React.Memo

리액트와 Memoization

Memoization

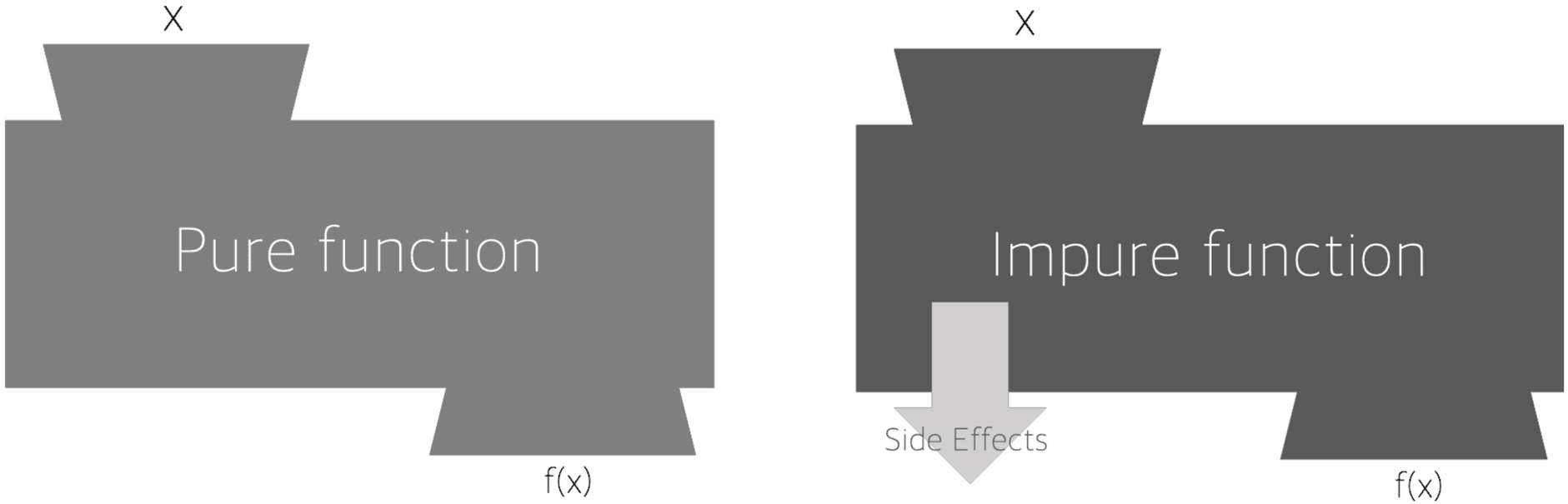
```
let memo = [0, 1];

function fibonacci(n) {
    if (memo[n] !== undefined) return memo[n];
    else {
        memo[n] = fibonacci(n - 1) + fibonacci(n - 2);
        return memo[n];
    }
}
```

이전에 계산한 값을 메모리에 저장해

동일한 계산의 반복 수행을 방지하는 기술

순수 함수



동일한 입력에 대해 **동일한 출력**

Side effect 가 존재해서는 안된다

Side effect

```
let sum = 0;

const add = (n) => {
  sum += n;
  return sum;
};

console.log(add(3)); // add(3) => 3
console.log(add(3)); // add(3) => 6
```

```
let sum = 0;

const add = (n) => {
  sum += n;
  return n;
};

const now = () => {
  return sum;
}

console.log(add(3)); // add(3) => 3
console.log(now()); // 3
console.log(add(3)); // add(3) => 3
console.log(now()); // 6
```

외부 상태를 수정하는 행위

전역변수, 파일, 데이터베이스 등

리액트와 순수함수

```jsx
export default function Component({name, age}) {
  return (
    <div>
      <div>{name}</div>
      <div>{age}</div>
    </div>
  )
}
```

컴포넌트가 props에만 의존하고 있어

props가 같으면 같은 화면을 출력하는 컴포넌트

리액트와 순수함수

```jsx
export default function Component({ arr }) {
  const sortedArr = arr.toSorted((a, b) => a - b);

  return (
    <ul>
      {sortedArr.map((value) => (
        <li>{value}</li>
      ))}
    </ul>
  );
}
```

정렬 결과는 같은 배열이 들어왔을 때 항상 같다

하지만 **리렌더링이 될 때마다 새롭게 계산**하게 된다

React.Memo

`memo(Component, arePropsEqual?)`

Wrap a component in `memo` to get a *memoized* version of that component. This memoized version of your component will usually not be re-rendered when its parent component is re-rendered as long as its props have not changed. But React may still re-render it: memoization is a performance optimization, not a guarantee.

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

const SomeComponent = memo(function SomeComponent(props) {
  // ...
});
```

React.Memo

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

const memoComponent = memo(function Component({ arr }) {
  const sortedArr = arr.toSorted((a, b) => a - b);

  return (
    <ul>
      {sortedArr.map((value) => (
        <li>{value}</li>
      ))}
    </ul>
  );
});

export default memoComponent;
```

memo 함수로 감싸주는 것만으로도
Memoization을 수행해준다

memo 함수의 return값으로 나온 값을
컴포넌트 형태로 사용해야한다

React.Memo

props에 객체가 들어오면
어떻게 비교해주는거지?

```
function ParentComponent() {
  const a = { name: 'John' };
  const b = { name: 'John' };

  return (
    <>
      <Component person={a} />
      <Component person={b} />
    </>
  );
}
```

React.Memo의 props 비교방식

```
function ParentComponent() {
  const a = { name: 'John' };
  const b = { name: 'John' };

  return (
    <>
      <Component person={a} />
      <Component person={b} />
    </>
  );
}
```

기본적으로는

Shallow Compare를 사용

a와 b는 메모리 참조 값이

다르기 때문에

다른 값으로 인식한다

React.Memo의 props 비교방식

비교 함수를 지정해줄 수 있다

`memo(Component, arePropsEqual?)`

Wrap a component in `memo` to get a *memoized* version of that component. This memoized version of your component will usually not be re-rendered when its parent component is re-rendered as long as its props have not changed. But React may still re-render it: memoization is a performance optimization, not a guarantee.

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

const SomeComponent = memo(function SomeComponent(props) {
  // ...
});
```

React.Memo의 props 비교방식

```jsx
const MemoComponent = memo(function Component(props) {
  return (
    <div>
      {props.name}
    </div>
  );
}, (prevProps, nextProps) => {prevProps.name === nextProps.name});

function ParentComponent() {
  const [state, setState] = useState({name : 'John', age : 5});

  useEffect(() => {
    setState({...state, age : 7})
  }, [])

  return (
    <>
      <MemoComponent {...state}/>
    </>
  );
}
```

MemoComponent의 props에 들어오는 state의 메모리 참조값은 변경되었다.

하지만 props의 name값이 이전과 같기 때문에 같은 인자가 들어왔다고 판단하고 렌더링을 다시 하지 않는다

React.Memo의 props 비교방식

```
const MemoComponent = memo(function Component(props) {
  return (
    <div>
      {props.name}
    </div>
  );
}, (prevProps, nextProps) => {prevProps.name === nextProps.name});
```

비교 형태가 prev와 next를 비교하는 형태?

↓

memoization을 **직전 상태**에 대해서만 적용할 수 있다

React.Memo가 유용한 경우

```
function ParentComponent() {
  const [state1, setState1] = useState({});
  const [state2, setState2] = useState({});
  const [state3, setState3] = useState({});
  const [state4, setState4] = useState({});

  return (
    <>
      <MemoComponent1 {...state1}/>
      <MemoComponent2 {...state2}/>
      <MemoComponent3 {...state3}/>
      <MemoComponent4 {...state4}/>
    </>
  );
}
```

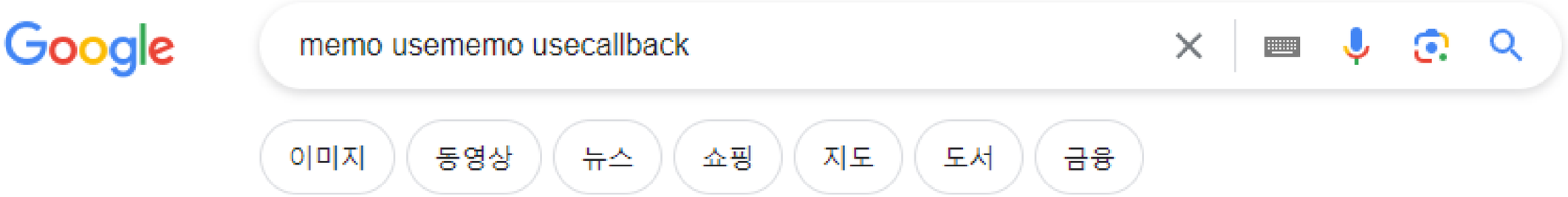
부모 컴포넌트의 상태 중
일부에만 의존적이거나

부모 컴포넌트의 다른
상태변화가 많은 경우

React.Memo 주의사항

```
function MyApp({ store, cookies }) {
  return (
    <div className="main">
      <header>
        <MemoizedLogout
          username={store.username}
          onLogout={() => cookies.clear()}
        />
      </header>
      {store.content}
    </div>
  );
}
```

MyApp이 리렌더링 될 때마다
새로운 callback이 생성되어
props로 넘어가기때문에
memoization이 불가능



velog
https://velog.io › React.memo-useMemo-useCallback-역...

React.memo, useMemo, useCallback 역할 및 차이점

2022. 3. 22. — **useMemo**는 함수의 연산량이 많을때 이전 결과값을 재사용하는 목적이고, **useCallback**은 함수가 재생성 되는것을 방지하기 위한 목적이다.

Developer
https://developer.kro.kr › blog › LKHcoding

React.memo, useMemo, useCallback 역할 및 차이점

2021. 8. 28. — **useMemo**는 함수의 연산량이 많을때 이전 결과값을 재사용하는 목적이고, **useCallback**은 함수가 재생성 되는것을 방지하기 위한 목적이다.

HOC 예시 (Auth에 따른 페이지... · React.memo의 사용법 · useMemo 사용법

Medium
https://medium.com › geekculture › great-confusion-abo...

React.memo, useMemo, useCallback | Geek Culture

2021. 9. 22. — The **useMemo** is used to memoize values, React.**memo** is used to wrap React Function components to prevent re-renderings. The **useCallback** is used to ...

GitHub
https://dawan0111.github.io › react › react---usecallback,...

react - useCallback, memo re-render 최적화

React의 Hook 중에 **useCallback**과 **useMemo**를 활용하여 최적화 하는 방법을 알아보았다. React가 렌더링을 실행하는 행동 Props가 변경되었을 때 State가 변경되었을 때 ...

React.Memo, useMemo, useCallback

세가지에 대해 비교하며 설명하라는

면접 질문도 자주 나오기 때문에

비교하며 공부하는걸 추천드립니다

참고자료

https://d2.naver.com/helloworld/9223303

https://react.dev/reference/react/memo#usage

https://ui.toast.com/weekly-pick/ko_20190731