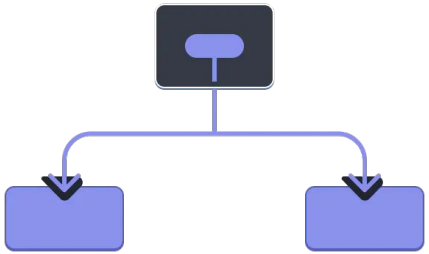
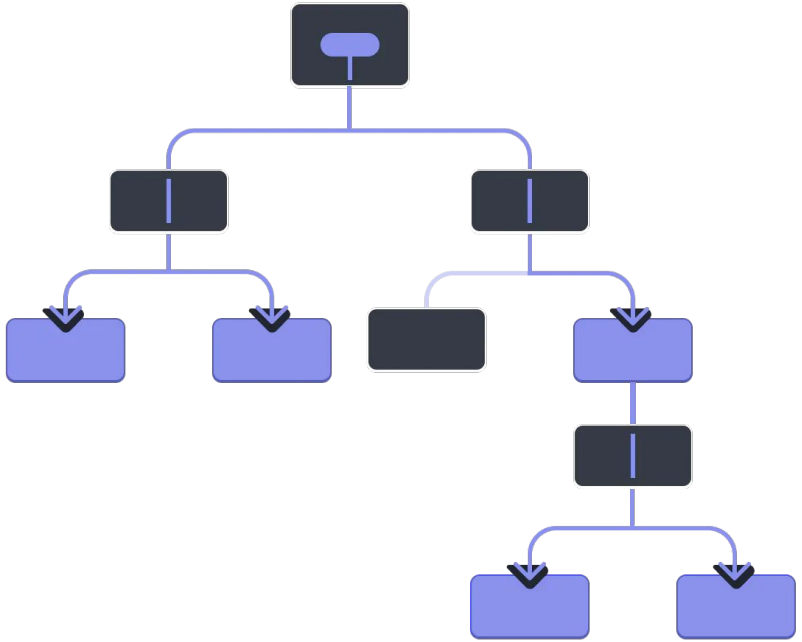
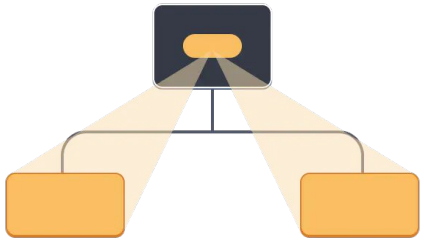
React

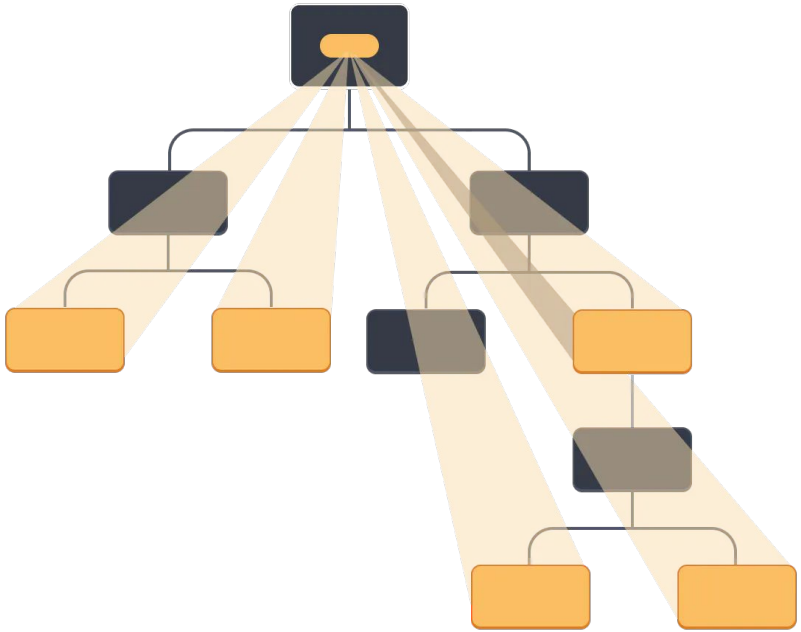
Redux





출처: https://redux.js.org/





출처: https://redux.js.org/

Redux

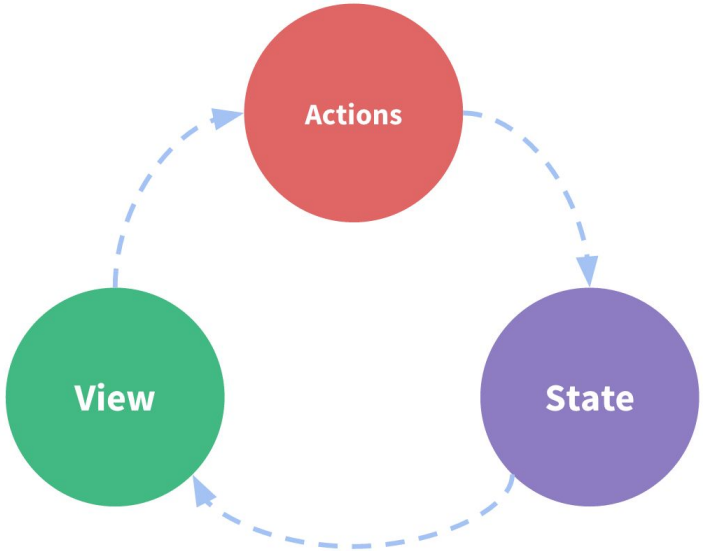
- Redux는 JavaScript 애플리케이션을 위한 상태(state) 컨테이너.
- 일반적으로 React에서는 컴포넌트 수준에서 상태(state)를 관리하고 속성(props)을 통해 상태를 전달.
- Redux를 사용하면 애플리케이션의 전체 상태(state)가 하나의 불변 객체(immutable object)로 관리.
- Redux 상태(state)를 업데이트할 때마다 상태(state) 섹션의 복사본과 새로운 변경 사항이 생성됨.

Redux를 쓰는 이유 vs 단점

- 전역 상태(state)를 쉽게 관리
  - Redux에 연결된 모든 컴포넌트에서 상태의 일부에 액세스하거나 업데이트
- Redux DevTools로 변경 사항을 쉽게 추적
  - 모든 액션(action) 또는 상태(state) 변경 사항을 추적하고 쉽게 따라갈 수 있음.
  - 응용 프로그램의 전체 상태가 각 변경 사항과 함께 추적된다는 사실은 변경 사항 사이를 앞뒤로 이동하는 시간 여행 디버깅을 쉽게 수행할 수 있음.
- Redux의 단점
  - 설정하고 유지해야 할 초기 상용구(boilerplate)가 많음(특히 Redux Toolkit 없이 일반 Redux를 사용하는 경우).
  - 작은 애플리케이션은 Redux가 필요하지 않을 수 있으며, 전역 상태 요구에 대해 Context API를 사용하는 것이 더 이점이 있음.

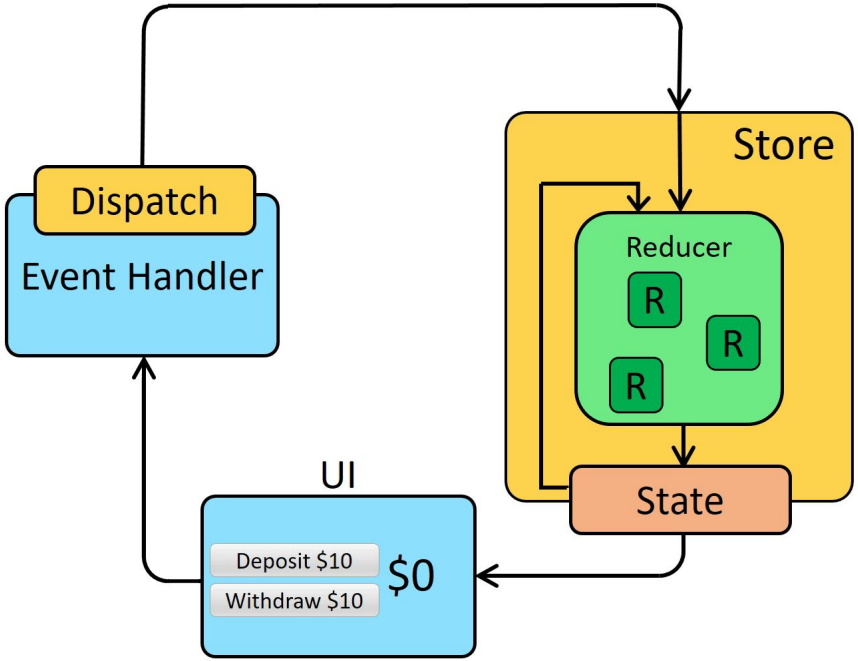
용어

- Actions
- Reducers
- Store
- Dispatch
- Connect



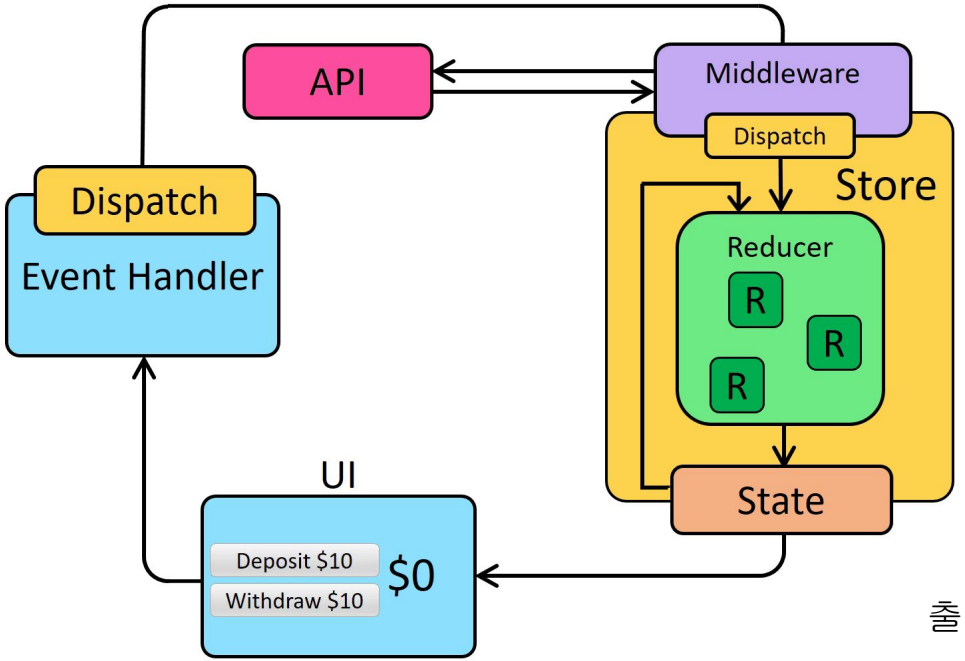
출처: https://redux.js.org/

용어



출처: https://redux.js.org/

용어



출처: https://redux.js.org/

용어 - Action

- Action은 애플리케이션에서 Redux 스토어로 데이터를 보냄.
- 액션은 일반적으로 `type`과 `payload`(선택 사항)라는 두 가지 속성이 있는 객체.
- `type`은 일반적으로 작업을 설명하는 대문자 문자열(상수).
- `payload`는 전달할 수 있는 추가 데이터.
- **Action creator**: action을 리턴하는 함수.

```
// action type
const DELETE_TODO = 'posts/deleteTodo'

// action
{
  type: DELETE_TODO,
  payload: id,
}

// action creator
const deleteTodo = (id) => ({
  type: DELETE_TODO,
  payload: id
})
```

용어 - Reducer

- Reducer는 상태(state)와 동작(action)이라는 두 가지 파라미터를 갖는 함수
  - 항상 전체 상태의 복사본을 반환.
  - 일반적으로 가능한 모든 action type을 처리하는 switch 문으로 구성.

```
const initialState = {
  todos: [
    { id: 1, text: 'Eat' },
  ],
  loading: false,
};

function todoReducer(state = initialState,
action) {
  switch (action.type) {
    case DELETE_TODO:
      return {
        ...state,
        todos: state.todos.filter((todo) =>
               todo.id !== action.payload),
      };
    default:
      return state
  }
}
```

용어 - Store

- Redux 애플리케이션 상태(state)는 리듀서에 의해 초기화되는 저장소(store)에 상주.
- Redux를 React와 함께 사용할 때, <Provider>가 애플리케이션을 감싸고, <Provider> 안의 모든 것은 Redux에 액세스할 수 있음.

```
import { createStore } from 'redux';
import { Provider } from 'react-redux';
import reducer from './reducers';

const store = createStore(reducer);

render(
  <Provider store={store}>
    <App />
  </Provider>,
  document.getElementById('root')
);
```

용어 - Dispatch

- Redux 상태를 업데이트하는 데 사용되는 객체를 받아들이는 저장소 객체(store object)에 사용할 수 있는 메서드.
- 일반적으로 이 객체는 action creator를 호출한 결과.

```
const Component = ({ dispatch }) => {
  useEffect(() => {
    dispatch(deleteTodo())
  }, [dispatch])
}
```

용어 - Connect

- connect() 함수는 React를 Redux에 연결하는 일반적인 방법 중 하나.
- 연결된 컴포넌트를 컨테이너라고도 함.

```
const mapStateToProps = (state) => ({
  post: state.post.post,
  loading: state.post.loading,
  hasErrors: state.post.hasErrors,
});

export default
connect(mapStateToProps)(PostPage);
```

RTK(Redux Toolkit)

RTK 장점:

- 쉬운 설정(적은 의존성)
- 상용구(boilerplate) 코드 감소.
  - 하나의 슬라이스(slice) 파일 vs 많은 파일들 (action과 reducer를 작성을 위해서)
- Redux Thunk, Redux DevTools 등이 빌트-인.
- immer 라이브러리를 사용하기 때문에, 직접 상태 변경이 가능.
  - {...state} 와 같은 코드를 더 이상 사용할 필요가 없음.