React 快速参考

组件、Hook、状态、副作用与常用模式

JSX 基础

表达式与属性

```
const name = "Alice";
const el = <h1>Hello, {name}!</h1>;
const img = <img src={url} alt="photo" />;
```

JSX 规则

{expression}	嵌入任意 JS 表达式
className	替代 HTML 的 **class**
htmlFor	替代 HTML 的 **for**
style={{color: 'red'}}	内联样式，值为对象
<Component />	自闭合标签（必须）
<> ... </>	Fragment（不产生额外 DOM 节点）

组件

函数组件

```
function Greeting({ name }) {
  return <h1>Hello, {name}!</h1>;
}

// Arrow function variant
const Greeting = ({ name }) => (
  <h1>Hello, {name}!</h1>
);
```

Props

```
function Card({ title, children }) {
  return (
    <div className="card">
      <h2>{title}</h2>
      {children}
    </div>
  );
}

<Card title="Welcome">
  <p>Content here</p>
</Card>
```

默认 Props

```
function Button({ label = "Click me", onClick }) {
  return <button onClick={onClick}>{label}</button>;
}
```

状态（useState）

基础状态

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

function Counter() {
  const [count, setCount] = useState(0);
  return (
    <button onClick={() => setCount(count + 1)}>
      Count: {count}
    </button>
  );
}
```

函数式更新

```
setCount(prev => prev + 1); // use prev state
setItems(prev => [...prev, newItem]); // arrays
setUser(prev => ({...prev, name: "Bob"})); // objects
```

状态规则

Immutable updates	永远不要直接修改 state
Async batching	更新可能被批量合并
Functional form	依赖上一个 state 时，使用 **prev =>** 形式

副作用（useEffect）

副作用模式

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

// Run on every render
useEffect(() => { /* ... */ });

// Run once on mount
useEffect(() => { /* ... */ }, []);

// Run when deps change
useEffect(() => { /* ... */ }, [count]);

// Cleanup on unmount
useEffect(() => {
  const id = setInterval(tick, 1000);
  return () => clearInterval(id);
}, []);
```

列表与 Key

```
function TodoList({ items }) {
  return (
    <ul>
      {items.map(item => (
        <li key={item.id}>{item.text}</li>
      ))}
    </ul>
  );
}
```

key 必须是稳定的唯一 ID，避免使用数组索引作为 key

事件处理

事件绑定

```
<button onClick={handleClick}>Click</button>
<button onClick={() => handleDelete(id)}>Del</button>
<input onChange={(e) => setVal(e.target.value)} />
<form onSubmit={(e) => {
  e.preventDefault();
  handleSubmit();
}}>
```

常用事件

onClick	鼠标点击
onChange	输入值变化
onSubmit	表单提交
onKeyDown	按键按下
onFocus / onBlur	获得焦点 / 失去焦点
onMouseEnter	鼠标进入元素

条件渲染

```
// Ternary
{isLoggedIn ? <Dashboard /> : <Login />}

// Logical AND (short-circuit)
{hasError && <ErrorBanner />}

// Early return
function Page({ user }) {
  if (!user) return <Login />;
  return <Dashboard user={user} />;
}
```

Hook

useRef

```
const inputRef = useRef(null);
// Access: inputRef.current.focus();
<input ref={inputRef} />
```

useRef 在渲染间保持值，但不会触发重新渲染

useMemo 与 useCallback

```
// Memoize expensive computation
const sorted = useMemo(
  () => items.sort(compareFn),
  [items]
);

// Memoize callback
const handleClick = useCallback(
  () => setCount(c => c + 1),
  []
);
```

useContext

```
import { useContext } from "react";
const value = useContext(MyContext);
```

自定义 Hook

```
function useLocalStorage(key, initial) {
  const [value, setValue] = useState(() => {
    const saved = localStorage.getItem(key);
    return saved ? JSON.parse(saved) : initial;
  });

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}

// Usage
const [name, setName] = useLocalStorage("name", "");
```

自定义 Hook 必须以 'use' 开头

Context API

创建与提供

```
import { createContext, useContext } from "react";

const ThemeCtx = createContext("light");

function App() {
  return (
    <ThemeCtx.Provider value="dark">
      <Page />
    </ThemeCtx.Provider>
  );
}
```

消费

```
function Page() {
  const theme = useContext(ThemeCtx); // "dark"
  return <div className={theme}>...</div>;
}
```