

# redux applyMiddleware

@Migo F2E Duncan

Create Store Flow

```
const newCreateStore = applyMiddleware(middlewareA)(createStore);  
const store = newCreateStore(reducer, {});
```

middlewareA

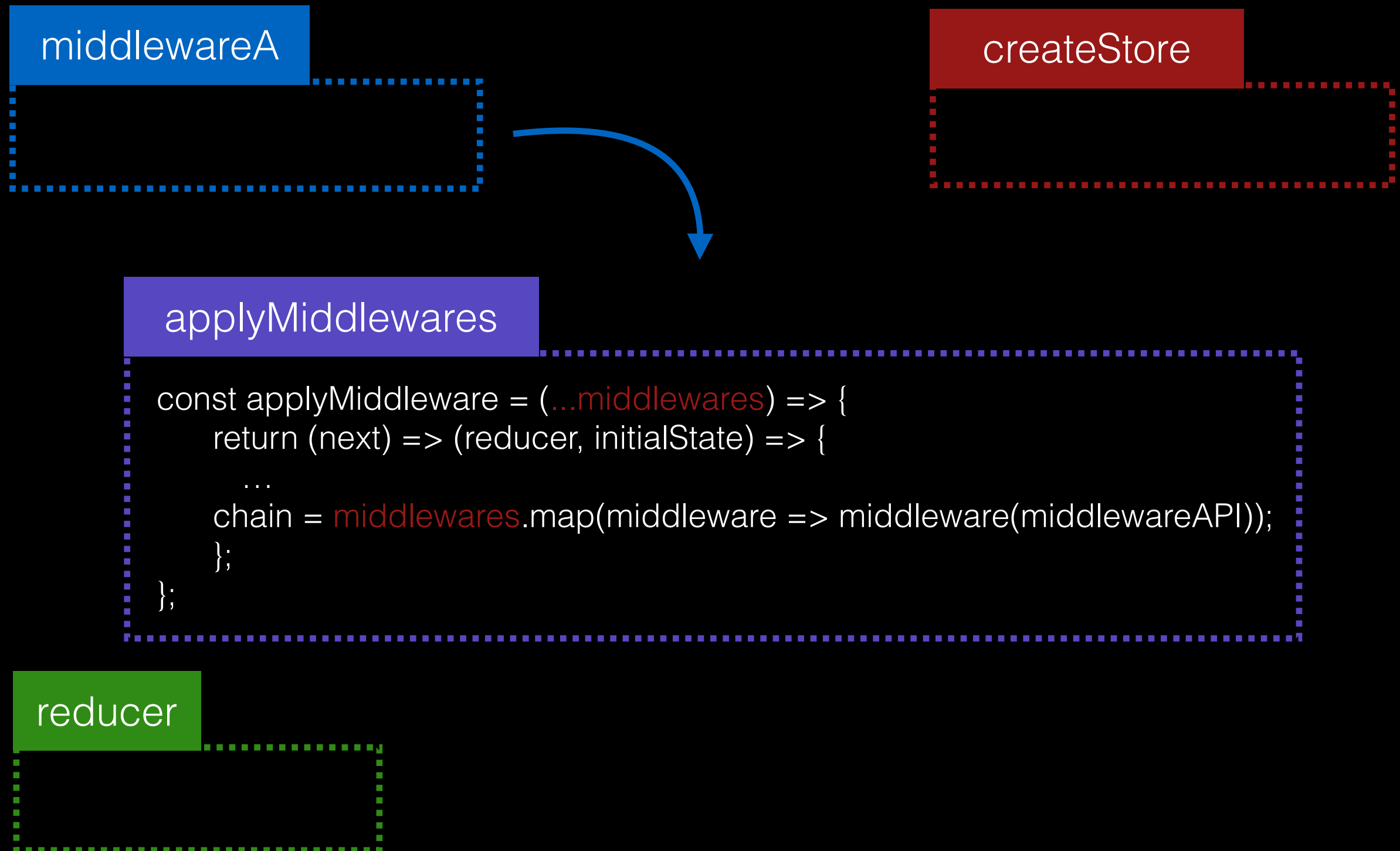
createStore

applyMiddlewares

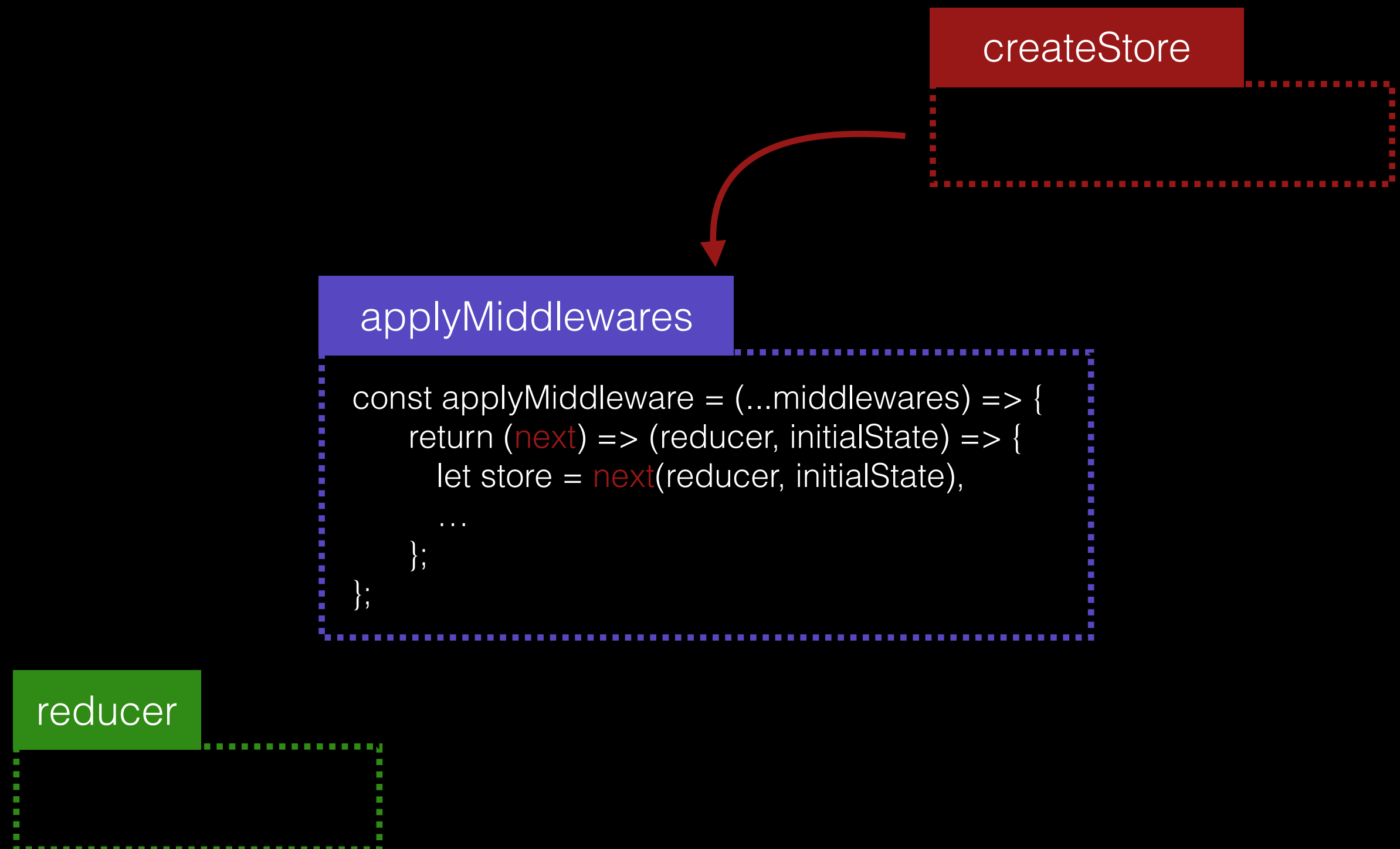
```
const applyMiddleware = (...middlewares) => {  
  return (next) => (reducer, initialState) => {  
    ...  
  };  
};
```

reducer

```
const newCreateStore = applyMiddleware(middlewareA)(createStore);  
const store = newCreateStore(reducer, {});
```



```
const newCreateStore = applyMiddleware(middlewareA)(createStore);  
const store = newCreateStore(reducer, {});
```



```
const newCreateStore = applyMiddleware(appMiddlewareA)(createStore);  
const store = newCreateStore(reducer, {});
```

applyMiddlewares

```
const applyMiddleware = (...middlewares) => {  
  return (next) => (reducer, initialState) => {  
    let store = next(reducer, initialState),  
  
    ...  
  };  
};
```

reducer



## applyMiddlewares

```
let store    = next(reducer, initialState),
    dispatch = store.dispatch,
    chain    = [];
let middlewareAPI = {
  getState: store.getState,
  dispatch: (action) => dispatch(action)
};
chain = middlewares.map(middleware => middleware(middlewareAPI));
dispatch = compose(...chain)(store.dispatch)
return {
  ...store,
  dispatch
};
```

view

```
store.dispatch({  
  type: 'TODO'  
});
```

middleware chain

store

```
const dispatch = (action) => {  
  state = reducer(state, action);  
  listeners.forEach(listener => listener());  
};
```

reducer

```
const reducer = (state = 0, action) => {  
  switch (action.type) {  
    case 'TODO':  
      return state + 1;  
  }  
};
```



view

```
store.dispatch({  
  type: 'TODO'  
});
```



middleware chain

middleware chain



The diagram illustrates the flow of data in Redux. On the left, a blue box labeled 'middleware chain' has a dashed blue border. A blue arrow points from this box to a red box on the right labeled 'store'. The 'store' box has a dashed red border and contains a code snippet for the dispatch function. The code is color-coded: 'const dispatch' is red, '(action)' is blue, '=>' is white, '{' is white, 'state = reducer' is green, '(state, action);' is white, 'listeners.forEach' is green, '(listener => listener())' is white, and ';' is white.

store

```
const dispatch = (action) => {  
  state = reducer(state, action);  
  listeners.forEach(listener => listener());  
};
```

store

```
const dispatch = (action) => {  
  state = reducer(state, action);  
  listeners.forEach(listener => listener());  
};
```



reducer

```
const reducer = (state = 0, action) => {  
  switch (action.type) {  
    case 'TODO':  
      return state + 1;  
  }  
};
```

store

```
const dispatch = (action) => {  
  state = reducer(state, action);  
  listeners.forEach(listener => listener());  
};
```

reducer


```
const reducer = (state = 0, action) => {  
  switch (action.type) {  
    case 'TODO':  
      return state + 1;  
  }  
};
```

view

```
store.subscribe(() => {  
  const state = store.getState();  
});
```

store

```
const dispatch = (action) => {  
  state = reducer(state, action);  
  listeners.forEach(listener => listener());  
};
```



view

```
store.subscribe(() => {  
  const state = store.getState();  
});
```

store

```
const getState = () => state;
```

view

```
store.subscribe(() => {  
  const state = store.getState();  
});
```

store

```
const getState = () => state;
```

