<WA1/>
<AW1/>
2021

Hooks

Supercharge function components

Fulvio Corno
Luigi De Russis
Enrico Masala

Outline

- Hooks: Why and What?
- Main hooks
  - `useState`
  - `useEffect`
  - `useContext`
- The “Rules” of Hooks
- Custom Hooks

Outline

Part 1

- Hooks: Why and What?
- Main hooks
  - `useState`
  - `useEffect`
  - `useContext`
- The “Rules” of Hooks
- Custom Hooks

Full Stack React, “Appendix C: React Hooks”

React Handbook, chapter “Hooks”

https://reactjs.org/docs/hooks-intro.html

Why? and What?

HOOKS

Classes vs. Functions (2nd Act)

Function component

- Simple
- Pure function (props->render)
- No state
- No side effects
- No lifecycle
- May define handler functions (not very useful, in absence of state)

Class component

- More complex
- ‘bind’ issues
- May have state
- Has lifecycle methods
- May define handler functions
- No side effects in render()
- Side effects in handlers and lifecycle

Classes vs. Functions (2nd Act)

Function component

- Simple
- Pure function (props->render)
- No state
- No side effects
- No lifecycle
- May define handler functions (not very useful, in absence of state)

Class component

- More complex
- ‘bind’ issues
- May have state
- Has lifecycle methods
- May define handler functions
- No side effects in render()
- Side effects in handlers and lifecycle

Hooks

- Proposed in October 2018 – https://youtu.be/dpw9EHDh2bM
  - Stable since React 16.8 (February 2019)

- Additions to function components to access advanced features
  - Special mechanism for overcoming some limitations of “pure” functions, in a controlled way
  - Managing state, accessing external resources, having side-effects, …
- One *hook* call for each requested functionality
  - Hooks = special functions called by function components

Most popular Hooks

Hook	Purpose
`useState`	Define a state variable in the component
`useEffect`	Define a side-effect during the component lifecycle
`useContext`	Act as a context consumer for the current component
`useReducer`	Alternative to useState for Redux-like architectures or complex state logic
`useMemo`	“Memoizes” a value (stores the result of a function and recomputes it only if parameters change)
`useCallback`	Creates a callback function whose value is memoized
`useRef`	Access to childrens’ ref properties
`useLayoutEffect`	Like useEffect, but runs after DOM mutations
`useDebugValue`	Shows a value in the React Developer Tools

https://reactjs.org/docs/hooks-reference.html

Full Stack React, “Appendix C: React Hooks”

React Handbook, chapter “Hooks”

https://reactjs.org/docs/hooks-state.html

Providing function components with a state

USESTATE HOOK

useState

- Creates a new state variable
  - Usually, a “simple” value
  - May be an object
  - Does not need to represent the whole complete component state
- You may access
  - The current value
  - A function to update the state value
- Update
  - With the new value
  - With a callback function

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

function ShortText(props) {
 const [hidden, setHidden] = useState(true);
 return (
    <span>
      {hidden ?
        `${props.text.substr(0, props.
maxLength)}...` : props.text }
      {hidden ? (
        <a onClick={() => setHidden(false)}>more</a>
      ) : (
        <a onClick={() => setHidden(true)}>less</a>
      )}
    </span>
  );
}
```

https://daveceddia.com/usestate-hook-examples/

Creating a state variable

- `import{ useState } from 'react';`
- `const [hidden, setHidden] = useState(true);`
  - Creates a new state variable
  - `hidden`: name of the variable
  - `setHidden`: update function
  - `true`: default (initial) value
  - Array destructuring assignment to assign 2 values at once

- Creates a state variable of any type
  - Remembered across function calls!
- The default value sets the initial type and value
- The variable name can be used inside the function (to affect rendering)
- The `setVariable()` function will replace the current state with the new one
  - And trigger a re-render

Updating the state

- With a new value
  - Dependent on props and constant values
  - Will **replace** the current one
  - Should have the same type (for consistent rendering)

```
setHidden(false) ;
```

- With a function

  `(oldstate) => { return newstate; }`
  - Executed as a callback
  - When *new state depends on old state*
  - The function return value will **replace** the current state

```
setSteps(oldSteps => oldSteps + 1);
```

Updating the state

- With a new value
  - Dependent values
  - Will **replac**
  - Should hav consistent

If the new state depends on the current state, we **must** provide a callback, otherwise updates may be lost.

✕ `setSteps(steps+1)` ✕

```
setHidden(false) ;
```

- With a function

  `(oldstate) => { return newstate; }`

  - Executed as a callback
  - When *new state depends on old state*
  - The function return value will **replace** the current state

```
setSteps(oldSteps => oldSteps + 1);
```

The default value

- Used during the first render of the component
  - Never used in successive renders
- May be computed from the props
  - But will not update if the props change
- May be a value, or a function
  - The function is called only during the initial render

Example

```
function Counter(props) {
  const [count, setCount] = useState(props.initialCount);
  return (
    <>
      Count: {count}
      <button onClick={() => setCount(props.initialCount)}>Reset</button>
      <button onClick={() => setCount(prevCount => prevCount - 1)}>-</button>
      <button onClick={() => setCount(prevCount => prevCount + 1)}>+</button>
    </>
  );
}
```

Multiple State Variables

- Do not use a single object for holding many (unrelated) properties
- Create as many state variables as needed, they are all independent
- Component will re-render if any state changes
- Children components will re-render only if their props change

```
function Example(props) {

  [hidden, setHidden] = useState(true) ;
  [count, setCount] = useState(0) ;
  [mode, setMode] = useState('view') ;

  . . .

  setHidden(false) ;
  . . .
  setCount( c => c+1 ) ;
  . . .
  setMode('edit') ;
  . . .

}
```