Formulare und Validierung

Manfred Steyer
SOFTWAREarchitekt.at

Inhalt

- Ansätze
- Template-getriebene Formulare
- Reaktive Formulare
- Validierung

Ansätze in Angular

Template-getrieben	• ngModel im Template • Angular erzeugt Objektgraph für Formular • FormsModule
Reaktiv	• Anwendung erzeugt Objektgraph • Mehr Kontrolle • ReactiveFormsModule
Daten-getrieben	• Angular generiert Formular für Datenmodell • An Community übergeben

SOFTWARE *architekt.at*

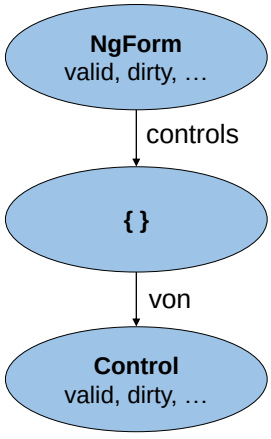
Template-getriebene Formulare

Template-getriebene Formulare

```
export class FlugSuchenComponent {

   von: string;
   nach: string;

   constructor(flugService: FlugService) {

         von = 'Graz';
         nach = 'Hamburg';
   }
}
```

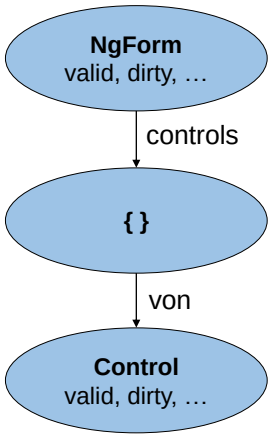

View

```
<form>

   <input type="text" name="von"
      [(ngModel)]="von" required minlength="3">

   […]

</form>
```



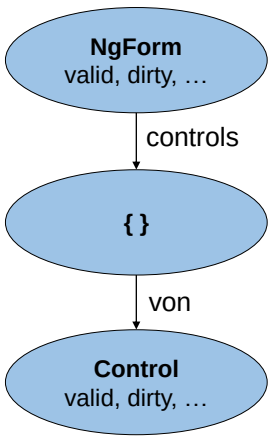
View

```
<form #f="ngForm">

   <input type="text" name="von"
      [(ngModel)]="von" required minlength="3">

   […]

</form>
```



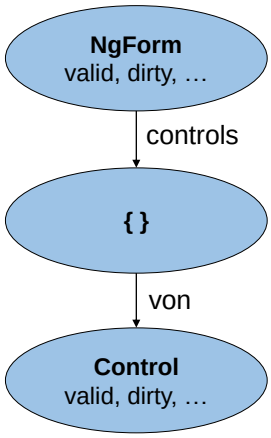
View

```
<form #f="ngForm">

   <input type="text" name="von"
      [(ngModel)]="von" required minlength="3">

   <div *ngIf="!f.controls['von'].valid">
      …Error…
   </div>

</form>
```



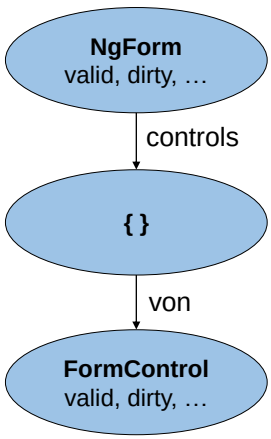
View

```
<form #f="ngForm">

   <input type="text" name="von"
      [(ngModel)]="von" required minlength="3">

   <div *ngIf="!f?.controls['von']?.valid">
      …Error…
   </div>

</form>
```



View

```
<form #f="ngForm">

   <input type="text" name="von"
      [(ngModel)]="von" required minlength="3">

   <div *ngIf="!f?.controls['von']?.valid">
      …Error…
   </div>

   <div
      *ngIf="f?.controls['von'].hasError('required')">
      …Error…
   </div>
</form>
```



DEMO

Eigene Validierungs-Regeln

Direktiven

- Fügen Verhalten zur Seite hinzu
- Beispiel: ngModel, ngClass, ngIf, ngFor
- Kein Template im Gegensatz zu Komponenten

Validierungs-Direktive

<input [(ngModel)]="von" name="von" **ort**>

Validierungs-Direktive

```
@Directive({
    selector: 'input[ort]'


})
export class OrtValidatorDirective implements Validator {

    validate(c: AbstractControl): object {
        let value = c.value;
        […]
        if (…) return { ort: true };
        return {}; // Kein Fehler
    }
}
```

Validierungs-Direktive

```
@Directive({
    selector: 'input[ort]',
    providers: [{ provide: NG_VALIDATORS,
                  useExisting: OrtValidatorDirective, multi: true}]
})
export class OrtValidatorDirective implements Validator {

    validate(c: AbstractControl): object {
        let value = c.value;
        […]
        if (…) return { ort: true };
        return {}; // Kein Fehler
    }
}
```

.hasError('ort')

Attribute berücksichtigen

```
<input [(ngModel)]="von" name="von"
                [ort]="['Graz', 'Hamburg', 'Zürich']">
```

Attribute berücksichtigen

```
@Directive({
    selector: 'input[ort]',
    providers: [{ provide: NG_VALIDATORS,
                  useExisting: OrtValidatorDirective,
                  multi: true }]
})
export class OrtValidatorDirective implements Validator {

    @Input() ort: string[];

    validate(c: AbstractControl): object {
        […]
    }
}
```


Attribute berücksichtigen

```
@Directive({
    selector: 'input[ort]',
    providers: [{ provide: NG_VALIDATORS,
                  useExisting: OrtValidatorDirective,
                  multi: true }]
})
export class OrtValidatorDirective implements Validator {

    @Input() ort: string;
    @Input() strategy: string;

    validate(c: AbstractControl): object {
        […]
    }
}
```

Attribute berücksichtigen

```
<input [(ngModel)]="von" name="von"
        [ort]="['Graz', 'Hamburg', 'Zürich']" [strategy]="'strict'">
```


DEMO

Multi-Field-Validatoren

```
@Directive({
    selector: 'form[roundTrip]',
    providers: [ … ]
})
export class RoundTripValidatorDirective implements Validator {

    validate(control: AbstractControl): object {
        […]
    }
}
```


Multi-Field-Validatoren

```
export class RoundTripValidatorDirective implements Validator {

    validate(control: AbstractControl): object {
        let group = control as FormGroup;

        let from = group.controls['from'];
        let to = group.controls['to'];

        if (!from || !to) return { };

        […]
}
```

Multi-Field-Validatoren

```
export class RoundTripValidatorDirective implements Validator {

    validate(control: AbstractControl): object {
        let group = control as FormGroup;

        let from = group.controls['from'];
        let to = group.controls['to'];

        if (!from || !to) return { };

        if (from.value === to.value) return { roundTrip: true };

        return { };
    }
}
```

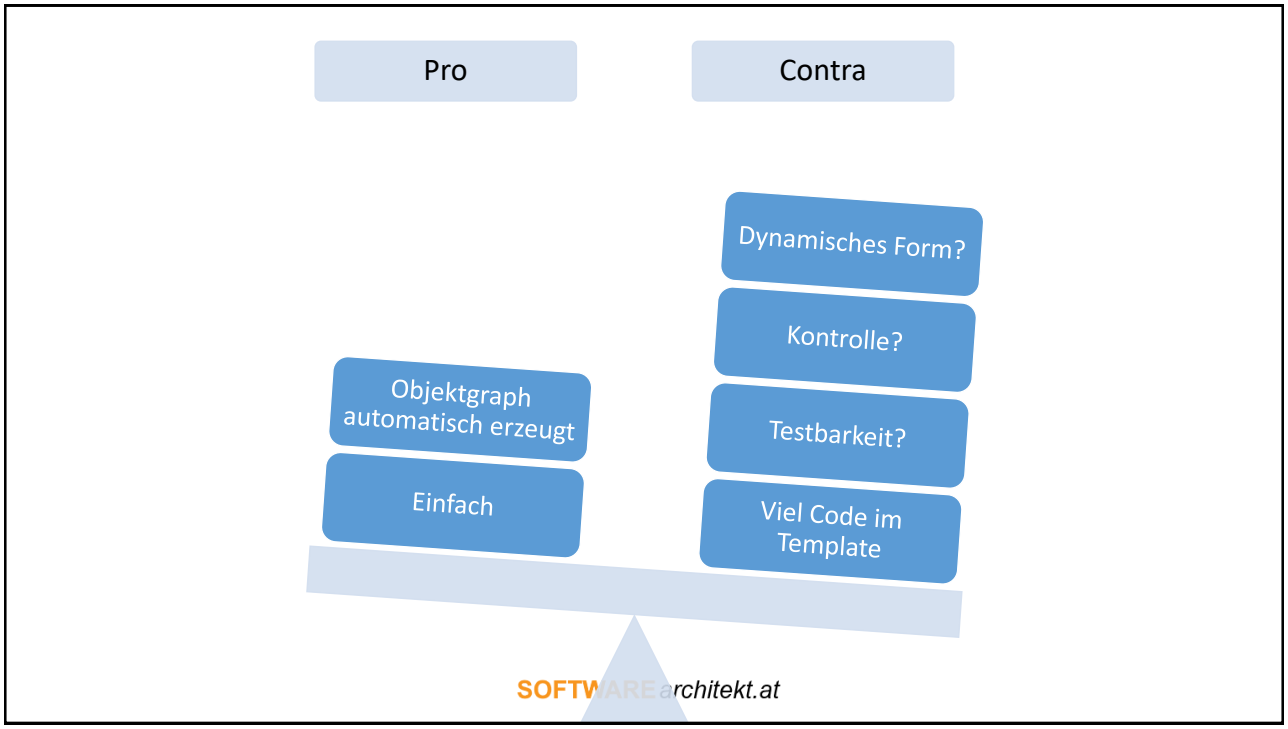

Asynchrone Validierungs-Direktiven

```
@Directive({
    selector: 'input[asyncCity]',
    providers: [ … ]
})
export class AsyncCityValidatorDirective {

    validate(control: AbstractControl): Observable<object> {
        […]
    }

}
```

Asynchrone Validierungs-Direktiven

Token: NG_ASYNC_VALIDATORS

DEMO



Reaktive Formulare

ReactiveFormsModule

```
@NgModule({
  imports: [
    ReactiveFormsModule,
    CommonModule,
    SharedModule,
    [...]
  ],
  [...]
})
export class FlightBookingModule { }
```


Reaktive Formulare

```
export class FlugSuchenComponent {

  form: FormGroup;

  […]
}
```

Reaktive Formulare

```
export class FlugSuchenComponent {

   form: FormGroup;

   constructor(…) {
       let fromControl = new FormControl('Graz');
       let toControl = new FormControl('Hamburg');
       this.form = new FormGroup({ from: fromControl, to: toControl});

       […]

   }
}
```


Reaktive Formulare

```
export class FlugSuchenComponent {

   form: FormGroup;

   constructor(…) {
       let fromControl = new FormControl('Graz');
       let toControl = new FormControl('Hamburg');
       this.form = new FormGroup({ from: fromControl, to: toControl});

       fromControl.validator = Validators.require;
       […]
   }
}
```

Reaktive Formulare

```
export class FlugSuchenComponent {

  form: FormGroup;

  constructor(…) {
      let fromControl = new FormControl('Graz');
      let toControl = new FormControl('Hamburg');
      this.form = new FormGroup({ from: fromControl, to: toControl});

      fromControl.validator =
              Validators.compose([Validators.require, Validators.minLength(3)]);
  }
}
```


Reaktive Formulare

```
export class FlugSuchenComponent {

  form: FormGroup;

  constructor(…) {
      let fromControl = new FormControl('Graz');
      let toControl = new FormControl('Hamburg');
      this.form = new FormGroup({ from: fromControl, to: toControl});

      fromControl.validator =
              Validators.compose([Validators.require, Validators.minLength(3)]);

     fromControl.asyncValidator =
              Validators.compose([…]);
  }
}
```

FormBuilder

```
export class FlugSuchenComponent {

   form: FormGroup;

   constructor(fb: FormBuilder, …) {
      this.form = fb.group({
         von: ['Graz', Validators.required],
         nach: ['Hamburg', Validators.required]
      });
      […]
   }

}
```


FormBuilder

```
export class FlugSuchenComponent {

   form: FormGroup;

   constructor(fb: FormBuilder, …) {
      this.form = fb.group({
         von: ['Graz', [Validators.required, Validators.minLength(3)] ],
         nach: ['Hamburg', Validators.required]
      });
      […]
   }

}
```

FormBuilder

```
export class FlugSuchenComponent {

   form: FormGroup;

   constructor(fb: FormBuilder, …) {
      this.form = fb.group({
         von: ['Graz', [Validators.required, Validators.minLength(3)], [ /* asyncValidator */ ] ],
         nach: ['Hamburg', Validators.required]
      });
      […]
   }

}
```


API

```
this.form.valueChanges.subscribe(change => {
    console.debug('formular hat sich geändert', change);
});
```

```
this.form.controls['von'].valueChanges.subscribe(change => {
    console.debug('von hat sich geändert', change);
});
```

```
let fromValue = this.form.controls['von'].value;
let toValue = this.form.controls['nach'].value;
```

```
let formValue = this.form.value;
```

Reaktive Formulare

```
<form [formGroup]="form">

   <input id="from" formControlName="von" type="text">

   […]

</form>
```


Reaktive Formulare

```
<form [formGroup]="form">

   <input id="from" formControlName="von" type="text">
   <div *ngIf="!form.controls.von.valid">…Error…</div>

   […]

</form>
```

DEMO

Validatoren für reaktive Formulare

Reaktive Validatoren == Funktionen

Ein einfacher Validator

```
function validate (c: AbstractControl): object  {
    if (c.value == 'Graz' || c.value == 'Hamburg') {
        return { };
    }
    return { city: true };
}
```

Validatoren anwenden

```
this.form = fb.group({
    von: [
        'Graz',
        [
            validate
        ],
        [
            /* asyncValidator */
        ]
    ],
    nach: ['Hamburg', Validators.required]
});
```


Parametrisierte Validatoren

```
function validateWithParams(allowedCities: string[]) {

        […]

}
```

Parametrisierte Validatoren

```
function validateWithParams(allowedCities: string[]) {

    return (c: AbstractControl): object => {
            […]
    };

}
```


Parametrisierte Validatoren

```
function validateWithParams(allowedCities: string[]) {

    return (c: AbstractControl): object => {

        if (allowedCities.indexOf(c.value) > -1) {
            return { }
        }

        return { city: true };

    };
}
```

Validatoren anwenden

```
this.form = fb.group({
    von: [
        'Graz',
        [
            validateWithParams(['Graz', 'Hamburg'])
        ],
        [
            /* asyncValidator */
        ]
    ],
    nach: ['Hamburg', Validators.required]
});
```


DEMO