Flux-Architektur

abseits von JavaScript

Manuel Mauky
@manuel_mauky

JUG Görlitz

Saxonia Systems
So geht Software.

Was ist Flux?

“Flux is the application architecture that Facebook uses for building client-side web applications.”

https://facebook.github.io/flux/

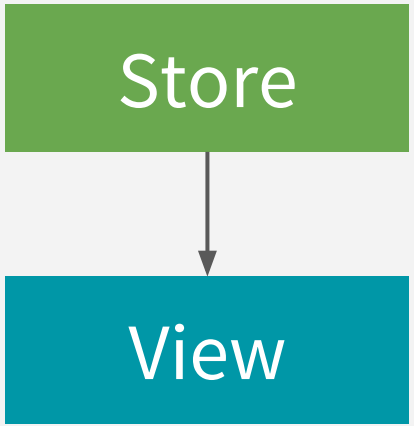
Was ist Flux?

- Architektur-Muster für React.JS
- Kernidee: **Unidirektionaler Datenfluss**
- klare View-Hierarchien
- Ziel: Einfacher Nachvollziehen, was in der Application abläuft

Store

- Applikations-Zustand
- Logik
- repräsentiert eine fachliche Einheit

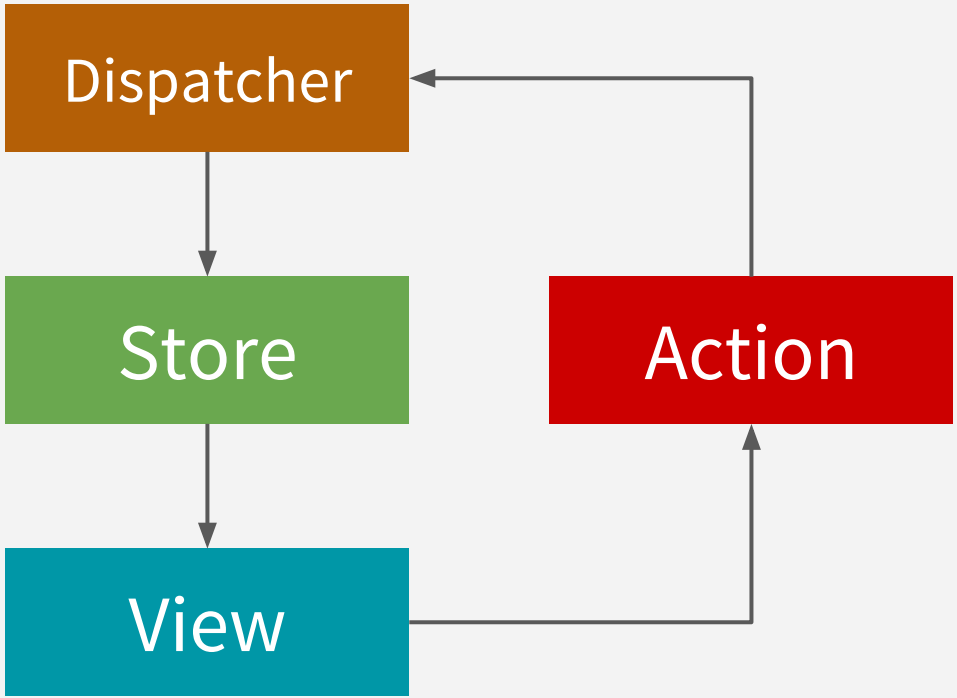
Datenfluss



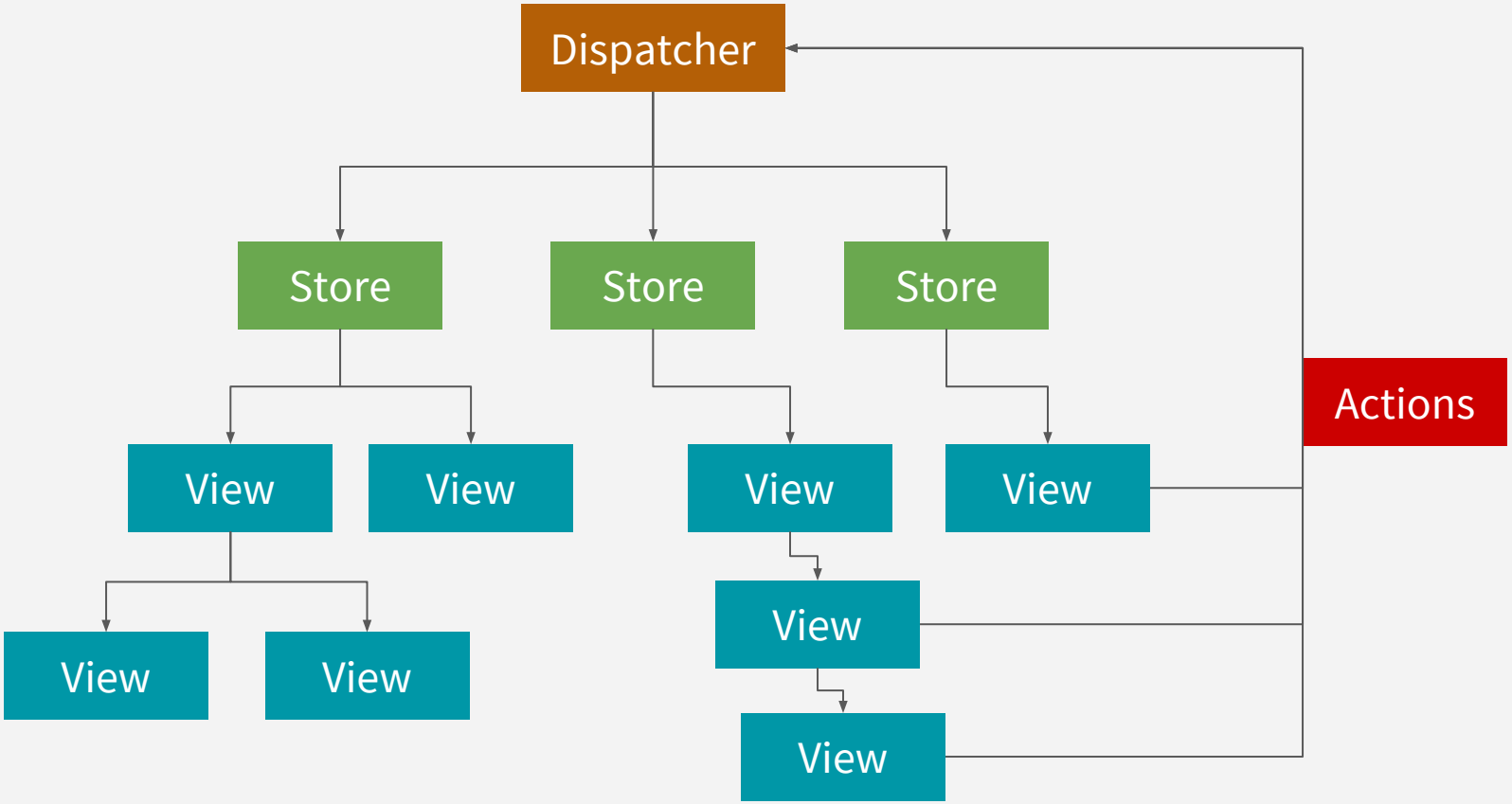
- View zeigt Daten des Stores
- Wenn sich Store ändert, wird View geupdated

Datenfluss

- View erzeugt “Actions” bei Userinteraktion
- Action repräsentiert fachliche Aktion
- vergleichbar Command-Pattern



- Dispatcher leitet **alle** Actions an **alle** Stores weiter
- Stores entscheiden selbst, ob und wie sie auf Actions reagieren


Dispatcher
Store
Store
Store
Actions
View
View
View
View
View
View
View
View

Exkurs: React.JS

Konzeptionell

- jede Änderung führt zu kompletten Re-Rendering der gesamten Komponente inkl. aller Unterkomponenten
- Anleihen aus funktionaler Programmierung
- einfaches Programmiermodell

Exkurs: React.JS

Konzeptionell

- jede Änderung führt zu kompletten Re-Rendering der gesamten Komponente inkl. aller Unterkomponenten
- Anleihen aus funktionaler Programmierung
- einfaches Programmiermodell

Tatsächlich

- Virtual-DOM
- Diff-Algorithmus berechnet minimal notwendige DOM-Änderungen
- dadurch sehr performant

Flux abseits von JavaScript?

Flux mit JavaFX

Herausforderung:

- JavaFX hat kein Virtual-DOM
- Komplettes Re-Rendering inperformant
- JavaFX ist recht Zustands-Zentriert

Flux mit JavaFX

Herausforderung:

- JavaFX hat kein Virtual-DOM
- Komplettes Re-Rendering inperformant
- JavaFX ist recht Zustands-Zentriert

Aber:

- JavaFX eignet sich für Reactive-Programming:
  - Databinding
  - Reactive-Streams (*RxJava* oder *ReactFX*)

Idee

- Stores geben Read-Only Properties nach außen

Idee

- Stores geben Read-Only Properties nach außen
- Views binden sich an Properties des Store

Idee

- Stores geben Read-Only Properties nach außen
- Views binden sich an Properties des Store
- Actions als Klassen
- Dispatcher als reactive Stream von Actions
- Stores subscriben den Stream von Actions

Beispiel: Counter

```
public class CounterStore {

    private IntegerProperty counter = new SimpleIntegerProperty();


    public ReadOnlyIntegerProperty counterValue() {
        return counter;
    }
}
```

```
public class CounterView {

    @FXML
    private Label valueLabel;

    @Inject
    private CounterStore store;

    public void initialize() {
        valueLabel.textProperty().bind(store.counterValue());
    }
}
```

```
public class IncreaseAction implements Action {

    private final int amount;

    public IncreaseAction(int amount) {
        this.amount = amount;
    }

    public int getAmount() {
        return amount;
    }

    // equals & hashcode
}
```

```
import org.reactfx.EventSource;
import org.reactfx.EventStream;

public class Dispatcher {

    private EventSource<Action> actionStream = new EventStream<>();


    public void dispatch(Action action) {
        actionStream.push(action);
    }

    public EventStream<Action> getActionStream() {
        return actionStream;
    }
}
```

```
public class CounterView implements View {

    ...

    @FXML
    public void increaseButtonPressed() {
        Dispatcher.getInstance().dispatch(new IncreaseAction(1));
    }

}
```

```
public class CounterStore {

    private IntegerProperty counter = new SimpleIntegerProperty();

    public CounterStore() {
        Dispatcher.getInstance()
            .getActionStream()
            .filter(a -> a.getClass().equals(IncreaseAction.class))
            .map(a -> (IncreaseAction) a)  // cast is needed
            .subscribe(this::increase);
    }

    private void increase(IncreaseAction action) {
        counter.set(counter.get() + action.getAmount());
    }

    public ReadOnlyIntegerProperty counterValue() {
        return counter;
    }
}
```

```
public class CounterStore extends Store {

    private IntegerProperty counter = new SimpleIntegerProperty();

    public CounterStore() {
        subscribe(IncreaseAction.class, this::increase);
    }

    private void increase(IncreaseAction action) {
        counter.set(counter.get() + action.getAmount());
    }

    public ReadOnlyIntegerProperty counterValue() {
        return counter;
    }
}
```

```
@Test
public void test() {
    // given
    assertThat(store.counter()).hasValue(0);

    // when
    Dispatcher.getInstance().dispatch(new IncreaseAction(1));

    // then
    assertThat(store.counter()).hasValue(1);
}
```

Ausblick

- Virtual-SceneGraph?
- Einfacher in funktionalen Sprachen? Groovy? Kotlin? Frege?
- Redux - komplett funktionale Variante von Flux

Q & A

Example Code:
https://github.com/lestard/FluxFX

Manuel Mauky

manuel.mauky@saxsys.de
http://lestard.eu
@manuel_mauky