상태 관리가 rendering에 미치는 영향

1회차 주요내용

1. DOMTree의 node들은 VirtualDOM에서 Fiber로 나타낸다
2. Fiber는 key로 구분된다
    a. array.map() 사용 시 0번째 인덱스에 값을 추가하는 것은 지양한다
    b. 다른 array들 끼리는 key가 중복되어도 괜찮다
3. 불필요한 tag를 추가하는 것을 지양한다

agenda

1. useState vs useReducer
2. useMemo vs useCallback
3. React 18에서 새롭게 소개된 hook들
4. 전역 상태관리 툴 소개 (Context API, Redux, Recoil)
5. [아하!모먼트] 사수 없는 환경에서 성장하기 위한 노력

useState vs useReducer

1. useState
    a. 간단한 상태 관리
        i. 값이 하나인 경우
        ii. 상태들이 서로 관련이 없는 경우
    b. 컴포넌트 내에서 사용
2. useReducer
    a. 복잡한 상태관리
        i. 상태들이 서로 관련이 있거나, 참조가 필요한 경우
        ii. 로그인된 사용자의 권한을 확인해서 다른 화면을 보여준다거나 하는 경우
    b. 여러 컴포넌트에서 상태가 공유되어야 할 때
        i. ContextAPI(useContext) 사용 시 사용하는 것이 일반적
    c. reducer를 따로 선언하는 것이 일반적

useState vs useReducer

```
`const [state, setState] = useState(initialState);`
```

```
`const [state, dispatch] = useReducer(reducer, initialState);`
```

useState

```jsx
import React, { useState } from 'react';

function Counter() {
  const [count, setCount] = useState(0);

  return (
    <div>
      <p>Count: {count}</p>
      <button onClick={() => setCount(count + 1)}>Increment</button>
      <button onClick={() => setCount(count - 1)}>Decrement</button>
    </div>
  );
}
```

```jsx
import React, { useState } from 'react';

function SimpleForm() {
  const [username, setUsername] = useState('');
  const [email, setEmail] = useState('');

  const handleSubmit = (e) => {
    e.preventDefault();
    console.log(`Username: ${username}, Email: ${email}`);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input
        type="text"
        placeholder="Username"
        value={username}
        onChange={(e) => setUsername(e.target.value)}
      />
      <input
        type="email"
        placeholder="Email"
        value={email}
        onChange={(e) => setEmail(e.target.value)}
      />
      <button type="submit">Submit</button>
    </form>
  );
}
```

useState 사용시 주의사항

1. State as a Snapshot
2. 함수형 프로그래밍의 원칙
3. `batch`
4. Update queue
5. 반복문 안에서 상태를 업데이트 하는 경우

useReducer

1. e-commerce에서 제품, 카테고리, 주문 정보, 사용자 정보, 쿼리정보 등을 담아야 할 때

```
const initialState = {
  products: [],
  categories: [],
  orders: [],
  user: null,
  searchQuery: '',
  filters: {},
  sorting: { field: 'name', direction: 'asc' },
  pagination: { currentPage: 1, itemsPerPage: 10 },
  theme: 'light',
  loading: false,
};
```

```
function reducer(state, action) {
  switch (action.type) {
    case 'SET_PRODUCTS':
      return { ...state, products: action.payload };
    case 'SET_CATEGORIES':
      return { ...state, categories: action.payload };
    case 'SET_ORDERS':
      return { ...state, orders: action.payload };
    case 'SET_USER':
      return { ...state, user: action.payload };
    case 'SET_SEARCH_QUERY':
      return { ...state, searchQuery: action.payload };
    case 'SET_FILTERS':
      return { ...state, filters: action.payload };
    case 'SET_SORTING':
      return { ...state, sorting: action.payload };
    case 'SET_PAGINATION':
      return { ...state, pagination: action.payload };
    case 'TOGGLE_THEME':
      return { ...state, theme: state.theme === 'light' ? 'dark' : 'light'
    case 'SET_LOADING':
      return { ...state, loading: action.payload };
    default:
      return state;
  }
}
```

```
function EcommerceDashboard() {
  const [state, dispatch] = useReducer(reducer, initialState);

  // ... component logic to fetch data, handle user interactions

  return (
    // ... JSX to render the dashboard, display data, and handle
  );
}
```

useState vs useReducer

1. useState만 써서 충분한 경우가 대부분
2. useReducer를 사용하는 것이 효율적인 경우
    a. 관리해야하는 상태가 많을 때
    b. 상태들이 서로 관련이 있는 경우
    c. 비즈니스 로직 분리
    d. Immutability
3. useContext 설명할 때 예제코드와 같이 상세히 다룰 예정

useMemo vs useCallback

1. useMemo
   a. 함수의 결과를 cache하기 위해 사용
      i. “Expensive” computation
      ii. React는 1ms 이상 걸리면 `expensive`라고 칭함
      iii. “Expensive”하지 않다면 굳이 필요 없다.
   b. 조건에 따른 컴포넌트를 리턴할 때 사용하거나 특정 변수를 계산할 때
   c. 초기렌더링보다는 cache되는 것을 고려할 때 re-rendering에 유리함
      i. https://github.com/yeonjuan/dev-blog/blob/master/JavaScript/should-you-really-use-usememo.md?utm_source=substack&utm_medium=email
      ii. 요약하면 시간복잡도를 n으로 칭할 때, n > 100일때만 심하게 유리함
2. useCallback
   a. 함수 자체를 cache하기 위해 사용
   b. dependency를 확인해야 하는 함수일 때
   c. ChildComponent에 prop으로 넘겨주는 함수일 때

useCallback

1. 함수를 왜 cache하는가?
2. JavaScript 언어 특성

```jsx
import React, { useState, useCallback } from 'react';

function ChildComponent({ onClick }) {
  console.log('ChildComponent re-rendered');
  return <button onClick={onClick}>Click me</button>;
}

function ParentComponent() {
  const [count, setCount] = useState(0);

  const handleClick = useCallback(() => {
    setCount((prevCount) => prevCount + 1);
  }, []);

  console.log('ParentComponent re-rendered');

  return (
    <div>
      <h1>Count: {count}</h1>
      <ChildComponent onClick={handleClick} />
    </div>
  );
}
```

useMemo vs useCallback

1. Sendbird repo에서 사용법 확인
2. 좋은 코드를 계속해서 보는 것이 중요함
    a. 처음 프로젝트 아키텍처 잡았던 경험
3. https://github.com/sendbird/quickstart-calls-reactjs

React18 에서 새로 소개된 hook

1. useId
2. useTransition
3. useDeferredValue
4. useSyncExternalStore
5. useInsertionEffect

기타 hook들

1. useEffect
   a. useEffect 왠만하면 필요없다
2. useLayoutEffect
3. useRef

전역 상태 툴 - Context API

1. 기본 제공 툴
2. Provider를 사용해서 components들에 `state`를 `provide` 하는 방식
3. 장점
    a. 매우 간단함
    b. 추가 패키지를 설치하지 않아도 됨

전역 상태 툴 - Context API

1. 단점
    a. 비즈니스 로직에 따라서 Provider를 생성해줘야함
        i. 코드가 복잡해질 수 있음
    b. 렌더링 효율에 좋지 않음
2. Sendbird repo 참고 설명
    a. Custom hook을 선언해서 reducer를 사용하는 방식
    b. https://github.com/sendbird/quickstart-calls-reactjs/tree/main/sample-01

전역 상태 툴 - Redux

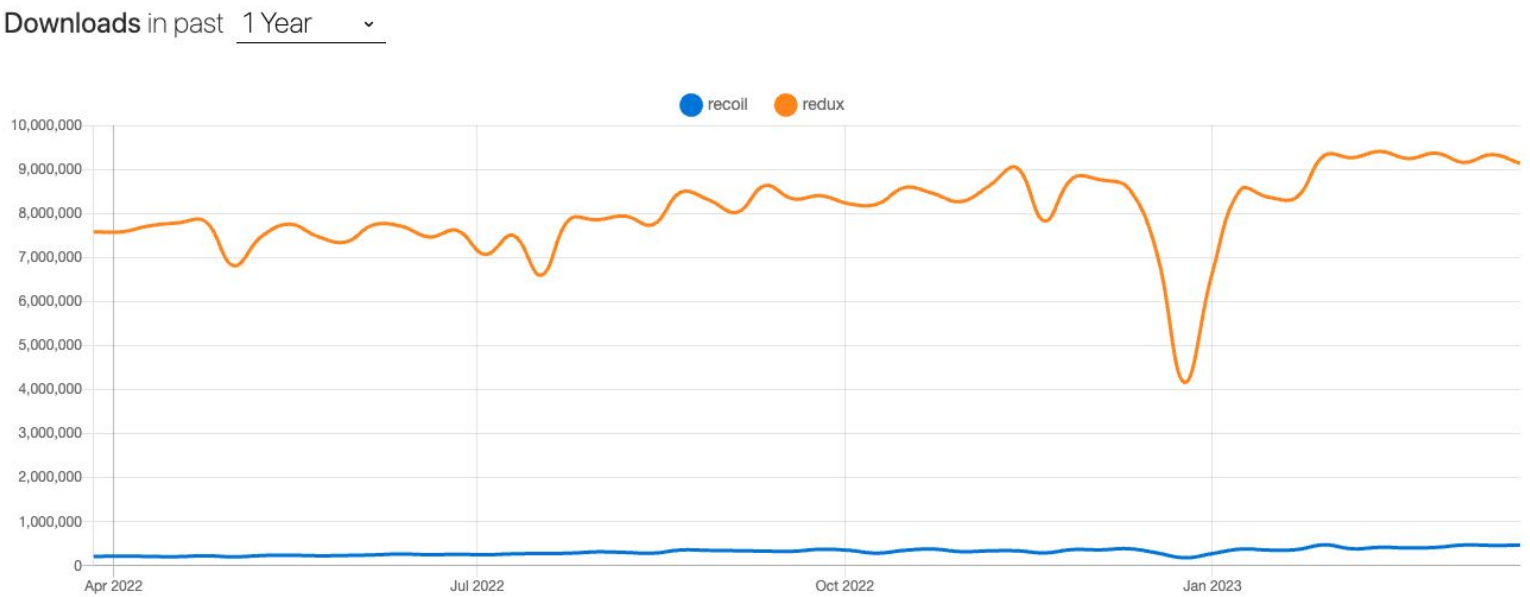
1. 모든 상태를 store에 저장함
   a. Context API가 context별로 reducer를 따로 사용하는 것과 반대됨
2. Read-only states
   a. useReducer와 유사하게 `dispatch`를 통해서만 상태를 업데이트 할 수 있음
   b. store를 직접적으로 mutate할 수 없음
3. 장점
   a. Redux DevTools등을 활용한 비교적 쉬운 debugging
   b. Middleware: saga, thunk, persistent
4. 단점
   a. 구조가 복잡함
   b. 사이즈가 작을경우 불필요한 overhead
5. https://github.com/reduxjs/redux/tree/master/examples

전역 상태 툴 - Recoil

1. Facebook에서 만들었음
2. atoms and selectors라는 개념
   a. 내가 필요한 값만 `subscribe`하는 느낌
   b. atom은 일반적인 state와 유사한 개념
   c. selector는 atom을 Manipulate 해야하는 경우 사용됨
3. 장점
   a. 구조가 간단해서 적용하기 쉬움
   b. ContextAPI의 rendering 비효율을 개선함
   c. React 상태관리를 위해서 만들어짐

전역 상태 툴 - Recoil

1. 단점
   a. 사용자가 아직은 그렇게 많지 않음
   b. middleware의 부재



전역 상태 툴 - Recoil

https://github.com/saseungmin/Recoil_ToDo