Redux practice. Part 1

|  |  |  |  |
| --- | --- | --- | --- |
| № урока: | 3 | Курс: | React Advanced |
|  | | |  |
| Средства обучения: | | | Текстовый редактор или IDE, браузер, Node.js |

# Обзор, цель и назначение урока

В этом уроке мы начинаем знакомство с Redux на практике, напишем простенькое приложение для ознакомления с различными методами Redux API и увидим на живом примере как осуществляется data-flow в Redux. Также мы познакомимся с библиотекой react-redux, благодаря которой мы сможем связать наши React компоненты с Redux.

**Изучив материал данного занятия, учащийся сможет:**

* Разбираться в принципе работы Redux
* Иметь понятие о предназначении различных методов и функций Redux
* Понимать как осуществляется data-flow в Redux
* Связывать React c Redux

# Содержание урока

1. Связывание React с Redux.
2. Создание Store и Reducers.
3. Прокидывание данных из стора в компоненты Redux.
4. Как вызывать Actions.
5. Что такое Action Creators и зачем они нам нужны.
6. Redux Middleware.

# Резюме

Для того, чтобы подключить **Redux**, нам нужно воспользоваться компонентом **Provider** из библиотеки react-redux, для этого оборачиваем top-level компонент в нашем приложении Провайдером и передаем в кач-ве **prop** в него **store**.

**Store** создается при помощи функции **createStore** из библиотеки **redux**. В кач-ве аргумента он принимает **Reducer**.

**Reducer –** это чистая функция, которая принимает на вход **state** и **action.**

**Reducer** обязательно должен в конце вернуть **state,** а также он может иметь какие-либо условия для изменения нашего **state** (но всегда нужно возвращать новый объект **state**, никаких изменений в текущем).

**State** можно проинициализировать каким-то дефолтным значением, сделать это можно при помощи **ES2015 Default Arguments.**

Чтобы передать в компонент данные из **Store** нужно воспользоваться функцией connect и б-ки react-redux. **Connect** – это такая ф-я, которая возвращает указатель на другую ф-ю. В кач-ве аргумента возвращаемой мы передаем наш компонент, а уже в функции высшего порядка мы возвращаем объект с данными из нашего **state** (благодаря тому, что эта функция принимает на вход сам state) и этот state по-сути эквивалентен **store.getState()**.

Если мы все сделали верно, то кроме самих данных в кач-ве prop в наш компонент попадет функция **dispatch**, с помощью которой можно вызывать различные **Actions**.

Но на самом деле такая практика является плохой (потому что компоненту незачем знать о существовании **dispatch**, он должен просто дергать какой-то callback), поэтому зачастую создают две функции **MapStateToProps** и **MapDispatchToProps**, которые возвращают объекты с данными и Actions соответственно. И обе эти функции мы передаем в кач-ве аргументов в наш connect.

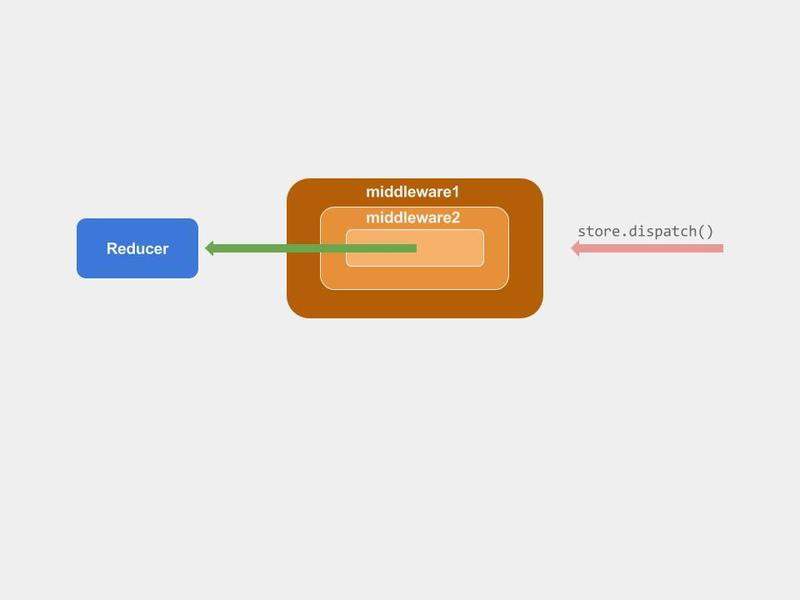
В Redux есть метод **subscribe**, который будет вызывать переданный в него callback каждый раз, когда что-то поменяется в **Store**. За счёт этого у нас обновляются данные в наших компонентах (в частности благодаря компоненту **Provider**).

**Action Creators** упрощают нашу разработку и изолируют **Actions** отдельно от самих компонентов. **Actions Creators** создаются при помощи **bindActionCreators**.

Также в нашем приложении может быть множество **reducers**, и благодаря методу **combineReducers** их можно объединить.

**Redux Middlewares**

Мы уже с вами знаем, что после того, как мы вызвали **store.dispatch(..)** наш **Action** попадает в **Reducer** где мы его как-то обрабатываем. Но благодаря **middlewares** мы можем встраиваться в этот процесс. Для чего это нужно? Допустим мы можем просто логировать события (выводить какой был вызван **Action**) и что случилось с нашим **state** после его вызова. Или же мы можем обрабатывать какие-то асинхронные операции. Это удобный механизм во многом помогающий нам.



Внешне **Middleware** выглядит как функция с аргументом **store**, возвращающая другую функцию с аргументом **next**, которая в свою очередь возвращает другую функцию с аргументом **action**. Выглядит устрашающе, но благодаря **arrow-functions** смотрится это все посимпатичнее.

В конце мы обязательно должны вызвать **next(action)** дабы наша цепочка не оборвалась (и данные пошли дальше – либо в следующий middleware, либо непосредственно в **reducer**).

Для **Redux** написано много различных готовых **middlewares**, и самым популярным является **redux-thunk**, который позволяет синхронно вызывать много **dispatch** в рамках одного **Action**.

# Закрепление материала

* Для чего нужна библиотека react-redux?
* Как передать данные из Redux в компонент
* Допустим наш State это массив. Можно ли в reducer делать arrayName.push(someDate) ?
* Зачем нужен метод subscribe?
* Почему лучше использовать Action Creators?
* Что нужно обязательно делать в конце функции middleware?
* Как называется один из самых популярных middleware для Redux и что он делает?

# Рекомендуемые ресурсы

Репозиторий с примерами

<https://github.com/fnnzzz/react-advanced-itvdn>

Официальная документация Redux

<http://redux.js.org/>

Библиотека faker.js для генерации фейковых данных

https://github.com/marak/<https://github.com/marak/Faker.js/>Faker.js/

redux-thunk middleware

https://github.com/[gaearon](https://github.com/gaearon/redux-thunk)/redux-thunk