

day50-rn-useMemo

☰ 태그	
📅 날짜	@2022년 12월 9일

useMemo는 동일한 연산의 반복 수행을 제거해서 성능을 최적화하는 데 사용한다.

- 기본형태

useMemo(() => {}, []);

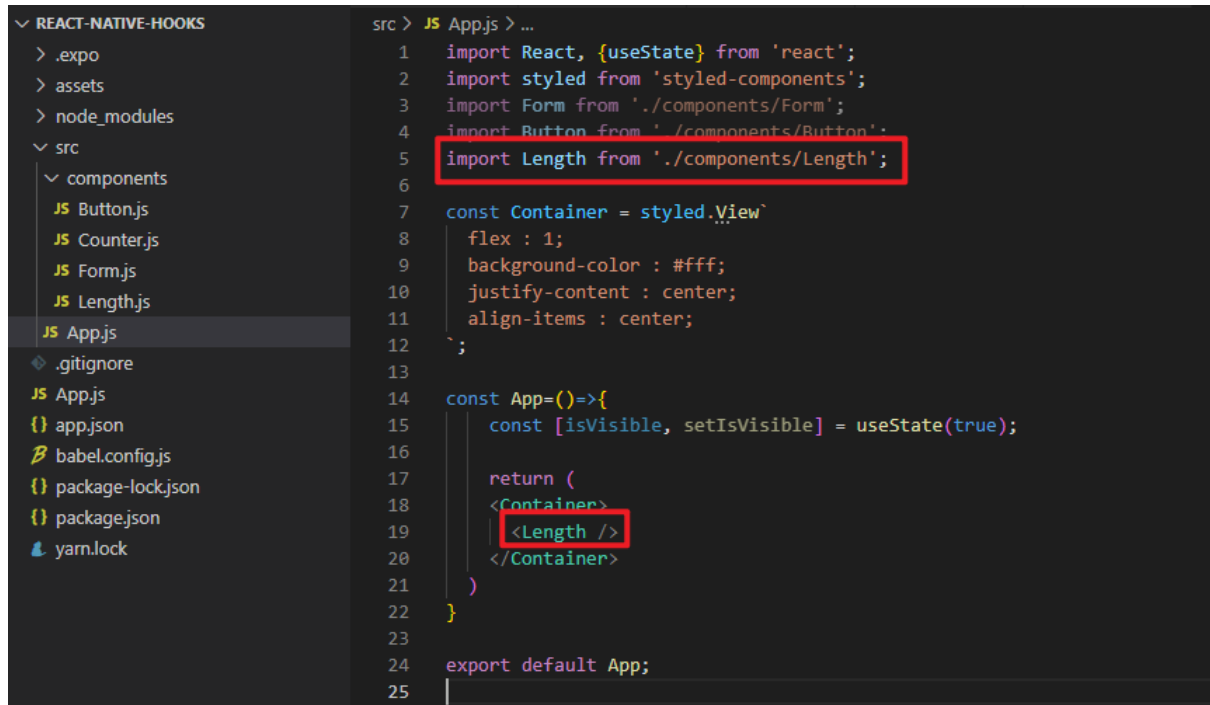
첫 번째 파라미터에는 함수를 전달하고, 두 번째 파라미터에는 함수 실행 조건을 배열로 전달하면 지정된 값에 변화가 있는 경우에만 함수가 호출된다.

먼저 components폴더 밑에 Length.js파일을 생성하고 아래처럼 작성하자.

```
> .expo
> assets
> node_modules
▼ src
  ▼ components
    JS Button.js
    JS Counter.js
    JS Form.js
    JS Length.js
  JS App.js
  .gitignore
  JS App.js
  {} app.json
  babel.config.js
  {} package-lock.json
  {} package.json
  yarn.lock

1  import React, {useState} from "react";
2  import styled from "styled-components/native";
3  import Button from "../Button";
4
5  const StyledText = styled.Text`
6    font-size : 24px;
7  `;
8  const getLength = text => {
9    console.log(`Target Text : ${text}`);
10   return text.length;
11 };
12 const list = ['JavaScript', 'Expo', 'Expo', 'React Native'];
13
14 let idx = 0;
15 const Length = ()=>{
16   const [text, setText] = useState(list[0]);
17   const [length, setLength] = useState('');
18
19   const _onPress = ()=>{
20     setLength(getLength(text));
21     ++idx;
22     if(idx < list.length)
23       setText(list[idx]);
24   };
25   return(
26     <>
27       <StyledText>Text : {text}</StyledText>
28       <StyledText>Length : {length}</StyledText>
29       <Button title="Get Length" onPress={_onPress} />
30     </>
31   );
32 };
33 export default Length;
```

const list. 4칸짜리 배열을 생성하고, 버튼을 클릭할 때마다 배열을 순환하며 문자열의 길이를 구하는 컴포넌트를 작성했다. 이제 App.js에 컴포넌트를 추가해보자.



```
src > JS App.js > ...
1  import React, {useState} from 'react';
2  import styled from 'styled-components';
3  import Form from './components/Form';
4  import Button from './components/Button';
5  import Length from './components/Length';
6
7  const Container = styled.View`
8    flex : 1;
9    background-color : #fff;
10   justify-content : center;
11   align-items : center;
12 `;
13
14  const App=()=>{
15     const [isVisible, setIsVisible] = useState(true);
16
17     return (
18       <Container>
19         <Length />
20       </Container>
21     )
22   }
23
24   export default App;
25
```

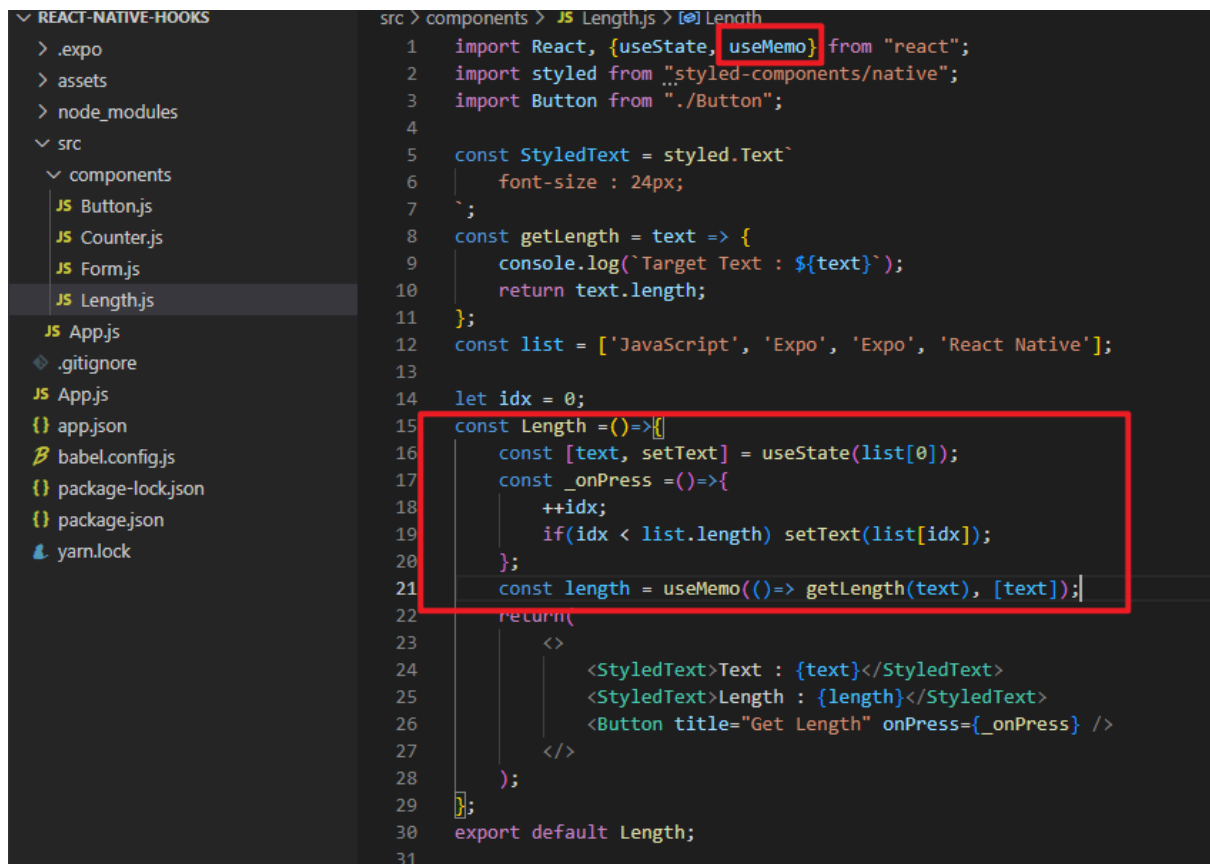


화면에서 버튼을 클릭하면 현재 문자열의 길이를 구하는 것을 확인할 수 있다. 마지막 문자열 이후에는 더 이상 문자열의 변화가 없기 때문에 같은 문자열의 길이를 반복해서 구한다.

```
LOG Target Text : JavaScript
LOG Target Text : Expo
LOG Target Text : Expo
LOG Target Text : React Native
LOG Target Text : React Native
LOG Target Text : React Native
LOG Target Text : React Native
```

특별한 문제 없이 잘 동작하지만 아쉬운 부분이 있다. 두 번째와 세 번째는 expo라는 동일한 문자열의 길이를 계산 했고, 배열의 마지막 값 이후에는 문자열의 변화가 없는데도 계속해서 `getLength` 함수가 호출된다.

이러한 상황에서 `useMemo`를 이용하면 계산하는 값에 변화가 있을 경우에만 함수가 호출되므로 중복되는 불필요한 연산을 제거할 수 있다. `Length` 컴포넌트에 `useMemo`를 사용해서 계산할 값에 변화가 없는 경우 `getLength` 함수가 호출되지 않도록 수정한다.



```
src > components > JS Length.js > Length
1  import React, {useState, useMemo} from "react";
2  import styled from "styled-components/native";
3  import Button from "../Button";
4
5  const StyledText = styled.Text`
6    font-size : 24px;
7  `;
8  const getLength = text => {
9    console.log(`Target Text : ${text}`);
10   return text.length;
11 };
12 const list = ['JavaScript', 'Expo', 'Expo', 'React Native'];
13
14 let idx = 0;
15 const Length = () => {
16   const [text, setText] = useState(list[0]);
17   const _onPress = () => {
18     ++idx;
19     if(idx < list.length) setText(list[idx]);
20   };
21   const length = useMemo(() => getLength(text), [text]);
22   return (
23     <>
24       <StyledText>Text : {text}</StyledText>
25       <StyledText>Length : {length}</StyledText>
26       <Button title="Get Length" onPress={_onPress} />
27     </>
28   );
29 };
30 export default Length;
```

useMemo를 사용해 text의 값이 변경되었을 때만 text길이를 구하도록 수정했다.

결과를 보면 동일한 문자열인 Expo를 다시 계산하지 않고, 배열의 마지막 값 이후에는 문자열의 변화가 없기 때문에 더 이상 getLength 함수를 호출하지 않는다는 것을 확인할 수 있다.

```
LOG Target Text : JavaScript
LOG Target Text : Expo
LOG Target Text : React Native
```

useMemo를 이용하면 이와 같이 특정 값에 변화가 있는 경우에만 함수를 실행하고, 값이 변하지 않으면 이전에 연산했던 결과를 이용해 중복된 연산을 방지함으로써 성능을 최적화 할 수 있다.