Redux : App 전체의 state를 Store로 관리하며 순수함수인 Reducer로만 상태 변경 가능

* Reducer : type과 payloads를 속성으로 갖는 단순 객체. Action 이벤트 발생 시 동작.
* Action Event : dispatch 함수에 단순 객체인 Action을 전달하여 발생.
* State 변화 추적 가능 -> 변화 예측 및 디버깅 용이
* 순수 함수로 test 코드 작성 가능
* Generator 객체를 사용하여 비동기 처리
* Dispatch(Action) -> Reducer Run -> Store state 변경 -> 변경된 State가 이를 subscribe 중인 컴포넌트에 전달
* Action은 단순 객체로 동적 처리를 할 수 없어 Action을 생성하는 함수인 action creator를 생성하여 사용함. 이를 편하게 하기 위해 미들웨어 사용. (redux thunk, redux saga…)

================================================================

Redux-Saga : Redux 미들웨어. Redux 비동기 처리를 지원해주는 라이브러리

* Action = Event / Reducer = Event Process

takeEvery() : GET 메소드에 사용

* Import { takeEvery } from ‘redux-saga/effects’

function\* watchMessages() {

yield takeEvery('ADD\_MESSAGE', postMessageToServer)

}

* ‘ADD\_MESSAGE’ Type이 호출될 때 마다 postMessageToServer 함수 호출

takeLatest() : PUSH, PUT, DELETE와 같은 리소스 변경 메소드에 사용

* takeEvery()와 유사하지만 한 번에 하나의 함수 핸들러만 실행
* 동시성 회피
* 처리기 실행 중 다른 작업 시작 시, 기존 처리기 취소 후 최신 데이터로 재실행

take()

* 해당 동작 발생 후 정지

put() : 액션 호출, dispatch로 취급 가능

* Redux Store에 저장시키는 함수

yield put({ type: 'INCREMENT' })

yield put({ type: "USER\_FETCH\_SUCCEEDED", data: data })

call()

* 호출하는 함수를 래핑하여 객체 반환
* 객체를 분석하여 테스트 용이

call(delay, 1000)

호출 시

{ CALL: {fn: delay, args: [1000]}}

객체 반환.

all()

* call()함수 동시 실행 후 전부 반환

(call로 동일 함수 호출 시 순차적으로 함수 실행하며 반환)

import { all, call } from 'redux-saga/effects'

const [todos, user] = yield all([

call(fetch, '/api/todos'),

call(fetch, '/api/user')

])

* 함수 병행 실행 가능
* 모든 함수 호출이 종료되면 반환

race()

* 하나의 함수가 리턴되면 그대로 종료되며 다른 함수들 취소
* 일반적으로 백그라운드 작업 종료할 때 사용