<WA1/>

2020

Hooks

Supercharge function components

Enrico Masala
Fulvio Corno
Luigi De Russis

Outline

- 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 lifecycle
- May define handler functions (not very useful, in absence of state)
- No side effects

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 lifecycle
- May define handler functions (not very useful, in absence of state)
- No side effects

Class component

- More complex
- 'bind' issues
- May have state
- Has lifecycle methods

Can we retain function components' simplicity and add the possibility of managing state, lifecycle and side effects?

Reusing stateful logic

- If you need to share the same state-management logic in two different (class) components, you have to repeat it twice
  - E.g., keeping the state of an <input> up to date
- Current solutions are: higher-order components or render props, but they add significant complexity

Can we find a way to “encapsulate” some functionality, in a way that it’s easy to import in different components?

Lifecycle methods are confusing

- Component lifecycle may be difficult to understand
- The code related to a functionality is usually *split across* several methods
  - E.g., set a state in `constructor`, fetch the updated value in `componentDidMount`, remove subscriptions in `componentWillUnmount`
  - Each method contains a mix of different concerns
  - Each functionality is spread in different parts of the code

Can we keep the code related to a single functionality in a single place?

Hooks

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

- Additions to function components to access advanced features
- One hook call for each requested functionality
- In many cases, they replace class components
- Easy to extend and customize
- 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 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’;
import ReactDOM from 'react-dom’;

function ShortText({ text, maxLength }) {
 const [hidden, setHidden] = useState(true);
 return (
    <span>
      {hidden ?
        `${text.substr(0, maxLength)}...` : 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(prevState => prevState + 1);
```

Updating the state

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

Replace, not **merge**

⚠ Different from this.setState() ⚠

- With a function
) => { return newstate; }
  ed as a callback
  new state depends on old state
  - The function return value will **replace** the current state

```
setHidden(false) ;
```

```
setSteps(prevState => prevState + 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

Class *vs* Function+Hooks

```
function Example() {

  // Declare a new state variable, called "count"
  const [count, setCount] = useState(0);




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

```
class Example extends React.Component {

  constructor(props) {
    super(props);
    this.state = {
      count: 0
    };
  }

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

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
- No state merging is needed
- 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') ;
  . . .

}
```

Full Stack React, “Appendix C: React Hooks”

React Handbook, chapter “Hooks”

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

Simple Way to Create a Context Consumer

USECONTEXT HOOK

Remember Context API?

- Create a new context:

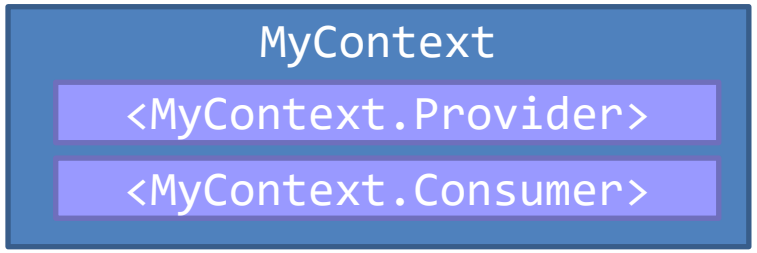
```
const NumberContext = React.createContext();
```

- Define a Context Provider component

```
<NumberContext.Provider value={42}>
  <Display />
</NumberContext.Provider>
```

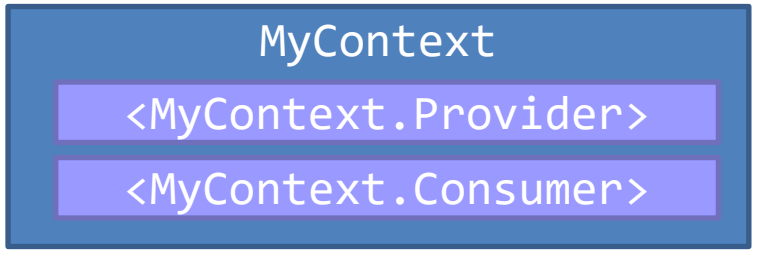
- Define one or more Context Consumer components

```
<NumberContext.Consumer>
  {value => <div>The answer is {value}.</div>}
</NumberContext.Consumer>
```



Accessing context with Hooks

- The `useContext` hook allows the current component to *consume* the context
- The argument is a Context object
  - Must have been created by `React.createContext();`
- The value depends on the closest enclosing provider
  - Must be nested inside `<MyContext.Provider>`

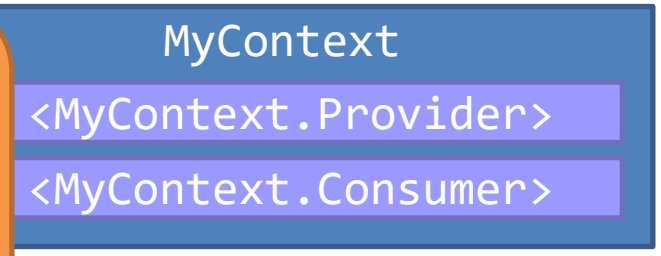



```
function Display() {

  const value = useContext(NumberContext);

  return <div>The answer is {value}.</div>;
}
```

Accessing context with Hooks

- The `useContext` hook allows the current compo… *consume* the conte…
- The argument is a C…
  - Must have been cre… `React.createContext();`
- The value depends on the closest enclosing provider
  - Must be nested inside `<MyContext.Provider>`

There is no way to create a **new** context, or to create a context **provider**, with Hooks




```
…splay() {

  const value = useContext(NumberContext);

  return <div>The answer is {value}.</div>;
}
```

Accessing multiple contexts

- May call `useContext` more than once
- All the context variables will be available
- No need to nest components

```
function HeaderBar() {
  const user = useContext(CurrentUser);
  const notif = useContext(Notifications);

  return (
    <header>
      Welcome back, {user.name}!
      You have {notif.length} notifications.
    </header>
  );
}
```

https://daveceddia.com/usecontext-hook/

Accessing multiple contexts

```
function HeaderBar() {
  return (
    <CurrentUser.Consumer>
      {user =>
        <Notifications.Consumer>
          {notif =>
            <header>
              Welcome back, {user.name}!
              You have {notif.length}
              notifications.
            </header>
          }
        </Notifications.Consumer>
      }
    </CurrentUser.Consumer>
  );
}
```

Without Hooks

```
function HeaderBar() {
  const user = useContext(CurrentUser);
  const notif = useContext(Notifications);

  return (
    <header>
      Welcome back, {user.name}!
      You have {notif.length} notifications.
    </header>
  );
}
```

With Hooks

Full Stack React, “Appendix C: React Hooks”

React Handbook, chapter “Hooks”

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

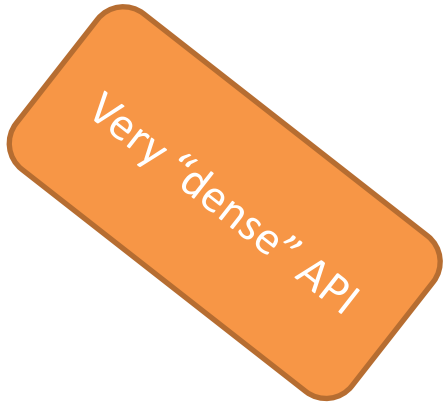
Side-effects and Life Cycle in Functional Components

USEEFFECT HOOK

Side effects in function components

- Examples of desired side effects
  - **Data fetching**
  - Setting up a subscriptions (handlers, etc.), or removing them
  - Manually changing the DOM in React components
  - …
- Think of the `useEffect` Hook as sort of "combined":
  - **componentDidMount**
  - componentDidUpdate
  - componentWillUnmount

How to `useEffect`



- `useEffect(fn, [])`
- Pass a function `fn` to do the work
  - May have side effects (change state, call external URLs, …)
  - Function returns:
    - Nothing
    - A function, that will be called before the component leaves the screen, to do cleanup (e.g., of subscription)
  - Unlike componentDidMount and componentDidUpdate, `fn` fires **after** layout and paint, during a deferred event
- Pass an optional array with the list of variables to monitor to determine if `fn` must be called or not
  - Can be omitted (default): `fn` will be called after every completed render

Four ways to call `useEffect`

- Once, when component mounts
  - `useEffect( ()=>callOnce(), [] )`
- On every component render
  - `useEffect( ()=>callEveryRender() )`
- On every component render, if some values changed
  - `useEffect( ()=>callIfAnyDepChange(dep1,dep2), [dep1,dep2])`
- When component unmounts
  -
```
useEffect( ()=>{w.addListener();
                return ()=>w.removeListener();}, [])
```

https://dev.to/spukas/4-ways-to-useeffect-pf6

When are Effects executed?

- The useEffect function fires after layout and paint (after the render is committed to the screen), during a deferred event
  - Non blocking behavior
  - But before the next Render phase
- The clean-up function runs before the component is removed from the UI
  - Additionally, if a component renders multiple times, the **previous effect is cleaned up before executing the next effect**
  - To avoid repeated cleanup, ensure you specify the dependency array

`useEffect` optional array caveats

- Make sure the array includes **all** values from the component scope (such as props and state) that change over time and that are used by the effect. Otherwise, your code will reference stale values from previous renders
  - every value *referenced inside the effect* function should also appear in the dependencies array
- If the array includes variables that always change when executing the effect, you risk having an infinite loop

Tips about useEffect array

- Do not use [] just because you are lazy: it is a common source of bugs when some values are actually used
- If needed, other strategies (useReducer and useCallback Hooks) can remove the need for a dependency instead of incorrectly omitting it
- Do I need to specify functions as effect dependencies or not?
  - hoist functions that don't need props or state outside of your component
  - pull the ones that are used only by an effect inside of that effect
  - otherwise, wrap them into useCallback hook where they're defined
  - important since functions can "see" values from props and state

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

Example: data fetch

```
function App() {
  const [data, setData] = useState({ hits: [] });
  const [url, setUrl] = useState('https://hn.algolia.com/api/v1/search?query=mysearchterm');
  const [isLoading, setIsLoading] = useState(false);

  useEffect(() => {
    const fetchData = async () => {
      setIsLoading(true);
      const result = await axios(url);  // fetch "equivalent"
      setData(result.data);
      setIsLoading(false);
    };
    fetchData();
  }, [url]);              // When setUrl is called, data will be fetched
```

https://www.robinwieruch.de/react-hooks-fetch-data

Using useEffect correctly is difficult

- How do I replicate componentDidMount with useEffect?
- How do I correctly fetch data inside useEffect? What is []?
- Do I need to specify functions as effect dependencies or not?
- Why do I sometimes get an infinite refetching loop?
- Why do I sometimes get an old state or prop value inside my effect?

- Also, React is being extended to better support the async data fetching case (Suspense for Data Fetching etc.)

https://overreacted.io/a-complete-guide-to-useeffect/

Full Stack React, “Appendix C: React Hooks”

React Handbook, chapter “Hooks”

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

Peeking Under the Hood

THE RULES OF HOOKS

Quiz

- What is the “magic” behing useState?
- How can the same function return different state variables?
- How can the values be persisted across function calls?

```
function Example(props) {

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

  . . .

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

}
```

Answer

- React associates to each functional component an array of Hook “slots”
  - Slots are stored with the function, therefore they are persistent
- Each time you call a Hook, a new “slot” is used
  - The first time, it’s created
  - The other times, it’s reused

```
function Example(props) {

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

  . . .

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

}
```

Corollary

- React must “know” which functions may host Hooks
- Hooks must always be called in the same order each time a component renders

```
function Example(props) {

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

  . . .

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

}
```

Hook usage rules

- Only Call Hooks at the Top Level
  - Always call Hooks at the top level of your React function
  - Don’t call Hooks inside loops, conditions, or nested functions
- Only Call Hooks from React Functions
  - Don’t call Hooks from regular JavaScript functions
  - You may call Hooks from React function components
  - You may call Hooks from custom Hooks

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

Example

```
function Form() {
  // 1. Use the name state variable
  const [name, setName] = useState('Mary');

  // 2. Use an effect for persisting the form
  useEffect(function persistForm() {
    localStorage.setItem('formData', name);
  });

  // 3. Use the surname state variable
  const [surname, setSurname] = useState('Poppins');

  // 4. Use an effect for updating the title
  useEffect(function updateTitle() {
    document.title = name + ' ' + surname;
  });

  // ...
}
```

```
// ------------
// First render
// ------------
useState('Mary')
  // 1. Initialize the name state variable with 'Mary'
useEffect(persistForm)
  // 2. Add an effect for persisting the form
useState('Poppins')
  // 3. Initialize the surname state variable with 'Poppins'
useEffect(updateTitle)
  // 4. Add an effect for updating the title

// -------------
// Second render
// -------------
useState('Mary')
  // 1. Read the name state variable (argument is ignored)
useEffect(persistForm)
  // 2. Replace the effect for persisting the form
useState('Poppins')
  // 3. Read the surname state variable (argument is ignored)
useEffect(updateTitle)
  // 4. Replace the effect for updating the title
```

https://reactjs.org/docs/hooks-rules.html#explanation

Full Stack React, “Appendix C: React Hooks”

React Handbook, chapter “Hooks”

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

Extending the reach of Hooks-based programming patterns

CUSTOM HOOKS

Reusable application logic

- Traditional React: share stateful logic between components via render props and/or higher-order components
- Building your own Hooks lets you extract component logic into reusable functions without forcing you to add more components to the tree
  - Extract shared logic to a third function
- A custom Hook is a JavaScript function whose name starts with "**use**" and that may call other Hooks
- You can decide what it takes as arguments, and what, if anything, it should return

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

Reusable application logic

- Every time you use a custom Hook, all state and effects inside of it are fully isolated
  - Two components using the same Hook do NOT share state
- However, since Hooks are functions, we can pass information between them
  - For instance, pass the value returned by a setState as a parameter to another Hook

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

Custom Hooks: custom fetch application logic

```
const useDataApi = (initialUrl, initialData) => {
  const [data, setData] = useState(initialData);
  const [url, setUrl] = useState(initialUrl);
  const [isLoading, setIsLoading] = useState(false);
  const [isError, setIsError] = useState(false);

  useEffect(() => {
    const fetchData = async () => {
      setIsError(false);
      setIsLoading(true);
      try {
        const result = await axios(url);  // fetch "equivalent"
        setData(result.data);
      } catch (error) {
        setIsError(true);
      }
      setIsLoading(false);
    };
    fetchData();
  }, [url]);
  return [{ data, isLoading, isError }, setUrl];
};
```

```
function App() {
  const [query, setQuery] = useState('mysquerystring');
  const [{ data, isLoading, isError }, doFetch] = useDataApi(
    'https://hn.algolia.com/api/v1/search?query=myquerystring',
    { hits: [] },
  );

  return (
    <Fragment>
      <form
        onSubmit={event => {
          doFetch(
           `http://hn.algolia.com/api/v1/search?query=${query}`,
          );
          event.preventDefault();
        }}
      >
    ...
```

https://www.robinwieruch.de/react-hooks-fetch-data

Hooks are quickly expanding

A LOOK AT THE FUTURE

Latest additions

- Many common problems are being addressed in a more standardized way
- Support for Hooks is entering in many libraries: Hooks for react router
  - useHistory, useLocation, useParams, useRouteMatch
  - https://reacttraining.com/react-router/web/api/Hooks
- Automatically handle suspension of rendering for data fetching and display a loading indicator: <React.Suspense fallback={<Spinner />}>
  - https://reactjs.org/blog/2018/11/27/react-16-roadmap.html#react-16x-mid-2019-the-one-with-suspense-for-data-fetching
- …

References

- 4 Examples of the useState Hook, https://daveceddia.com/usestate-hook-examples/
- How the useContext Hook Works, https://daveceddia.com/usecontext-hook/
- 4 Ways to useEffect(), https://dev.to/spukas/4-ways-to-useeffect-pf6
- How the useEffect Hook Works, https://daveceddia.com/useeffect-hook-examples/
- How to fetch data with React Hooks?, https://www.robinwieruch.de/react-hooks-fetch-data

License

- These slides are distributed under a Creative Commons license “**Attribution-NonCommercial-ShareAlike 4.0 International (CC BY-NC-SA 4.0)**”
- **You are free to:**
  - **Share** — copy and redistribute the material in any medium or format
  - **Adapt** — remix, transform, and build upon the material
  - The licensor cannot revoke these freedoms as long as you follow the license terms.
- **Under the following terms:**
  - **Attribution** — You must give appropriate credit, provide a link to the license, and indicate if changes were made. You may do so in any reasonable manner, but not in any way that suggests the licensor endorses you or your use.
  - **NonCommercial** — You may not use the material for commercial purposes.
  - **ShareAlike** — If you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original.
  - **No additional restrictions** — You may not apply legal terms or technological measures that legally restrict others from doing anything the license permits.
- https://creativecommons.org/licenses/by-nc-sa/4.0/