MobX

@Phurunbit Tech Ltd, 2020

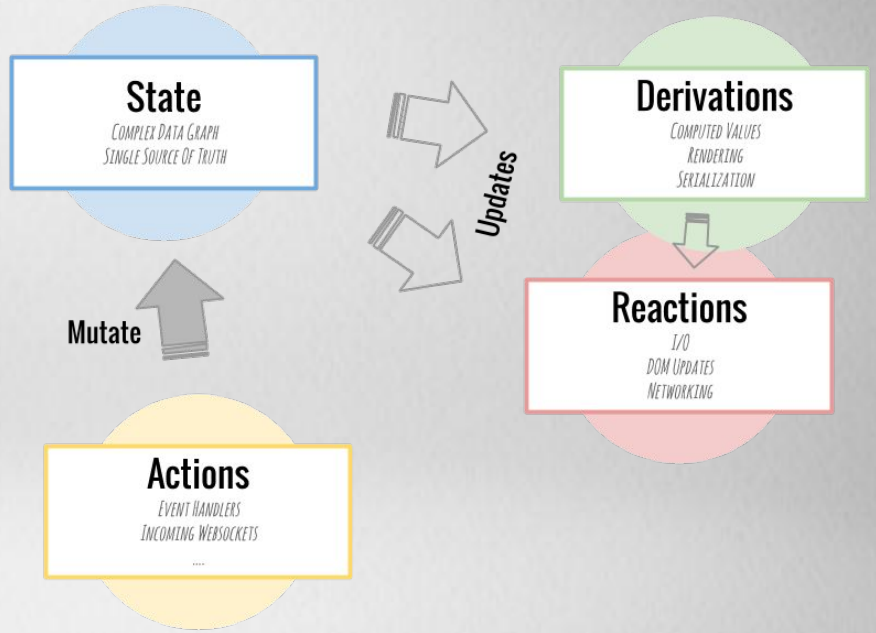
Agenda

1. Introduction
2. Mobx Core
3. MobX vs Redux

Introduction

What is MobX?

MobX is a battle tested library that makes state management simple and scalable by transparently applying functional reactive programming (TFRP).



Philosophy

Straightforward

Write minimalistic, boilerplate free code that captures your intent.

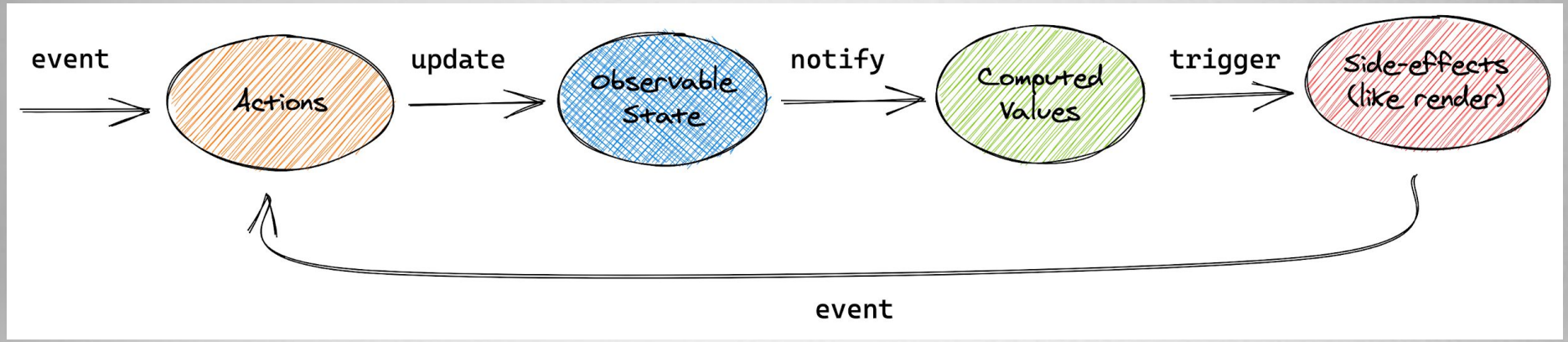
Effortless optimal rendering

All changes to and uses of your data are tracked at runtime, building a dependency tree that captures all relations between state and output.

Architectural freedom

MobX is unopinionated and allows you to manage your application state outside of any UI framework.

How it works?


event
Actions
update
Observable State
notify
Computed Values
trigger
Side-effects (like render)
event

Installation

Yarn: yarn add mobx

NPM: npm install --save mobx

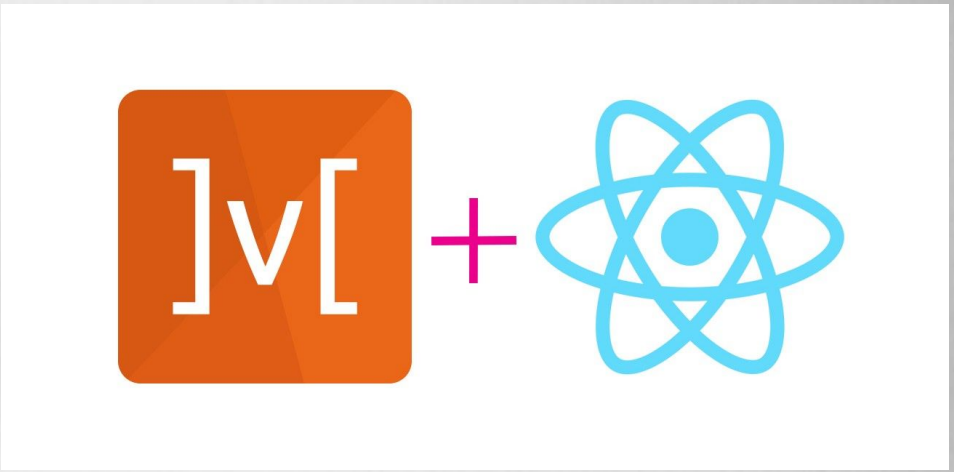
Supported Frameworks

MobX.dart: MobX for Flutter / Dart

lit-mobx: MobX for lit-element

mobx-angular: MobX for angular

mobx-vue: MobX for Vue



MobX Core

1. Observable state (1)

Observable

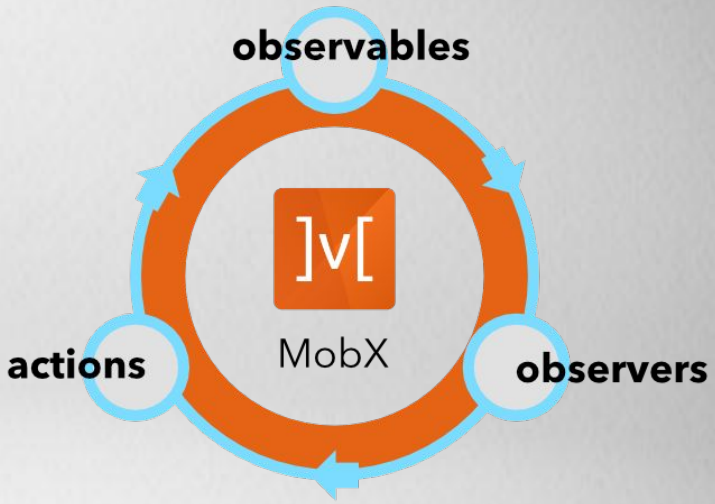
defines a trackable field that stores the state.

Action

marks a method as action that will modify the state.

Computed

marks a getter that will derive new facts from the state and cache its output.



1. Observable state (2)

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

class Doubler {
    value

    constructor(value) {
        makeObservable(this, {
            value: observable,
            double: computed,
            increment: action
        })
        this.value = value
    }

    get double() {
        return this.value * 2
    }

    increment() {
        this.value++
    }
}
```

2. Updating state using actions

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

class Doubler {
    value = 0

    constructor(value) {
        makeObservable(this, {
            value: observable,
            increment: action
        })
    }

    increment() {
        // Intermediate states will not become visible to observers.
        this.value++
        this.value++
    }
}
```

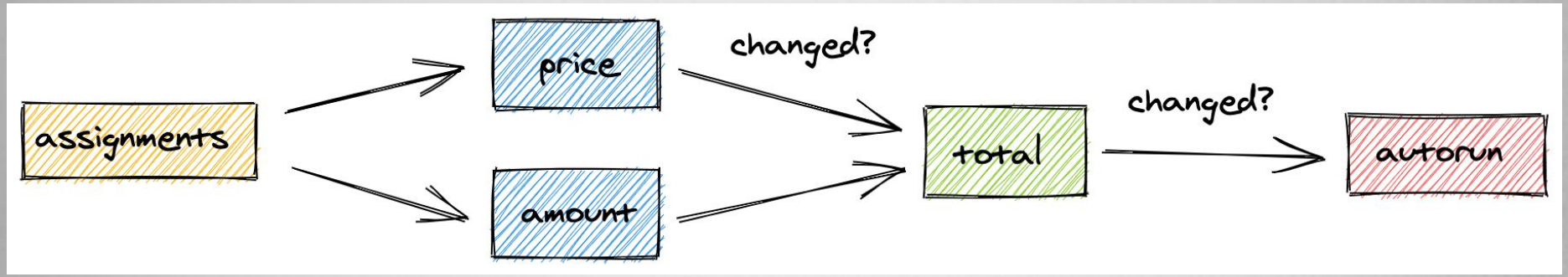
3. Deriving information with computeds (1)

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

class OrderLine {
    price = 0
    amount = 1

    constructor(price) {
        makeObservable(this, {
            price: observable,
            amount: observable,
            total: computed
        })
        this.price = price
    }

    get total() {
        console.log("Computing...")
        return this.price * this.amount
    }
}
```

3. Deriving information with computeds (2)

```
const order = new OrderLine(0)

const stop = autorun(() => {
    console.log("Total: " + order.total)
})
// Computing...
// Total: 0

console.log(order.total)
// (No recomputing!)
// 0

order.amount = 5
// Computing...
// (No autorun)

order.price = 2
// Computing...
// Total: 10

stop()

order.price = 3
// Neither the computation nor autorun will be recomputed.
```

3. Deriving information with computeds (3)



4. Reactions

Autorun

```
autorun(effect: (reaction) => void)
```

Reaction

```
reaction(() => value, (value, previousValue, reaction) => { sideEffect }, options?)
```

When

```
when(predicate: () => boolean, effect?: () => void, options?)
when(predicate: () => boolean, options?): Promise
```

... ...

Quick Example

```
import React from "react"
import ReactDOM from "react-dom"
import { makeAutoObservable } from "mobx"
import { observer } from "mobx-react"

// Model the application state.
class Timer {
    secondsPassed = 0

    constructor() {
        makeAutoObservable(this)
    }

    increase() {
        this.secondsPassed += 1
    }

    reset() {
        this.secondsPassed = 0
    }
}

const myTimer = new Timer()

// Build a "user interface" that uses the observable state.
const TimerView = observer(({ timer }) => (
    <button onClick={() => timer.reset()}>Seconds passed: {timer.secondsPassed}</button>
))

ReactDOM.render(<TimerView timer={myTimer} />, document.body)

// Update the 'Seconds passed: X' text every second.
setInterval(() => {
    myTimer.increase()
}, 1000)
```

MobX vs Redux

- Multi Stores	Multiple Stores	Single Store
- Easy to learn	Easy	Not so easy
- Scalability & Maintenance	Use in small project	Use in big project
- Debugging	Not so easy	Easy
- Community & Popularity	19k stars, 155 contributors on github	48k stars, 672 contributors on github
- Others	*object oriented programming*	-

Thank you