ReactJS REDUX Summary

Using stores and reducers

A simple example

```
import {createStore} from 'redux'

var reducer_1 = (state, action) => {
  console.log('reducer_0 was called with state', state, 'and action', action);
};
var store_1 = createStore(reducer_1);
```

Real world example

```
import {createStore} from 'redux'

var reducer_1 = (state = {}, action) => {
  switch(action.type) {
  case '':
  return {
    ...state,
    message: action.value
  };
  default:
    return state;
  }
};
var store_1 = createStore(reducer_1);
```

Combining reducers

```
import {combineReducers, createStore} from 'react';

var userReducer = (state = {}, action) => {
  switch(action) {
    case 'ADD_USER':
    return {
      //Return modified state
  };
  default:
    return state;
  }
};

var itemReducer = (state = [], action) => {
  switch(action) {
    case 'ADD_ITEM':
      return {
        //Return modified state
      };
    default:
      return state;
  }
};

var reducers = combineReducers(user: userReducer, item: itemReducer);
var store = createStore(reducers);
console.log(store.getState());
// {
// user: {}, // {} is the slice returned by our userReducer
// items: [] // [] is the slice returned by our itemsReducer
// }
```

Dispatching an action

Flow of application: ActionCreator -> Action -> dispatcher -> reducer

Without action creator

```
import {combineReducers, createStore} from 'react';

var userReducer = (state = {}, action) => {
  switch(action) {
  case 'ADD_USER':
    return {
      //Return modified state
    };
    default:
      return state;
  }
};

var itemReducer = (state = [], action) => {
  switch(action) {
    case 'ADD_ITEM':
      return {
        //Return modified state
      };
      default:
        return state;
    }
};
```

```
var reducers = combineReducers(user: userReducer, item: itemReducer);
var store = createStore(reducers);
store.dispatch({type: 'ACTION'});
```

With Action Creator (Adopted from Flux)

```
import {combineReducers, createStore} from 'react';

var userReducer = (state = {}, action) => {
  switch(action) {
    case 'ADD_USER':
      return {
        //Return modified state
      };
    default:
      return state;
  }
};

var itemReducer = (state = [], action) => {
  switch(action) {
    case 'ADD_ITEM':
      return {
        //Return modified state
      };
    default:
      return state;
  }
};

var reducers = combineReducers(user: userReducer, item: itemReducer);
var store = createStore(reducers);
var addItemActionCreator = (name) => {
  return {
    item: name,
    type: 'ADD_ITEM'
  };
};
store.dispatch(addItemActionCreator);
```

Async Action with Middleware

```
import {combineReducers, createStore, applyMiddleware} from 'react';

var userReducer = (state = {}, action) => {
  switch(action) {
    case 'ADD_USER':
      return {
        //Return modified state
      };
    default:
      return state;
  }
};

var itemReducer = (state = [], action) => {
  switch(action) {
    case 'ADD_ITEM':
      return {
        //Return modified state
      };
    default:
      return state;
    }
};

//Set the state after 2s
var addItemActionCreator = (name) => {
  return (dispatch) => {
    setTimeout(() => {
      dispatch({
        item: name,
        type: 'ADD_ITEM'
      });
    }, 2000);
  };
};

// 1) The first level provide the dispatch function and a getState function (if your
// middleware or your action creator needs to read data from state) to the 2 other levels
// 2) The second level provide the next function that will allow you to explicitly hand over
// your transformed input to the next middleware or to Redux (so that Redux can finally call all
// 3) the third level provides the action received from the previous middleware or from your dis
// and can either trigger the next middleware (to let the action continue to flow) or process
// the action in any appropriate way.
var thunkMiddleware = ({dispatch, getState}) => {
  return (next) => {
    return (action) => {
      return typeof action === 'function' ? action(dispatch, getState) : next(action);
    }
  };
};

var reducers = combineReducers(user: userReducer, item: itemReducer);
var finalCreateStore = applyMiddleware(thunkMiddleware)(createStore);
var store = finalCreateStore(reducers);
store.dispatch(addItemActionCreator);
```

Subscribing to a Store

```
import {createStore} from 'redux'
```

```
var reducer_1 = (state = {}, action) => {
  switch(action.type) {
    case '':
      return {
        ...state,
        message: action.value
      };
    default:
      return state;
  }
};
var store_1 = createStore(reducer_1);
store_1.subscribe(() => {
  //Update react views
});
```

Revision #1
Created 5 April 2017 22:59:50 by Tingwai
Updated 5 April 2017 23:10:43 by Tingwai