MobX 6

1. 概述

MobX 是一个简单的可扩展的状态管理库，无样板代码风格简约。

目前最新版本为 6，版本 4 和版本 5 已不再支持。

在 MobX 6 中不推荐使用装饰器语法，因为它不是 ES 标准，并且标准化过程要花费很长时间，但是通过配置仍然可以启用装饰器语法。

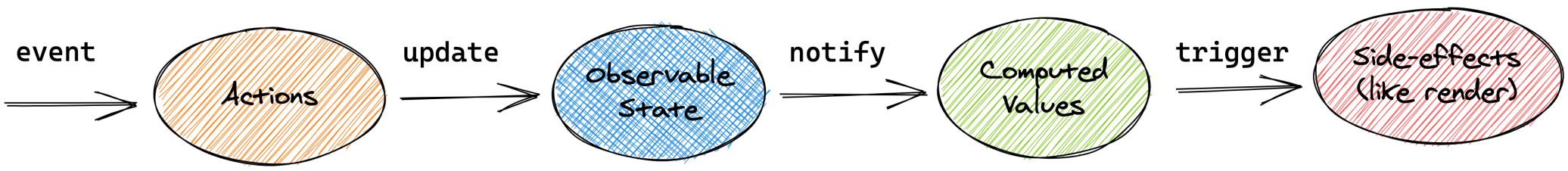
MobX 可以运行在任何支持 ES5 的环境中，包含浏览器和 Node。

MobX 通常和 React 配合使用，但是在 Angular 和 Vue 中也可以使用 MobX。

2. 核心概念

1. observable：被 MobX 跟踪的状态。
2. action：允许修改状态的方法，在严格模式下只有 action 方法被允许修改状态。
3. computed：根据现有状态衍生出来的状态。
4. flow：执行副作用，它是 generator 函数。可以更改状态值。

3. 工作流程



4. 下载

- mobx：MobX 核心库
- mobx-react-lite：仅支持函数组件
- mobx-react：既支持函数组件也支持类组件

```
yarn add mobx@6.3.1 mobx-react-lite@3.2.0
```

5. 案例驱动之计数器

在组件中显示数值状态，单击[+1]按钮使数值加一，单击[-1]按钮使数值减一。

1. 创建用于存储状态的 Store

```
export default class CounterStore {
  constructor() {
    this.count = 0
  }
}
```

2. 创建用于修改状态的方法

```
export default class CounterStore {
  constructor() {
    this.count = 0
  }
  increment() {
    this.count += 1
  }
  decrement() {
    this.count -= 1
  }
}
```

3. 让 MobX 可以追踪状态的变化
    1. 通过 observable 标识状态，使状态可观察
    2. 通过 action 标识修改状态的方法，状态只有通过 action 方法修改后才会通知视图更新

```
import { action, makeObservable, observable } from "mobx"

export default class CounterStore {
  constructor() {
    this.count = 0
    makeObservable(this, {
      count: observable,
      increment: action,
      decrement: action
    })
  }
  increment() {
    this.count += 1
  }
  decrement() {
    this.count -= 1
  }
}
```

4. 创建 Store 类的实例对象并将实例对象传递给组件

```
// App.js
import Counter from "./Counter"
import CounterStore from "../store/Counter"
const counterStore = new CounterStore()

function App() {
  return <Counter counterStore={counterStore} />
}

export default App
```

5. 在组件中通过 Store 实例对象获取状态以及操作状态的方法

```
function Counter({ counterStore }) {
  return (
    <Container>
      <Button onClick={() => counterStore.increment()}>
        INCREMENT
      </Button>
      <Button>{counterStore.count}</Button>
      <Button onClick={() => counterStore.decrement()}>
        DECREMENT
      </Button>
    </Container>
  )
}

export default Counter
```

6. 当组件中使用到的 MobX 管理的状态发生变化后，使视图更新。通过 observer 方法包裹组件实现目的

```
import { observer } from "mobx-react-lite"

function Counter() { }

export default observer(Counter)
```

7. 简化组件代码

```
function Counter({ counterStore }) {
  const { count, increment, decrement } = counterStore
  return (
    <Container>
      <Button border="left" onClick={increment}>
        INCREMENT
      </Button>
      <Button>{count}</Button>
      <Button border="right" onClick={decrement}>
        DECREMENT
      </Button>
    </Container>
  )
}
```

8. 当代码简化后，修改状态的方法中的 this 指向出现了问题，通过 action.bound 强制绑定 this，使 this 指向 Store 实例对象

```
import { action, makeObservable, observable } from "mobx"

export default class CounterStore {
  constructor() {
    this.count = 0
    makeObservable(this, {
      count: observable,
      increment: action.bound,
      decrement: action.bound
    })
  }
  increment() {
    this.count += 1
  }
  decrement() {
    this.count -= 1
  }
}
```

9. 总结：状态变化更新视图的必要条件
    1. 状态必须被标记为 `observable`
    2. 更改状态的方法必须被标记为 `action`
    3. 组件必须通过 `observer` 方法包裹
10. 创建 RootStore

    在应用中可存在多个 Store，多个 Store 最终要通过 RootStore 管理，在每个组件都需要获取到 RootStore。

```
// store/index.js
import { createContext, useContext } from "react"
import CounterStore from "./Counter"

class RootStore {
  constructor() {
    this.counterStore = new CounterStore()
  }
}
const rootStore = new RootStore()
const RootStoreContext = createContext()

export const RootStoreProvider = ({ children }) => {
  return (
    <RootStoreContext.Provider value={rootStore}>
      {children}
    </RootStoreContext.Provider>
  )
}

export const useRootStore = () => {
  return useContext(RootStoreContext)
}
```

```
// App.js
import { RootStoreProvider } from "../store"
import Counter from "./Counter"

function App() {
  return (
    <RootStoreProvider>
      <Counter />
    </RootStoreProvider>
  )
}

export default App
```

```
import { observer } from "mobx-react-lite"
import { useRootStore } from "../store"

function Counter() {
  const { counterStore } = useRootStore()
  const { count, increment, decrement } = counterStore
  return (
    <Container>
      <Button onClick={increment}>
        INCREMENT
      </Button>
      <Button>{count}</Button>
      <Button onClick={decrement}>
        DECREMENT
      </Button>
    </Container>
  )
}

export default observer(Counter)
```

6. 案例驱动之 Todo

6.1 创建 Store

1. 创建用于管理 Todo 任务的 Store

```
import { makeObservable, observable } from "mobx"

export default class Todo {
  constructor(todo) {
    this.id = todo.id
    this.title = todo.title
    this.isCompleted = todo.isCompleted || false
    this.isEditing = false
    makeObservable(this, {
      title: observable,
      isCompleted: observable,
      isEditing: observable
    })
  }
}
```

2. 创建用于管理 Todo 任务列表的 Store

```
import { makeObservable, observable } from "mobx"

export default class TodoStore {
  constructor() {
    this.todos = []
    makeObservable(this, {
      todos: observable
    })
  }
}
```

6.2 添加任务

1. 创建向 todo 任务列表中添加 todo 任务的方法

```
import { action, makeObservable, observable } from "mobx"
import Todo from "./Todo"

export default class TodoStore {
  constructor() {
    this.todos = []
    makeObservable(this, {
      todos: observable,
      addTodo: action.bound
    })
  }
  addTodo(title) {
    this.todos.push(new Todo({ title, id: this.generateTodoId() }))
  }
  generateTodoId() {
    if (!this.todos.length) return 1
    return this.todos.reduce((id, todo) => (id < todo.id ? todo.id : id), 0) + 1
  }
}
```

2. 在组件中实现添加任务的逻辑

```
import { useState } from "react"
import { useRootStore } from "../../store"

function Header() {
  const [title, setTitle] = useState("")
  const { todoStore } = useRootStore()
  const { addTodo } = todoStore
  return (
    <header className="header">
      <input
        value={title}
        onChange={e => setTitle(e.target.value)}
        onKeyUp={e => {
          if (e.key !== "Enter") return
          addTodo(title)
          setTitle("")
        }}
      />
    </header>
  )
}

export default Header
```

6.3 显示任务列表

```
import { observer } from "mobx-react-lite"
import { useRootStore } from "../../store"
import Todo from "./Todo"

function Main() {
  const { todoStore } = useRootStore()
  const { todos } = todoStore
  return (
    <section className="main">
      <ul className="todo-list">
        {todos.map(todo => (
          <Todo key={todo.id} todo={todo} />
        ))}
      </ul>
    </section>
  )
}

export default observer(Main)
```

```
function Todo({ todo }) {
  return (
    <li>
      <div className="view">
        <input className="toggle" type="checkbox" />
        <label>{todo.title}</label>
        <button className="destroy" />
      </div>
      <input className="edit" />
    </li>
  )
}

export default Todo
```

6.4 加载远端任务

1. 下载 json-server： `yarn add json-server@0.16.3`
2. 创建 db.json

```
{
  "todos": [
    {
      "id": 1,
      "title": "吃饭",
      "isCompleted": false
    },
    {
      "id": 2,
      "title": "睡觉",
      "isCompleted": false
    },
    {
      "id": 3,
      "title": "打豆豆",
      "isCompleted": false
    }
  ]
}
```

3. 在 package.json 文件中添加启动命令

```
"scripts": {
    "json-server": "json-server --watch ./db.json --port 3001"
  }
```

4. 启动 json-server： `npm run json-server`
5. 在 todoStore 中添加加载任务列表的方法

```
import axios from "axios"
import { flow, makeObservable, observable } from "mobx"
import Todo from "./Todo"

export default class TodoStore {
  constructor() {
    this.todos = []
    makeObservable(this, {
      todos: observable,
      loadTodos: flow
    })
    this.loadTodos()
  }
  *loadTodos() {
    let response = yield axios.get("http://localhost:3001/todos")
    response.data.forEach(todo => this.todos.push(new Todo(todo)))
  }
}
```

6.5 更改任务状态

1. 在 Todo 类中添加修改任务是否已经完成的方法

```
export default class Todo {
  constructor() {
    makeObservable(this, {
      modifyTodoIsCompleted: action.bound
    })
  }
  modifyTodoIsCompleted() {
    this.isCompleted = !this.isCompleted
  }
}
```

2. 创建 `TodoCompleted` 组件实现逻辑

```
import { observer } from "mobx-react-lite"

function TodoCompleted({ todo }) {
  const { isCompleted, modifyTodoIsCompleted } = todo
  return (
    <input
      className="toggle"
      type="checkbox"
      checked={isCompleted}
      onChange={modifyTodoIsCompleted}
    />
  )
}

export default observer(TodoCompleted)
```

3. 在 `Todo` 组件中引用 `TodoCompleted` 组件并根据条件决定是否为 `li` 添加 `completed` 类名

```
import { observer } from "mobx-react-lite"
import TodoCompleted from "./TodoCompleted"

function Todo({ todo }) {
  return (
    <li className={todo.isCompleted ? "completed" : ""}>
      <div className="view">
        <TodoCompleted todo={todo} />
      </div>
    </li>
  )
}

export default observer(Todo)
```

6.6 删除任务

1. 在 `todoStore` 中添加实现删除任务的方法

```
import axios from "axios"
import { action, makeObservable,  } from "mobx"

export default class TodoStore {
  constructor() {
    makeObservable(this, {
      removeTodo: action.bound
    })
  }
  removeTodo(id) {
    this.todos = this.todos.filter(todo => todo.id !== id)
  }
}
```

2. 创建 `TodoDelete` 组件实现删除 todo 任务逻辑

```
import { useRootStore } from "../../store"

function TodoDelete({ id }) {
  const { todoStore } = useRootStore()
  const { removeTodo } = todoStore
  return <button className="destroy" onClick={removeTodo.bind(null, id)} />
}

export default TodoDelete
```

3. 在 `Todo` 组件调用 `TodoDelete` 组件并传入 todo ID

```
import { observer } from "mobx-react-lite"
import TodoDelete from "./TodoDelete"

function Todo({ todo }) {
  return (
    <li>
      <div className="view">
        <TodoDelete id={todo.id} />
      </div>
    </li>
  )
}

export default observer(Todo)
```

6.7 编辑任务

1. 在 todoStore 中添加更改任务是否处于编辑状态的方法

```
import { action, makeObservable } from "mobx"

export default class Todo {
  constructor(todo) {
    makeObservable(this, {
      modifyTodoIsEditing: action.bound,
    })
  }
  modifyTodoIsEditing() {
    this.isEditing = !this.isEditing
  }
}
```

2. 添加 `TodoTitle` 组件展示任务标题并为其添加双击事件，当事件发生时将任务更改为可编辑状态

```
function TodoTitle({ todo }) {
  const { title, modifyTodoIsEditing } = todo
  return <label onDoubleClick={modifyTodoIsEditing}>{title}</label>
}

export default TodoTitle
```

3. 在 `Todo` 组件中调用 `TodoTitle` 组件，并为 `li` 添加 `editing` 类名

```
import { observer } from "mobx-react-lite"
import TodoTitle from "./TodoTitle"
import classnames from "classnames"

function Todo({ todo }) {
  return (
    <li className={classnames({ completed: todo.isCompleted, editing: todo.isEditing })} >
      <div className="view">
        <TodoTitle todo={todo} />
      </div>
    </li>
  )
}

export default observer(Todo)
```

4. 创建 `TodoEditing` 组件实现编辑 todo 任务标题

```
import { useRef, useEffect } from "react"

function TodoEditing({ todo }) {
  const { title, modifyTodoTitle, isEditing } = todo
  const ref = useRef(null)
  useEffect(() => {
    if (isEditing) ref.current.focus()
  }, [isEditing])
  return (
    <input
      ref={ref}
      className="edit"
      defaultValue={title}
      onBlur={e => modifyTodoTitle(e.target.value)}
    />
  )
}

export default TodoEditing
```

5. 在 `Todo` 组件中调用 `TodoEditing` 组件并传递 todo 任务

```
import { observer } from "mobx-react-lite"
import TodoTitle from "./TodoTitle"
import classnames from "classnames"
import TodoEditing from "./TodoEditing"

function Todo({ todo }) {
  return (
    <li className={classnames({ completed: todo.isCompleted, editing: todo.isEditing })} >
      <div className="view">
        <TodoTitle todo={todo} />
      </div>
      <TodoEditing todo={todo} />
    </li>
  )
}

export default observer(Todo)
```

6.8 计算未完成任务数量

1. 在 todoStore 中添加获取未完成任务数量的派生状态

```
import axios from "axios"
import { makeObservable, computed } from "mobx"

export default class TodoStore {
  constructor() {
    makeObservable(this, {
      unCompletedTodoCount: computed
    })
  }
  get unCompletedTodoCount() {
    return this.todos.filter(todo => !todo.isCompleted).length
  }
}
```

2. 创建 `UnCompletedTodoCount` 组件实现逻辑

```
import { observer } from "mobx-react-lite"
import { useRootStore } from "../../store"

function UnCompletedTodoCount() {
  const { todoStore } = useRootStore()
  const { unCompletedTodoCount } = todoStore
  return (
    <span className="todo-count">
      <strong>{unCompletedTodoCount}</strong> item left
    </span>
  )
}

export default observer(UnCompletedTodoCount)
```

3. 在 `Footer` 组件中调用 `UnCompletedTodoCount` 组件

```
import UnCompletedTodoCount from "./UnCompletedTodoCount"

function Footer() {
  return (
    <footer className="footer">
      <UnCompletedTodoCount />
    </footer>
  )
}

export default Footer
```

6.9 任务过滤

1. 在 `todoStore` 中添加存储过滤条件的属性以及更改过滤条件的方法

```
import axios from "axios"
import { action, makeObservable, observable, } from "mobx"

export default class TodoStore {
  constructor() {
    this.filterCondition = "All"
    makeObservable(this, {
      modifyFilterCondition: action.bound,
      filterCondition: observable,
    })
  }
  modifyFilterCondition(filterCondition) {
    this.filterCondition = filterCondition
  }
}
```

2. 创建 `TodoFilter` 组件，为过滤按钮添加事件以更改过滤条件，根据过滤条件为按钮添加 `selected` 类名

```
import classNames from "classnames"
import { observer } from "mobx-react-lite"
import { useRootStore } from "../../store"

function TodoFilter() {
  const { todoStore } = useRootStore()
  const { filterCondition, modifyFilterCondition } = todoStore
  return (
    <ul className="filters">
      <li>
        <button
          onClick={() => modifyFilterCondition("All")}
          className={classNames({ selected: filterCondition === "All" })}
        >
          All
        </button>
      </li>
      <li>
        <button
          onClick={() => modifyFilterCondition("Active")}
          className={classNames({ selected: filterCondition === "Active" })}
        >
          Active
        </button>
      </li>
      <li>
        <button
          onClick={() => modifyFilterCondition("Completed")}
          className={classNames({ selected: filterCondition === "Completed" })}
        >
          Completed
        </button>
      </li>
    </ul>
  )
}

export default observer(TodoFilter)
```

3. 在 `Footer` 组件中调用 `TodoFilter` 组件

```
import TodoFilter from "./TodoFilter"

function Footer() {
  return (
    <footer className="footer">
      <TodoFilter />
    </footer>
  )
}

export default Footer
```

4. 在 `TodoStore` 中添加派生状态，根据条件获取过滤后的 todo 列表

```
import axios from "axios"
import { action, flow, makeObservable, observable, computed } from "mobx"
import Todo from "./Todo"

export default class TodoStore {
  constructor() {
    makeObservable(this, {
      filterTodos: computed
    })
  }
  get filterTodos() {
    switch (this.filterCondition) {
      case "Active":
        return this.todos.filter(todo => !todo.isCompleted)
      case "Completed":
        return this.todos.filter(todo => todo.isCompleted)
      default:
        return this.todos
    }
  }
}
```

5. 在 Main 组件获取 `filterTodos` 派生状态

```
import { observer } from "mobx-react-lite"
import { useRootStore } from "../../store"
import Todo from "./Todo"

function Main() {
  const { todoStore } = useRootStore()
  const { filterTodos } = todoStore
  return (
    <section className="main">
      <ul className="todo-list">
        {filterTodos.map(todo => (
          <Todo key={todo.id} todo={todo} />
        ))}
      </ul>
    </section>
  )
}

export default observer(Main)
```

6.10 清除已完成任务

1. 在 `TodoStore` 中添加清除已完成任务的方法

```
import axios from "axios"
import { action, makeObservable, } from "mobx"

export default class TodoStore {
  constructor() {
    makeObservable(this, {
      clearCompleted: action.bound
    })
  }
  clearCompleted() {
    this.todos = this.todos.filter(todo => !todo.isCompleted)
  }
}
```

2. 创建 `ClearCompleted` 组件实现清除已完成任务功能

```
import { useRootStore } from "../../store"

function ClearCompleted() {
  const { todoStore } = useRootStore()
  const { clearCompleted } = todoStore
  return (
    <button className="clear-completed" onClick={clearCompleted}>
      Clear completed
    </button>
  )
}

export default ClearCompleted
```

3. 在 `Footer` 组件中调用 `ClearCompleted` 组件

```
import ClearCompleted from "./ClearCompleted"

function Footer() {
  return (
    <footer className="footer">
      <ClearCompleted />
    </footer>
  )
}

export default Footer
```