

React: (Side) Effects

Phase 2 // Week 1, Day 4

Sakib Rasul | July 7, 2023

Today's Objectives

Today, we'll answer the following questions:

1. *What's a **pure function**?*
2. *How (and why) should we write **pure components** in React?*
3. *What's a **(side) effect** in React?*
4. *How do we cause **(side) effects** in React?*

Pure Functions

- A **pure function** is one that, given the same input, returns the same output, each and every time.
- Take `function double(x) { return x * 2 }`.
 - If we pass in `x = 2`, `double(x)` will always be 4.
 - If we pass in `x = 32`, `double(x)` will always be 64.
 - ...and so on! Any given `x` will always output the same number.

Pure Components in React

- In React, a **pure component** is one that, given the same props and state, renders the same JSX.
- React assumes that all components are pure.
- Why does React care about purity? 🤔

Pure Components in React

- In React, a **pure component** is one that, given the same props and state, renders the same JSX.
- React assumes that all components are pure.
- Why does React care about purity?
 - It's how React knows *what* to re-render, *when*!

An Impure Component (Don't do this!)

```
let capacity = 0;
```

```
function Venue() {  
  capacity = capacity + 100;  
  return <h1>This venue holds {capacity} people.</h1>  
}
```

```
function Venues() {  
  return (<>  
    <Venue/>  
    <Venue/>  
  </>)  
}
```


An Impure Component (Don't do this!)

```
let capacity = 0;
```

```
function Venue() {  
  capacity = capacity + 100;  
  return <h1>This venue holds {capacity} people.</h1>  
}
```

```
function Venues() {  
  return (  
    <Venue/> { /* This venue holds 200 people. */}  
    <Venue/> { /* This venue holds 400 people. */}  
  )  
}
```

A Pure Component

```
function Venue({ capacity }) {  
  return <h1>This venue holds {capacity} people.</h1>  
}
```

```
function Venues() {  
  return (  
    <Venue capacity={100} />  
    <Venue capacity={200} />  
  )  
}
```


A Pure Component

```
function Venue({ capacity }) {  
  return <h1>This venue holds {capacity} people.</h1>  
}
```

```
function Venues () {  
  return (  
    <Venue capacity={100} /> { /* This venue holds 100 people. */}  
    <Venue capacity={200} /> { /* This venue holds 200 people. */}  
  )  
}
```

Side Effects in React

- In React, a **side effect** happens outside of rendering.
- Since side effects don't run during a render, they can be impure.
- Can you think of an impure side effect? 🤔

Side Effects in React

- In React, a **side effect** happens outside of rendering.
- Since side effects don't run during a render, they can be impure.
- Can you think of an impure side effect?
 - The most common side effect in React is setting state.
 - The most common place to write side effects is in event handlers.
 - ...but not all side effects are triggered by events 🤔

(Side) Effects in React

- A small minority of side effects are triggered by rendering itself.
- In React, these side effects are known as **effects**.
- Can you think of an effect you might want to write in React? 🤔

(Side) Effects in React

- A small minority of side effects are triggered by rendering itself.
- In React, these side effects are known as **effects**.
- Can you think of an effect you might want to write in React? 🤔
 - The most common effect in React is a GET request.
 - We usually want to fetch data after a component's first render.
 - i.e., We want the component's first render to trigger a `fetch()`.

Writing an Effect in React

- To write an effect, follow these steps:
 1. `import { useEffect } from 'react'.`
 2. Invoke `useEffect()` with either one or two parameters.

Writing an Effect in React

- To write an effect, follow these steps:
 1. `import { useEffect } from 'react'.`
 2. Invoke `useEffect()` with either one or two parameters.
 - A. The first parameter should be a function that runs effects.
 - B. The second parameter, if any, should be an array that lists the effect's dependencies.

Writing an Effect in React

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

export default function Component() {

  useEffect(() => {
    /* Write your effect(s) here */
  }, dependencies);

  return <></>;
}
```


The Dependency Array

- An effect's dependencies dictate when that effect will run.
- More specifically, the dependency array lets us choose which of a component's renders will trigger the execution of an effect.
- If we omit the dependency array, our effect runs after every re-render.
- If we pass an empty array `[]`, our effect runs only **on mount**.
- If we pass a non-empty array `[a, b]`, our effect runs on mount and whenever a or b change.
- Can we make any variable a dependency? 🤔

The Dependency Array

- If we omit the dependency array, our effect runs after every re-render.
- If we pass an empty array `[]`, our effect runs only **on mount**.
- If we pass a non-empty array `[a, b]`, our effect runs on mount and whenever a or b change.
- Can we make any variable a dependency?
 - Only props and state make sense as dependencies, because only they can change between renders.
 - More specifically, your dependencies should be any and all props and/or state referenced by your effect.

Fetching Data in React: An Example

```
export default function Museum() {  
  const [galleries, setGalleries] = useState([]);  
  useEffect(/* What goes here? */);  
  return (  
    <>{galleries.map(gallery => <Gallery [...] />)}</>  
  )  
}
```

Fetching Data in React: An Example

```
export default function Museum() {  
  const [galleries, setGalleries] = useState([]);  
  useEffect(() =>  
    fetch("http://museum.com/galleries")  
      .then(response => response.json())  
      .then(data => setGalleries(data))  
  ), []);  
  return <>{galleries.map(gallery => <Gallery [...] />)}</>  
}
```


The “Effect” of an Effect’s Dependency Array

Dependency Array	Outcome
none	Run <u>after every</u> render.
[]	Run <u>only on mount</u> , i.e. after the first render.
[a, b, ...]	Run on mount <u>and</u> whenever a dependency changes.

Other Effects

- In general, effects are used to synchronize one, some, or all of a component's renders with some external system.
- Fetching data on mount synchronizes a component's first render with an external resource (e.g. a JSON server or an API).
- Others include DOM methods, animations, and subscriptions.
- Some effects require cleanup.

and now, a demo



Thanks!

© Sakib Rasul 2023