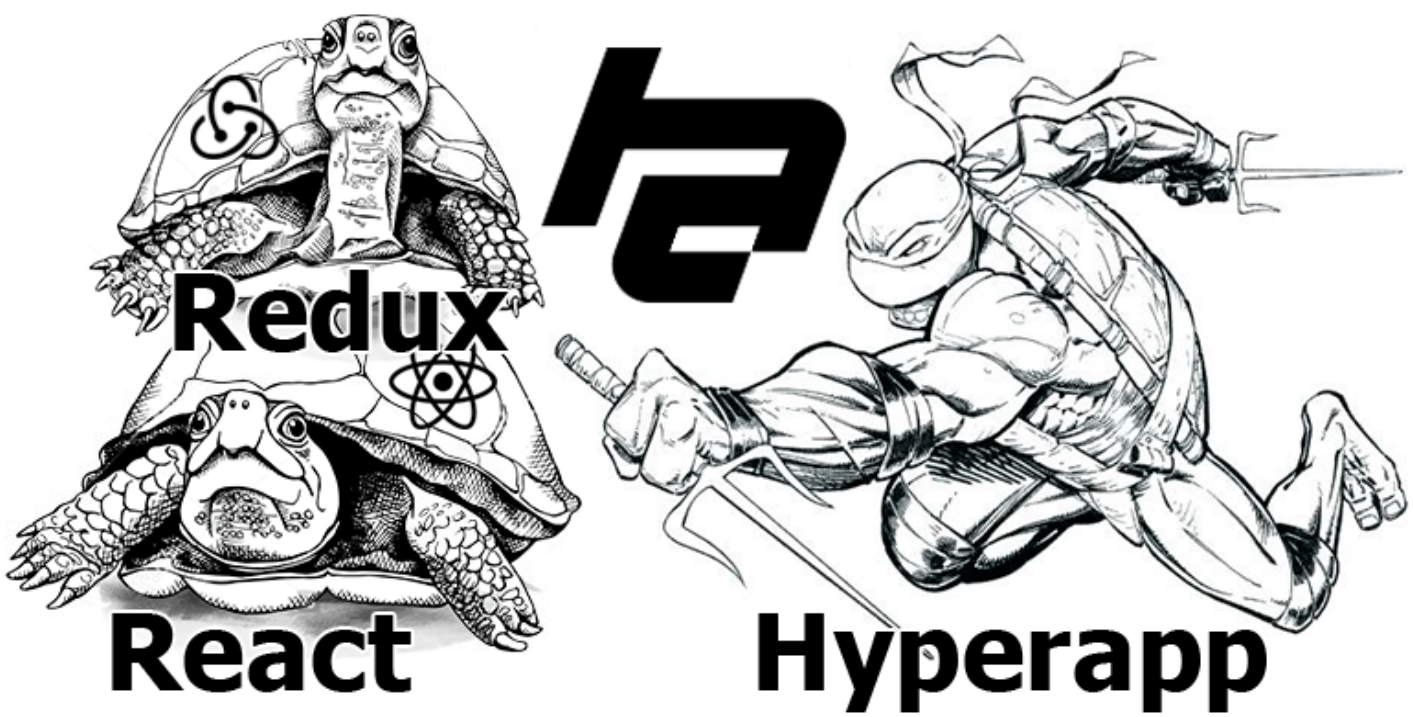
ПРОГРАММИРОВАНИЕ*, REACTJS*, NODE.JS*, JAVASCRIPT*, БЛОГ КОМПАНИИ DEVEXPRESS

Hyperapp для беженцев с React/Redux

ПЕРЕВОД

MrCheater 26 февраля в 08:51 10,7k

Оригинал: Wolfgang Wedemeyer



Я люблю Redux

Именно благодаря Redux для меня началось путешествие в мир удивительного функционального программирования. И это первое из функциональщины, что я попробовал в production. Прошли те времена, когда я использовал DOM для хранения состояния и неуверенно манипулировал им с помощью jQuery.

Redux — это инструмент для управления состоянием приложения (state), который позволяет полностью отделить его от представления (view). Представление (view) становится производным состояния (state), которое предоставляет пользователю интерфейс для его изменения. Действия пользователя (actions) не изменяют состояние (state) напрямую. Вместо этого они попадают в редюсер (reducer). Это такая чистая функция, которая на основе предыдущего состояния (state) и действия (action) генерирует следующее состояние (state). Такой подход к обновлению данных во многом был вдохновлен архитектурой языка программирования Elm и концепцией однонаправленного потока данных Flux. Это, возможно, самая популярная JavaScript-библиотека для иммутабельного изменения состояния из тех, что существуют сегодня. Авторы Redux сфокусировались на решении одной единственной проблемы — управление состоянием приложения (state), и сделали это хорошо. Redux получился достаточно модульным, чтобы работать с различными библиотеками для отображения представления (view).

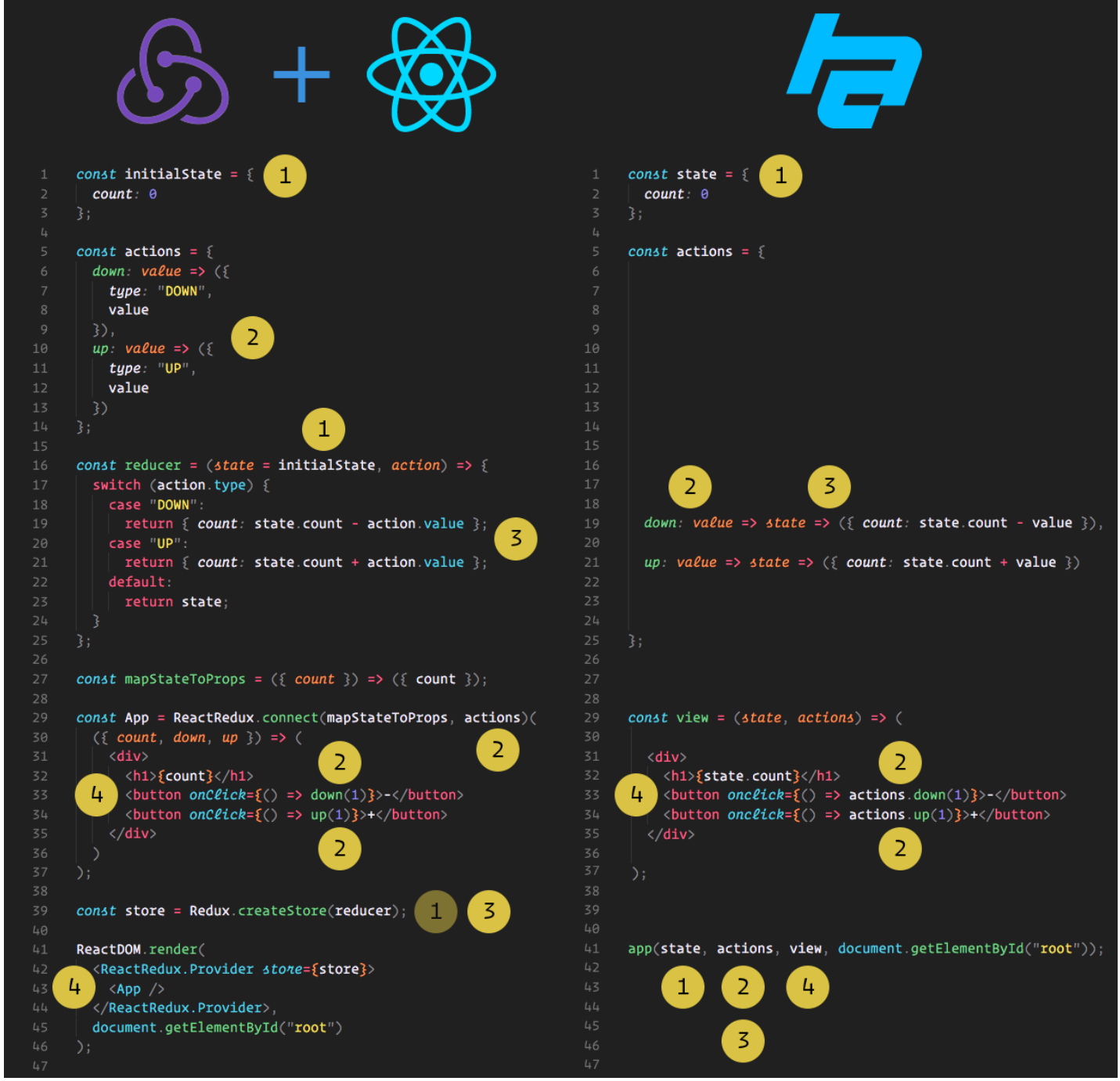
React использует аналогичный сфокусированный подход для представления (view), имеет эффективный виртуальный DOM, который можно подключить к DOM браузера, нативным мобильным приложениям, VR и прочим платформам.

Что бы создавать надежные, функциональные и легко отлаживаемые web-приложения, можно использовать React и Redux. Правда, потребуются вспомогательные библиотеки вроде react-redux и куча boilerplate-кода. А можно попробовать Hyperapp.

Hyperapp представляет собой единую библиотеку, которая обеспечивает управление состояним приложения (state) и иммутабельность, как в Redux/Elm, в сочетании с отображением представления (view) и Virtual DOM, как в React. Hyperapp использует подходы функционального программирования при управлении своим состоянием, но более гибко подходит к разрешению побочных эффектов (side effects), асинхронных действий и манипуляций с DOM. Hyperapp предоставляет мощную абстракцию для создания веб-приложений, но при этом дает вам полный доступ к нативным API, чтобы не ограничивать вас.

Код скажет больше, чем тысяча слов

Простое приложение-счетчик на React + Redux против эквивалента на Hyperapp:



Давайте пройдемся по каждому из пронумерованных блоков кода и разберемся, что к чему.

1. Состояние (state)

В Redux состояние (state) может быть любого типа, хотя настоятельно рекомендуется выбирать такой тип данных, который легко сериализовать. Подавляющее большинство разработчиков в

качестве начального состояния (state) для редюсера (reducer) использует пустой объект.

Не отражено в коде, но вызов `Redux.createStore` принимает в качестве необязательного аргумента начальное состояние (state). В Hyperapp состояние (state) всегда является объектом. Вместо двух разных способов инициализации состояния (state) здесь один.

2. Действия (actions)

В Redux генераторы действий (action creators) — это функции, которые возвращают действия (actions), как объекты JavaScript. Обычно генераторы действий (action creators) подключаются к Redux хранилищу (store) с помощью `bindActionCreators`, либо автоматически, либо вручную, используя аргумент `mapDispatchToProps` для `ReactRedux.connect`. Действия (actions) обычно определяются как множественный экспорт из одного файла, который затем втягивается в одно пространство имен, используя `import * as actions from "./actions"` при использовании модулей ES6.

В Hyperapp — генераторы действий (action creators), редюсеры (reducer) и `bindActionCreators` не нужны. Действия (actions) это чистые функции, которые иммутабельно меняют состояние (state) и имеют все данные необходимые для этого.

3. Изменение состояния (state)

В Redux изменение состояния (state) происходит в редюсере (reducer), который является чистой функцией, принимает состояние (state) и действие (action), возвращая следующее состояние (state). Действие (action) может обновить state (состояние) в любом редюсере (reducer).
Функция изменения состояния (state) имеет следующий вид:

```
(state, action) => nextState
```

Hyperapp использует функцию изменения состояния (state) такого вида:

```
(action) => (state [, actions]) => nextState
```

Не отражено в коде, но Hyperapp выполняет слияние (merge) состояния. Поэтому вместо `Object.assign` или `{... state, key: "value"}` достаточно просто `return: {key: "value"}`.

4. Представление (view)

В Redux представление (view) должно быть вручную подключено к состоянию (state) и генераторам действий (action creators). Для этого приходится использовать функцию высшего порядка (HOC) `ReactRedux.connect`, которая обертывает ваш компонент для подключения его к Redux хранилищу (store). Чтобы это работало, вы также должны обернуть свое приложение в `<ReactRedux.Provider>`, что делает ваше хранилище (store)

доступным для любых компонентов, которые хотят подключиться к нему.

В Hyperapp ваше состояние (state) и действия (actions) автоматически подключаются к вашему представлению (view), и только компоненты верхнего уровня имеют к ним доступ.

Slices

Redux и Hyperapp поддерживают состояние с вложенными пространствами имен, однако они делают это, используя несколько разные подходы. В Hyperapp это идет из коробки — в Redux требуется вручную использовать `combineReducers`.

Код с использованием Redux:

```
const potatoReducer = (potatoState = initialPotatoes, action) => {
  switch (action.type) {
    case FRY:
    // ...
  }
}
const tomatoReducer = (tomatoState = initialTomatoes, action) => {
  switch (action.type) {
    case GRILL:
    // ...
  }
}
const rootReducer = combineReducers({
  potato: potatoReducer,
  tomato: tomatoReducer
})
// This would produce the following state object
{
  potato: {
    // ...potatoes
    // and other state managed by the potatoReducer...
```

```
  },
  tomato: {
    // ...tomatoes
    // and other state managed by the tomatoReducer...
    // maybe some nice sauce?
  }
}
```

Эквивалентный код с использованием Hyperapp:

```
const rootState = {
  potato: {
    // ...just potato things
  },
  tomato: {
    // ...just tomato things
    // maybe some nice sauce?
  }
}
const rootActions = {
  potato: {
    // these actions receive only
    // the potato state slice and actions
  },
  tomato: {
    // these actions receive only
    // the tomato state slice and actions
  }
}
```

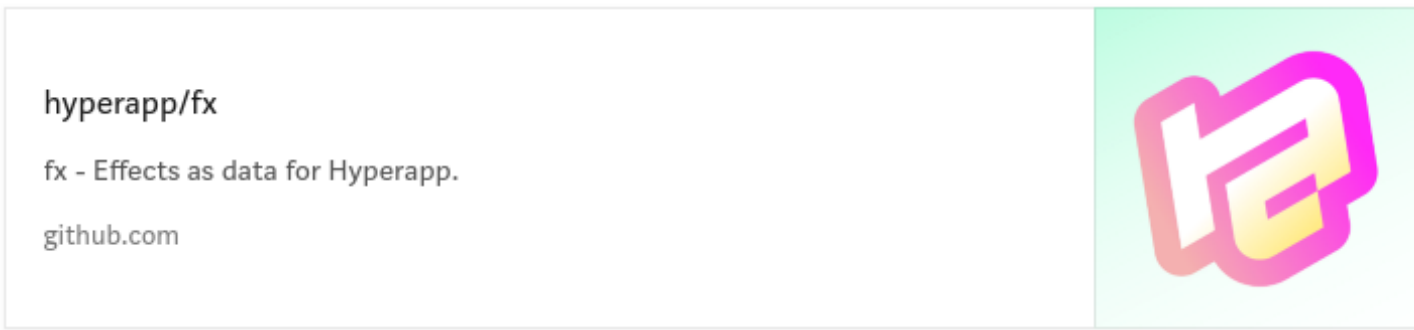
Async Actions

Пример организации асинхронных действий (acions)

```
const actions = {
  upLater: value => (state, actions) => {
    setTimeout(actions.up, 1000, value)
  },
  // Called one second after upLater
```

```
  up: value => state => ({ count: state.count + value })
}
```

Effects




```
import { withEffects, http } from "hyperapp-effects"
const state = {
  // ...
}
const actions = {
  foo: () => http("/data", "dataFetched"),
  dataFetched: data => {
    // data will have the JSON-decoded response from /data
  }
}
withEffects(app)(state, actions).foo()
```

Можно добавлять собственные эффекты.

Middleware

Для расширения возможностей генераторов действий (action creators) Redux предполагает использование `applyMiddleware` на уровне создания хранилища (store).

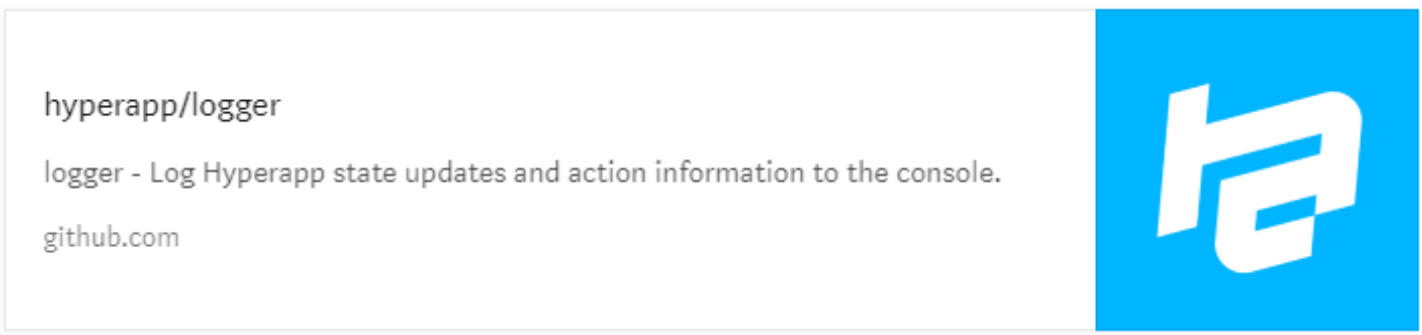
Hyperapp предполагает ручную композицию actions (действий) и middleware.

```
// Manual composition
hoa3(hoa2(hoa1(app)))(state, actions, view, document.body)

// Or with a standard-issue compose function
compose(hoa3, hoa2, hoa1)(app)(state, actions, view, document.body)

// Compose plays nicely with using different HOAs per environment
const hoas = NODE_ENV === "production" ? productionHoas : devHoas
compose(...hoas)(app)(state, actions, view, document.body)
```

Простым примером middleware является hyperapp-logger, который выводит информацию на консоль при вызове любого из ваших действий (actions):




```
logger(options)(app)(state, actions, view, document.body)
```

Завершение

Hyperapp воспринимает простоту так же серьезно, как и Redux. Сделать сложное простым, а большее меньшим возможно. Исходный код Hyperapp составляет ~300 строк кода, который я могу прочитать, когда у меня возникают вопросы, или при отладке, когда у меня есть проблемы. Размер библиотеки всего 1,4 кБ.

Я бегу от своей функциональной родины Redux не потому, что он мне не нравится, а из-за всей боли и страданий, которые вызывают у меня его соседи. Если вы любите Redux, как и я, и ищете лучшего баланса между простым функциональным миром преобразования данных и сложным внешним императивом миром, то я рекомендую вам дать шанс Hyperapp.

Похожие публикации

Честный MVC на React + Redux 44

MrCheater • 18 июля 2016 в 04:59

Иммутабельность в JavaScript 56

MrCheater • 30 мая 2016 в 06:18

Стилизация React-компонентов 10

ИЗ ПЕСОЧНИЦЫ

MrCheater • 11 мая 2016 в 12:13

Популярное за сутки

Наташа — библиотека для извлечения структурированной информации из текстов на русском языке

14

alexkuku • вчера в 16:12

Unit-тестирование скриншотами: преодолеваем звуковой барьер. Расшифровка доклада

4

lahmatiy • вчера в 13:05

Люди не хотят чего-то действительно нового — они хотят привычное, но сделанное иначе

25

ПЕРЕВОД

Smileek • вчера в 10:32

Руководство по SEO JavaScript-сайтов. Часть 2. Проблемы, эксперименты и рекомендации

2

ПЕРЕВОД

ru_vds • вчера в 12:04

Как адаптировать игру на Unity под iPhone X к апрелю

0

P1CACHU • вчера в 16:13

Лучшее на Geektimes

Стивен Хокинг, автор «Краткой истории времени», умер на 77 году жизни — 33

HostingManager • вчера в 13:49

Обзор рынка моноколес 2018 — 70

lozga • вчера в 06:58

«Битва за Telegram»: 35 пользователей подали в суд на ФСБ — 40

alizar • вчера в 15:14

Стивен Хокинг и его работа — что дал ученый человечеству? — 8

marks • вчера в 14:46

Sunlike — светодиодный свет нового поколения — 17

AlexeyNadezhin • вчера в 20:32