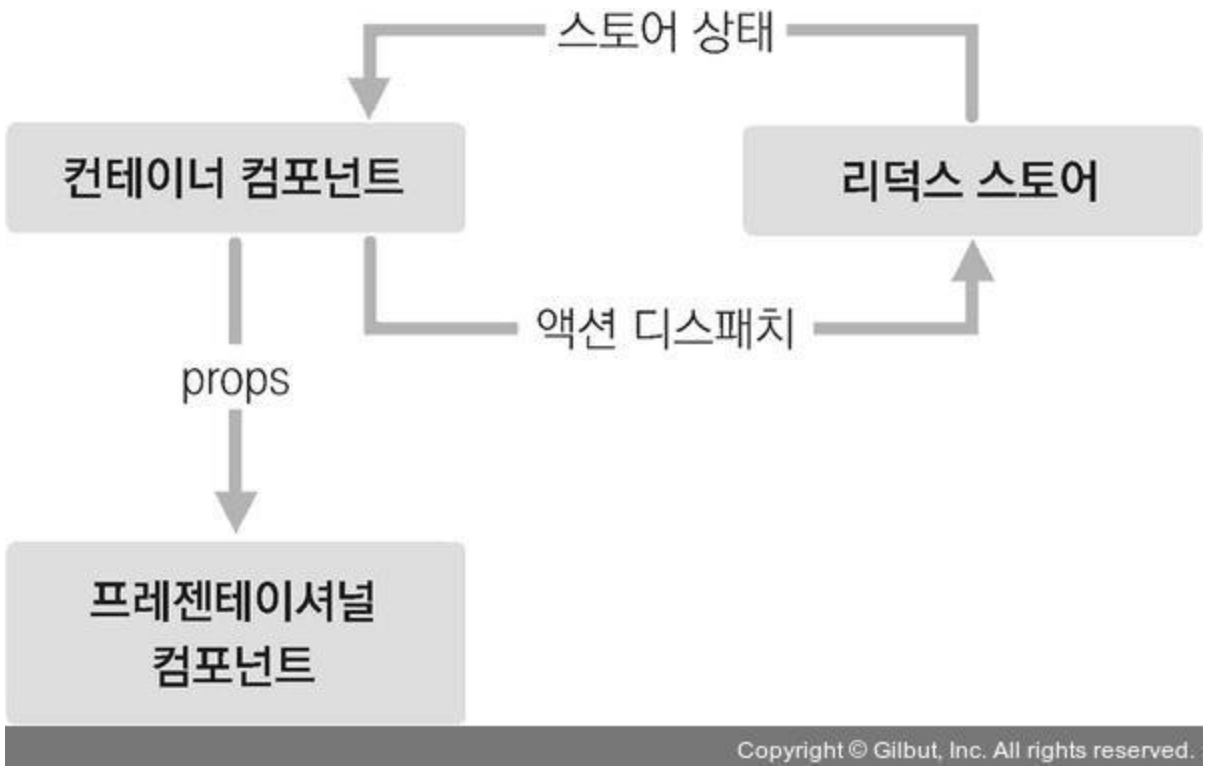
리덕스를 사용하여 리액트 애플리케이션 상태 관리하기

redux

- 상태 업데이트에 관한 로직을 모듈로 따로 분리하여 컴포넌트 파일과 별개로 관리할 수 있음
- store를 직접 사용하기보다
  react-redux 라이브러리에서 제공하는 `conect` , `Provider` 를 사용하여 redux 작업을 함

```
yarn add redux react-redux
```



- **presentational component:** `src/compoents`
  - 상태 관리가 이루어 지지 않음, props를 받아 화면에 보여주기만 함
- **container component:** `src/containers`
  - 리덕스로부터 상태를 받고 스토어에 액션을 디스패치함

1. presentational component 작성

1. `git checkout 5189de3`
2. `./components`, `App.js` 확인

2. 리덕스 관련 코드 작성

일반적인 구조

```
├── actions
│   ├── counter.js
│   └── todos.js
├── constants
│   └── ActionTypes.js
└── reducers
    ├── counter.js
    └── todos.js
```

- 기능별로 파일을 하나씩 만드는 방식
- 새로운 액션을 만들 때마다 세 종류의 파일을 모두 수정해야 함
- 공식 문서 및 Mad Talk에서 사용하는 방식

Ducks 패턴

```
└── modules
    ├── counter.jsx
    ├── index.jsx
    └── todos.jsx
```

- 기능별로 파일 하나에 다 몰아서 작성하는 방식
- 본 예제에서 사용 함
- `action`, `action creator`, `reducer` 작성
- 하나의 reducer로 합치기 위해 `combineReducers` 사용

3. 리액트 애플리케이션에 리덕스 적용

1. `./src/index.js`에서 `store`를 만들어 리덕스 적용
2. `Provider` 컴포넌트로 감싸기
3. Redux DevTools 사용
    i. google chrome에서 Redux DevTools 확장프로그램 설치
    ii. `composeWithDevTools()` 적용
    iii. `yarn add redux-devtools-extension`

4. container component 만들기

`containers/CounterCountainer.jsx`

- `mapStateToProps(state => {})` : redux store 안의 state를 컴포넌트의 props로 넘김
- `mapDispatchToPros(dispatch => {})` : action creator를 컴포넌트의 props로 넘김
- `concet(mapStateToProps, mapDispatchToPros)(연동할 컴포넌트)` :
  리덕스와 연동된 컴포넌트 생성
- `bindActionCreators(actionCreators, dispatch)` : dispatch로 감싸는 작업을 쉽게
- `connect` 에서 `mapDispatchToProps` 를 함수 형태가 아닌 액션 생성 함수로 이루어진 객체로 넣을 수 있음

`containers/TodosContainer.jsx` , `components/Todos.jsx` 구현

- `git checkout 6625e18`

리덕스 더 편하게 사용하기

`./modules`

redux-actions

- `yarn add redux-actions`
- `createAction(payloadCreator)` : action 생성
- `handleActions(reducer, defaultState)` : reducer 생성
  - `{ payload: name }` 비구조화 문법을 사용

immer

- `yarn add immer`
- 상태의 구조가 깊을 수록 불변성을 지키기 까다롭기 떄문에 편리한 상태 관리를 위해 사용

Hooks를 사용하여 컨테이너 컴포넌트 만들기

`connect()`를 사용하는 대신 `react-redux`에서 제공하는 Hooks를 사용할 수 있음

- `useSelector(mapStateToProps)` : state를 획득하는 함수
- `useDispatch()` : 내부 스토어의 내장 함수 `dispatch()`를 사용할 수 있게 함
  성능 최적화를 위해 `useCallback()`으로 감싸주는 것 이 좋음
- `useStore()` : 직접 리덕스 스토어에 접근할 수 있음. 사용하는 상황은 흔치 않음

```
const store = useStore();
store.dispatch({ type: 'SAMPLE_ACTION' });
store.getState();
```

connect 함수와의 주요 차이점

- `connect()` : 컨테이너 컴포넌트의 부모 컴포넌트가 리렌더링 될 때 해당 컨테이너 컴포넌트의 props가 바뀌지 않았다면 리렌더링이 자동으로 방지 됨
- `useSelector()` : 최적화 자동이 자동으로 이루어 지지 않기 때문에, `React.memo` 를 사용해 주아야 함
    - `TodosContainer` 의 부모 컴포넌트가 리렌더링 하지 않기 때문에 불필요한 최적화임