } from "react-dom/client";

const root = document.getElementById("root");

createRoot(root).render(<App/>);
```

- **src/App.jsx**

```
export function App() {
    return (
        <div className="p-32 bg-rose-500">
            <h1 className="text-6xl md:text-7xl lg:text-9xl duration-300">
                <span className="font-mono text-white">React</span>
            </h1>
        </div>
    )
}
```

2.2.4 动态类样式

> npm install clsx tailwind-merge

- **package.json**

```
{
    "name": "react",
    "version": "1.0.0",
    "scripts": {
        "dev": "vite",
        "build": "vite build"
    },
```

```
    "keywords": [],
    "author": "",
    "license": "ISC",
    "description": "",
    "dependencies": {
        "clsx": "^2.1.1",
        "react": "^19.1.1",
        "react-dom": "^19.1.1",
        "tailwind-merge": "^3.3.1"
    },
    "devDependencies": {
        "@tailwindcss/vite": "^4.1.12",
        "@vitejs/plugin-react": "^5.0.1",
        "tailwindcss": "^4.1.12",
        "vite": "^7.1.3"
    },
    "type": "module"
}
```

- **src/tailwind/index.js**

```
import { clsx } from "clsx";
import { twMerge } from "tailwind-merge";

// 部分 UI 库带有该函数
export function cn(...classes) {
    return twMerge(clsx(...classes));
}
```

- **clsx-for-tailwind**

```
npm install clsx-for-tailwind

import { cn } from "clsx-for-tailwind";
```

- **src/App.jsx**

```
import { cn } from "./lib/tailwind";

export function App() {
    return (
        <div className={ cn("p-32 bg-rose-500", 2 > 1 && "p-48 font-bold") }>
            <h1 className="text-6xl md:text-7xl lg:text-9xl duration-300">
                <span className="font-mono text-white">React</span>
            </h1>
        </div>
    )
}
```

典型

- **src/App.jsx**

```
export function App() {
    return (
        <div className="bg-slate-200 p-10">
            <div className="flex h-80 flex-col md:flex-row gap-10">

                <div className="w-full min-h-full bg-linear-to-tr from-cyan-500
to-purple-500">
                    <div className="size-full flex justify-center items-center">
                        <h1 className="font-mono text-6xl text-white">React</h1>
                    </div>
                </div>

                <div className="w-full min-h-full" style={ bgLines }>
                    <div className="size-full flex justify-center items-center">
                        <h1 className="font-mono text-6xl">Tailwind</h1>
                    </div>
                </div>

            </div>
        </div>
    )
}

const bgLines = {
    backgroundImage: "linear-gradient(#000 2px, transparent 2px)",
    backgroundSize: "20px 20px",
    backgroundPosition: "center center",
```

```
}
```

03. Jsx

> 特殊的 JavaScript & HTML 混写语法形式、以组织 React 组件逻辑

3.1 基本行为

> JSX 文件以 .jsx 名称作为文件后缀 有且仅有一个根元素 组件名首字母须大写

- **src/App.jsx**

```
export function App() {
    return (
        <div className="p-32 bg-rose-500">
            <h1 className="text-8xl text-white">jsx</h1>
        </div>
    )
}
```

- **src/App.jsx**

```jsx
export function App() {
    return (
        <>
            <h1 className="text-9xl bg-rose-500 p-10 text-white">react</h1>
            <h1 className="text-9xl bg-rose-500 p-10 text-white">jsx</h1>
        </>
    )
}
```

3.2 变量绑定

> { 表达式 } 方式绑定变量 仅在 return (<element> </element>) 范围生效

☑ 表达式须有明确的返回值: number | string | boolean | array 等

- **src/App.jsx**

```jsx
export function App() {

    const username = "天气预报";

    function fn() {
        return "Hello World";
    }

    const user = {
        username: "天气预报",
        gender: "male",
    }

    const jsx = (
        <div>
            <i>element</i>
        </div>
    );

    const innerHTML = "<i>element</i>"

    return (
```

```
        <div className="p-20 bg-rose-500 space-y-10 text-white text-4xl">
            <p>{ username }</p>
            <p>{ fn() }</p>
            <p>{ 665 + 1 }</p>

            {/* 不可直接绑定布尔变量 3 > 2 */ }
            <p>{ 3 > 2 ? "Yes" : "No" }</p>

            {/* 不可直接绑定对象变量 { user } */ }
            <p>{ user.username } : { user.gender }</p>

            <div>{ jsx }</div>

            <p>{ innerHTML }</p>

            <div dangerouslySetInnerHTML={ { __html: innerHTML } }/>

            <p>{ [ 1, 2, 3, 4, 5, 6, 7, 8, 9, ] }</p>
        </div>
    )
}
```

3.3 条件渲染

> React 组件允许条件逻辑组织返回 JSX | 或者表达式获取 JSX

- **src/App.jsx**

```
export function App() {

    const boolean = false;

    if (boolean) {
        return (
            <div>
                <h1>react yes</h1>
            </div>
        )
    }

    return (
        <div>
            <h1>react no</h1>
        </div>
```

```
    )

}
```

- **src/App.jsx**

```
export function App() {

    const boolean = true;

    return (
        <>
            {
                boolean && (
                    <div>
                        <h1>react yes</h1>
                    </div>
                )
            }
        </>
    )
}
```

- **src/App.jsx**

```
export function App() {

    const boolean = true;

    return boolean && (
        <div>
            <h1>react yes</h1>
        </div>
    )
}
```

- **src/App.jsx**

```
export function App() {

    const boolean = true;

    return boolean ? (<h1>react yes</h1>) : (<h1>react no</h1>)
}
```

3.4 列表渲染

> 列表渲染逻辑为 将数组中的每一个元素 映射为 对应的 JSX 变量

- [x] 需为每一项 JSX 变量标记 key 属性 以使 React 内部区分每一项
- [x] key 属性通常使用数据中的 唯一标识 作为标识值 仅当无唯一标识时使用 遍历 index
- [x] 遍历项因 UI 操作使得顺序、数量产生变化时 不得使用 index 作为 key

- **src/index.jsx**

```
export function App() {

    const arr = [ 1, 2, 3, 4, 5, 6, 7, ];

    return (
        <ul className="flex gap-10 p-10">
            {
                arr.map((element, index) => (
                    <li className="w-32 aspect-video bg-rose-500 hover:bg-cyan-
400 p-2"
                        key={ index }>
                        { element }
                    </li>
                ))
            }
        </ul>
    )
}
```

- **src/index.jsx**

```jsx
export function App() {

    const userList = [
        {
            id: 1,
            username: "John",
        },
        {
            id: 2,
            username: "Jackson",
        },
        {
            id: 3,
            username: "Mary",
        },
    ];

    return (
        <ul className="flex gap-10 p-10">
            {
                userList.map(({ id, username }) => (
                    <li className="w-32 aspect-video bg-rose-500 hover:bg-cyan-
400 p-2"
                        key={ id }>
                        { username }
                    </li>
                ))
            }
        </ul>
    )
}
```

- **src/App.jsx**

```jsx
import { v4 as uuid } from "uuid";

export function App() {

    const userList = [
        {
            username: "John",
            gender: "male",
        },
        {
            username: "Jackson",
```

```
            gender: "male",
        },
        {
            username: "Mary",
            gender: "female",
        },
    ];


    // 将原始数据处理为包含 ID 的数据、后续使用新数据
    const users = userList.map((user) => ({ ...user, id: uuid() }));

    return (
        <ul className="flex gap-10 p-10">
            {
                users.map(({ id, username, gender }) => (
                    <li className="w-32 aspect-video bg-rose-500 hover:bg-cyan-
400 p-2"
                        key={ id }>
                        { username } : { gender }
                    </li>
                ))
            }
        </ul>
    )
}
```

=== key 值变化将会使得绑定区域重新渲染

- **src/App.jsx**

```
export function App() {

    return (
        <div className="bg-slate-100 p-32" key={ 1 change to 2 ... }>
            <input className="block w-1/2 p-5 border-2 border-emerald-600"
                   type="text"/>
        </div>
    )

}
```

3.5 事件绑定

> onClick | onChange | on + 事件名大写 ...

☑ 事件回调函数要么无参 要么有且仅有一个 event 事件对象作为参数

- src/App.jsx

```
export function App() {

    return (
        <div className="bg-slate-300 p-32">
            <button className="bg-black px-10 py-4 text-white text-xl cursor-
pointer"
                    onClick={ () => console.log(Math.random()) }>
                click
            </button>
        </div>
    )
}
```

- src/App.jsx

```
export function App() {

    return (
        <div className="bg-slate-300 p-32">
            <button className="bg-black px-10 py-4 text-white text-xl cursor-
pointer"
                    onClick={ (event) => console.log(event.target) }>
                click
            </button>
        </div>
    )

}
```

> 多参数时 将逻辑处理函数 放入事件回调函数中调用即可

- **src/App.jsx**

```
export function App() {

    function clickCallback(event) {
        clickLogic(event, 100);
    }

    function clickLogic(event, other) {
        console.log(event.target, other);
    }

    return (
        <div className="bg-slate-300 p-32">
            <button className="bg-black px-10 py-4 text-white text-xl cursor-
pointer"
                    onClick={ clickCallback }>
                click
            </button>
        </div>
    )

}
```

3.6 嵌套组件

> <Nav/> | <Banner> ... </Banner> 将组件作为 HTML 元素使用即可

- **src/components/Nav.jsx**

```
export function Nav() {
    return (
        <nav className="bg-rose-500">
            <section className="max-w-7xl mx-auto h-20 flex items-center">
                <ul className="flex gap-5">
```

```
                    <li><a className="px-5 py-3 text-white text-xl font-mono"
href="">react</a></li>
                    <li><a className="px-5 py-3 text-white text-xl font-mono"
href="">next</a></li>
                    <li><a className="px-5 py-3 text-white text-xl font-mono"
href="">tailwind</a></li>
                    <li><a className="px-5 py-3 text-white text-xl font-mono"
href="">motion</a></li>
                    <li><a className="px-5 py-3 text-white text-xl font-mono"
href="">zustand</a></li>
                </ul>
            </section>
        </nav>
    )
}
```

- **src/components/Banner.jsx**

```
export function Banner() {
    return (
        <div className="bg-slate-300 py-48">
            <h1 className="text-center text-9xl font-mono">Example</h1>
        </div>
    )
}
```

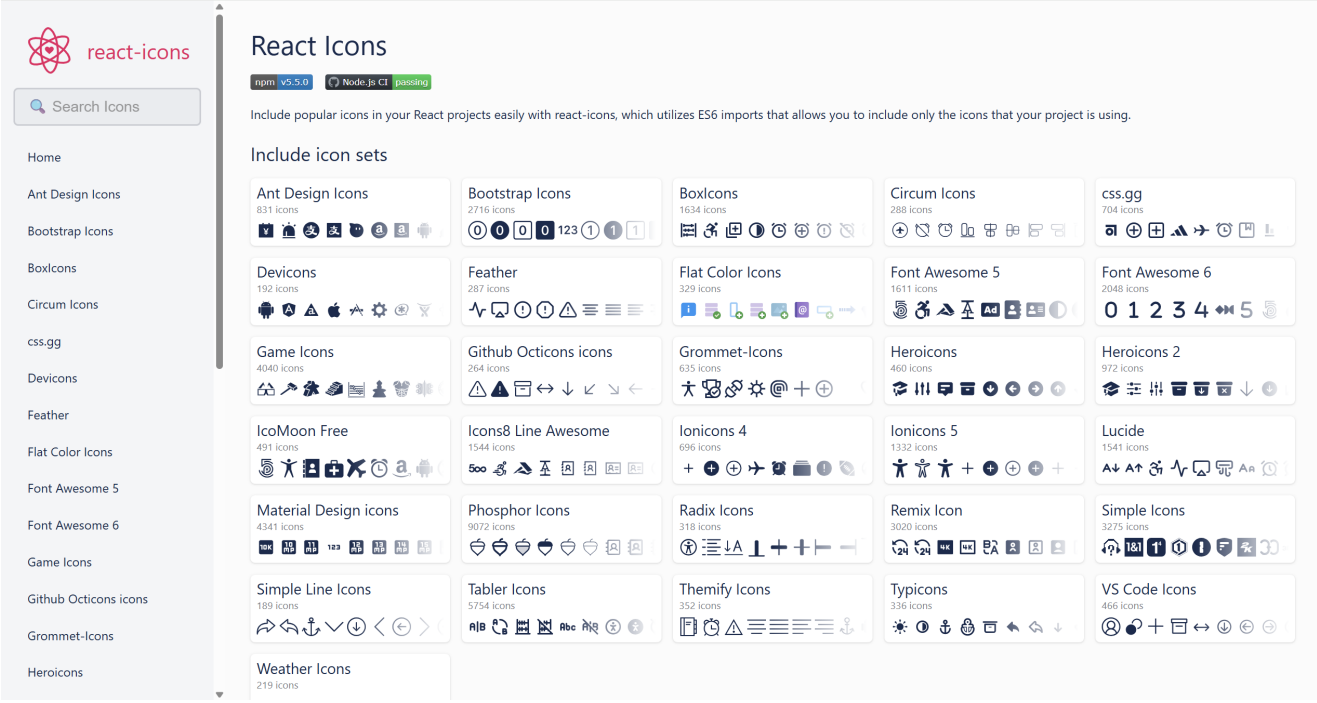
- **src/App.jsx**

```jsx
import { Nav } from "./components/Nav.jsx";
import { Banner } from "./components/Banner.jsx";

export function App() {

    return (
        <div>
            <Nav/>
            <Banner/>
        </div>
    )
}
```

3.7 图标

> https://react-icons.github.io/react-icons/

- UI



> npm install react-icons

- **src/App.jsx**

```
import {
    FaApple,
    FaTwitter,
    FaYahoo,
    FaGoogle,
    FaMicrosoft,
    FaFacebook
} from "react-icons/fa";

export function App() {

    return (
        <div className="p-32 flex gap-10 bg-black">
            <FaApple     size={ 48 } color={ "white" } />
            <FaTwitter   size={ 48 } color={ "white" } />
            <FaYahoo     size={ 48 } color={ "white" } />
            <FaGoogle    size={ 48 } color={ "white" } />
            <FaMicrosoft size={ 48 } color={ "white" } />
            <FaFacebook  size={ 48 } color={ "white" } />
        </div>
    )
}
```

04. Props

> 组件函数参数被称为 PROP [React 赋予概念]

4.1 声明方式

- **src/components/Link.jsx**

```
import { cn } from "../lib/tailwind";

export function Link(props) {
    return (
        <a className={ cn("block w-fit px-10 py-3 bg-slate-100 text-xl",
props.className) }
           href={ props.href }>
            { props.text }
        </a>
    )
}
```

- **src/components/Link.jsx [解构形式]**

```
import { cn } from "../lib/tailwind";

export function Link({ className, href, text }) {
    return (
        <a className={ cn("block w-fit px-10 py-3 bg-slate-100 text-xl",
className) }
           href={ href }>
            { text }
        </a>
    )
}
```

- **src/App.jsx**

```
import { Link } from "./components/Link.jsx";

export function App() {

    return (
        <div className="p-10">
            <Link className="bg-rose-500 text-white"
                  href="https://www.example-plus.work/"
                  text="example"/>
        </div>
    )
}
```

4.2 key

> 组件总是具备 key 属性、一般用于触发重新渲染、遍历项标识

- **src/components/Link.jsx**

```
import { Link } from "./components/Lnik.jsx";

export function App() {

    return (
        <div className="p-10">
            <Link className="bg-rose-500 text-white"
                  href={ "https://www.example-plus.work/" }
                  text={ "example" }
                  key="unique-key-name"/>
        </div>
    )
}
```

4.3 children

> 组件总是具备 children 属性、表示 <Component> 此处内容 </Component>

- src/components/Link.jsx

```
import { cn } from "../lib/tailwind";

export function Link({ className, href, children }) {
    return (
        <a className={ cn("block w-fit p-5 bg-slate-100 text-xl", className) }
           href={ href }>
            { children }
        </a>
    )
}
```

- src/App.jsx

```
import { Link } from "./components/Link.jsx";

export function App() {

    return (
        <div className="p-10">
            <Link className="bg-black text-white"
                  href="https://www.example-plus.work/"
                  key="unique-key-name">

                <span className="text-purple-600 font-mono">example</span>

            </Link>
        </div>
    )
}
```

4.4 数据类型

> string | number ... object | ref | function ... JSX

- src/components/Component.jsx

```
export function Component({ number, string, user, fn, jsx }) {
    return (
        <div className="p-10 bg-rose-500 space-y-10">
            <p>{ number }</p>
            <p>{ string }</p>
            <p>{ user.username } : { user.age }</p>
            <p onClick={ fn }>click</p>
            <p>{ jsx }</p>
        </div>
    )
}
```

- src/App.jsx

```
import { Component } from "./components/Component.jsx";

export function App() {

    return (
        <div className="p-10 bg-slate-200">
            <Component number={ 100 }
                       string="hello"
                       user={ { username: "John", age: 18 } }
                       fn={ () => console.log(Math.random()) }
                       jsx={ (<i>element</i>) }/>
        </div>
    )
}
```

05. Hooks

> REACT 组件渲染过程中预留给开发者的 钩子函数 被赋予特定执行功能

☑ HOOK 是内置预留约定函数、必须在组件顶层作用域调用、且推荐调用置于组件开头

5.01 useRef

> 相对组件而言的全局变量 | DOM 元素引用句柄

5.1.1 存储数据

> const ref = useRef(value);

- **src/App.jsx**

```jsx
import { useRef } from "react"

export function App() {

    const ref = useRef(100);

    function handle() {
        ref.current += 1;
        console.log(ref.current);
    }

    return (
        <div className="p-32 bg-rose-500">
            ref.current : { ref.current }
            <button className="p-5 bg-blue-500" onClick={ handle }>+1</button>
        </div>
    )
```

```
}
```

5.1.2 引用元素

> const ref = useRef(null); | <element ref={ ref }>

- **src/App.jsx**

```jsx
import { useRef } from "react"

export function App() {

    const ref = useRef(null);

    return (
        <div className="p-32 flex">
            <input className="block p-5 border-2"
                   type="text"
                   ref={ ref }/>
            <button className="p-5 bg-blue-500 text-white"
                    onClick={ () => ref.current.focus() }>
                focus
            </button>
        </div>
    )
}
```

5.02 useId

> 组件内的快捷 短ID、不同组件内调用、ID一定不同、通常用作拼接 ID 前缀

- **src/App.jsx**

```
import { useId } from "react"

export function App() {

    const id = useId();

    const usernameId = id + "username";
    const passwordId = id + "password";

    return (
        <div className="p-32">
            <form className="space-x-5" action="">
                <input className="border-2" id={ usernameId } name="username"
type="text"/>
                <input className="border-2" id={ passwordId } name="password"
type="password"/>
            </form>
            <p>{ id } : { usernameId } : { passwordId }</p>
        </div>
    )
}
```

5.03 useState

> 相对组件而言的全局状态变量、值变更将触发组件再次渲染

5.3.1 重新渲染

> 状态变量一旦发生变化、将使得组件内部所有代码再次执行一遍、UI 同时更新

- **src/App.jsx**

```
import { useState } from "react"

export function App() {
```

```
    const [ number, setNumber ] = useState(100);

    console.log(Math.random());

    return (
        <div className="p-32 bg-slate-200">
            <p className="text-6xl">{ number }</p>
            <p>
                <button className="p-5 bg-amber-500"
                        onClick={ () => setNumber(number + 1) }>+1
                </button>
            </p>
        </div>
    )
}
```

5.3.2 重新渲染影响

> 重新渲染不会影响父组件、不会影响 children 子组件、但会影响嵌套子组件

- **src/App.jsx [不会影响 children 子组件]**

```
import { Child, Current } from "./components/Current.jsx";

export function App() {

    return (
        <div >
            <Current>
                <Child/>
            </Current>
        </div>
    )
}
```

- **src/components/Current.jsx**

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

```
export function Child() {
    console.log("Child : " + Math.random());

    return (
        <div className="p-10 bg-slate-300">
            <h1>Child</h1>
        </div>
    )
}

export function Current({ children }) {

    const [ number, setNumber ] = useState(1);

    console.log("Current : " + Math.random());

    return (
        <div className="p-10 bg-slate-200">
            <h1>Current : { number }</h1>
            <button className="p-5 bg-amber-500"
                    onClick={ () => setNumber(prev => prev + 1) }>+1
            </button>
            <div>{ children }</div>
        </div>
    )
}
```

- **src/components/Current.jsx**

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

export function Child() {
    console.log("Child : " + Math.random());

    return (
        <div className="p-10 bg-slate-300">
            <h1>Child</h1>
        </div>
    )
}

export function Current() {

    const [ number, setNumber ] = useState(1);

    console.log("Current : " + Math.random());

    return (
        <div className="p-10 bg-slate-200">
```

```
            <h1>Current : { number }</h1>
            <button className="p-5 bg-amber-500"
                    onClick={ () => setNumber(prev => prev + 1) }>+1
            </button>
            <Child/>
        </div>
    )
}
```

- **src/App.jsx [会影响嵌套子组件]**

```
import {  Current } from "./components/Current.jsx";

export function App() {

    return (
        <div>
            <Current/>
        </div>
    )
}
```

5.3.3 正确更新状态变量

> setState(value) 处于复杂嵌套函数或异步操作时、调用需注意以下区别

✅ 不依赖旧 state ⇒ setState(newState)
✅ 需依赖旧 state ⇒ setState(prev ⇒ 返回 prev 运算结果)

- **src/App.jsx**

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

```
export function App() {

    const [ number, setNumber ] = useState(100);

    console.log(Math.random());

    return (
        <div className="p-32 bg-slate-200">
            <p className="text-6xl">{ number }</p>
            <p>
                <button className="p-5 bg-amber-500"
                        onClick={ () => setNumber(prev => prev + 1) }>+1
                </button>
            </p>
        </div>
    )
}

// () => setNumber(number + 1)
// () => setNumber(prev => prev + 1) 此处两种方式都无问题 简单情况时 可使用第一种方式

// 一些特殊情况：比如某些 hook 仅执行一次 假设为 useFn()
    useFn() {
        setState(number 此方式的 number 被锁定为首次渲染值)
    }
```

5.3.4 典型运用一

> 按钮唯一性选择

- **src/App.jsx**

```
import { useState } from "react";
import { cn } from "./lib/tailwind/index.js";

export function App() {

    const [ selected, setSelected ] = useState(0);

    const arr = [ "tailwind", "react", "next", "motion", "zustand" ];

    return (
        <ul className="p-20 flex gap-5">
            {
```

```
                arr.map((title, index) => (
                    <li className={ cn("px-8 py-2 bg-green-500 text-white
cursor-pointer",
                                       selected === index && "bg-rose-500") }
                        onClick={ (() => setSelected(index)) }
                        key={ index }>
                        { title }
                    </li>
                ))
            }
        </ul>
    )
}
```

5.3.5 典型运用二

> 图片循环轮播

- **src/App.jsx**

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

import { FaAngleLeft, FaAngleRight } from "react-icons/fa";

export function App() {

    const [ index, setIndex ] = useState(3);

    function wrap(min, max, value) {
        const range = max - min;
        return ((((value - min) % range) + range) % range) + min;
    }

    const images = [
        "https://cdn.pixabay.com/photo/2023/03/15/16/17/feather-
7854908_1280.jpg",
        "https://cdn.pixabay.com/photo/2022/08/05/10/22/purple-bells-
7366491_1280.jpg",
        "https://cdn.pixabay.com/photo/2023/03/02/03/01/bird-7824442_1280.jpg",
        "https://cdn.pixabay.com/photo/2022/10/24/09/31/flower-
7543035_1280.jpg",
        "https://cdn.pixabay.com/photo/2024/12/18/01/27/lightning-
9274136_1280.jpg",
        "https://cdn.pixabay.com/photo/2025/06/23/18/09/blossom-
9676411_1280.jpg",
```

```
        "https://cdn.pixabay.com/photo/2023/03/08/13/20/nature-
7837757_1280.jpg",
        "https://cdn.pixabay.com/photo/2019/01/22/16/38/winter-
3948461_1280.jpg",
    ];

    return (
        <div className="w-7xl mx-auto h-160 mt-10 relative">
            <img className="size-full object-center select-none" src={
images[index] } alt=""/>

            <FaAngleLeft className="absolute left-10 top-1/2 -translate-y-1/2
                                   text-white cursor-pointer"
                         size={ 64 }
                         onClick={ () => setIndex(wrap(0, images.length, index -
1)) }/>
            <FaAngleRight className="absolute right-10 top-1/2 -translate-y-1/2
                                    text-white cursor-pointer"
                          size={ 64 }
                          onClick={ () => setIndex(wrap(0, images.length, index
+ 1)) }/>
        </div>
    )
}
```

5.3.6 二次封装

> 二次 HOOK 必须有对 内置 HOOK 的调用

- **src/lib/hooks/index.js**

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

export function useToggle(boolean) {
    const [ value, setValue ] = useState(!!boolean);

    function toggle() {
        setValue(prev => !prev);
    }

    return {
        value, toggle,
    }
}
```

```
export function useCounter(value) {

    // check type ...

    const [ count, setCount ] = useState(value ?? 0);

    function increment() {
        setCount(prev => prev + 1);
    }

    function decrement() {
        setCount(prev => prev - 1);
    }

    return {
        count, increment, decrement,
    }

}
```

- **src/App.jsx**

```
import { useToggle } from "./lib/hooks";

export function App() {

    const { value, toggle } = useToggle(true);

    return (
        <div className="p-32">
            <button className="p-5 bg-rose-500 text-white" onClick={ toggle }>
                show / hidden
            </button>
            {
                value && (
                    <div className="size-48 bg-rose-500 mt-10">Box</div>
                )
            }
        </div>

    )
}
```

5.04 useEffect

> 异步执行便捷钩子、可跟踪变量再次触发参数函数执行

5.4.1 异步执行

> useEffect(fn, []) ⇒ fn 将被异步执行

- **src/App.jsx**

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

export function App() {

    console.log("A");

    useEffect(() ⇒ console.log("B"), []);

    console.log("C");

    // console log  A C B

    return (
        <div className="p-32 bg-rose-500"/>
    )
}
```

5.4.2 变量跟踪

> useEffect(fn, []) ⇒ [] 无依赖变量仅在组件首次渲染执行 可跟踪非 state 变量

- **src/App.jsx [状态变量]**

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

export function App() {

    const [ number, setNumber ] = useState(100);

    useEffect(() => console.log(Math.random()), [ number ]);

    return (
        <div className="p-32 bg-rose-500">
            <button className="p-5 bg-green-500"
                    onClick={ () => setNumber(number + 1) }>+1
            </button>
        </div>
    )
}
```

- **src/App.jsx [非状态变量]**

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

let counter = 0;

export function App() {

    const [ number, setNumber ] = useState(100);

    useEffect(() => console.log(Math.random()), [ counter ]);

    return (
        <div className="p-32 bg-rose-500">
            <button className="p-5 bg-green-500"
                    onClick={ () => {
                        setNumber(number + 1);
                        counter++;
                    } }>+1
            </button>
        </div>
    )
}
```

5.4.3 组件卸载

> useEffect(() => { return fn }, []) fn 将在组件卸载时执行

- src/components/Box.jsx

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

export function Box() {

    useEffect(() => {

        // ...

        return () => {
            console.log(Math.random()); // 此处使用外部变量易产生错误
        }

    }, []);

    return (
        <div className="p-10 bg-green-500">Box</div>
    )
}
```

- src/App.jsx

```
import { useState } from "react";
import { Box } from "./components/Box.jsx";

export function App() {

    const [ show, setShow ] = useState(true);

    return (
        <div className="p-32 bg-slate-300 space-y-10">
            <button className="p-5 bg-rose-500" onClick={()=> setShow(!show)}>
                show / hidden
            </button>
            {
                show && (
                    <Box/>
```

```
                )
            }
        </div>
    )
}
```

5.4.4 二次封装

> 下案例易错说明：useEffect 在首次渲染将持有回调函数引用不再更新、useRef 保证重新渲染获取实时函数引用

- **src/lib/hooks/index.js**

```
import { useRef, useEffect } from "react";

export function useUnmount(callback) {

    const ref = useRef(callback);
    ref.current = callback;

    useEffect(() => () => {
        ref.current()
    }, []);

}
```

- **src/components/Box.jsx**

```
import { useUnmount } from "../lib/hooks";

export function Box() {

    useUnmount(()=> console.log(Math.random()));

    return (
        <div className="p-10 bg-green-500">Box</div>
    )
}
```

5.05 useMemo

> 缓存耗时计算结果、即使组件被多次重新渲染、即贯穿组件多次渲染周期

- **src/App.jsx**

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

const a = 2;
const b = 3;

export function App() {

    const [ number, setNumber ] = useState(100);

    function add(one, other) {
        console.log("add function called ...");
        return one + other;
    }

    const sum = add(a, b);

    const cache = useMemo(() => {
        console.log("useMemo function called ...");
        return a + b;
    }, [ a, b ]);

    return (
        <div className="p-32 bg-rose-500">

            <p>sum : { sum }</p>
```

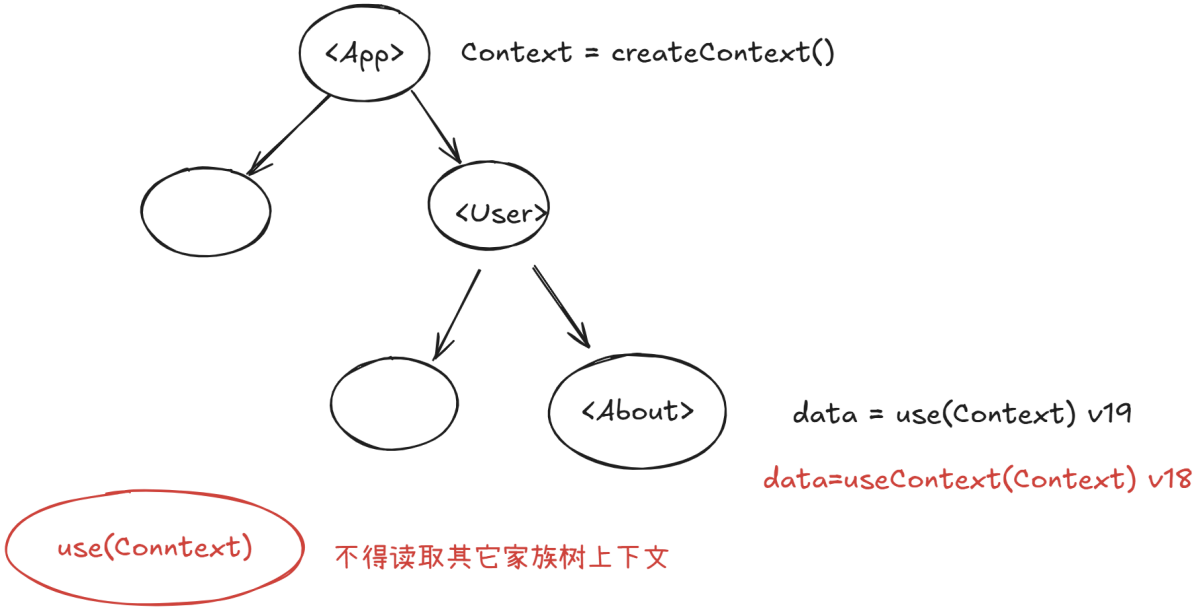
```
            <p>sum : { sum }</p>
            <p>sum : { sum }</p>

            <p>cache : { cache }</p>
            <p>cache : { cache }</p>
            <p>cache : { cache }</p>

            <p>number : { number }</p>

            <button className="p-5 bg-green-500"
                    onClick={ () => setNumber(number + 1) }>+1
            </button>
        </div>
    )
}
```

5.06 useContext

> createContext | useContext | use 深层级组件树传值和取值

- ☑ createContext | use 是 API、但 use() 仍需遵守 hook 顶层调用约定
- ☑ 上下文必须自上而下 不得倒置

- **UI**



- **src/App.jsx**

```jsx
import { createContext, useState } from "react";

import { User } from "./components/User.jsx";

export const AnyContext = createContext(null);

export function App() {

    const [ number, setNumber ] = useState(1000);

    return (
        <AnyContext value={ number }>
            <div className="p-32 bg-rose-500">
                <p>{ number }</p>
                <button className="p-5 bg-amber-500"
                        onClick={ () => setNumber(number + 1) }>+1
                </button>

                <User number={ number }/>

            </div>
        </AnyContext>
    )
}

// <AnyContext.Provider value> v18
// <AnyContext          value> v19
```

- **src/components/User.jsx**

```jsx
import { About } from "./About.jsx";

export function User({ number }) {
    return (
        <div className="p-5 bg-slate-300">
            <p>User : { number }</p>
            <About/>
        </div>
    )
}
```

- **src/components/About.jsx**

```
import { use } from "react";
import { AnyContext } from "../App.jsx";

export function About() {

    const number = use(AnyContext);

    return (
        <div className="p-5 bg-slate-600">
            <p>About : { number }</p>
        </div>
    )
}
```

5.07 useCallback

> 缓存函数对象自身、以避免组件重新渲染过程反复创建函数

☑ cacheFn = useCallback(fn, [dependencies]) 一般用在二次 hook 作为优化手段

5.7.1 使用方式

> 通常较少在组件内主动使用

- **src/App.jsx**

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

```
export function App() {

    const [ number, setNumber ] = useState(1000);

    const increment = useCallback(() => {
        setNumber(prev => prev + 1);
    }, []);

    return (
        <div className="p-32 bg-rose-500">
            <p>{ number }</p>
            <button className="p-5 bg-amber-500"
                    onClick={ increment }>+1
            </button>
        </div>
    )
}
```

5.7.2 二次封装

> 一般用作优化手段

- **src/lib/hooks/index.js**

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

export function useCounter(initial) {
    const [ count, setCount ] = useState(initial ?? 0);

    const increment = useCallback(() => {
        setCount(x => x + 1);
    }, [])

    const decrement = useCallback(() => {
        setCount(x => x - 1);
    }, [])

    return {
        count,
        increment,
        decrement,
    }
}
```

> 仍需注意使用外部变量造成的错误陷阱

- **src/App.jsx**

```jsx
import { useState, useCallback } from "react";

export function App() {

    const [ number, setNumber ] = useState(1000);

    const increment = useCallback(() => {
        // setNumber(prev => prev + 1);
        setNumber(number + 1);
    }, []);

    return (
        <div className="p-32 bg-rose-500">
            <p>{ number }</p>
            <button className="p-5 bg-amber-500"
                    onClick={ increment }>+1
            </button>
        </div>
    )
}
```

5.08 useReducer

> 较 useState 允许将 setState 复杂逻辑提取到 一个 Fn

☑ `const [state, dispatch] = useReducer(fn, initValue, initMapFn?)`

- **src/App.jsx**

```jsx
import { useReducer } from "react";
```

```
function reducer(user, action) {

    if (action.type === "+") {
        return {
            ...user,
            age: user.age + action.number
        }
    }

    if (action.type === "-") {
        return {
            ...user,
            age: user.age - action.number
        }
    }

    return user;

}


export function App() {

    const [ user, dispatch ] = useReducer(reducer, { username: "jackson", age:
18 }, undefined);

    return (
        <div className="p-32 bg-rose-500">
            <p>{ user.username } : { user.age }</p>
            <button className="p-5 bg-amber-500"
                    onClick={ () => dispatch({ type: "+", number: 10 }) }>age +
10</button>
            <br/>
            <button className="p-5 bg-amber-500"
                    onClick={ () => dispatch({ type: "-", number: 20 }) }>age -
20</button>
        </div>
    )
}
```

5.09 useOptimistic

> === 检测状态更新时立即更新 UI、不再等待 更新状态所需的后续操作

- ✅ const [state, triggerStateFn] = useOptimistic(initState, updateStateFn);
- ✅ 仅 react server component [RSC] 环境下可用

- **Page.tsx [next]**

```tsx
"use client"

import { useOptimistic } from "react";

interface Goods {
    id: number;
    name: string;
}

export default function Page() {

    const goodsList: Goods[] = [
        {
            id: 1,
            name: "飞科剃须刀"
        },
        {
            id: 2,
            name: "JavaScript实战编程书籍"
        },

    ];

    function updateGoodsList(goodsList: Goods[], id: number) {
        return goodsList.filter(goods => goods.id !== id);
    }

    const [optimisticGoodsList, addOptimisticGoodsId] = useOptimistic(goodsList,
updateGoodsList);

    async function deleteGoodsById(id: number) {
        addOptimisticGoodsId(id);
        await new Promise(resolve => setTimeout(resolve, 4000));
    }

    return (
        <div className="p-10 bg-amber-200">
            <ul>
                {
                    optimisticGoodsList.map(({ name, id }: Goods) => (
                        <li key={ id } className="flex items-center">
                            <span>{ name }</span>
                            <form action={ () => deleteGoodsById(id) }>
                                <button className="p-5 bg-amber-300"
                                        type="submit">delete</button>
                            </form>
                        </li>
                    ))
                }
            </ul>
```

```
            </div>
        )
    }
```

5.10 useTransition

> useTransition | startTransition 异步更新状态、且返回耗时标识

- **src/App.jsx**

```
import { useTransition } from "react";
import { useCounter } from "usehooks-ts";

export function App() {

    const { count, increment } = useCounter(1000);

    const [ isPending, startTransition ] = useTransition();

    function h() {
        startTransition(async () => {
            await new Promise(resolve => setTimeout(resolve, 1000));
            increment();
        });
    }

    return (
        <div className="p-10">
            <p>{ count }</p>
            <button className="p-5 bg-green-500 disabled:bg-slate-500
disabled:cursor-not-allowed"
                    onClick={ h } disabled={ isPending }>
                increment
            </button>
        </div>
    )
}
```

- **src/App.jsx**

```
import { useCounter } from "usehooks-ts";
import { startTransition } from "react";

export function App() {

    const { count, increment } = useCounter(1000);

    function h() {
        startTransition(async () => {
            await new Promise(resolve => setTimeout(resolve, 1000));
            increment();
        })
    }

    return (
        <div className="p-10">
            <p>{ count }</p>
            <button className="p-5 bg-green-500"
                    onClick={ h }>
                increment
            </button>
        </div>
    )
}
```

5.11 useLayoutEffect

> useLayoutEffect 是 useEffect 同步版本会在真实 DOM 绘制之前执行 [会阻止渲染UI]

- **src/App.jsx**

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

export function App() {

    useLayoutEffect(() => {

        function task() {
            const start = Date.now();
            while (Date.now() - start < 3000) {
                Math.sqrt(Math.random() ** 2);
            }
```

```
            console.log("task end ...")
        }

        task();

    }, []);

    return (
        <div className="p-32 bg-rose-500" />
    )
}
```

06. Hooks Libary

> React Hooks 生态较为丰富、Hooks 自行封装极易出错、推荐使用三方库

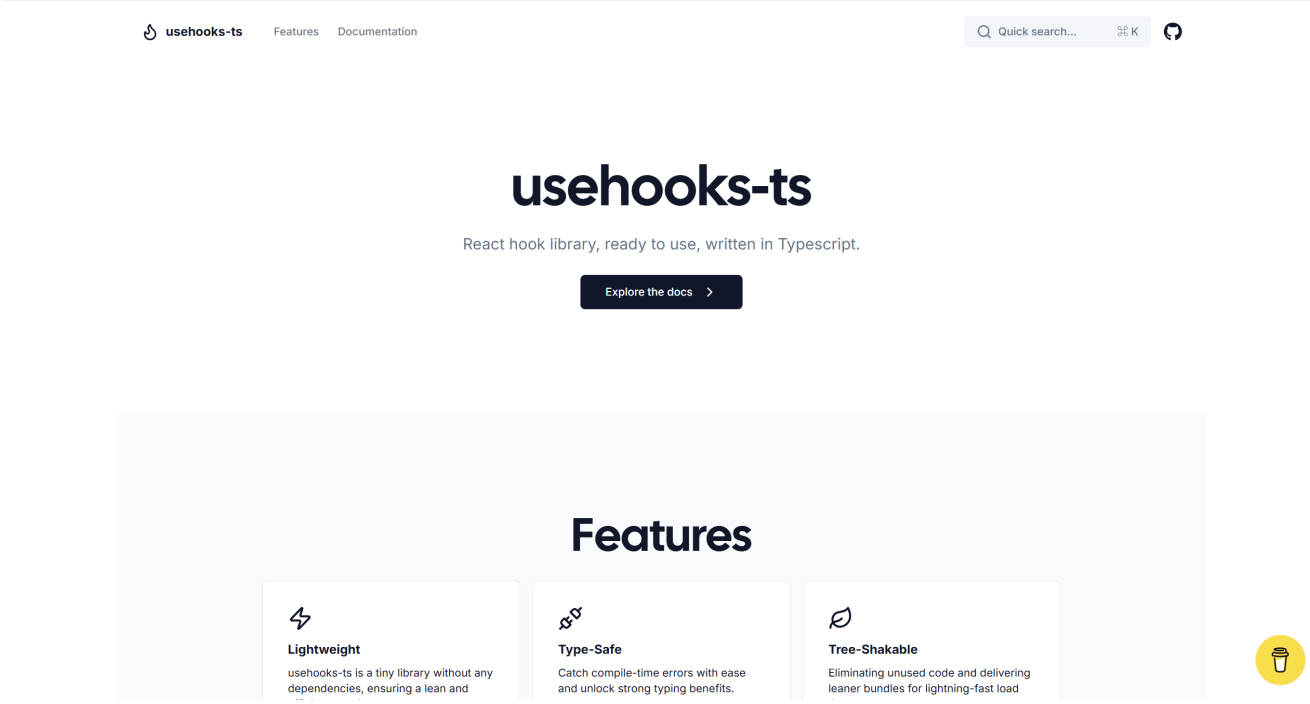
6.1 流行库

> usehooks | rooks | react-use ...

6.1.1 usehooks

> https://usehooks-ts.com/

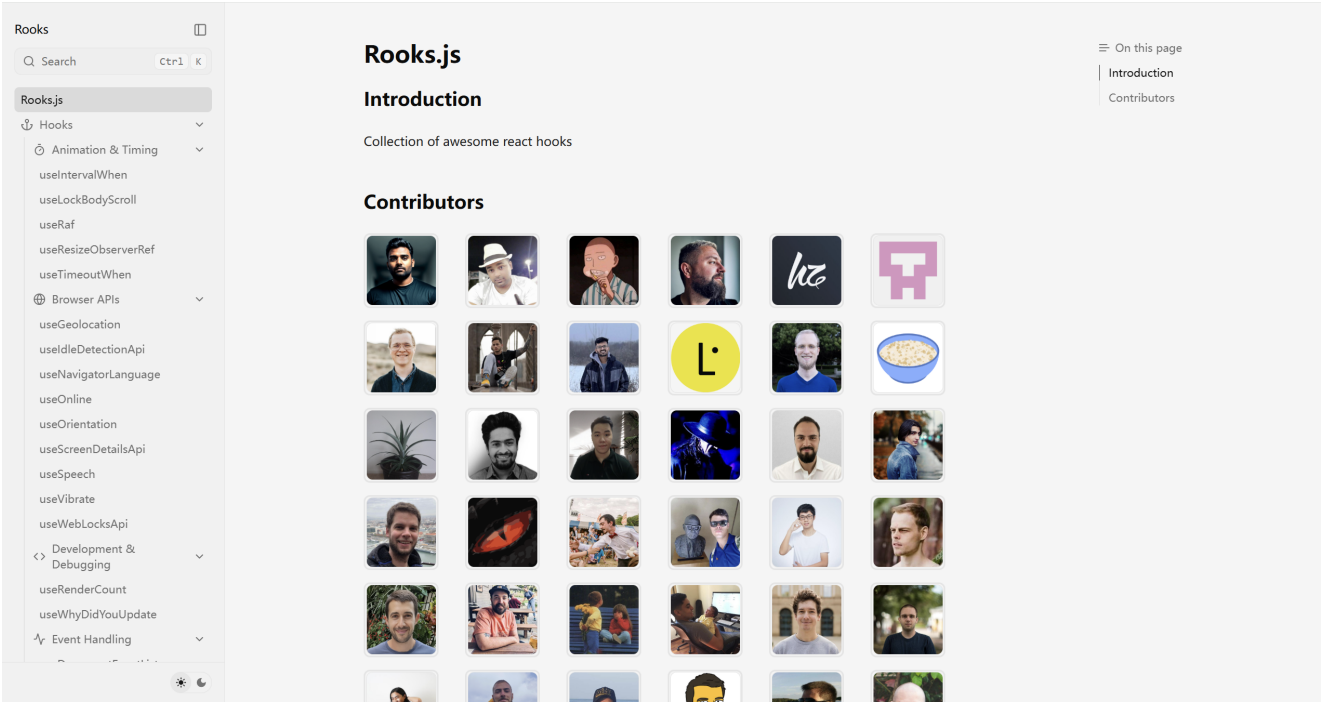
- UI



6.1.2 rooks

> https://rooks.vercel.app/docs

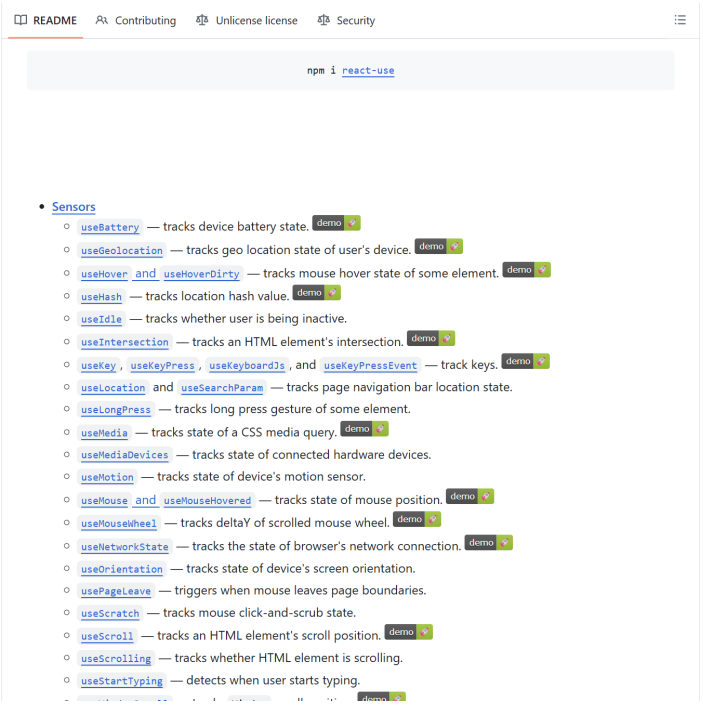
- UI



6.1.3 react-use

> https://github.com/streamich/react-use

- UI



6.2 usehooks

> https://usehooks-ts.com/

- npm

```
npm insatll usehooks-ts
```

6.2.1 useHover

> JavaScript Hover 控制

- **src/App.jsx**

```
import { useHover } from "usehooks-ts";
import { useRef } from "react";

export function App() {

    const ref = useRef(null);
    const isHover = useHover(ref);

    return (
        <div className="p-20 bg-rose-500">
            <h1 className="bg-green-500 p-10 cursor-pointer" ref={ ref }>
                { isHover ? "Hover" : "Not Hover" }
            </h1>
        </div>
    )
}
```

- **src**

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

import type { RefObject } from "react";

import { useEventListener } from "usehooks-ts";

export function useHover<T extends HTMLElement = HTMLElement>(elementRef:
RefObject<T>): boolean {
    const [ value, setValue ] = useState<boolean>(false);

    const handleMouseEnter = () => {
        setValue(true);
    }
    const handleMouseLeave = () => {
        setValue(false);
    }
```

```
    useEventListener("mouseenter", handleMouseEnter, elementRef);
    useEventListener("mouseleave", handleMouseLeave, elementRef);

    return value;
}
```

6.2.2 useInterval

> 无须清理的定时器实现

- **src/App.jsx**

```
import { useState } from "react";
import { useInterval } from "usehooks-ts";
import dayjs from "dayjs";

export function App() {

    const [ datetime, setDateTime ] = useState(new Date());

    useInterval(() => setDateTime(new Date()), 1000);

    return (
        <div className="p-20 bg-rose-500">
            <h1 className="bg-green-500 p-10 cursor-pointer">
                { dayjs(datetime).format("YYYY-MM-DD HH:mm:ss") }
            </h1>
        </div>
    )
}
```

- **src**

```
import { useEffect, useRef } from "react";

import { useIsomorphicLayoutEffect } from "usehooks-ts";

export function useInterval(callback: () => void, delay: number | null) {
    const savedCallback = useRef(callback);

    useIsomorphicLayoutEffect(() => {
        savedCallback.current = callback
    }, [ callback ]);

    useEffect(() => {
        if (delay === null) {
            return;
        }

        const id = setInterval(() => {
            savedCallback.current();
        }, delay);

        return () => {
            clearInterval(id);
        }
    }, [ delay ]);
}
```

6.2.3 useToggle

> 布尔切换

- **src/App.jsx**

```
import { useToggle } from "usehooks-ts";

export function App() {

    const [ visibility, toggle ] = useToggle(true);

    return (
        <div className="p-20">
            <button className="bg-black p-5 text-white" onClick={ toggle }>
                show / hidden
            </button>
            {
                visibility && (
```

```
                    <div className="w-60 aspect-3/4 bg-cyan-400 mt-10"/>
                )
            }
        </div>
    )
}
```

- **src**

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

import type { Dispatch, SetStateAction } from "react";

export function useToggle(defaultValue?: boolean): [
    boolean,
    () => void,
    Dispatch<SetStateAction<boolean>> ] {

    const [ value, setValue ] = useState(!!defaultValue);

    const toggle = useCallback(() => {
        setValue(x => !x);
    }, [])

    return [ value, toggle, setValue ];
}
```

6.2.4 useMediaQuery

> 媒体查询

- **src/App.jsx**

```
import { useMediaQuery } from "usehooks-ts";

export function App() {

    const matches = useMediaQuery("(max-width: 640px)");
```

```
    if (matches) {
        return (
            <div className="p-10 bg-rose-500">mobile</div>
        )
    }

    return (
        <div className="p-10 bg-blue-500">PC</div>
    )
}
```

6.2.5 useIntersectionObserver

> 元素视口检测

- **src/App.jsx**

```
import { useIntersectionObserver } from "usehooks-ts";
import { cn } from "clsx-for-tailwind";

function Box() {

    const { isIntersecting, ref } = useIntersectionObserver({
        threshold: 0.8
    });

    return (
        <div className={ cn("w-80 aspect-3/4 bg-slate-600 opacity-30 duration-
300 scale-90",
            isIntersecting && "bg-purple-700 opacity-100 scale-100") } ref={ ref
}/>
    )
}

export function App() {

    return (
        <div className="p-20 bg-black space-y-40">
            <Box/>
            <Box/>
            <Box/>
```

```
            <Box/>
            <Box/>
            <Box/>
            <Box/>
            <Box/>
            <Box/>
        </div>
    )
}
```

6.2.6 useCopyToClipboard

> 文本复制

- **src/App.jsx**

```
import { useCopyToClipboard } from "usehooks-ts";
import { useRef } from "react";

export function App() {

    const [ copiedText, copyFn ] = useCopyToClipboard();
    const ref = useRef(null);

    return (
        <div className="p-20 bg-rose-500 space-y-20">
            <p className="text-6xl text-white" ref={ ref }>千里之行、始于足下。</p>
            <p className="text-6xl text-white">
                <span onClick={ async () => await
copyFn(ref.current.textContent) }>
                    点击复制到此处:
                </span>
                { copiedText }
            </p>
        </div>
    )
}
```

6.2.7 useDebounceValue

> 防抖状态

- src/App.jsx

```
import { useDebounceValue } from "usehooks-ts";

export function App() {

    const [ debounce, setDebounce ] = useDebounceValue(18, 1000); // value, ms

    return (
        <div className="p-10 bg-rose-500">
            <p>{ debounce }</p>
            <button className="p-5 bg-cyan-400 text-white"
                    onClick={ () => setDebounce(debounce + 1) }>+1
            </button>
        </div>
    )
}
```

6.2.8 useDocumentTitle

> 文档标题

- src/App.jsx

```
import { useDocumentTitle } from "usehooks-ts";

export function App() {

    useDocumentTitle("App - Home");

    return (
        <div className="p-20 bg-rose-500">
           <h1 className="text-6xl text-white">Home</h1>
        </div>
    )
}
```

6.2.9 useTimeout

> 无须清理延时实现

- **src/App.jsx**

```
import { useTimeout } from "usehooks-ts";

export function App() {

    useTimeout(() => alert("delay 3s"), 3000);

    return (
        <div className="p-10 bg-rose-500"/>
    )
}
```

07. 注意事项

7.1 根元素

> react 要求组件必须有且唯一根元素

- src/App.jsx

```
export function App() {
    return (
        <>
            <div className="p-20 bg-rose-500">
                <h1 className="text-6xl text-white">root</h1>
            </div>
        </>
    )
}
```

- src/App.jsx

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

export function App() {
    return (
        <div className="p-10 bg-slate-200">
            {
                [ ...Array(6) ].map((_, i) => (
                    <Fragment key={ i }>
                        <p>p : { i }</p>
                        <span>span : { i }</span>
                    </Fragment>
                ))
            }
        </div>
    )
}
```

- **src/App.jsx**

```
import { Fragment, useState } from "react";
import { cn } from "clsx-for-tailwind";
import { FiRefreshCcw } from "react-icons/fi";

function Refresh({ children, className }) {

    const [ randomKey, setRandomKey ] = useState(Math.random());

    return (
        <Fragment key={ randomKey }>
            <div className={ cn("max-w-7xl mx-auto py-20 relative", className)
}>
                { children }
                <button className="absolute top-5 right-5 p-2 cursor-pointer
                                   hover:rotate-180 duration-300"
                        onClick={ () => setRandomKey(Math.random) }>
                    <FiRefreshCcw className="text-white text-2xl"/>
                </button>
            </div>
        </Fragment>
    )
}

export function App() {
    return (
        <>
            <Refresh className="bg-slate-900 px-5">
                <div>something</div>
            </Refresh>
        </>
    )

}
```

自定义根元素

- **src/App.jsx**

```
function Box({ as = "div", children, className, ...props }) {

    if (as === "h1" ||
        as === "h2" ||
```

```
        as === "h3" ||
        as === "h3" ||
        as === "h4" ||
        as === "h5" ||
        as === "h6" ||
        as === "p"  ||
        as === "div") {

        // 不可省略该操作 确保自定义 Tag 首字母大写
        const Tag = as;

        return (
            <Tag className={ className } { ...props }>{ children }</Tag>
        )

    }

    return null;

}

export function App() {

    return (
        <div className="p-20 bg-rose-500">
            <Box as="h2" className="bg-purple-700 p-10"><i>e</i></Box>
        </div>
    )

}
```

=== jsx/jsxs 函数

- **src/App.jsx**

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

export function Box({ as }) {

    if (as === "h1" || as === "h2" || as === "h3" || as === "h4" || as === "h5"
|| as === "h6") {

        const title = "jsx-runtime";
        const content = "react/jsx-runtime";

        return jsxs(as,
            {
```

```
                children: [
                    jsx("p",
                        {
                            children: title,
                            className: "p-5 bg-rose-500 text-white",
                        }),
                    jsx("p",
                        {
                            children: content,
                            className: "p-5 bg-rose-400 text-white",
                        }),
                ],
                className: "bg-slate-300 p-20"
            });
    }

    return null;

}

export function App() {
    return (
        <div>
            <Box as="h1"/>
        </div>
    )
}
```

7.2 延迟组件加载

> `=== lazy(() => import(""))`

☑ import 可当作类似函数调用 实现对模块异步导入

- **src/App.jsx**

```
export function App() {

    async function readPackageJson() {
```

```
        const json = await import("../package.json").then(module =>
module.default);
        console.log(json);
    }

    readPackageJson();

    return (
        <div className="p-20 bg-rose-500"/>
    )

}
```

- **src/components/Box.jsx**

```
export default function Box() {
    return (
        <div className="p-20 bg-slate-200">box</div>
    )
}
```

- **src/App.jsx**

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

const Box = lazy(() => import("./components/Box.jsx"));

export function App() {

    return (
        <div className="p-20 bg-rose-500">
            <Box/>
        </div>
    )

}
```

7.3 组件缓存

> props 未产生变化的被动重新渲染、组件卸载后渲染挂载引起的状态丢失

7.3.1 被动重新渲染

> 如直接被嵌套的组件会因当前组件的重新渲染而渲染、期望 PROP 不变而触发重新渲染使用

- src/components/Box.jsx

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

export const Box = memo(function ({ number }) {

        console.log(Math.random());

        return (
            <div className="bg-white">box : { number }</div>
        )
    }
);
```

- src/App.jsx

```
import { useCounter } from "usehooks-ts";

import { Box } from "./components/Box.jsx";

export function App() {

    const { count, increment } = useCounter(100);
```

```
    return (
        <div className="p-20 bg-rose-500">
            <p>{ count }</p>
            <button onClick={ increment }>increment</button>
            <Box number={ 100 or count }/>
        </div>
    )

}
```

7.3.2 组件卸载

> 组件卸载后重新挂载是一次全新的渲染、即上次渲染的状态全部丢失

☑ 推荐优先使用 display: none 思路 | 使用父组件状态或数据

- **src/App.jsx**

```
import { useCounter, useToggle } from "usehooks-ts";

export function App() {

    const { count, increment } = useCounter(100);
    const [ value, toggle ] = useToggle(true);

    return (
        <div className="p-20 bg-rose-500">
            <button onClick={ toggle }>show / hidden</button>
            <div style={ {
                display: value ? "block" : "none",
            } }>
                <p>{ count }</p>
                <button onClick={ increment }>increment</button>
            </div>
        </div>
    )

}
```

- **src/App.jsx**

```jsx
import { useState } from "react";
import { cn } from "clsx-for-tailwind";

export function App() {

    const titles = [ "react", "next", "motion", "gsap", "zustand" ];
    const [ selected, setSelected ] = useState("react");

    // state ...

    return (
        <div className="p-20">
            <ul className="flex gap-5">
                {
                    titles.map((title) => (
                        <li key={ title }>
                            <button
                                className={ cn("p-5 bg-slate-900 text-white
cursor-pointer",
                                               selected === title && "bg-rose-500")
}
                                onClick={ () => setSelected(title) }>
                                { title }
                            </button>
                        </li>
                    ))
                }
            </ul>
            <div className="my-10">
                {
                    selected === "react" && (
                        <div className="p-32 bg-cyan-400">react</div>
                    )
                }

                {
                    selected === "next" && (
                        <div className="p-32 bg-black text-white">next</div>
                    )
                }

                {
                    selected === "motion" && (
                        <div className="p-32 bg-yellow-400">motion</div>
                    )
                }

                {
                    selected === "gsap" && (
                        <div className="p-32 bg-green-500">gsap</div>
                    )
                }
```

```
                {
                    selected === "zustand" && (
                        <div className="p-32 bg-slate-500">zustand</div>
                    )
                }

            </div>
        </div>
    )

}
```

```
=== npm install react-activation
```

- **src/App.jsx**

```
import { useCounter, useToggle } from "usehooks-ts";

import { KeepAlive, AliveScope } from "react-activation";

function Box() {

    const { count, increment } = useCounter();

    return (
        <div className="p-20 bg-slate-500">
            <p>{ count }</p>
            <button onClick={ increment }>increment</button>
            <div className="h-60 overflow-y-auto">
                <div className="h-screen"/>
            </div>
        </div>
    )
}


export function App() {

    const [ show, toggle ] = useToggle(true);

    return (

        <div className="p-20 bg-rose-500">
            <button className="cursor-pointer" onClick={ toggle }>show /
hidden</button>
            <AliveScope>
```

```
                {
                    show && (
                        <KeepAlive>
                            <Box/>
                        </KeepAlive>
                    )
                }
            </AliveScope>
        </div>
    )

}
```

7.4 后备组件

> 逻辑变量变更、数据请求前后、组件渲染耗时期望 Loading 组件代替目标组件

7.4.1 数据处理

> 设计需借助 状态变量 触发重新渲染以获得目标 UI

- **src/App.jsx**

```
import { useState } from "react";
import { useTimeout } from "usehooks-ts";

export function App() {

    const [ number, setNumber ] = useState(0);

    useTimeout(() => setNumber(1), 3000);

    return (
        <div>
```

```
            {
                number === 1 && (
                    <div className="p-10 bg-rose-500">ui</div>
                )
            }
            {
                number === 0 && (
                    <div className="p-10 bg-blue-500">loading</div>
                )
            }
        </div>
    )

}
```

> **Note**
>
> ☑ 注意产生数据请求时一般是三个状态：请求前 -1 | 请求到期望数据 1 | 请求到非期望数据 0

7.4.2 Suspense

> === Suspense React 内置后备组件手段、仅支持异步组件 Promise<Component>

☑ Suspense 通过确认异步组件 待定状态 控制后备组件和目标组件的切换

- **src/App.jsx**

```
import { lazy, Suspense } from "react";

const Box = lazy(() => new Promise(resolve => setTimeout(() => {
        resolve({
            default: function () {
                return (
                    <div className="p-10 bg-blue-500 text-white">Box</div>
                )
```

```
            }
        });
    }, 3000))
);

function Loading() {
    return (
        <div className="flex items-center gap-5">
            <span className="size-8 border-2 border-white border-r-transparent
                              rounded-full animate-spin"/>
            <span className="text-white">loading</span>
        </div>
    )
}

export function App() {
    return (
        <div className="p-10 bg-rose-500">
            <Suspense fallback={ <Loading/> }>
                <Box/>
            </Suspense>
        </div>
    )
}
```

7.5 数据请求

```
=== npm install swr
```

- **src/App.jsx**

```
import useSWR from "swr";

const fetcher = (url) => fetch(url).then(response => response.json())

export function App() {

    const url = "https://jsonplaceholder.typicode.com/posts/1";

    const { data, isLoading } = useSWR(url, fetcher);

    if (isLoading) {
        return (
            <div className="p-20 bg-green-500">Loading...</div>
```

```
        )
    }

    return (
        <div className="p-20 bg-rose-500">
            { JSON.stringify(data, null, 2) }
        </div>
    )

}
```

7.6 文档结构

> createPortal(Component, selector) 指定 DOM 位置渲染

- **src/App.jsx**

```
import { createPortal } from "react-dom";

function Box() {
    return (
        <div className="size-48 bg-green-500"></div>
    )
}

export function App() {

    return (
        <div className="p-10 bg-rose-500">
            { createPortal(<Box/>, document.body) }
        </div>
    )
}
```

08. 状态管理

> 全局的状态变量、可在任意无关联的组件中使用状态

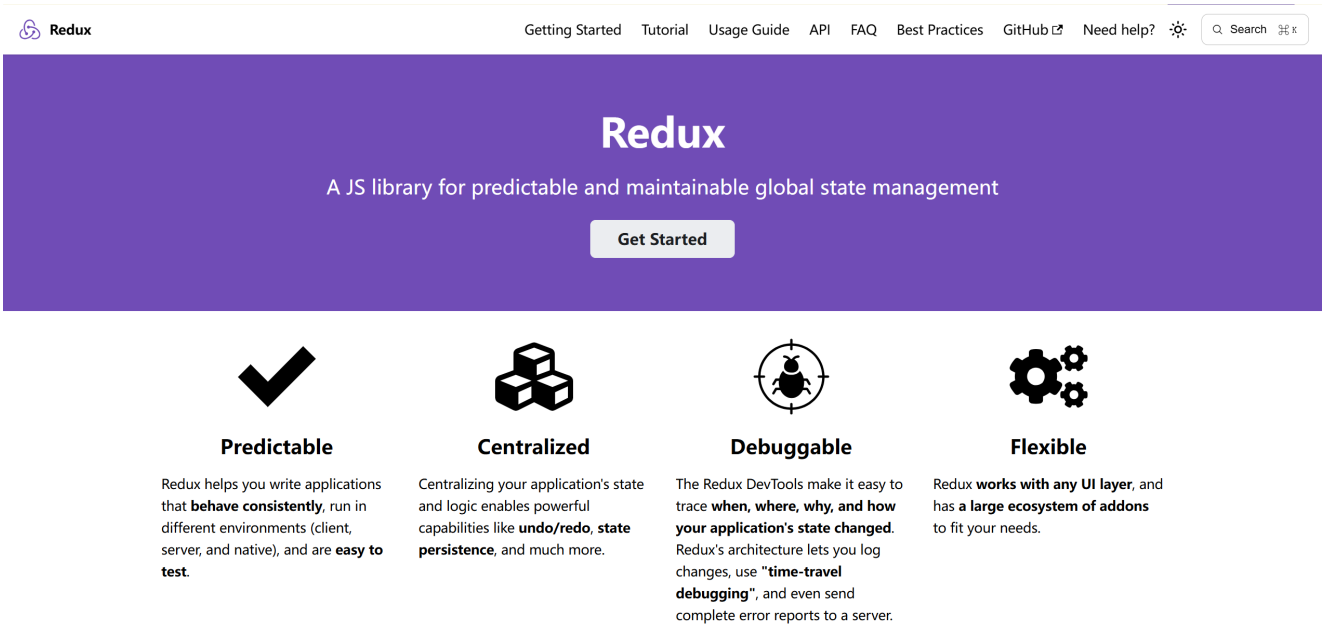
8.1 流行库

> redux | mobx | zustand | jotai

8.1.1 Redux

> https://redux.js.org/

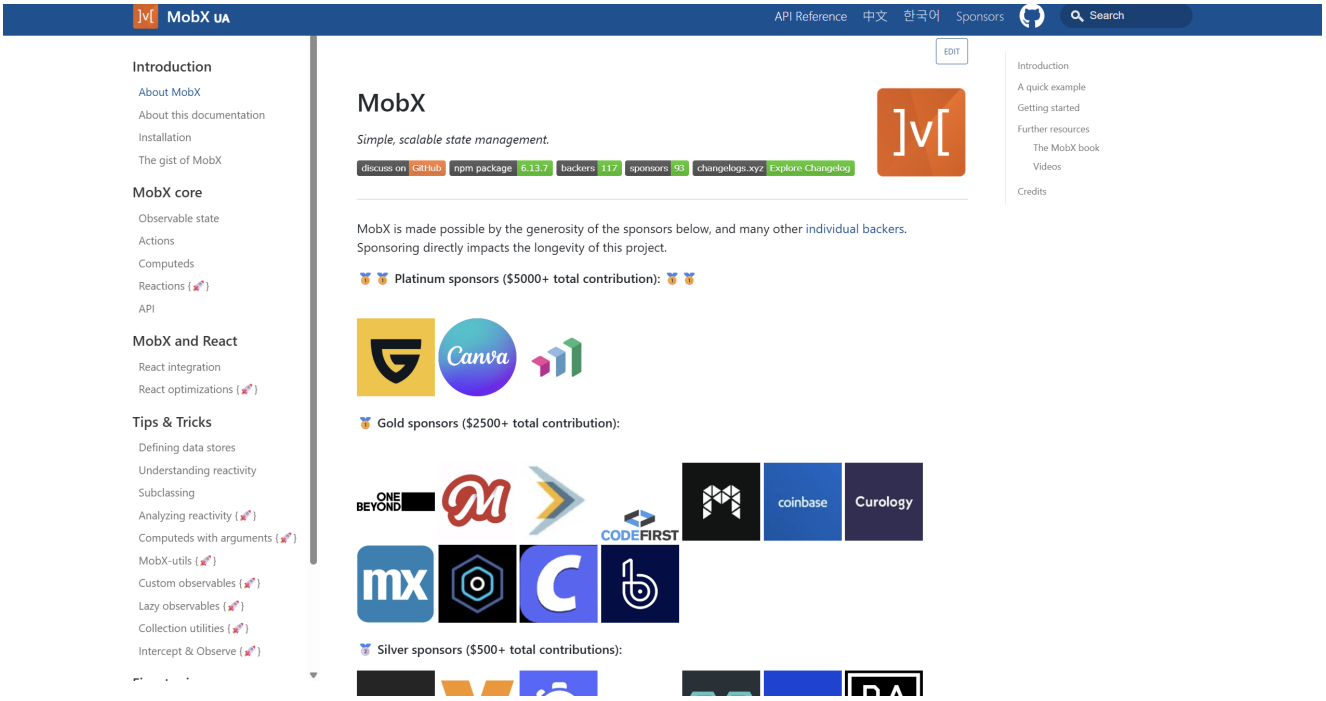
- UI



8.1.2 Mobx

> https://mobx.js.org/README.html

- UI



8.1.3 Zustand

> https://zustand-demo.pmnd.rs/

- UI

8.1.4 Jotai

> https://tutorial.jotai.org/

- UI

8.2 Zustand

```
npm install zustand
```

8.2.1 定义

```
create( (set)=> ({ 状态定义 }) )
```

- **src/zustand/index.js**

```
import { create } from "zustand";

export const globalUserState = create((set) => ({

    counter: 100,

    increment: () => {
        set((state) => ({ counter: state.counter + 1 }))
    },

    decrement: () => {
        set((state) => ({ counter: state.counter + 1 }))
    },

    update: (value) => {
        set((state) => ({ counter: state.counter + value }))
    },

}));
```

8.2.2 使用

```
const property = globalUserState(state => state.property)
```

- **src/App.jsx**

```
import { globalUserState } from "./zustand/user";
import { User } from "./components/User.jsx";

export function App() {

    const counter = globalUserState(state => state.counter);
    const increment = globalUserState(state => state.increment);

    return (
        <div className="p-20 bg-rose-500">
            <p>{ counter }</p>
            <button onClick={ increment }>increment</button>
            <User/>
        </div>
    )

}
```

8.2.3 持久化

> import { persist } from "zustand/middleware"; [根据业务可选]

- **src/zustand/user/index.js**

```
import { create } from "zustand";
import { persist, createJSONStorage } from "zustand/middleware";

export const globalUserState = create()(persist((set) => ({

    counter: 100,

    increment: () => {
        set((state) => ({ counter: state.counter + 1 }))
    },

    decrement: () => {
        set((state) => ({ counter: state.counter + 1 }))
    },
```

```
    update: (value) => {
        set((state) => ({ counter: state.counter + value }))
    },

}), {
    name: "user-state-in-session",
    storage: createJSONStorage(() => window.sessionStorage),
}));
```

8.2.4 注意事项

> **Note**
>
> ☑ zustand 操作的状态对象始终是不可变的

- **src/zustand/user/index.js**

```
import { create } from "zustand";

export const globalUserState = create((set) => ({

    user: {
        username: null,
        age: 18,
    },

    incrementAge: () => {
        set((state) => ({ ...state.user, age: state.user.age + 1 }));
    },

    decrementAge:
        () => {
            set((state) => ({ ...state.user, age: state.user.age - 1 }));
        },

    updateAge:
        (value) => {
            set((state) => ({ ...state.user, age: state.user.age + value }));
        },

}));
```

09. 面试题型

9.1 useState

> 考察重新渲染的正确认识

- [x] 判断以下代码是否正确运行并解释原因

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

export function App() {
    const [ number, setNumber ] = useState(0);

    function init() {
        setNumber(1);
    }

    init();

    return (
        <div>
            <p>{ number }</p>
            <button onClick={ () => setNumber(number + 1) }>+1</button>
        </div>
    )
}
```

> **Note**

- [x] 以上代码无法正确运行、会陷入死循环
- [x] 调用 init() 时组件还未完成渲染、并再次触发重新渲染、React 不允许渲染阶段触发重新渲染

9.2 useEffect

> 考察 useEffect 执行时机

☑ 解答输出顺序并解释 useEffect(fn) fn 执行时机

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

export function App() {

    console.log(1);
    console.log(2);

    useEffect(() => {
        console.log(4);
    }, []);

    async function asyncFn() {
        console.log(3);
    }

    asyncFn();

    return (
        <div/>
    )
}
```

> **Note**
>
> ☑ 程序输出: 1 2 3 4
>
> ☑ 组件解析流程 1. 翻译 jsx | 2. 调用组件内代码 | 3. 生成虚拟DOM | 4. 挂载真实DOM | 5. useEffect

9.3 useEffect

> useEffect(fn) | useEffect(fn, []) 区别

- ☑ useEffect(fn)　　-- 组件首次挂载执行、且任意形式的组件重新渲染依然执行
- ☑ useEffect(fn, [])　-- 仅组件首次挂载执行、明确通知 React 无依赖项无须再次执行