

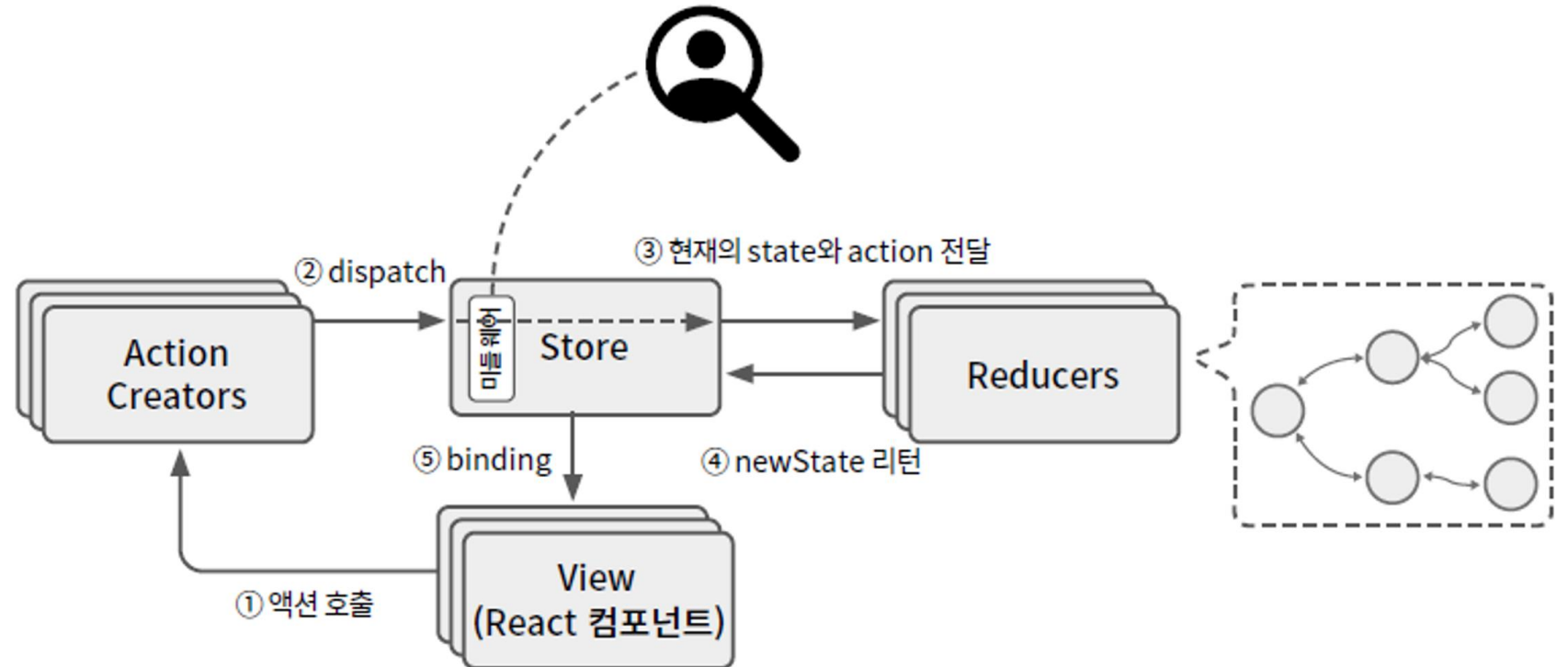
- Redux 미들웨어
- 비동기 처리



# 1. Redux Middleware란?

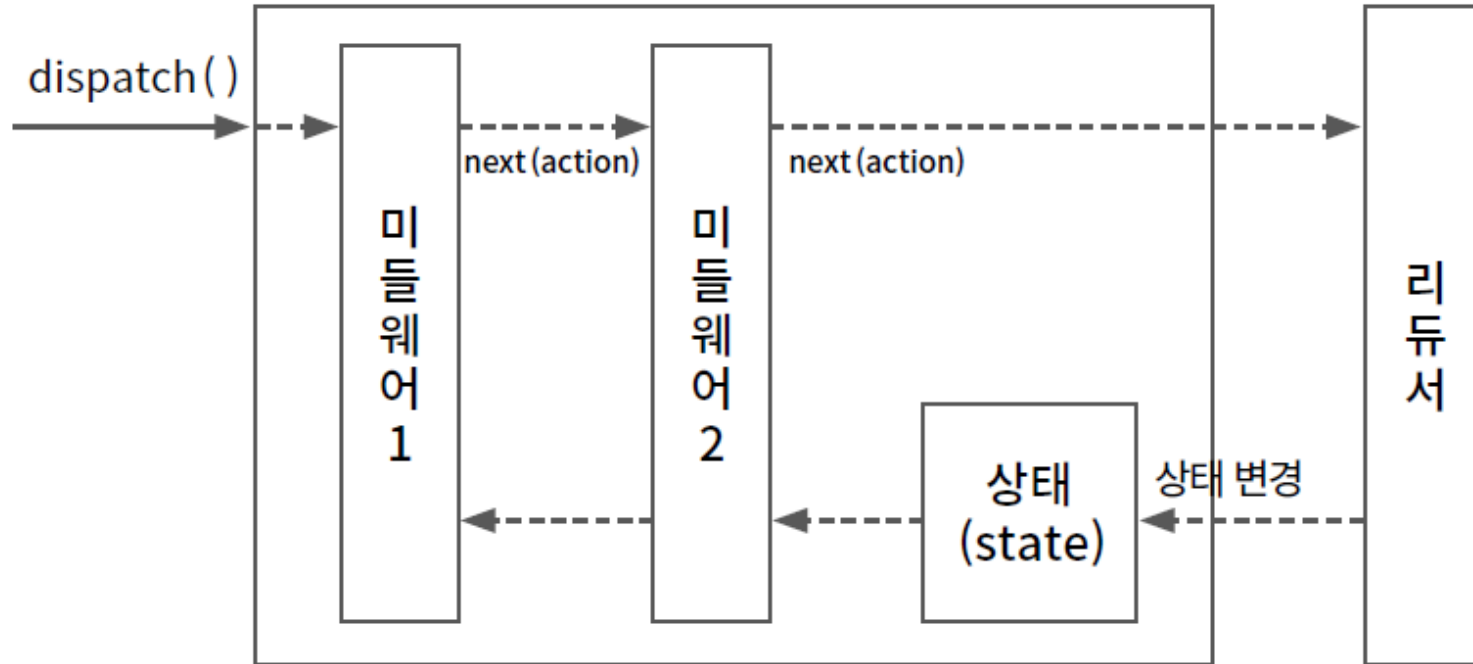
## ❖리덕스 미들웨어란?

- 액션이 스토어로 dispatch 된 후 리듀서에 도달하기 전과 상태 변경이 완료된 후 수행할 중앙집중화된 작업을 지정할 수 있는 함수
- 단일 스토어 내부에 등록함
  - 모든 액션이 스토어를 거쳐감
  - 상태는 스토어에 저장



# 1. Redux Middleware란?

❖ 좀 더 상세하게 보자면...



- 미들웨어1의 전처리 실행 → `next(action)`
- 미들웨어2의 전처리 실행 → `next(action)`
- 리듀서 실행 → 새로운 상태 리턴 → 스토어의 새로운 상태로 설정
- 미들웨어2의 후처리 실행
- 미들웨어1의 후처리 실행

# 1. Redux Middleware란?

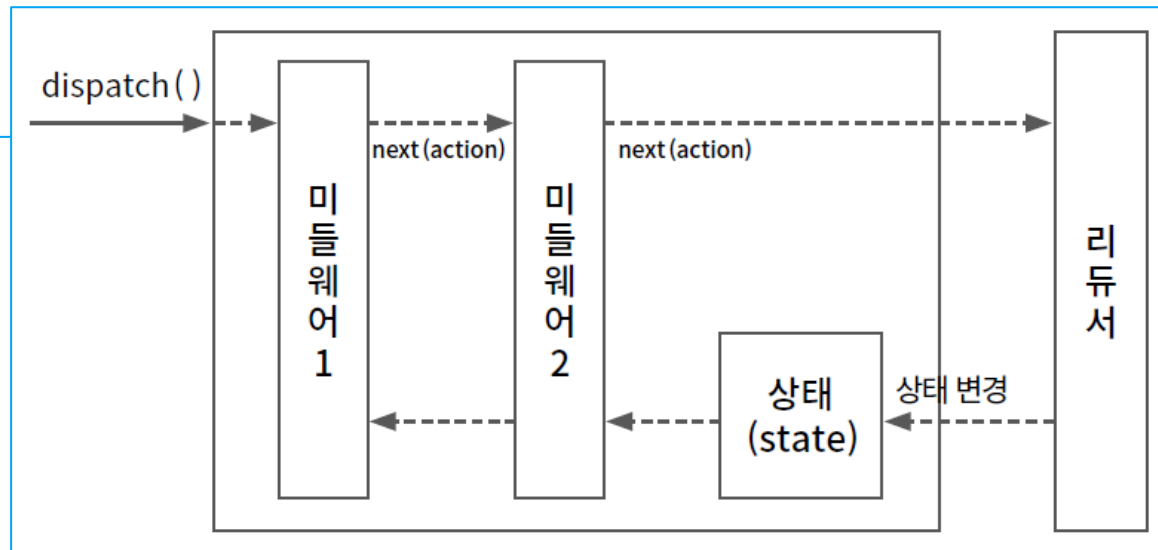
## ❖ 좀 더 상세하게 보자면...

// ES6 문법 표현

```
const middleware1 = (store)=>(next)=>(action)=> {  
  //다음 미들웨어 또는 리듀서로 전달되기 전  
  next(action);  
  //스토어의 새로운 상태 설정 후  
}
```

// ES5 문법 표현

```
var middleware1 = function middleware1(store) {  
  return function (next) {  
    return function (action) {  
      //다음 미들웨어 또는 리듀서로 전달되기 전  
      next(action);  
      //스토어의 새로운 상태 설정 후  
    };  
  };  
};
```

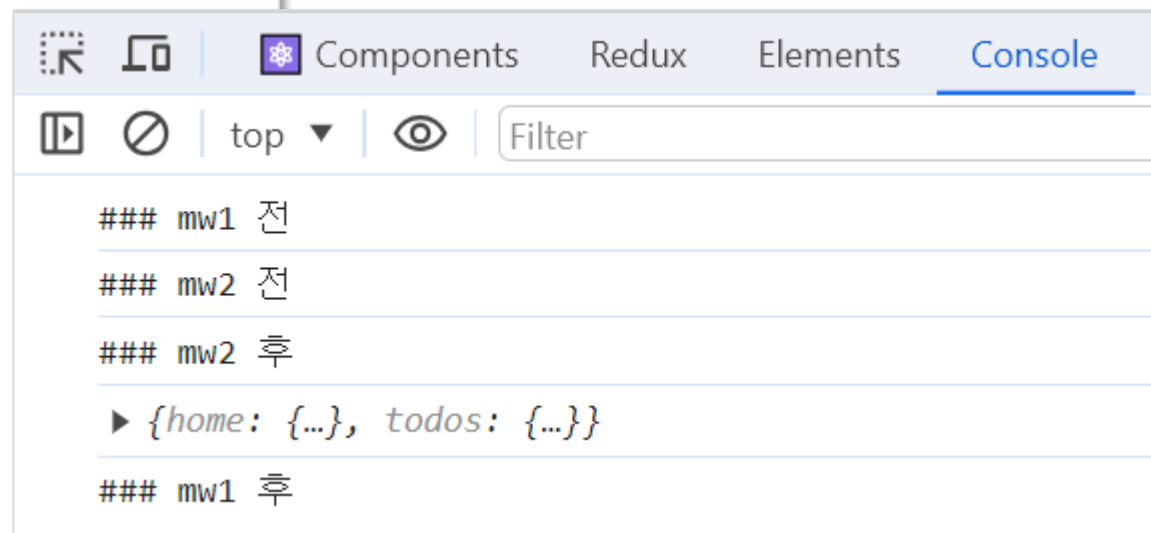


- 미들웨어1의 전처리 실행 → next(action)
- 미들웨어2의 전처리 실행 → next(action)
- 리듀서 실행 → 새로운 상태 리턴 → 스토어의 새로운 상태로 설정
- 미들웨어2의 후처리 실행
- 미들웨어1의 후처리 실행

## 2. 간단한 Logger 미들웨어

### ❖ Logger 작성하기 전에 테스트 미들웨어

```
.....  
const mw1 = (store) => (next) => (action) => {  
  console.log("### mw1 전");  
  next(action);  
  console.log("### mw1 후");  
};  
  
const mw2 = (store) => (next) => (action) => {  
  console.log("### mw2 전");  
  next(action);  
  console.log("### mw2 후");  
  console.log(store.getState());  
};  
  
const AppStore = configureStore({  
  reducer: RootReducer,  
  middleware: (getDefaultMiddleware) => {  
    return getDefaultMiddleware({ serializableCheck: false })  
      .concat(mw1).concat(mw2);  
  }  
});  
.....
```

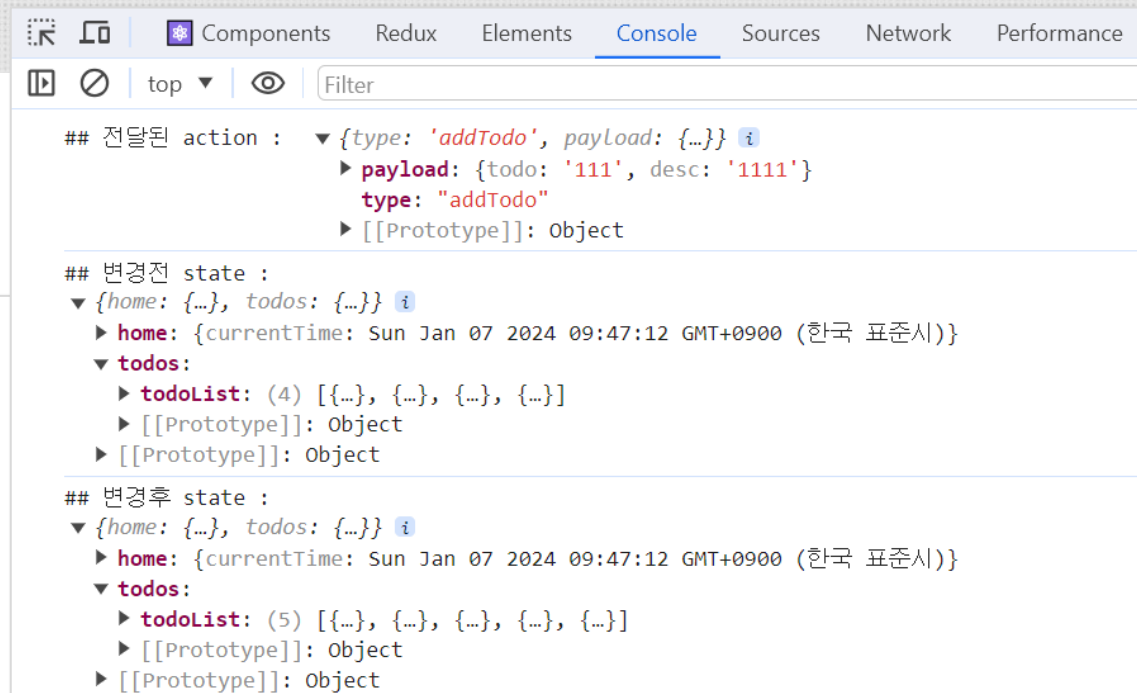


## 2. 간단한 Logger 미들웨어

### ❖ Logger 작성

- 기존 테스트 미들웨어를 제거하고 작성

```
.....  
const logger = (store)=>(next)=>(action)=> {  
  console.log("## 전달된 action :", action);  
  console.log("## 변경전 state :", store.getState());  
  next(action);  
  console.log("## 변경후 state :", store.getState());  
}  
  
const AppStore = configureStore({  
  reducer: RootReducer,  
  middleware: (getDefaultMiddleware) => {  
    return getDefaultMiddleware({ serializableCheck: false }).concat(logger);  
  }  
});  
  
.....
```



### 3. 미들웨어와 비동기 처리

#### ❖복잡하고 긴 처리 시간이 필요한 작업

- 동기적으로 처리하면?
  - 처리 시간이 길어지면 그 시간 동안 브라우저가 먹통이 됨
  - 따라서 비동기적으로 처리해야 함
- 대표적인 예
  - 백엔드 API 서비스와의 통신 : 네트워크를 통해 전송되는 시간 + 백엔드에서의 실행 시간
  - setTimeout()을 이용해 일정 시간 뒤에 실행하도록 처리

#### ❖리덕스를 사용하는 애플리케이션에서는 어느 지점에서 비동기 처리를 할까?

- 리듀서? No! 순수함수!!
- 액션 생성자
  - 좋은 위치이긴 하지만 액션 생성자는 액션(메시지 객체)을 생성하여 리턴함
  - 액션 생성자 함수는 값을 리턴하기 때문에 동기적으로 작동!
  - 이런 이유로 미들웨어를 이용해야 함
  - redux-thunk, redux-saga



## 4. redux-thunk 미들웨어

### ❖redux-thunk란?

- 비동기 처리를 위한 redux용 미들웨어
- thunk
  - 컴퓨터 프로그램에서 다른 서브루틴 또는 함수로 연산 기능을 주입시킬 때 사용하는 함수
  - 지연된 실행을 위해 표현식으로 만든 함수
  - ActionCreator가 Action 메시지 대신에 thunk 함수를 리턴함
- 패키지 참조 : `npm install redux-thunk`
  - 리덕스 툴킷을 사용한다면 redux-thunk가 이미 포함되어 있으므로 추가설치할 필요 없음

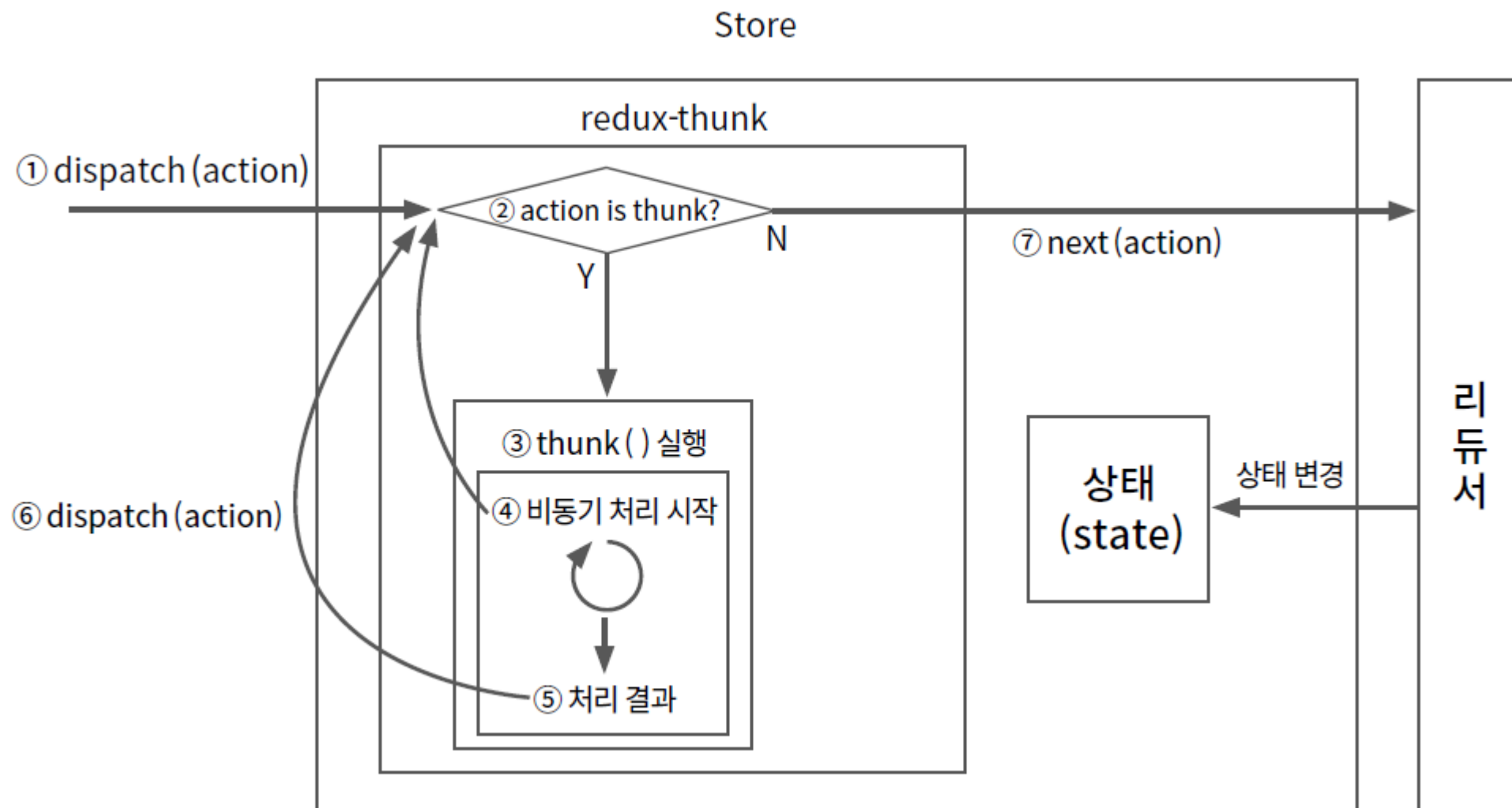
### ❖redux-thunk의 적용 방법

- store 객체에서 미들웨어 등록



## 4. redux-thunk 미들웨어

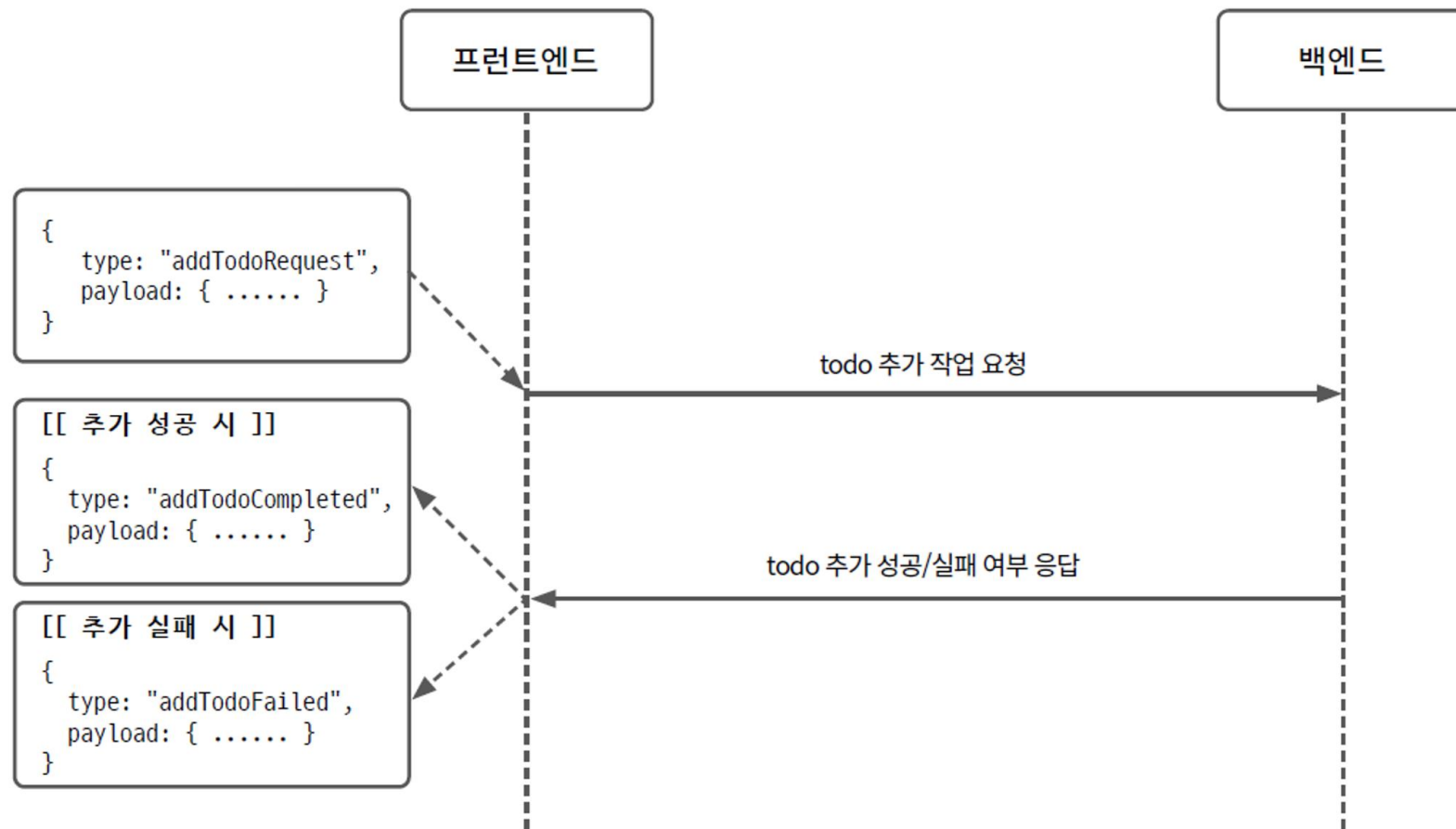
### ❖redux-thunk 미들웨어 아키텍처



## 4. redux-thunk 미들웨어

### ❖ 상태 변경이 필요한 시점

- 요청시점, 성공 응답, 실패 응답



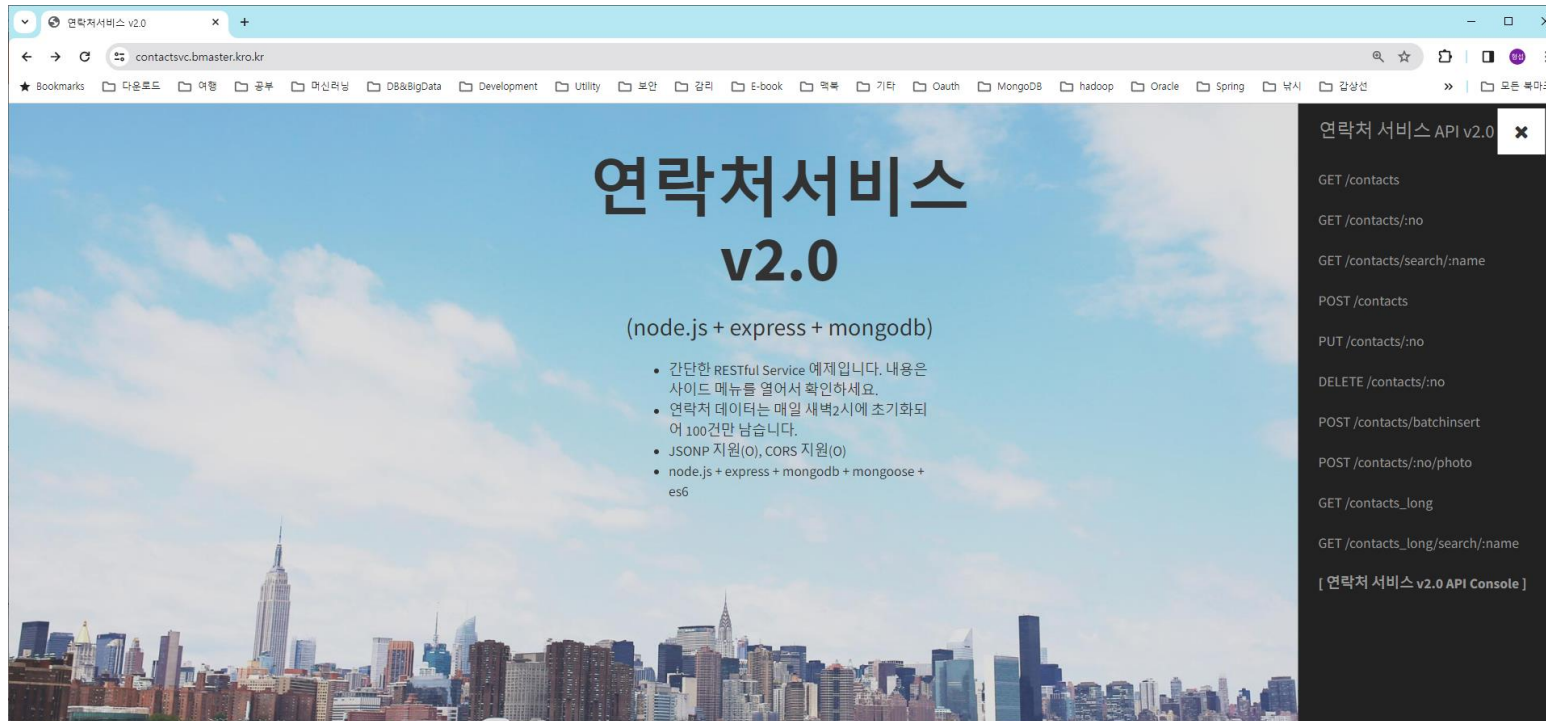
\* 시점별 액션명은 달라질 수 있음

- request : pending
- completed : fulfilled
- failed : rejected

## 5. 1단계-redux-thunk 적용

### ❖테스트용 백엔드 API

- <https://contactsvc.bmaster.kro.kr>



- 이름 검색 테스트 : 1초의 지연 시간 후 응답
  - [https://contactsvc.bmaster.kro.kr/contacts\\_long/search/ja](https://contactsvc.bmaster.kro.kr/contacts_long/search/ja)
  - [https://contactsvc.bmaster.kro.kr/contacts\\_long/search/se](https://contactsvc.bmaster.kro.kr/contacts_long/search/se)

## 5. 1단계-redux-thunk 적용

❖ 준비된 프로젝트 (contacts-client-2-시작전)를 이용해 시작

- App 표현 컴포넌트가 미리 제공됨
- 작성해야 할 요소 : Redux 구성요소(스토어, 액션생성자, 리듀서), App 컨테이너 컴포넌트

❖ src/redux/ContactActionCreator.js

```
import { createAction } from "@reduxjs/toolkit";
import axios from "axios";

export const ContactActionCreator = {
  searchContactsPending : createAction("searchContactsPending"),
  searchContactsFulfilled : createAction("searchContactsFulfilled"),
  searchContactsRejected: createAction("searchContactsRejected"),
  asyncSearchContacts: (name) => {
    return async (dispatch, getState) => {
      let url = "https://contactsvc.bmaster.kro.kr/contacts_long/search/" + name;
      try {
        dispatch(ContactActionCreator.searchContactsPending({ name }));
        const response = await axios.get(url);
        dispatch(ContactActionCreator.searchContactsFulfilled({ contacts : response.data }));
      } catch (error) {
        dispatch(ContactActionCreator.searchContactsRejected({ status : error }));
      }
    };
  },
}
```

thunk 함수

## 5. 1단계-redux-thunk 적용

### ❖src/redux/ContactReducer.js

```
import { createReducer } from "@reduxjs/toolkit";
import { ContactActionCreator } from "../ContactActionCreator";

const initialState = { contacts: [], isLoading: false, status: "" };

export const ContactReducer = createReducer(initialState, (builder) => {
  builder
    .addCase(ContactActionCreator.searchContactsPending, (state, action) => {
      state.contacts = [];
      state.isLoading = true;
      state.status = `조회중 : 검색명( ${action.payload.name} )`;
    })
    .addCase(ContactActionCreator.searchContactsFulfilled, (state, action) => {
      state.contacts = action.payload.contacts;
      state.isLoading = false;
      state.status = "조회 완료";
    })
    .addCase(ContactActionCreator.searchContactsRejected, (state, action) => {
      state.contacts = [];
      state.isLoading = false;
      state.status = "조회 실패 : " + action.payload.status;
    });
});
```

## 5. 1단계-redux-thunk 적용

### ❖src/redux/ContactStore.js

```
import { configureStore } from "@reduxjs/toolkit";
import { ContactReducer } from "../ContactReducer";

//디버깅 목적으로 logger 등록
const logger = (store) => (next) => (action) => {
  console.log("## 전달된 action :", action);
  next(action);
};

const ContactStore = configureStore({
  reducer: ContactReducer,
  middleware: (getDefaultMiddleware) => {
    return getDefaultMiddleware().concat(logger);
  }
});

export default ContactStore;
```

## 5. 1단계-redux-thunk 적용

### ❖src/App.jsx 변경

```
import { useDispatch, useSelector } from "react-redux";
import { useState } from "react";
import { ContactActionCreator } from "../redux/ContactActionCreator";

const App = ({ contacts, isLoading, status, asyncSearchContacts }) => {
  .....(생략)
};

export const AppContainer = () => {
  const dispatch = useDispatch();
  const contacts = useSelector((state)=>state.contacts);
  const isLoading = useSelector((state)=>state.isLoading);
  const status = useSelector((state)=>state.status);

  const asyncSearchContacts = (name)=>dispatch(ContactActionCreator.asyncSearchContacts(name));

  return <App contacts={contacts} isLoading={isLoading} status={status}
    asyncSearchContacts={asyncSearchContacts} />
}

export default App;
```



## 5. 1단계-redux-thunk 적용

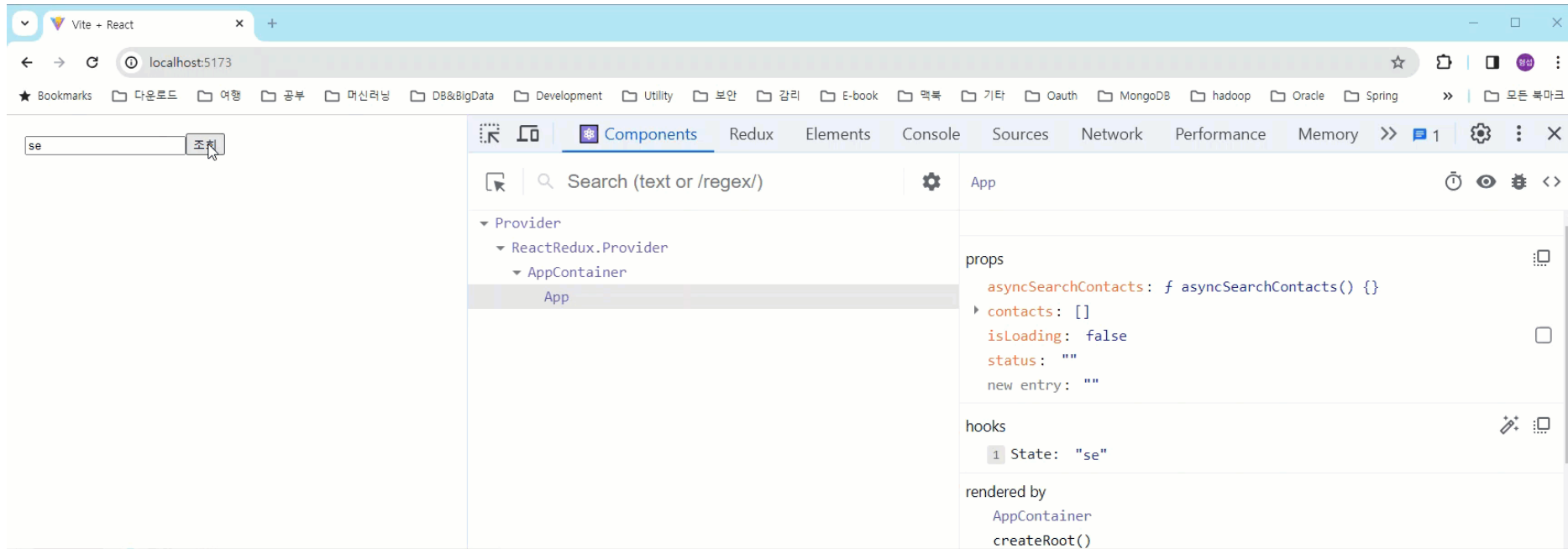
### ❖src/main.jsx

```
import React from "react";
import ReactDOM from "react-dom/client";
import { AppContainer } from "./App.jsx";
import "./index.css";
import ContactStore from "./redux/ContactStore.js";
import { Provider } from "react-redux";

ReactDOM.createRoot(document.getElementById("root")).render(
  <React.StrictMode>
    <Provider store={ContactStore}>
      <AppContainer />
    </Provider>
  </React.StrictMode>
);
```

## 5. 1단계-redux-thunk 적용

### ❖ 실행 결과



#### ■ 이제까지 예제의 문제점

- 비동기 처리시의 각 시점마다 상태를 변경해줄 액션 생성자를 모두 만들어야 하나?
- Thunk로 비동기 처리할 때 각 시점에 액션을 직접 dispatch 해주어야 하나?
- 이러한 이유로 Redux Toolkit의 createAsyncThunk() 함수를 사용함.

## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

### ❖리덕스 툴킷에는...

- redux-thunk 패키지 이미 포함
- redux-thunk가 이미 defaultMiddleware로 등록되어 있음

### ❖createAsyncThunk 툴킷 함수

- 요청 시작, 요청 완료 시점에 직접 dispatch 하지 않아도 됨.
- 내부적으로 ActionType과 액션 생성자를 만들어냄
  - 액션명을 "searchPerson"으로 지정했다면...

```
const asyncAction = createAsyncThunk("액션명", async (arg, thunkAPI) => {  
  //args는 비동기 처리할 때 필요한 아규먼트입니다.  
  //비동기 처리 후 마지막에 리턴하는 값이  
  //최종적으로 완료했을 때 전달하는 action의 페이로드가 됩니다.  
  return payload  
})
```

시점	액션명	액션 생성자 함수
비동기 작업 시작	searchPerson/pending	asyncAction.pending
비동기 작업 완료	searchPerson/fulfilled	asyncAction.fulfilled
비동기 작업 실패	searchPerson/rejected	asyncAction.rejected

## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

### ❖createAsyncThunk 함수의 형태

```
const asyncAction = createAsyncThunk("액션명", async (arg, thunkAPI) => {  
  //args는 비동기 처리할 때 필요한 아규먼트입니다.  
  //비동기 처리 후 마지막에 리턴하는 값이  
  //최종적으로 완료했을 때 전달하는 action의 페이로드가 됩니다.  
  return payload  
})
```

- 두번째 인자 함수 : payloadCreator라는 비동기 처리 수행 함수
  - 요청/응답 시점별로 dispatch(action)하지 않아도 됨
  - 만일 직접 dispatch하고 싶다면 thunkAPI 인자를 이용하여 dispatch, fulfillWithValue, rejectWithValue, getState 등의 함수를 이용해 상태를 확인하고 액션을 직접 전달할 수 있음
  - payloadCreator 함수는 Promise 기반이므로 Promise, async/await 기반으로 작성되어야 함

## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

### ❖ src/redux/ContactActionCreator.js 변경

```
import { createAsyncThunk } from "@reduxjs/toolkit";
import axios from "axios";

export const ContactActionCreator = {
  asyncSearchContacts: createAsyncThunk("searchContacts", async({ name }, thunkAPI)=> {
    let url = "https://contactsvc.bmaster.kro.kr/contacts_long/search/" + name;
    const response = await axios.get(url);
    return { contacts : response.data };
  })
}
```

- argument가 하나이므로 전달해야할 인자가 여러개인 경우 객체형태로 전달해야 함 : { name }
- 액션 자동 생성
  - ContactActionCreator.asyncSeachContacts.pending
  - ContactActionCreator.asyncSeachContacts.fulfilled
  - ContactActionCreator.asyncSeachContacts.rejected
- 자동으로 dispatch하는 경우 pending, rejected에는 action payload가 전달되지 않음
  - rejected 시점에는 action.error, pending 시점에는 action.meta 가 전달됨

## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

### ❖src/redux/ContactReducer.js 변경

```
import { createReducer } from "@reduxjs/toolkit";
import { ContactActionCreator } from "../ContactActionCreator";

const initialState = { contacts: [], isLoading: false, status: "" };

export const ContactReducer = createReducer(initialState, (builder) => {
  builder
    .addCase(ContactActionCreator.asyncSearchContacts.pending, (state, action) => {
      state.contacts = [];
      state.isLoading = true;
      state.status = `조회중 (name:${action.meta.arg.name})`;
    })
    .addCase(ContactActionCreator.asyncSearchContacts.fulfilled, (state, action) => {
      state.contacts = action.payload.contacts;
      state.isLoading = false;
      state.status = "조회 완료";
    })
    .addCase(ContactActionCreator.asyncSearchContacts.rejected, (state, action) => {
      state.contacts = [];
      state.isLoading = false;
      state.status = "조회 실패 : " + action.error.message;
    });
});
```

## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

### ❖src/App.jsx 변경

```
.....  
const App = ({ contacts, isLoading, status, asyncSearchContacts }) => {  
  const [name, setName] = useState("");  
  const search = () => {  
    //argument 형식에 맞춰서 전달  
    asyncSearchContacts({ name });  
    setName("");  
  };  
  
  return (  
    .....(생략)  
  );  
};  
.....
```



## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

❖만일 각 요청, 응답 시점에 Action Payload를 전달해야 한다면?

- thunkAPI 인자를 사용함
  - thunkAPI.dispatch()
  - thunkAPI.getState()
  - thunkAPI.rejectWithValue(value, [meta])
  - thunkAPI.fulfillWithValue(value, meta)

❖기존 예제에 다음 기능을 추가해보자

- 요청 시점(pending)에 action payload를 전달
- 실패 응답 시점(rejected)에 action.payload.status 를 전달

## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

### ❖src/redux/ContactActionCreator.js 변경

```
import { createAsyncThunk } from "@reduxjs/toolkit";
import axios from "axios";

export const ContactActionCreator = {
  asyncSearchContacts: createAsyncThunk("searchContacts", async({ name }, thunkAPI)=> {
    try {
      thunkAPI.dispatch({ type: "additionalAction", payload: { msg:"추가적인 액션" }});
      let url = "https://contactsvc.bmaster.kro.kr/contacts_long/search/" + name;
      const response = await axios.get(url);
      return { contacts : response.data };
    } catch(e) {
      return thunkAPI.rejectWithValue({ status: e.message })
    }
  })
}
```

## 6. 2단계-@reduxjs/toolkit의 createAsyncThunk() 함수 적용

### ❖src/redux/ContactReducer.js 변경

```
.....(생략)
export const ContactReducer = createReducer(initialState, (builder) => {
  builder
    .addCase(ContactActionCreator.asyncSearchContacts.pending, (state, action) => {
      state.contacts = [];
      state.isLoading = true;
      state.status = `조회중 (name: ${action.meta.arg.name})`;
    })
    .addCase(ContactActionCreator.asyncSearchContacts.fulfilled, (state, action) => {
      state.contacts = action.payload.contacts;
      state.isLoading = false;
      state.status = "조회 완료";
    })
    .addCase(ContactActionCreator.asyncSearchContacts.rejected, (state, action) => {
      state.contacts = [];
      state.isLoading = true;
      state.status = "조회 실패 : " + action.payload.status;
    });
});
```

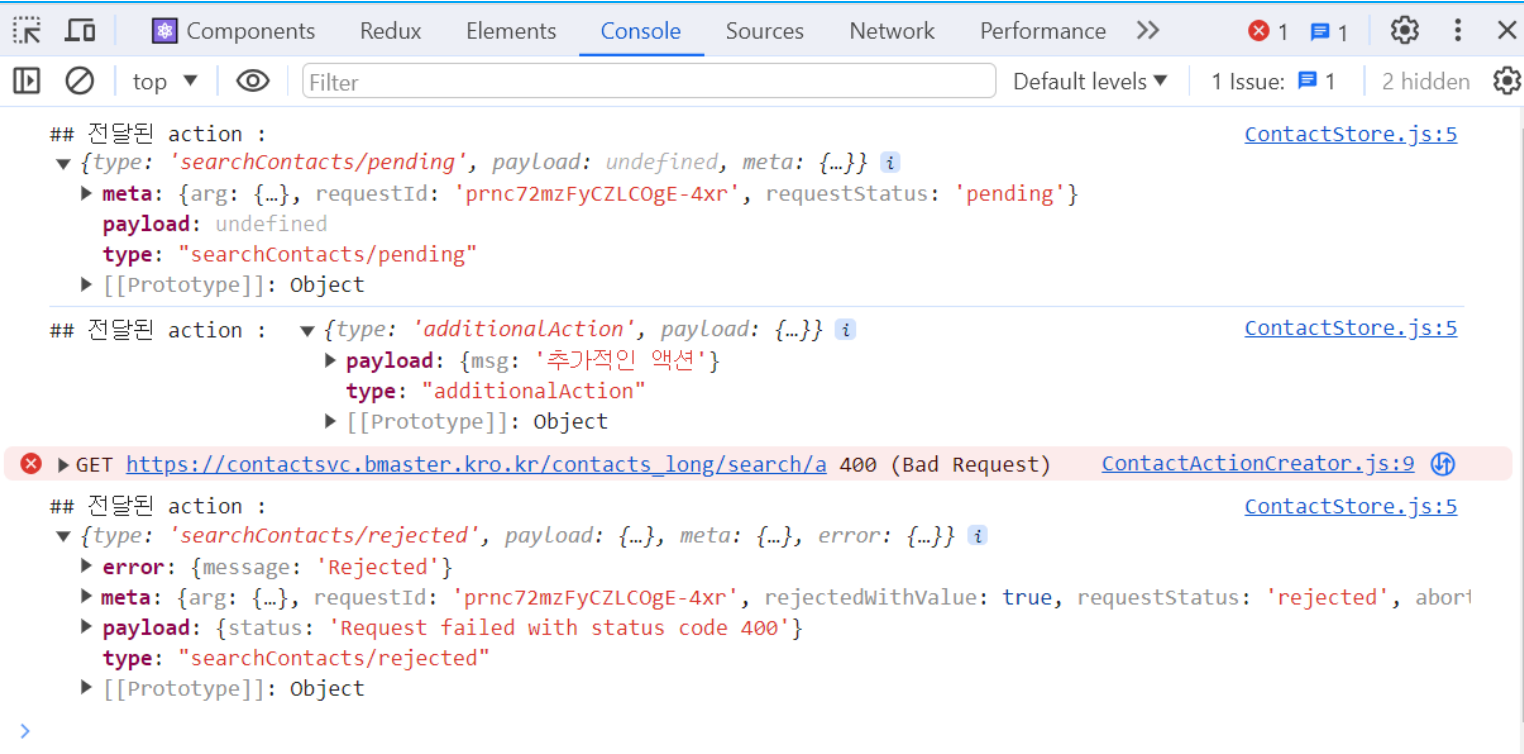
## 6. 2단계-@reduxjs/toolkit의 creatAsyncThunk() 함수 적용

### ❖ 실행 결과1

- 이름으로 한글자만 입력하고 조회 시도

조회

조회 실패 : Request failed with status code 400



```
## 전달된 action : ContactStore.js:5
▼ {type: 'searchContacts/pending', payload: undefined, meta: {...}} i
  ► meta: {arg: {...}, requestId: 'prnc72mzFyCZLC0gE-4xr', requestStatus: 'pending'}
  ► payload: undefined
  ► type: "searchContacts/pending"
  ► [[Prototype]]: Object

## 전달된 action : ContactStore.js:5
▼ {type: 'additionalAction', payload: {...}} i
  ► payload: {msg: '추가적인 액션'}
  ► type: "additionalAction"
  ► [[Prototype]]: Object

✖ GET https://contactsvc.bmaster.kro.kr/contacts_long/search/a 400 (Bad Request) ContactActionCreator.js:9 ⓘ

## 전달된 action : ContactStore.js:5
▼ {type: 'searchContacts/rejected', payload: {...}, meta: {...}, error: {...}} i
  ► error: {message: 'Rejected'}
  ► meta: {arg: {...}, requestId: 'prnc72mzFyCZLC0gE-4xr', rejectedWithValue: true, requestStatus: 'rejected', abort
  ► payload: {status: 'Request failed with status code 400'}
  ► type: "searchContacts/rejected"
  ► [[Prototype]]: Object
```

## 6. 2단계-@reduxjs/toolkit의 creatAsyncThunk() 함수 적용

### ❖ 실행 결과 2

- 두 글자 이상의 영문을 입력하여 검색 : se, ja, an 등

조회중 (name: se)  
조회

Components Redux Elements Console Sources Network Performance Memory >> 1

top Filter

## 전달된 action : {type: 'searchContacts/pending', payload: undefined, meta: {...}} ContactStore.js:5  
## 전달된 action : {type: 'additionalAction', payload: {...}} ContactStore.js:5  
>

Jesse Powell : 010-3456-8296 : 서울시  
Rose Scott : 010-3456-8266 : 서울시  
Rosebud James : 010-3456-8263 : 서울시  
Sean Hall : 010-3456-8261 : 서울시  
Serin Rogers : 010-3456-8228 : 서울시  
조회

Components Redux Elements Console Sources Network Performance Memory >> 1

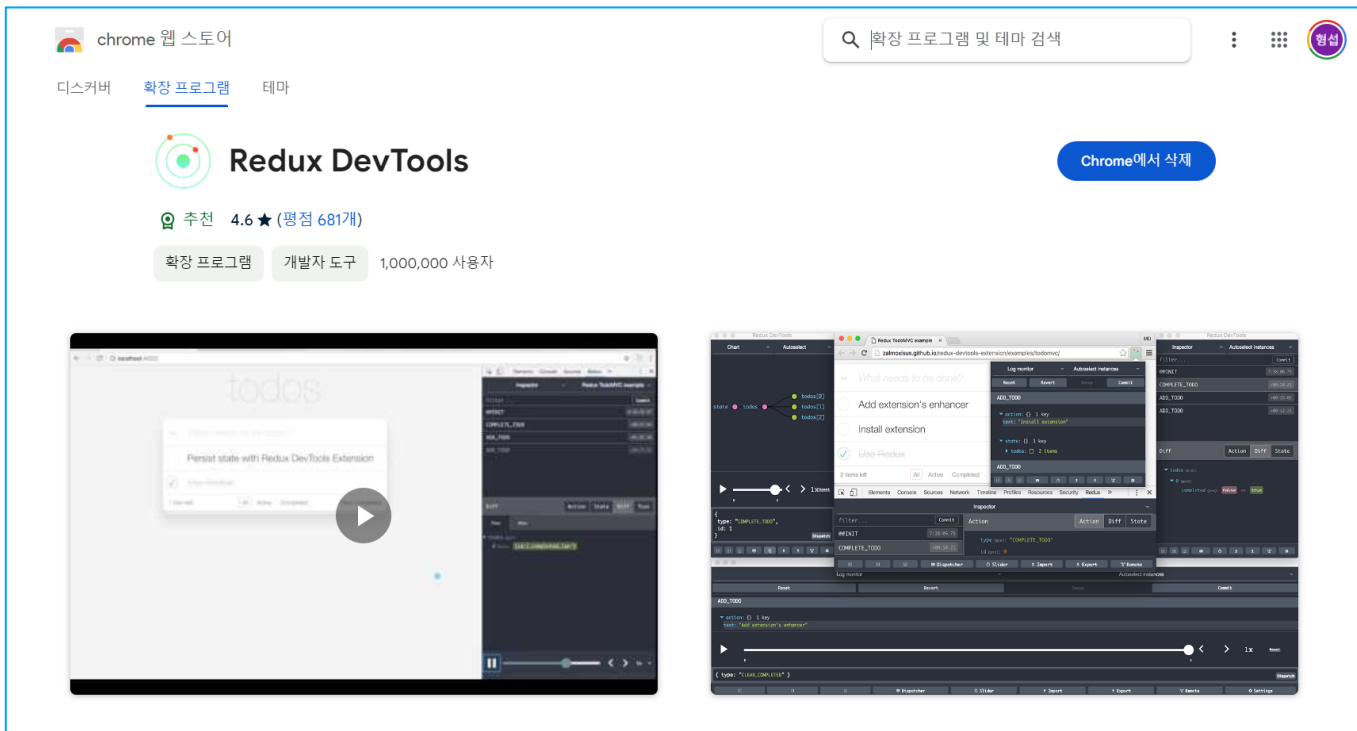
top Filter

## 전달된 action : {type: 'searchContacts/pending', payload: undefined, meta: {...}} ContactStore.js:5  
## 전달된 action : {type: 'additionalAction', payload: {...}} ContactStore.js:5  
## 전달된 action : {type: 'searchContacts/fulfilled', payload: {...}, meta: {...}} i  
    ▶ meta: {arg: {...}, requestId: 'qln2SBPGUWAT3HKFoveBC', requestStatus: 'fulfilled'}  
    ▶ payload: {contacts: Array(5)}  
      type: "searchContacts/fulfilled"  
    ▶ [[Prototype]]: Object

# 7. Redux Devtools

## ❖ Redux Devtools

- Redux를 이용한 앱을 개발할 때 개발을 강력하게 지원하는 개발 패키지 도구
  - Redux의 상태와 액션 정보를 시각화하며, 상태 변경을 추적할 수 있도록 함.
- 크롬 확장 프로그램 설치
  - Redux Devtools로 구글링하여 크롬 확장 프로그램 설치



## 7. Redux Devtools

### ❖ Redux Dev tools는 미들웨어로 작성되었음

- Store에 미들웨어 설정만으로 적용 끝
  - @reduxjs/toolkit에서는 미들웨어의 등록이 이미 되어 있음
- Redux의 불변성 --> 시간 여행 디버깅을 가능하게 함 --> Redux Devtools
- 개발환경일 때만 사용하도록 하기 위한 설정이 필요함

### ❖ todolist-app-router 예제에 적용

```
//logger를 사용하지 않도록 설정
const AppStore = configureStore({
  reducer: RootReducer,
  middleware: (getDefaultMiddleware) => {
    return getDefaultMiddleware({ serializableCheck: false });
  },
  devTools: process.env.NODE_ENV !== "production"
});

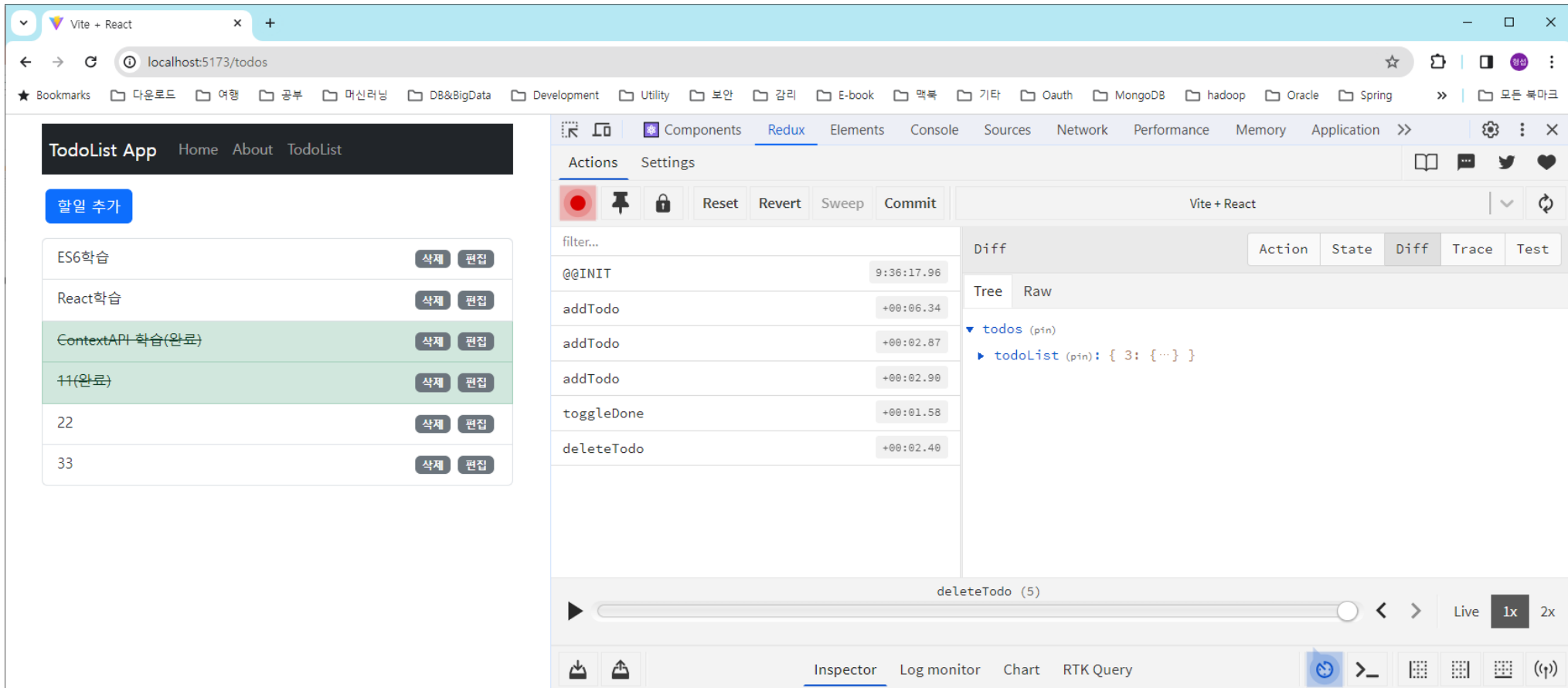
export default AppStore;
```



# 7. Redux Devtools

## ❖ Demo

- 시간 여행 디버깅
- 상태 변경 추적, Diff, Chart





Q&A