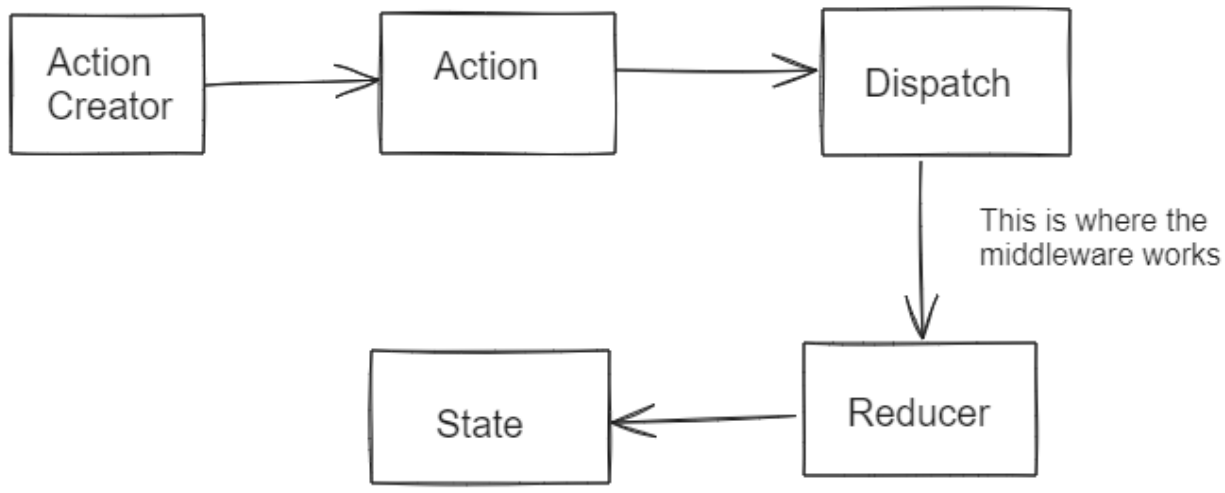
Redux-Thunk

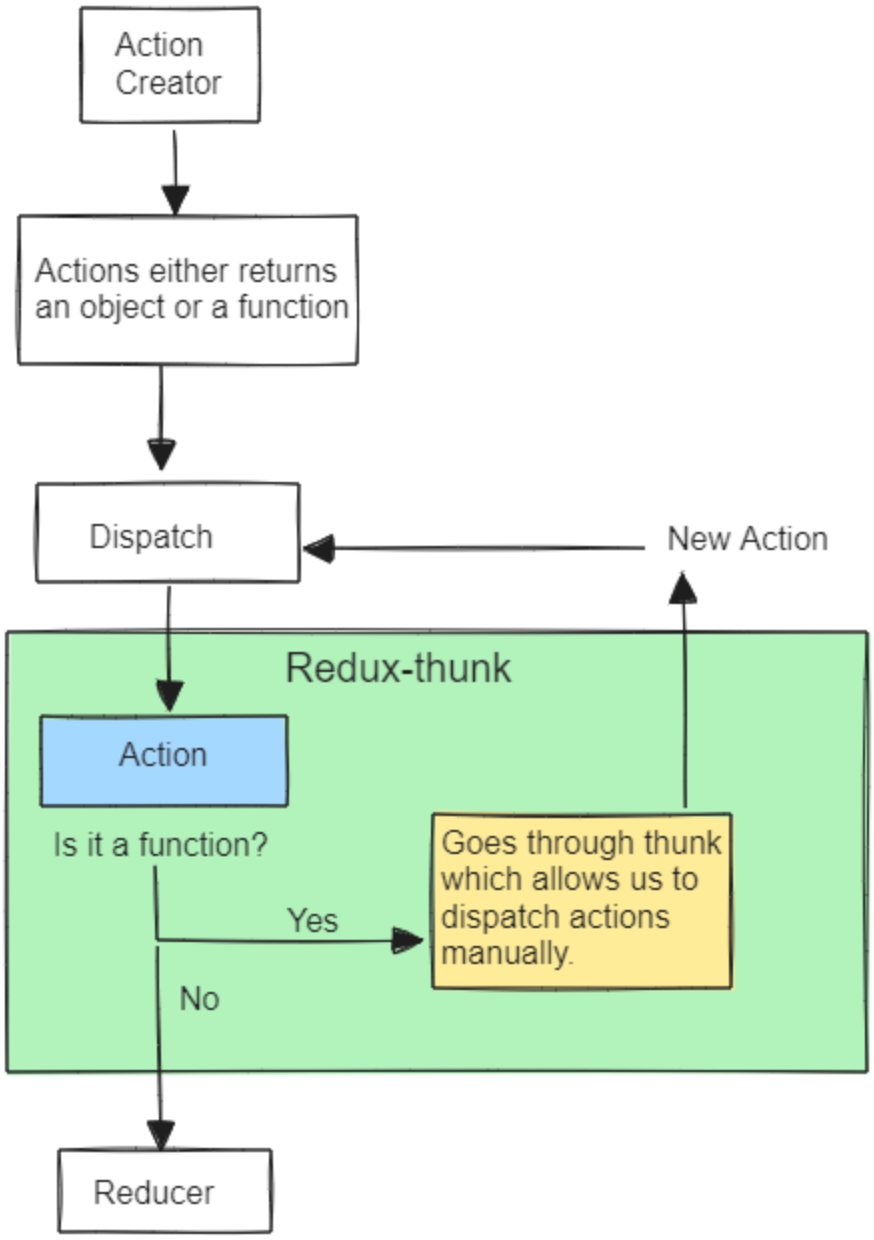
Redux-thunk is a middleware for Redux, a popular state management library for JavaScript applications. It enables asynchronous actions in Redux by allowing action creators to return functions instead of plain objects. This is particularly useful for handling asynchronous operations such as API requests.

Plain redux doesn't allow complex logic inside action functions, you can only perform simple synchronous updates by dispatching actions. This middleware extends its ability and lets you write complex logic that interacts with the store. Thunk doesn't interfere with the action until it returns a function. Thunk allows us to dispatch actions manually, which gives us the power to incorporate some logic or run some asynchronous code before dispatching an action.

Redux flow without thunk:



Redux flow with thunk:



Here's an overview of how redux-thunk works and how to use it:

1. Understanding Redux Middleware:

Middleware in Redux provides a way to interact with actions that are dispatched to the store before they reach the reducer.

Middlewares can be used for various purposes, such as logging, crash reporting, and handling asynchronous actions.

2. Installing redux-thunk:

You need to install redux-thunk as a dependency in your project:

```
npm install redux-thunk
```

or

```
yarn add redux-thunk
```

3. Configuring redux-thunk in Your Redux Store:

Apply redux-thunk middleware when creating your Redux store using applyMiddleware from redux.

```
import { createStore, applyMiddleware } from 'redux';

import thunk from 'redux-thunk';

import rootReducer from './reducers' ;

 const store = createStore(rootReducer, applyMiddleware(thunk));
```

4. Using redux-thunk for Asynchronous Actions:

With redux-thunk, action creators can return functions instead of plain objects.

These functions receive the **dispatch** and **getState** functions as arguments, allowing them to dispatch multiple actions and access the current state.

The inner function can be synchronous or asynchronous, making it suitable for handling asynchronous operations.

```
// Example of an asynchronous action using redux-thunk
```

```
const fetchData = () => {
  return (dispatch, getState) => {
        // You can dispatch multiple actions, including asynchronous API requests
        dispatch({ type: 'FETCH_DATA_REQUEST' });

        // Simulate an API request (e.g., using fetch)
    fetch('https://api.example.com/data')
        .then(response => response.json())
        .then(data => {
        // Dispatch another action with the received data
        dispatch({ type: 'FETCH_DATA_SUCCESS', payload: data });
        })
        .catch(error => {
        // Dispatch an error action in case of failure
        dispatch({ type: 'FETCH_DATA_FAILURE', error: error.message });
        });
  };
};
```

5. Dispatching Thunk Actions:

You can dispatch thunk actions just like any other actions in your application.

```
import { useDispatch } from 'react-redux';
import { fetchData } from './actions';
 const MyComponent = () => {
  const dispatch = useDispatch();
   const handleClick = () => {
        // Dispatch the thunk action
    dispatch(fetchData());
  };
   return (
        <button onClick={handleClick}>Fetch Data</button>
  );
};
```

6. Async/Await with redux-thunk:

You can also use async functions with redux-thunk by returning a function that uses async/await syntax.

```
const fetchData = () => {
  return async (dispatch, getState) => {
        try {
```

```
        dispatch({ type: 'FETCH_DATA_REQUEST' });

        const response = await fetch('https://api.example.com/data');
        const data = await response.json();

        dispatch({ type: 'FETCH_DATA_SUCCESS', payload: data });
        } catch (error) {
        dispatch({ type: 'FETCH_DATA_FAILURE', error: error.message });
        }
  };
};
```

In summary, redux-thunk is a middleware that allows you to write action creators that return functions, enabling the handling of asynchronous operations in your Redux applications. It's a powerful tool for managing complex state logic, especially when dealing with asynchronous data fetching and API requests.