Redux学习总结

Created and last modified by 王文磊 on Mar 11, 2022

一、简介

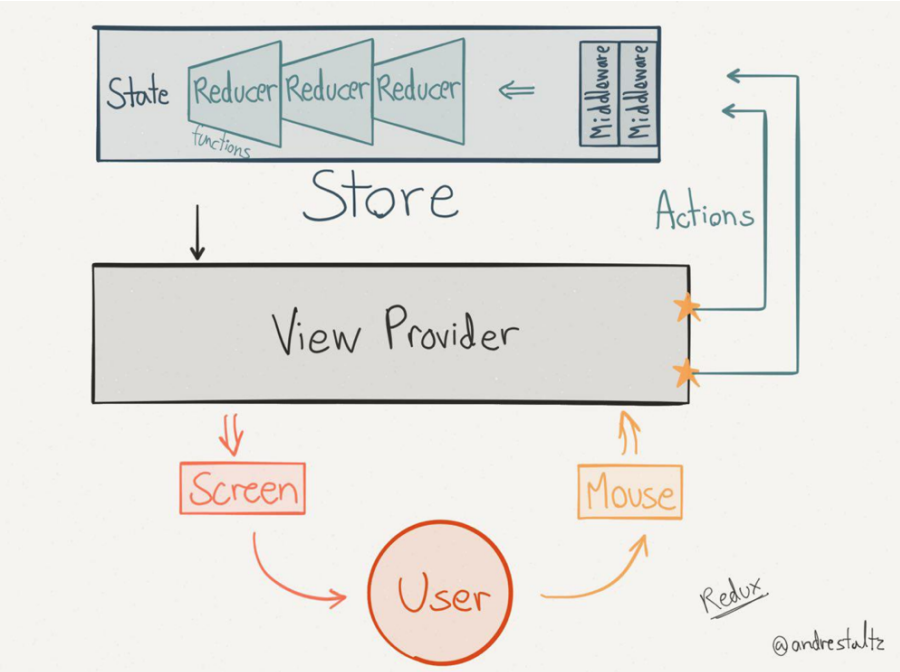
背景：

随着基于 JS 开发的多端应用日趋复杂，前端项目需要管理更多的 state（状态）。如：服务器响应数据、缓存数据、本地生成尚未持久化到服务器的数据（表单、图片等上传资源），以及 UI 所依赖的状态：路由 path 、选中的标签、列表数据、展示控制、分页等等。

管理不断变化的 state 是非常困难的。如果一个 model 与另一个 model 耦合，当对应的 view 变化时就可能引起对应 model 以及另一个 model 的变化，以至于另一个 view 也相应变化。当项目体积随着迭代开发而越来越大时，如此多耦合、复杂的 state 及变化会让问题排查、新功能的开发如履薄冰。

Recat在视图层禁止异步和直接操作DOM来解决了这个问题，美中不足的是依旧把处理 state 中数据的问题留给了开发者，而 Redux 就是这类问题的解决方式之一。

核心概念：



Redux 维护了一个全局唯一的 state，这个 state 就像 Model，区别是它没有 setter。因此其他代码不能随意修改它。

想要更新 state 需要发起一个 action，Action就是一个普通的JS对象：

```
{ type: 'ADD_TODO', text: 'Go to swimming pool' }
{ type: 'TOGGLE_TODO', index: 1 }
{ type: 'SET_VISIBILITY_FILTER', filter: 'SHOW_ALL' }
```

强制使用 action 来更新 state 的好处是可以清晰知道应用中到底发生了什么。如果 state 中某个数据发生改变了，开发者就可以清晰的知道是哪个或哪几个 action 发生了，而不用全局搜索何处有对该数据的 set。

最后，为了把 action 和 state 串联起来，我们还需要开发一些函数即 reducer，reducer 接收 上一次的 state 以及 dispatch 发起的 action：

```
function visibilityFilter(state = 'SHOW_ALL', action) {
  if (action.type === 'SET_VISIBILITY_FILTER') {
    return action.filter;
  } else {
    return state;
  }
}

function todos(state = [], action) {
  switch (action.type) {
  case 'ADD_TODO':
    return state.concat([{ text: action.text, completed: false }]);
  case 'TOGGLE_TODO':
    return state.map((todo, index) =>
      action.index === index ?
        { text: todo.text, completed: !todo.completed } :
        todo
   )
  default:
    return state;
  }
}
```

可以再用一个 reducer 将这两个 reducer 结合：

```
function todoApp(state = {}, action) {
  return {
    todos: todos(state.todos, action),
    visibilityFilter: visibilityFilter(state.visibilityFilter, action)
  };
}
```

Like Be the first to like this