REACT HOOKS (II)

YT Course by The Net Ninja / June 2019

Mind map by Ana-Maria Dobre

More about useEffect

We can run as many useEffect fct as many times as we want (1)

=> you can run 1 callback when height is changed and the other callback when songs is changed.

=> On initial render, 2 callback fct have run now, one for the songs and one for the height.
// because they both run on initial render, nothing to do with adding new data yet; but if you add a new song, only one of the callbacks will run.
Similarly, if you increment the height, only one of the callbacks will run.

(1)

```
import React, {useEffect, useState} from "react";
import {v4 as uuidv4} from 'uuid'
import NewSongForm from "./NewSongForm";

const SongList = () => {
  const [songs, setSongs] = useState([
    {title: 'Lo que te conté mientras te hacias la dormida', id: 1},
    {title: 'Rosas', id: 2},
    {title: 'Las paz de tus ojos', id: 3}
  ]);

  const [height, setHeight] = useState(3600);
  const addSong=(title) => {
    setSongs([...songs, {title: title, id: uuidv4()}])
  }

  useEffect(()=> {
    console.log('useEffect hook ran', songs);
  }, [songs])

  useEffect(()=> {
    console.log('useEffect hook ran', height);
  }, [height])

  return(
    <div className="song-list">
      <ul>
        {songs.map(song => {
          return(
            <li key={song.id}>{song.title}</li>
          );
        })}
      </ul>
      <NewSongForm addSong={addSong} />
      <button onClick={() => setHeight(height + 1)}>Increment height by 1: {height}</button>
    </div>
  );
}

export default SongList;
```

Practice (2)

Limit when to use callback

We might not want to run the callback fct every time ANY data is updated.
Image we only wanted to run the callback when the songs-related data changes, not the age-related data changes.

=> we can do that by adding a second parameter into the useEffect hook.
That second param is going to be an array of data that we essentially want to watch for change.

(1)

```
import React, {useEffect, useState} from "react";
import {v4 as uuidv4} from 'uuid'
import NewSongForm from "./NewSongForm";

const SongList = () => {
  const [songs, setSongs] = useState([
    {title: 'Lo que te conté mientras te hacias la dormida', id: 1},
    {title: 'Rosas', id: 2},
    {title: 'Las paz de tus ojos', id: 3}
  ]);

  const [height, setHeight] = useState(3600);
  const addSong=(title) => {
    setSongs([...songs, {title: title, id: uuidv4()}])
  }

  useEffect(()=> {
    console.log('useEffect hook ran', songs);
  }, [songs])

  return(
    <div className="song-list">
      <ul>
        {songs.map(song => {
          return(
            <li key={song.id}>{song.title}</li>
          );
        })}
      </ul>
      <NewSongForm addSong={addSong} />
      <button onClick={() => setHeight(height + 1)}>Increment height by 1: {height}</button>
    </div>
  );
}

export default SongList;
```

useEffect HOOK

useEffect (overview)
- like a lifecycle method that we'd normally use inside a React CC
- when we use FC, we don't have access to those lifecycle methods

=> if we wanted to run some code when the component updates, we'd normally need to use a CC and hook into a lifecycle method

useEffect features
- fct
- takes a callback fct as param
- callback will run every time the comp (re)renders (1) i.e. every time the data inside our component changes and also on the initial render
- typically use for: communicating with DB, or an API endpoint, etc

Example (2)
Add a useEffect call to log something onto the console; also output the songs.
How to test:
Enter new song in NewSongForm, then click the "Add song" button

=> useEffect hook runs again, every time the comp renders, or every time the data updates inside the comp (like when adding a new song)

(1)

```
useEffect(() => {})
```

(2)

```
import React, {useEffect, useState} from "react";
import {v4 as uuidv4} from 'uuid'
import NewSongForm from "./NewSongForm";

const SongList = () => {
  const [songs, setSongs] = useState([
    {title: 'Lo que te conté mientras te hacias la dormida', id: 1},
    {title: 'Rosas', id: 2},
    {title: 'Las paz de tus ojos', id: 3}
  ]);

  const addSong=(title) => {
    setSongs([...songs, {title: title, id: uuidv4()}])
  }

  useEffect(()=> {
    console.log('useEffect hook ran', songs);
  })

  return(
    <div className="song-list">
      <ul>
        {songs.map(song => {
          return(
            <li key={song.id}>{song.title}</li>
          );
        })}
      </ul>
      <NewSongForm addSong={addSong} />
    </div>
  );
}

export default SongList;
```

Practice (1)

Create additional state using useState (1)

=> we can use useState as many times as we want inside a comp, storing different values.
// Initialise the height by 3600.
Create a new button, with caption {height}
Add onClick event handler, with an inline arrow fct inside.
// onClick={() => setHeight(height + 1)}

How to test:
Click on "Increment height by 1" button => notice the height on the caption increments, and useEffect hook runs (see console).

(1)

```
import React, {useEffect, useState} from "react";
import {v4 as uuidv4} from 'uuid'
import NewSongForm from "./NewSongForm";

const SongList = () => {
  const [songs, setSongs] = useState([
    {title: 'Lo que te conté mientras te hacias la dormida', id: 1},
    {title: 'Rosas', id: 2},
    {title: 'Las paz de tus ojos', id: 3}
  ]);

  const [height, setHeight] = useState(3600);
  const addSong=(title) => {
    setSongs([...songs, {title: title, id: uuidv4()}])
  }

  useEffect(()=> {
    console.log('useEffect hook ran', songs);
  })

  return(
    <div className="song-list">
      <ul>
        {songs.map(song => {
          return(
            <li key={song.id}>{song.title}</li>
          );
        })}
      </ul>
      <NewSongForm addSong={addSong} />
      <button onClick={() => setHeight(height + 1)}>Increment height by 1: {height}</button>
    </div>
  );
}

export default SongList;
```

Legend

FC: functional components
CC: class components