Lifecycle hooks useEffect

Friday, March 25, 2022 4:52 PM

- Now react function components can manage their lifecycle using useEffect hook
- It provides all necessary functionality with more flexibility

useEffect

- The function takes a callback which is called to manage the life cycle
- It is called in several different formats

1. Unconditonal

```
const Component = (props) =>{

    useEffect(  () => {

        //called after first and every render
        if( book==null){
            setBook(fetchBook(book) )
        }

    });

    return <div>your JSX here </div>;

}
```

- The function is a combo of
  - componentDidMount
  - componentDidUpdate
- Called after
  - first update
  - Every other render

1. First component function will return the JSX
2. useEffect will called
   - Most likely it will cause re-render
3. Be careful. You may end up in infinite loop of useEffect and render
   - Write a conditional logic

2 Use Effect with dependency

```
const Component = (props) =>{

    useEffect(  () => {

        //called after first and every render

        setBook(fetchBook(book) )


    },  [book, props.isbn ]);

    return <div>your JSX here </div>;

}
```

- We pass a second parameter containing a list of dependencies
- This useEffect will be called
  1. after first render (componentDidMount)
  2. Everytime any of the given property changes
     - If none of given property changes it is not called again
     - If book or book.isbn changes this would be called again.
- You can think there is a built-in condition check here

3. useEffect with empty dependency list

- Empty dependency list is not same as having no dependency list

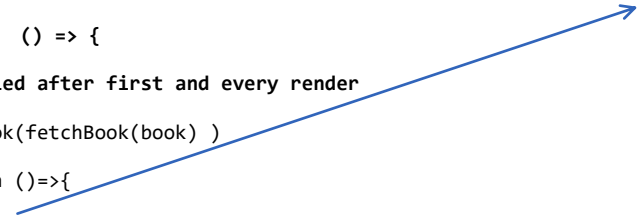
```
const Component = (props) =>{

    useEffect(  () => {

        //called after first and every render

        setBook(fetchBook(book) )


    },  []);

    return <div>your JSX here </div>;

}
```

- Called after first render
  - useEffect are always called first render
- Now it should be called whenever one of the dependency changes
  - Since dependency list is empty, it will never be called again
- It will act as componentDidMount

Cleanup Logic

- The callback can return a function
- If callback returns function it is called for cleanup before next useEffect

```
const Component = (props) =>{

    useEffect(  () => {

        //called after first and every render

        setBook(fetchBook(book) )

        return ()=>{


        }


    },  [ ]);

    return <div>your JSX here </div>;

}
```



- Called to clean up the current effect
- It is called before
  - Next useEffect is called
  - Component is unmounted

- In this case this function acts as componentDidUnmount