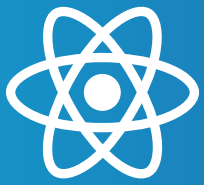
REACT **HOOKS** **CHEAT** SHEETS

Xebia × publicis sapient



SUMMARY

#1 | Manage state

useState

```
const [count, setCount] = useState(initialCount);
```

useReducer

```
const [state, dispatch] = useReducer(
  reducer,
  initialState,
  initialDispatch
);
```

#2 | Handle side effects

useEffect

```
useEffect(() => {
  applyEffect(dependencies);
  return () => cleanupEffect();
}, [dependencies]);
```

useLayoutEffect

```
useLayoutEffect(() => {
  applyBlockingEffect(dependencies);
  return cleanupEffect();
}, [dependencies]);
```

#3 | Use the Context API

useContext

```
const ThemeContext = React.createContext();
const contextValue = useContext(ThemeContext);
```

#4 | Memoize everything

useMemo

```
const memoizedValue = useMemo(
  () => expensiveFn(dependencies),
  [dependencies]
);
```

useCallback

```
const memoizedCallback = useCallback(
  expensiveFn(dependencies),
  [dependencies]
);
```

#5 | Use refs

useRef

```
const ref = useRef();
```

useImperativeHandle

```
useImperativeHandle(
  ref,
  createHandle,
  [dependencies]
)
```

#6 | Reusability

Extract reusable behaviour into custom hooks

#1 | Manage state

useState

Class way

```
const initialCount = 0;

class Counter extends Component {
  constructor(props) {
    super(props);
    this.state = { count: initialCount };
  }

  render() {
    return (
      <div>
        <p>You clicked {this.state.count} times</p>
        <button
            onClick={() => this.setState(({count}) => ({ count: count + 1 }))}
        >
          Click me
        </button>
      </div>
    );
  }
}
```

Hook way

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

const initialCount = 0;
function Counter() {
  const [count, setCount] = useState(initialCount);

  return (
    <div>
      <p>You clicked {count} times</p>
      <button
        onClick={() => setCount((c) => c + 1)}
      >
        Click me
      </button>
    </div>
  );
}
```

#1 | Manage state

useReducer

An alternative to useState. Use it in the components that need complex state management, such as multiple state values being updated by multiple methods

```
const initialState = {count: 0};

function reducer(state, action) {
  switch (action.type) {
    case 'increment':
      return {count: state.count + 1};
    case 'decrement':
      return {count: state.count - 1};
    default
      throw new Error();
  }
}

function Counter({initialState}) {
  const [state, dispatch] = useReducer(reducer, initialState);
  return (
    <>
      Count: {state.count}
      <button onClick={() => dispatch({type: 'increment'})}>+</button>
      <button onClick={() => dispatch({type: 'decrement'})}>+</button>
    </>
  );
}
```

#2 | Handle side effects

useEffect

Class way

```
class FriendStatus extends React.Component {

  state = { isOnline: null };

  componentDidMount() {
    ChatAPI.subscribeToFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }

  componentWillUnmount() {
    ChatAPI.unsubscribeFromFriendStatus(
      this.props.friend.id,
      this.handleStatusChange
    );
  }

  componentDidUpdate(prevProps) {
    if(this.props.friend.id !== prevProps.id) {
      ChatAPI.unsubscribeFromFriendStatus(
        prevProps.friend.id,
        this.handleStatusChange
      );
      ChatAPI.subscribeToFriendStatus(
        this.props.friend.id,
        this.handleStatusChange
      );
    }
  }

  handleStatusChange = status => {
    this.setState({
      isOnline: status.isOnline
    });
  }

  render() {
    if (this.state.isOnline === null) {
      return 'Loading...';
    }
    return this.state.isOnline ? 'Online' : 'Offline';
  }
}
```

#2 | Handle side effects

Hooks way

Class way

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

function FriendStatus(props) {
 const [isOnline, setIsOnline] = useState(null); // Store friend's online
status

 const handleStatusChange = (status) => setIsOnline(status.isOnline); // Set
up a status change handler

 useEffect(
    () => {
      // Update status when the listener triggers
      ChatAPI.subscribeToFriendStatus(
        props.friend.id,
        handleStatusChange
      );

     // Stop listening to status changes every time we cleanup
     return function cleanup() {
       ChatAPI.unsubscribeFromFriendStatus(
          props.friend.id,
          handleStatusChange
        );
      };
    },
    [props.friend.id] // Cleanup when friend id changes or when we "unmount"
 );

 if (isOnline === null) {
   return 'Loading...';
 }
 return isOnline ? 'Online' : 'Offline';
}
```

#2 | Handle side effects

useLayoutEffect

Almost the same as useEffect, but fires synchronously after the render phase. Use this to safely read from or write to the DOM

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

function ColoredComponent({color}) {
  const ref = useRef();

  useLayoutEffect(() => {
    const refColor = ref.current.style.color;
    console.log(`${refColor} will always be the same as ${color}`);
    ref.current.style.color = "rgba(255,0,0)";
  }, [color]);

  useEffect(() => {
    const refColor = ref.current.style.color;
    console.log(
      `but ${refColor} can be different from ${color} if you play with the
DOM`
    );
  }, [color]);

  return (
    <div ref={ref} style={{ color: color }}>
      Hello React hooks !
    </div>
  );
}
```

```
<ColoredComponent color={"rgb(42, 13, 37)"} />
// rgb(42, 13, 37) will always be the same as rgb(42, 13, 37)
// but rgb(255, 0, 0) can be different from rgb(42, 13, 37) if you play with
the DOM
```

#3 | Use the Context API

useContext

Class way

```
class Header extends React.Component {
  public render() {
    return (
      <AuthContext.Consumer>
        {({ handleLogin, isLoggedIn}) => (
          <ModalContext.Consumer>
            {({ isOpen, showModal, hideModal }) => (
              <NotificationContext.Consumer>
                {({ notification, notify }) => {
                  return (
                    ...
                  )
                }}
              </NotificationContext.Consumer>
            )}
          </ModalContext.Consumer>
        )}
      </AuthContext.Consumer>
    );
  }
}
```

Hook way

```
import { useContext } from 'react';

function Header() {

  const { handleLogin, isLoggedIn} = useContext(AuthContext); //
  const { isOpen, showModal, hideModal } = useContext(ModalContext);
  const { notification, notify } = useContext(NotificationContext);

  return ...
}
```

⚠️ Use the created context, not the consumer ⚠️

```
const Context = React.createContext(defaultValue);

// Wrong
const value = useContext(Context.Consumer);

// Right
const value = useContext(Context);
```

#4 | Memoize everything

useMemo

```
function RandomColoredLetter(props) {
  const [color, setColor] = useState('#fff')
  const [letter, setLetter] = useState('a')
  const handleColorChange = useMemo(() => () => setColor(randomColor()), []);
  const handleLetterChange = useMemo(() => () => setLetter(randomColor()), []);
  return (
    <div>
      <ColorPicker handleChange={handleColorChange} color={color} />
      <LetterPicker handleChange={handleLetterChange} letter={letter} />
      <hr />
      <h1 style={{color}}>{letter}</h1>
    </div>
  )
}
```

useCallback

```
function RandomColoredLetter(props) {
  const [color, setColor] = useState('#fff')
  const [letter, setLetter] = useState('a')
  const handleColorChange = useCallback(() => setColor(randomColor()), []);
  const handleLetterChange = useCallback(() => setLetter(randomColor()), []);
  return (
    <div>
      <ColorPicker handleChange={handleColorChange} color={color} />
      <LetterPicker handleChange={handleLetterChange} letter={letter} />
      <hr />
      <h1 style={{color}}>{letter}</h1>
    </div>
  )
}
```

#5 | Use refs

useRef

useRef can just be used as a common React ref :

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

function TextInput() {
  const inputRef = useRef(null);
  const onBtnClick = () => inputRef.current.focus();

  return (
    <>
      <input ref={ref} />
      <button onClick={onBtnClick}>Focus the text input</button>
    </>
  )
}
```

But it also allows you to just hold a mutable value through any render. Also, mutating the value of ref.current will not cause any render

#5 | Use refs

useImperativeHandle

Allows you to customize the exposed interface of a component when using a ref. The following component will automatically focus the child input when mounted :

```
function TextInput(props, ref) {
  const inputRef = useRef(null);
  const onBtnClick = () => inputRef.current.focus();

  useImperativeHandle(ref, () => ({
    focusInput: () => inputRef.current.focus();
  }));

  return (
    <Fragment>
      <input ref={inputRef} />
      <button onClick={onBtnClick}>Focus the text input</button>
    </Fragment>
  )
}

const TextInputWithRef = React.forwardRef(TextInput);

function Parent() {
  const ref = useRef(null);

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

  return (
    <div>
      <TextInputWithRef ref={ref} />
    </div>
  );
}
```

#6 | Reusability

Extract reusable behaviour into custom hooks

```
import { useState, useRef, useCallback, useEffect } from "React";

// let's hide the complexity of listening to hover changes
function useHover() {
  const [value, setValue] = useState(false); // store the hovered state
  const ref = useRef(null); // expose a ref to listen to

  // memoize function calls
  const handleMouseOver = useCallback(() => setValue(true), []);
  const handleMouseOut = useCallback(() => setValue(false), []);

  // add listeners inside an effect,
  // and listen for ref changes to apply the effect again
  useEffect(() => {
    const node = ref.current;
    if (node) {
      node.addEventListener("mouseover", handleMouseOver);
      node.addEventListener("mouseout", handleMouseOut);

      return () => {
        node.removeEventListener("mouseover", handleMouseOver);
        node.removeEventListener("mouseout", handleMouseOut);
      };
    }
  }, [ref.current]);

  // return the pair of the exposed ref and it's hovered state
  return [ref, value];
}
```

```
const HoverableComponent = () => {
  const [ref, isHovered] = useHover();
  return (
    <span style={{ color: isHovered ? "blue" : "red" }} ref={ref}>
      Hello React hooks !
    </span>
  );
};
```