Angular 2 + ngrx/store

A Software Presentation From

Jeremy Zerr
Site: http://www.zerrtech.com
Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr

Angular 2 + ngrx/store

- ☐ What is ngrx/store?
- ☐ Advantages of a single store for app state
- ☐ How will my app change with ngrx/store?
- ☐ What types of new objects do we deal with?
- ☐ Comparison to Redux
- ☐ Resources

What is ngrx/store?

- [ ] ngrx aims to bring reactive extensions to Angular 2
- [ ] What does Reactive mean? Using Observables to turn everything into data sequences that you are pushed changes. (My simple definition)
- [ ] ngrx/store is one part of that project
- [ ] It was inspired by Redux, which aimed to create a single app store and used commonly with React
- [ ] ngrx/store brings a Redux-like single store for all of your app state to Angular 2

Advantages of using a single store for app state

- ☐ Single source of truth (centralized state)
- ☐ State is read-only
- ☐ Changes made with pure functions (immutable and testable)
- ☐ Changes happen in one place instead of scattered throughout the app
- ☐ A snapshot represents the entire state of the app at a single point in time
- ☐ (you should) Make the store immutable
- ☐ Detecting changes becomes much more simple, more performant, all changes happen at the store and are pushed down.
- ☐ You can watch only the parts of the store you care about

How will my app change with ngrx/store?

- [ ] Sample project that Lists timesheets from TSheets API
- [ ] Service that returns TSheets Timesheets
- [ ] List Container component that manages timesheets
- [ ] List Dumb component that displays the timesheets
- [ ] On Github - Angular 2 Webpack TSheets API Demo feature/demo branch

Developer Environment

- ☐ Angular 2 rc.4
- ☐ Typescript and ES2015
- ☐ ngrx/store 2
- ☐ Webpack
- ☐ Visual Studio Code

Service getTimesheets()

```
// return an Observable to timesheets HTTP request
getTimesheets(): Observable<Timesheet[]> {
  return this.http.get(this.getTimesheetsListUrl(), { headers: this.getHeaders() })
    .map((response) => this.extractData(response, []))
    .catch(this.handleError);
}
```

```
// fetch timesheets then update the store
getTimesheets(): Subscription {
  let getRequest = this.http.get(this.getTimesheetsListUrl(), { headers: this.getHeaders() })
    .map((response) => this.extractData(response, []))
    .catch(this.handleError);

  return getRequest.subscribe(timesheets => {
    this._store.dispatch({type: 'SET_TIMESHEETS', payload: timesheets});
  });
}
```

```
constructor (private http: Http, private _store: Store<AppStore>) {
  this.timesheets$ = _store.select(state => state.timesheets);
}
```

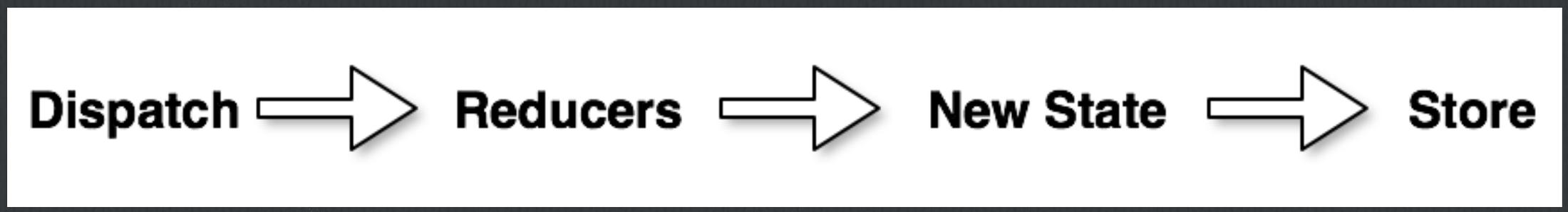
New Terminology

- [ ] Observables - a data collection you get push notifications from (think of an array over time)

- [ ] Subscription - you can subscribe to observables to be notified

How do we access the store?

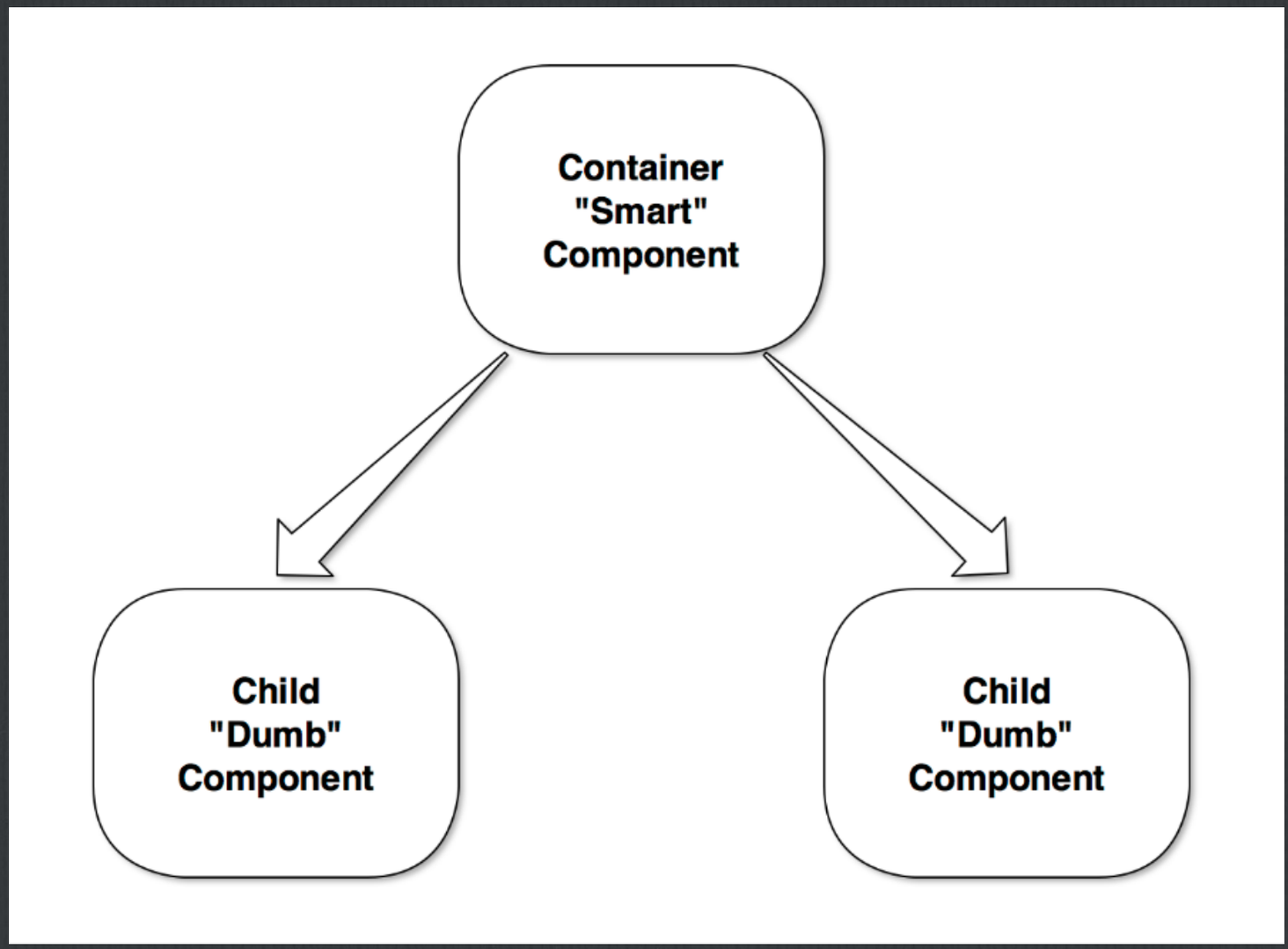
- ☐ We have Actions and Reducers
- ☐ Actions are dispatched and consist of a type and a payload of data
- ☐ Reducers are responsible for changing the state based on the action and the payload, and return a new state.
- ☐ We select an item in the Store to observe it



What can we do with Observables?

- ☐ Lots of the same things you can do with arrays
- ☐ Many of the functional transforms are available as Observable equivalents
- ☐ map, filter, reduce, etc
- ☐ rxjs operators link

What do I mean by container and dumb component?



Container Component

```
// if we were using regular Observables, we would pass that
// observable instead
this.timesheets$ = this.timesheetDirectService.getTimesheets()
  .map(timesheets => {
    return timesheets.sort((a, b) => {
      return b.date.getTime() - a.date.getTime();
    });
  });
```

```
// save the store observable and pass to component
// using the async pipe
this.timesheets$ = this.timesheetService.timesheets$
  .map(timesheets => {
    return timesheets.sort((a, b) => {
      return b.date.getTime() - a.date.getTime();
    });
  });
// this kicks off the initial request to populate the store
this.timesheetService.getTimesheets();
```

Container Component Template

```
<ts-list
  (onSelectTimesheet)="selectTimesheet($event)"
  (onSaveTimesheet)="saveTimesheet($event)"
  [timesheets]="timesheets$ | async"
  [selectedId]="selectedId">
</ts-list>
```

Dumb Component

```
@Input() timesheets: Timesheet[];
@Input() selectedId: number;
@Output() onSelectTimesheet: EventEmitter<any> = new EventEmitter();
@Output() onSaveTimesheet: EventEmitter<any> = new EventEmitter();
```

What does the store look like?

```ts
export interface AppStore {
  timesheets: Timesheet[];
}
```

What does an Action and Reducer look like?

```
this._store.dispatch({type: 'SET_TIMESHEETS', payload: timesheets});
```

```
export const timesheetsReducer: ActionReducer<Timesheet[]> = (state: Timesheet[] = [], action: Action) => {
  let state_copy: Timesheet[];
  switch (action.type) {
    case 'SET_TIMESHEETS':
      return action.payload;
    case 'ADD_TIMESHEETS':
      return [
        ...state,
        action.payload
      ];
    case 'EDIT_TIMESHEET':
      state_copy = [...state];
      let timesheet_id_to_edit: number = state_copy.findIndex((timesheet) => {
        if (timesheet._id === action.payload._id) {
          return true;
        } else {
          return false;
        }
      });
      if (timesheet_id_to_edit > -1) {
        state_copy[timesheet_id_to_edit] = _.merge(new Timesheet(), state_copy[timesheet_id_to_edit], action.payload);
      }
      return state_copy;
```

When do we call actions?

- [ ] We encapsulate all of the calls to Timesheet actions within the Timesheet service.
- [ ] When an API call is complete, we dispatch an action corresponding to the HTTP method
  - [ ] GET (all) = SET_TIMESHEETS - reset entire list
  - [ ] POST = ADD_TIMESHEETS - add new
  - [ ] PUT = EDIT_TIMESHEET - update existing
  - [ ] DELETE = DELETE_TIMESHEET - delete existing

Differences between ngrx/store and Redux

- ☐ ngrx/store has everything wrapped in Observables out of the box, have to use a middleware for Redux
- ☐ ngrx/store doesn't have the concept of middleware like Redux, approach is different, use meta reducers like ngrx/effects
- ☐ Almost all terminology is the same

Takeways/Tips/Gotchas

- ☐ So Angular 2 has Redux now just like React!
- ☐ HTTP doesn’t fire until you subscribe
- ☐ Make your store Immutable for the most efficient change detection
- ☐ Only subscribe to the parts of the store you need
- ☐ Observable variable$ naming convention

Resources

- [ ] Github ngrx/store
- [ ] Comprehensive Introduction to @ngrx/store - several images in presentation from this intro
- [ ] Github Angular 2 Webpack TSheets API Demo

Thanks!

A Software Presentation From

Jeremy Zerr
Site: http://www.zerrtech.com
Blog: http://www.jeremyzerr.com
LinkedIn: http://www.linkedin.com/in/jrzerr
Twitter: http://www.twitter.com/jrzerr