createReducer

Đây là cách tiếp cận reducer kiểu cũ

```
const initialState = { value: 0 }

function counterReducer(state = initialState, action) {
  switch (action.type) {
    case 'increment':
      return { ...state, value: state.value + 1 }
    case 'decrement':
      return { ...state, value: state.value - 1 }
    case 'incrementByAmount':
      return { ...state, value: state.value + action.payload }
    default:
      return state
  }
}
```

Cách này khá là phức tạp và dễ gây lỗi, những lỗi này có thể đến từ việc

- Bạn mutate state
- Bạn quên return một state mới
- Bạn quên khai báo default case

Với `createReducer` chúng ta có thể dễ dàng giải quyết các vấn đề trên:

- Chúng ta có thể mutate state nhờ vào thư viện immer được tích hợp bên trong
- Không cần thiết phải return một state mới
- Không cần khai báo default case

Cách dùng với "Builder Callback"

`createReducer` nhận vào 2 param

- **initialState** `State | () => State`: Là state khởi tạo hoặc một function khởi tạo state, function này khá hữu ích khi ta muốn lấy state từ localstorage
- **builderCallback** `(builder: Builder) => void`: Đây là callback nhận vào tham số là Builder object dùng để định nghĩa các case cho reducer

```
import {
  createAction,
  createReducer,
  AnyAction,
  PayloadAction
} from '@reduxjs/toolkit'
```

```
const increment = createAction<number>('increment')
const decrement = createAction<number>('decrement')

function isActionWithNumberPayload(
  action: AnyAction
): action is PayloadAction<number> {
  return typeof action.payload === 'number'
}

const reducer = createReducer(
  {
    counter: 0,
    sumOfNumberPayloads: 0,
    unhandledActions: 0
  },
  (builder) => {
    builder
      // Dùng addCase để thêm case trong trường hợp dùng createAction
      .addCase(increment, (state, action) => {
        // mutate state dễ dàng nhờ immer xử lý bên trong
        // không cần phải return state mới
        state.counter += action.payload
      })
      // Thêm case bằng cách dùng .addCase như muỗi chuỗi line
      .addCase(decrement, (state, action) => {
        state.counter -= action.payload
      })
      // addMatcher cho phép chúng ta thêm "matcher function"
      // nếu "matcher function" return true thì nó sẽ nhảy vào case này
      .addMatcher(isActionWithNumberPayload, (state, action) => {})
      // nếu muốn thêm default case khi không match case nào cả
      // thì dùng addDefaultCase
      .addDefaultCase((state, action) => {})
  }
)
```

Cách dùng với "Map Object"

Cách này thì ngắn hơn Builder Callback như chỉ hoạt động với Javascript, Typescript thì không hoạt động ổn và ít tương thích với các IDE. Team Redux khuyên dùng Builder Callback hơn là cách này

Nếu dispatch một action thông thường nào mà có type là 'increment' hoặc 'decrement' thì sẽ nhảy vào case dưới

```
const incrementAction = {
  type: 'increment'
}
```

```
const counterReducer = createReducer(0, {
  increment: (state, action) => state + action.payload,
  decrement: (state, action) => state - action.payload
})
```

Có thể dùng chung với action mà được tạo từ `createAction`

```
const increment = createAction('increment')
const decrement = createAction('decrement')

const counterReducer = createReducer(0, {
  [increment]: (state, action) => state + action.payload,
  [decrement.type]: (state, action) => state - action.payload
})
```

Log giá trị draft state trong reducer

draft state là thuật ngữ bên immer, nghĩa là state nháp, đang trong quá trình tính toán

Khi chúng ta log state này, trình duyệt sẽ hiển thị ra định dạng Proxy khó đọc.

Khi sử dụng `createReducer` hay `createSlice` thì bạn có thể import `current` để phục vụ cho việc log draft state

```
import { createSlice, current } from '@reduxjs/toolkit'

const slice = createSlice({
  name: 'todos',
  initialState: [{ id: 1, title: 'Example todo' }],
  reducers: {
    addTodo: (state, action) => {
      console.log('before', current(state))
      state.push(action.payload)
      console.log('after', current(state))
    }
  }
})
```