academeez

@angular/forms

7. ControlValueAccessor - Custom Form Controls

ControlValueAccessor

- In this lesson we will learn about ControlValueAccessor
- Understanding ControlValueAccessor will give you the power of creating custom form controls

```
<form>
    <input type="phoneNumber" name="phone-number" [formControl]="phoneNumber" ↗
    <input type="creditCard" name="phone-number" [formControl]="creditCard" ↗
    <input type="citiesAutocomplete" name="phone-number" [formControl]=“city" ↗
    <nz-address name=“fullAddress" [formControl]="address" ↗
    <button type=“submit”>Submit↙button>
↙form>
```

input <—> FormControl communication

academeez

- When we are placing a FormControl (either implicitly with **ngModel** or explicitly with reactive) there is a communication happening
  - The input will update the FormControl about the changes
  - The FormControl can update the state of the input
- Let’s examine how this communication is done

input with FormControl directive

academeez

- An input with a FormControl will trigger a directive.
- That directive is in charge of the input <— —> **FormControl** communication
- That directive should grab change events from the view and update the **FormControl** and grab state change from the **FormControl** and update the view

```
@Directive({
  selector:
      'input[type=checkbox][formControlName],input[type=checkbox][formControl],input[type=checkbox][ngModel]',
  // ...
})
export class CheckboxControlValueAccessor implements ControlValueAccessor {
  // ...
}
```

ControlValueAccessor

- So there is a middle directive to help us with the communication between the input and the **FormControl**
- To help us create that middle directive, angular has an interface contract which the middle directive needs to implement, these are methods that we need to implement for the input **FormControl** communication
- That is the job of the **ControlValueAccessor** interface

```
@Directive({
  selector:
      'input[type=checkbox][formControlName],input[type=checkbox][formControl],input[type=checkbox][ngModel]',
  // ...
})
export class CheckboxControlValueAccessor implements ControlValueAccessor {
  // ...
}
```

2 Directives on an input

- So basically the following input contains 2 directives
  - The directive that implements a ControlValueAccessor
  - The **[formControl]** directive

```
<input
    type="text"
    name="phone-number"
    [formControl]=“phoneNumber"
↗
```

ControlValueAccessor communicating with input

- The ControlValueAccessor can communicate with the input it is placed on with
  - ElementRef
  - @HostListener
  - host key in **@Directive** Metadata

```
@Directive({
  selector:
      'input[type=checkbox][formControlName],input[type=checkbox][formControl],input[type=checkbox][ngModel]',
  host: {'(change)': 'onChange($event.target.checked)', '(blur)': 'onTouched()'},
  // ...
})
//...
```

ControlValueAccessor communicating with [formControl]

- The ControlValueAccessor can communicate with the **[formControl]** it is placed on
- the **[formControl]** will ask the DI for the **ControlValueAccessor** and pass methods and state

```
@Directive({
  selector: '[formControl]',
  // ...
})
export class FormControlDirective extends NgControl implements OnChanges {
  // ...
  constructor(
      // ...
      @Optional() @Self() @Inject(NG_VALUE_ACCESSOR) valueAccessors: ControlValueAccessor[],
      // ...
      null) {
    // ...
  }
}
```

ControlValueAccessor register DI

- The fact that the **formControl** directive can ask for the **ControlValueAccessor** directive from the DI means that the **ControlValueAccessor** needs to register itself to the DI

```
@Directive({
  selector:
      'input[type=checkbox]
[formControlName],input[type=checkbox]
[formControl],input[type=checkbox][ngModel]',
  providers: [CHECKBOX_VALUE_ACCESSOR],
  // ...
})
export class CheckboxControlValueAccessor
    implements ControlValueAccessor {
        // ...
    }
```

```
export const CHECKBOX_VALUE_ACCESSOR: any = {
  provide: NG_VALUE_ACCESSOR,
  useExisting: forwardRef(() => CheckboxControlValueAccessor),
  multi: true,
};
```

ControlValueAccessor interface

academeez

- Now that we know how everyone communicates with everyone, let's see the interface that the middle directive needs to implement in order for the **FormControl** and input to communicate

```
export interface ControlValueAccessor {
    writeValue(obj: any): void;
    registerOnChange(fn: any): void;
    registerOnTouched(fn: any): void;
    setDisabledState?(isDisabled: boolean): void;
}
```

EX. Phone Number Control

- Let’s create the following Directive
- It will add a dash between the prefix and the phone

```
<input
        type="phoneNumber"
        name="phone-number"
        [formControl]="phoneNumber"
↗
```

Student EX. Credit Card input

- Your turn... Create the following directive
- It will add a dash every 4 digits

```
<input
        type="creditCard"
        name="creditCard"
        [formControl]="creditCard"
↗
```

Student EX. Autocomplete cities directive

academeez

- Create the following directive that will pop an autocomplete list that is taken from the server: http://nztodo.herokuapp.com/api/tasks/?format=json

```
<input
        type="cities"
        name="cities"
        [formControl]=“cities”
↗
```

Student EX. AddressComponent

- The same rules can be applied to a component as well
- Create a component that contains, city, address fields to pass the address

```
<nz-address [formControl]=“address” name=“address” >↙nz-address>
```

Summary

- Angular gives us the tools to create our own custom form control
- To do that we need to do the following:
  - Create a directive or component that implements the interface **ControlValueAccessor**
  - Following the interface declaration we will have to implement communication methods
  - Register our Component or directive with the DI.

academeez

Congratulations!

You are now an expert in building forms in angular!