React 상태 관리 기초 :
To do 프로젝트 상태관리 적용

글로벌 상태관리가 필요한 이유

글로벌 상태관리가 필요한 이유

01

Props drilling

리액트는 부모에서 자식으로 데이터가 흐릅니다
이로 인해 필요한 컴포넌트까지 데이터를 전달하기 위해 여러 중간 컴포넌트를 거쳐야 하는 경우가 발생합니다.
이러한 과정은 코드의 복잡성을 증가시키고 관리가 어렵습니다.

02

데이터 일관성

애플리케이션 전체에서 일관된 상태를 유지하는 것은 매우 중요합니다.

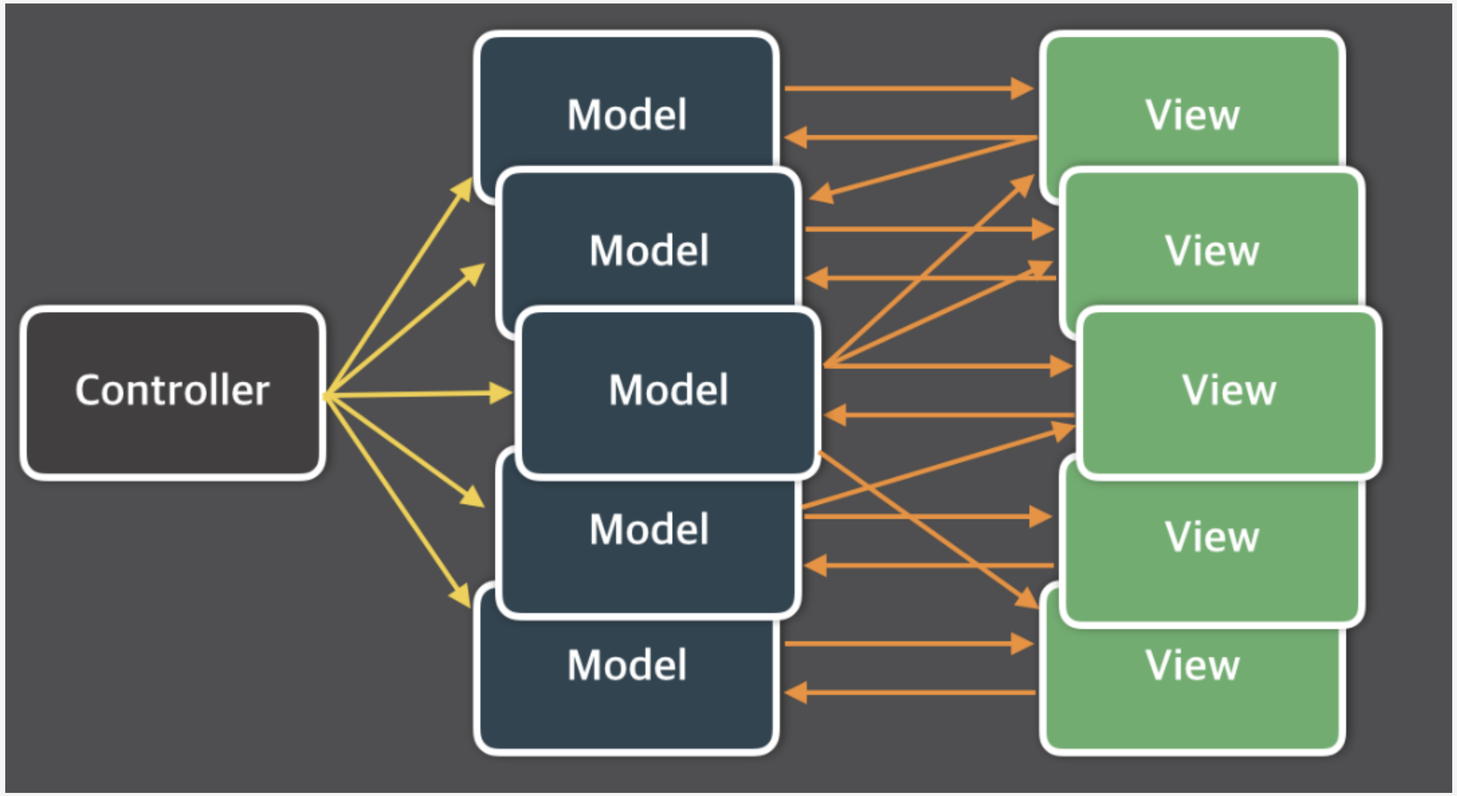
중앙에서 데이터가 관리되지 않으면 여러 컴포넌트에서 읽고 업데이트하는 과정에서 데이터가 불일치할 수 있습니다.

03

복잡성 감소

데이터를 여러 곳에서 관리하는 것보다 중앙에서 한 번에 관리하는 것이 더 효율적입니다.
이는 코드의 가독성을 높이고 유지보수를 용이하게 합니다.

Redux 기초


Controller
Model
Model
Model
Model
Model
View
View
View
View
View

Redux 탄생

3가지 원칙

01

Single Source of Truth

애플리케이션의 모든 상태를 하나의 중앙 저장소에서 관리합니다.
모든 상태 변화가 중앙에서 관리되므로, 상태의 일관성을 쉽게 유지할 수 있습니다.

02

읽기 전용

상태를 변경하는 유일한 방법은 "액션"이라는 객체를 디스패치(dispatch)하는 것입니다. 상태는 직접 변경할 수 없으며, 오직 액션을 통해서만 상태를 변경할 수 있습니다.

03

순수 함수로 작성

변화는 순서 함수을 통해 이뤄져야 합니다
Redux에서 상태 변화는 리듀서를 통해 이루어집니다. 리튜서는 이전 상태와 액션을 받아 새로운 상태를 반환합니다. 이를 통해 상태의 변화를 예측 가능하게 하고 관리 일관성을 유지합니다.

3가지 컨셉

01

Store

애플리케이션의 상태를 저장합니다.

02

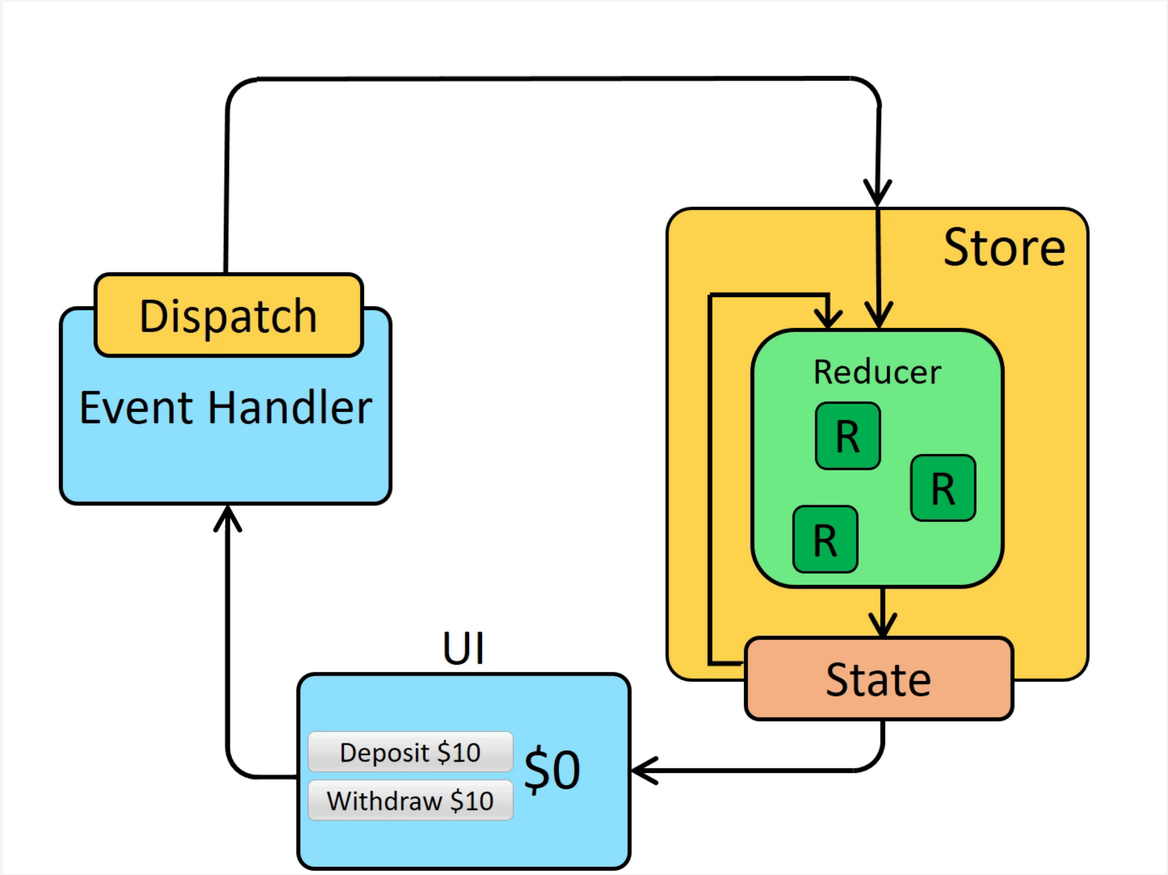
Action

애플리케이션에서 무엇이 발생했는지를 설명합니다.

03

Reducer

이전 상태와 액션을 받아 새로운 상태를 반환하는 순수 함수입니다.



Redux 데이터 흐름

출처: https://redux.js.org/tutorials/fundamentals/part-2-concepts-data-flow

Store

Redux 스토어는 애플리케이션의 데이터들을 저장하기 위한 저장소 역할을 합니다.

```
import { configureStore } from
'@reduxjs/toolkit';
import counterReducer from './counterSlice';

const store = configureStore({ reducer:
counterReducer });

console.log(store.getState());
// { value: 0 }
```

Action

액션은 단순한 JS 객체로,
type 속성을 가져야 합니다.

액션은 상태 변경하기 위한
의도를 나타냅니다.

```
const action = {
  type: "INCREASE",
  payload: {
    count: 1,
  },
};
```

Reducer

상태가 액션에 따라 어떻게 변화하는지를 명시하는 순수 함수입니다.

```
const initialState = {
  count: 10,
};

const reducer = (state = initialState, action) =>
{
  switch (action.type) {
    case "INCREASE":
      return {
        ...state,
        count: state.count +
action.payload.count,
      };
    default:
      return state;
  }
};
```

Redux Toolkit 소개 및 적용

Redux 문제점

```
// 액션 타입
const INCREMENT = 'INCREMENT';
const DECREMENT = 'DECREMENT';
// 액션 생성자
function increment() {
  return { type: INCREMENT };
}
function decrement() {
  return { type: DECREMENT };
}
// 리듀서
function counterReducer(state = initialState,
action) {
  switch (action.type) {
    case INCREMENT:
      return { count: state.count + 1 };
    case DECREMENT:
      return { count: state.count - 1 };
    default:
      return state;
  }
}
```

Store

```
import { configureStore } from
'@reduxjs/toolkit';
import counterReducer from './counterSlice';
import userReducer from './userSlice';
const store = configureStore({
  reducer: {
    counter: counterReducer,
    user: userReducer,
  }
});
export default store;
```

Slice

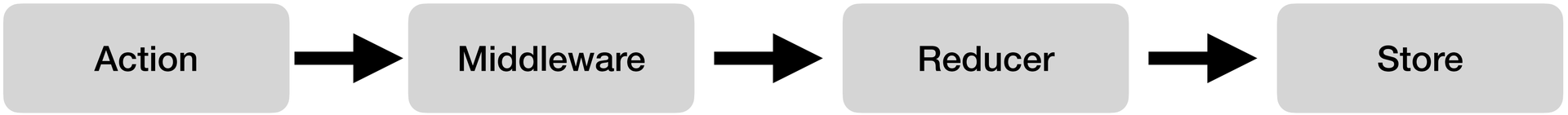
```
import { createSlice } from '@reduxjs/toolkit';

const counterSlice = createSlice({
  name: 'counter',
  initialState: { value: 0 },
  reducers: {
    increment: (state) => {
      state.value += 1;
    },
    decrement: (state) => {
      state.value -= 1;
    },
  },
});

export const { increment, decrement } =
counterSlice.actions;
export default counterSlice.reducer;
```

Redux toolkit 설치

npm install @reduxjs/toolkit react-redux // redux-toolkit 2.2.3 react-redux 9.1.2

Redux Thunk 소개

Middleware



액션이 디스패치되어 리듀서에서 처리되기 전에 특정 작업을 수행
예를 들어, 액션이 스토어에 들어가기전에 미들웨어를 사용하여 로깅 작업을 수행, 비동기 API 요청을 처리할 수 있습니다.

createAsyncThunk

```
const fetchTodo = createAsyncThunk(
    'todo/getTodo',
    async () => {
      const res = await axios.get('...')
      return res.data;
    }
)
```

extraReducers

```
const todoSlice = createSlice({
    ...,
    extraReducers: (builder) => {
        builder.addCase(
            fetchTodo.fulfilled,
            (state, action) => {
                state.list = action.payload
            }
        )
    }
})
```

Redux Thunk으로 Todo List 적용

json-server, axios 설치

npm install -D json-server //1.0.0-beta.1
npm install axios // 1.7.2

API 구성

GET /todos
POST /todos
PATCH /todos/:id
DELETE /todos/:id