2016-06-16

redux on Android

trivago

Warning!

This talk may be **opinionated** to at least some degree.

trivago

Giulio Petek

Android Developer

trivago Düsseldorf

Email: giulio.petek@trivago.com

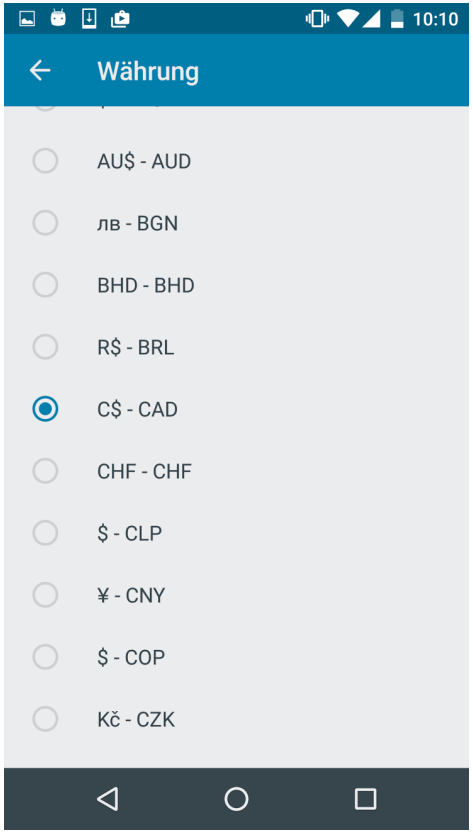
What do we cover?

1. Why are we here?
2. redux – a brief introduction.
3. What about Android?

So let’s get started!

… what’s the issue anyways?

Imagine…


10:10
Währung
AU$ - AUD
лв - BGN
BHD - BHD
R$ - BRL
C$ - CAD
CHF - CHF
$ - CLP
¥ - CNY
$ - COP
Kč - CZK

State... State Everywhere!

1. Activities/Fragments
2. View Models/Presenters
3. Controllers
4. Views

Why is that bad!?

1. Who is allowed to **mutate** state?
2. Who is responsible to **store** state?
3. When does state **propagate** through the app?
4. How to guarantee **consistent** state propagation?
5. What about **State Restoration**?
6. What about **testing**?

"We (kind of) already lost control over the **when**, **why**, and **how** of our **app's state**. Every **new requirement** makes the app more **fragile**."

One of my fellow colleagues

redux

"Redux attempts to make **state mutations predictable** [...].

@dan_abramov (Creator of redux)

3 Components

1. Single Source of Truth
2. State is Read-Only
3. Changes are made with **pure Functions**

#1 Single Source of Truth

The **state** of your whole application is stored in an object tree within a single **store**.

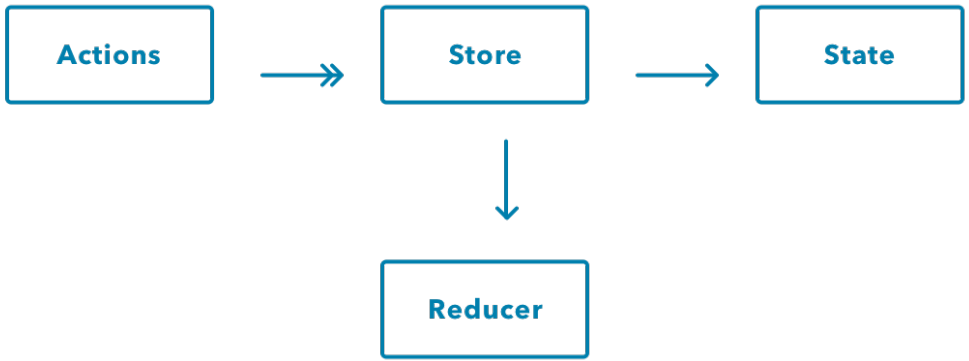
#2 State is Read-Only

The only way to **mutate state** is to emit an **action**, an object describing what happened.

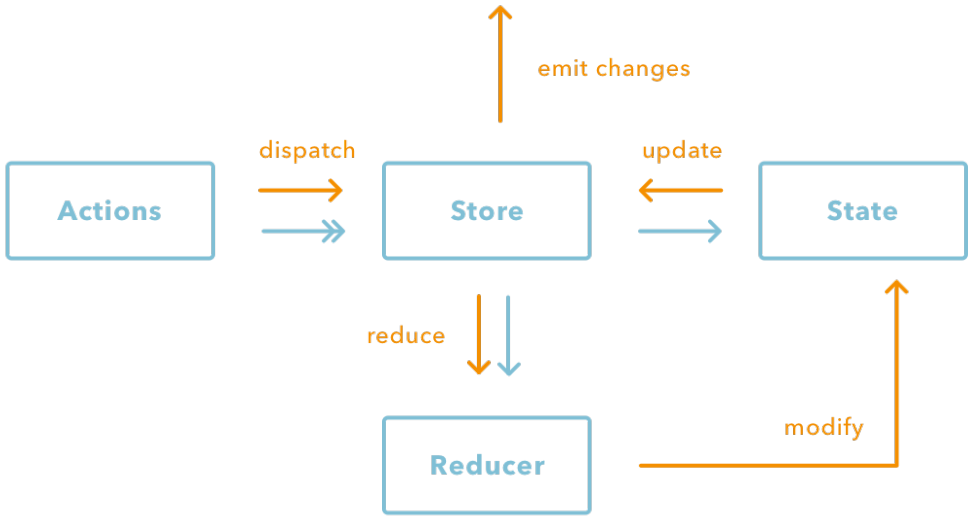
#3 Changes are made with Pure Functions

To specify how the state tree is transformed by **actions**, you write **pure reducers**.

The redux flow



The redux flow



So what do we gain?

1. Clean Flow of **Data** and **State**!
2. Improved **Testability** of Mutations
3. **Simplification** of Features

Going further…!

1. Replayable Crash Logs
2. Resource-based Routing
3. ...

So what about Android ?

3 Components

1. Actions
2. Reducer
3. State

#1 Action - Description

1. Used to describe a **redux** action
2. May contain **no**, **one** or **more** parameters

#3 Action - Code

```
public interface Action {

        // Public Api

        String getType();
}
```

#3 Action - Example

```
public final class DriveToAction implements Action {

        public static final String IDENTIFIER = ”id";
        public String destination;

        @Override
        public String getType() { return IDENTIFIER; }
    }
```

#2 Reducer - Description

1. **PURE** functions that uses a state and an action to modify the state
2. Use **compose()** to compose several reducers

#3 Reducer - Code

```
public interface Reducer<TState extends Serializable> {

        // Public Api

        TState reduce(TState pState, Action pAction);
}
```

#3 Reducer - Example

```
private static Reducer<CounterState> DRIVE_TO_REDUCER = (pState, pAction) -> {
        switch (pAction.getType()) {
            case DriveToAction.IDENTIFIER: {
                pState.destionation = ((DriveToAction)action).destionation;
            }
            break;
            // Handle other action...
        }
        return pState;
    };
```

#3 State - Description

1. **Object tree** that holds all the application's state
2. May contain **sub-states**

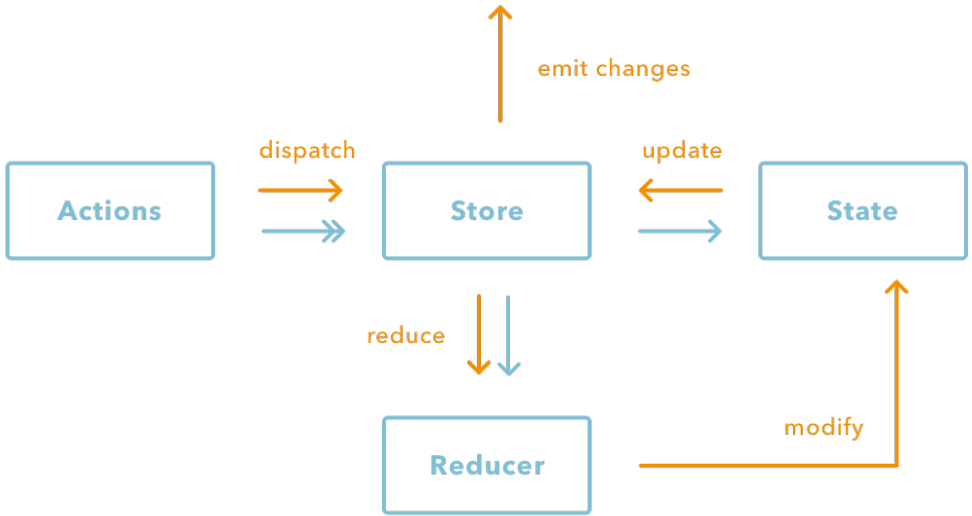
#3 State - Example

```
public final class AppState implements Serializable {

    // Public Api

    public DestionationDetails subState;
    public string destination;
}
```

The redux flow



Store - Description

1. **Combines** all other components
2. Dispatches **actions** and uses **reducer** to get new **state**
3. Emits **state changes**
4. Holds **current state**

#3 Store - Code

```
public class Store <TState extends Serializable> {

    // Public Api

    public void dispatch(Action pAction) {}
    public TState getState() {}
    public Observable<TState> state() {}
}
```

#3 Store - Example

```
Store<AppState> store = Store.createStore(REDUCER,
INITIAL_STATE);
store.state().subscribe(state ->
    // State changed
});
store.dispatch(MY_ACTION);
```

Problems?

trivago

#1 Rotation - Solution

1. Store is a "**Singleton**" and survives configuration changes
2. Store's **state()** observable is a **BehaviourSubject**

#2 Soft kill - Problem

1. **"Singleton"** is destroyed
2. **New application task** with a new identifier is created
3. **Activity stack** is restored

#2 Soft kill – Solution ?

1. Use **Android's** default way to restore instance
2. Works **per Activity/Fragment**

#2 Soft kill - ~~Solution~~

1. Use **Android's** default way to restore instance
2. Works **per Activity/Fragment**

Does not work as there should be **ONE STATE/STORE** not one per Activity

#2 Soft kill - Solution

1. Use File to store state
2. Read when app is started
3. Delete when app is started without an **SavedInstanceState**
4. Store state in **onSaveInstanceState**

#2 Soft kill - Solution

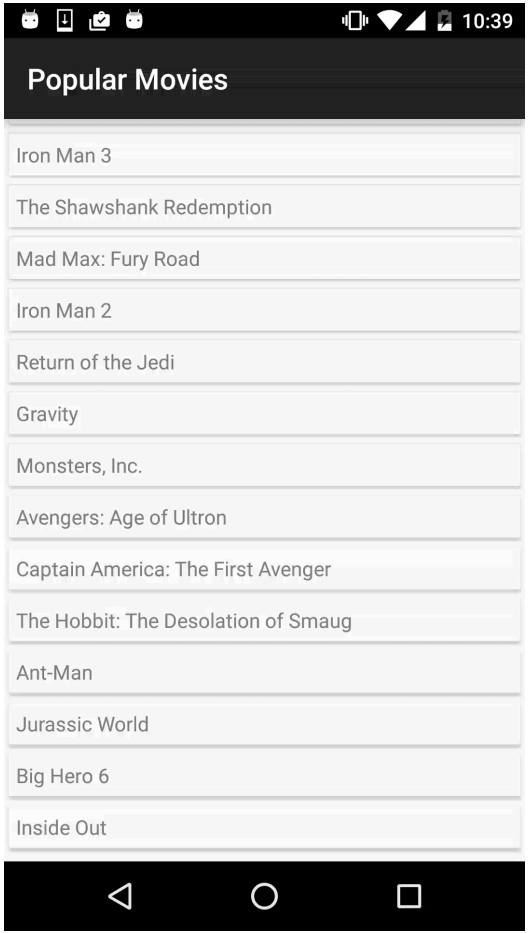
1. Use File to store state
2. Read when app is started
3. Delete when app is started without an **SavedInstanceState**
4. Store state in **onSaveInstanceState**

WORKS!

Really???

Yes! Yes! Yes!

https://github.com/rheinfabrik/android-redux-example



So do we use it in trivago?

1. **NO**. But for a new project – **YES.**
2. **Hard to integrate** into an existing project.
3. **Don't** fight the **framework!**

Questions & Discussion

@GiloTM