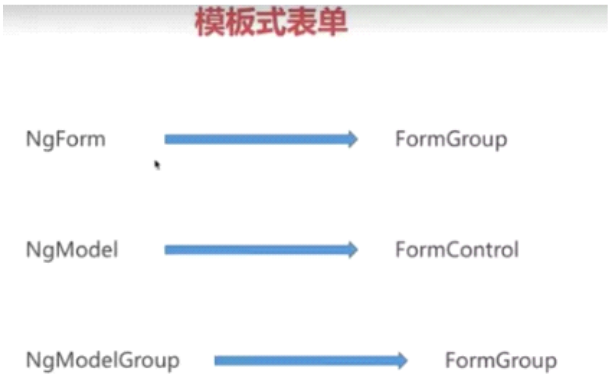
1. 模板式表单

星期五，十一月 23，2018　2:29 下午



使用模板式表单只能使用指令定义数据模型。NgForm、 NgModel 、NgModelGroup 这些指令来自于FormsModel模块。

一、NgForm：用来代表这个整个表单，在ng应用中自动的添加到每个form标签上，（在ng中的表单点击注册没有反应，不会在提交，是因为添加了ngForm，所以表单angular接管了。）隐式的创建一个FormGroup类的实例，代表表单数据模型并且存储表单的数据。标有form指令自动发现标有NgModel的子元素，添加到表单数据模型中。

注意：

a. ngForm可以在form标签外使用，下面两个图效果一样

```
<form action="/regist" method="post">
  <div>用户名:<input type="text" name="username"/></div>
  <div>密码: <input type="text" name="password"/></div>
  <div>确认密码:<input type="text" name="password"/></div>
  <div>手机号: <input type="text" name="mobile"/></div>
  <div><input type="submit" value="注册"/></div>
</form>
```

```
<div ngForm action="/regist" method="post">
  <div>用户名:<input type="text" name="username"/></div>
  <div>密码: <input type="text" name="password"/></div>
  <div>确认密码:<input type="text" name="password"/></div>
  <div>手机号: <input type="text" name="mobile"/></div>
  <div><input type="submit" value="注册"/></div>
</div>
```

b. 如果想让angular不接管这个表单，在后面加上ngNoForm即可。（点击注册能提交，是一个标准的html行为，不是一个ng行为）

```
<form action="/regist" method="post" ngNoForm>
  <div>用户名:<input type="text" name="username"/></div>
  <div>密码: <input type="text" name="password"/></div>
```

c. 可以被模板本地变量引用，以便在模板中访问ngForm实例。（#开头的变量等于ngForm，拿到了ngForm创建的对象了）声明了myForm模板变量，通过它访问我们的ngForm对象的属性。value属性保存着表单里面所有字段的值。

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm.value)">
  <div>用户名:<input type="text" name="username"/></div>
  <div>密码: <input type="text" name="password"/></div>
  <div>确认密码:<input type="text" name="password"/></div>
  <div>手机号: <input type="text" name="mobile"/></div>
  <div><input type="submit" value="注册"/></div>
</form>

<div>
  {{myForm.value | json}}
</div>
```

由于ngForm会拦截标准的表单提交事件，点击按钮没有效果，阻止提交。表单提交会有刷新，所以用(ngSubmit)事件绑定来代替

template.ts

```
onSubmit(value: any) {
  console.log(value);
```

}

这样就能在控制台打印出myForm.value值，但是值现在是空的，如果想显示，需要在每个标签中加上ngModel

二、NgModel：代表表单一个字段，隐式创建一个FormControl的实例来代表数据字段模型，用FormControl模型存储字段的值，标记了ngForm指令元素内来使用ngModel，不需要用括号，但是添加了ngModel元素必需指定一个name属性，name属性的值会成为ngFrom对象的value属性所对应的效果.

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm.value)">
  <div>用户名:<input type="text" ngModel name="username"/></div>
  <div>密码: <input type="text" ngModel name="password"/></div>
  <div>确认密码:<input type="text" ngModel name="confirmpass"/></div>
  <div>手机号: <input type="text" ngModel name="mobile"/></div>
  <div><input type="submit" value="注册"/></div>
</form>

<div>
  {{myForm.value | json}}
</div>
```

效果图：

用户名:dsada
密码: sadadsa
确认密码:dsds
手机号: dsdsd
注册
{ "username": "dsada", "password": "sadadsa", "confirmpass": "dsds", "mobile": "dsdsd" }

ngModel属性创建的对象能通过模板变量来引用

```
<div>用户名:<input #username="ngModel" ngModel  type="text" name="username"/></div>
<div>{{username.value}}</div> <!--通过这个获取用户名的值-->
```

效果图：

用户名:dsada
密码:
确认密码:
手机号:
注册
{ "username":"dsada" "password": "
dsada

三、NgModelGroup：NgModelGroup是表单的一部分，也会创建一个FormGroup的实例，会在ngForm的对象value属性中表现为一个嵌套的对象。

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm.value)">
  <div ngModelGroup="userInfo">
    <div>用户名:<input type="text" #username="ngModel" ngModel name="
    <div>密码: <input type="text" ngModel name="password"/></div>
  </div>
  <div>确认密码:<input type="text" ngModel name="confirmpass"/></div>
  <div>手机号: <input type="text" ngModel name="mobile"/></div>
  <div><input type="submit" value="注册"/></div>
</form>
```

效果：

用户名:ssaf
密码: fafaf
确认密码:
手机号:
注册
{"userInfo":{"username": "ssaf", "password": "fafaf"}}

从而将user那么和password嵌套到userInfo对象中

2. 响应式表单

星期五，十一月 23，2018　4:19 下午

一个数据模型，在ngForms模块中由三个类组成，FormControl,FormGroup,FormArray。
一、FromControl:

构成表单的基本单位，代表input元素，可以代表日历，下拉选择框，保存着元素当前的值，校验状态，是否修改过
演示：如何创建FromControl?

表单.ts

```
/* 接收一个参数，这个参数指定FormControl的初始值，如果它与页面的input连接时，
这个input的初始值即为这个参数值。而ngModel指定就会为附着的元素input创建一个FormContrl*/
username: FormControl = new FormControl( formState: "AAA");
```

二、FormGroup:

可以代表表单一部分，也可以用于代表整个表单，是多个Formcontrol的集合。一个Formcontrol是无效的，那个FormGroup也是无效的。他是代表从···到···的固定子集
演示：如何创建FormGroup?日期范围字段。

```
// 创建日期
  formModel: FormGroup = new FormGroup({
    form: new FormControl(), // 日期范围
    to: new FormControl()
  });
```

三、FormArray:

与FormGroup类似，但比FormGroup多一个长度属性。代表可以增长的字段集合。例如有email，可以用这个输入任意一个。

```
// 创建邮箱
/*与FormGroup的formContrl不同的是，没有key,只能靠序号来访问,第一个序号就是0*/
emails: FormArray = new FormArray( controls: [
  new FormControl( formState: 'aa@df.com'),
  new FormControl( formState: 'BB@DD.COM')
])
```

Note：与FormGroup的formContrl不同的是，FormArray没有key,只能靠序号来访问,第一个序号是0。

介绍下指令

响应式表单指令与模板指令完全不同，来自于ReactiveFormsModule

```
imports: [
  BrowserModule,
  FormsModule,
  ReactiveFormsModule
],
```

注意：

1. 指令是form开头，ng开头的是模板式表单；

2. 响应式form指令不可引用 #myForm="myform" .操作数据模型只能在代码中操作，模板指令是在代码中操作。模板表单变响应式，通常用formGroup代表整个表单。

一. 表单.html（formGroup代表整个表单）

```
<!--由于ts定义的FormControl不在formGroup里面故，
定义formControl也必须在formGroup外面定义，如下-->
<!--但由于username不在formGroup里面，故onSubmit点击事件后，
在控制台打印的信息，不会携带username的信息-->
<form [formGroup]="forModel" (submit)="onSubmit()">
  <!--表单处理方式变成了响应式， 所以做一个提交的方法-->
  <!--由于formGroup是个属性，所以这边需要用[]来绑定-->
  <div>
    起始日期<input type="date">
    截止日期<input type="date">
  </div>
  <div>
    <ul>
      <li>
        <input type="text">
      </li>
    </ul>
    <button type="button">增加Email</button>
  </div>
  <div>
    <button type="submit">保存</button>
  </div>
</form>
```

表单.ts

```
forModel: FormGroup = new FormGroup({}) /*表示整个表单的数据*/
onSubmit() {
    console.log(this.forModel.value);
 }
```

二. formGroupName链接一个group：加数据

```
// 创建日历
formModel: FormGroup = new FormGroup( controls: {
  // 定义一个单独的FormControl
  // 用FormGroupName来连接FormGroup,FormGroupName就是dateRange
  dateRange: new FormGroup( controls: {
    // FormControlName就是from和to，前端调用就用这两个
    from: new FormControl(), // 日期范围
    to: new FormControl()
  }),
});
```

html展示：

```
<div>username2:<input formControlName="username2"></div>
<!--formGroupName的值是个字符串，所以用字符串绑定就可以-->
<div formGroupName="dateRange">
  <!--formControlName也是字符串，前端的逻辑和ts一样，都是在嵌套进去-->
  <!--formControlName必须用在formGroupName的指令范围内-->
  起始日期<input type="date" formControlName="from">
  截止日期<input type="date" formControlName="to">
</div>
```

这么做的效果是可以把日期控件的值保存。[formGroup]="forModel"是后台的属性，formGroupName的值是一个字符串，formControlName的值也是一个字符串，用在formGroupName指令内

三、formAarryName

必须用在<form 的formGroup之内。

步骤1：将模板的ul和控制器的emails的formAarry里的formControl绑定了

```
formModel: FormGroup = new FormGroup( controls: {
  // 定义一个单独的FormControl
  username2: new FormControl( formState: "BBB"),
  // 用FormGroupName来连接FormGroup,FormGroupName就是dateRange
  dateRange: new FormGroup( controls: {
    // FormControlName就是from和to，前端调用就用这两个
    from: new FormControl(), // 日期范围
    to: new FormControl()
  }),
  emails: new FormArray( controls: [
    new FormControl( formState: 'aa@df.com'),
    new FormControl( formState: 'BB@DD.COM')
  ])
});
```

步骤2：点击按钮向eamils数组添加元素，添加输入框

```
addEmail(){
  /*添加一个邮箱输入框，需要拿到FormAarry as FormArray是转换为FormArray类型*/
  const emails = this.formModel.get('emails') as FormArray;
  /*模板根据数组循环，数组多一个元素，多一个输入框*/
  emails.push(new FormControl())
}
```

步骤3：html绑定：

```
<div>
  <ul formArrayName="emails">
    <!--由于formArray没有key，所以在html中一般与*ngFor联用，循环出来他的formControlName-->
    <!--this.formModel.get('emails').controls的controls是代表email的集合-->
    <!--需要当前循环下标-->
    <li *ngFor="let e of this.formModel.get('emails').controls; let i = index">
      <!--[formControlName]这里需要属性绑定的语法-->
      <input type="text" [formControlName]="i">
    </li>
  </ul>
  <!--点击按钮去增加一个输入块-->
  <button type="button" (click)="addEmail()">增加Email</button>
```

效果：就增加了邮箱输入框，保存成功

四、formControl

由于ts中单独定义的formControl不会放在FormGroup内部，所以需要在外部定义：

ts：

```
/* 接收一个参数，这个参数指定FormControl的初始值，如果它与页面的input连接时，
这个input的初始值即为这个参数值。而ngModel指定就会为附着的元素input创建一个FormContrl*/
username: FormControl = new FormControl( formState: "AAA");

// 创建日历
formModel: FormGroup = new FormGroup( controls: {
  // 用FormGroupName来连接FormGroup,FormGroupName就是dateRange
  dateRange: new FormGroup( controls: {
```

html：

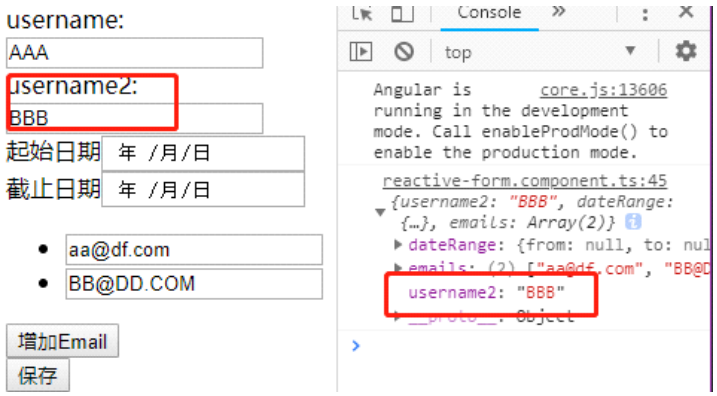
```
<!--由于ts定义的FormControl不在formGroup里面故，定义formControl也必须在formGroup外面定义，入下-->
<!--但由于username不在formGroup里面，故onSubmit点击事件在控制台打印的信息，不会携带username的信息-->
<input [formControl]="username">
<!--表单处理方式变成了响应式， 并做一个提交的方法-->
<!--由于formGroup是个属性，所以这边需要用[]来绑定-->
<form [formGroup]="formModel" (submit)="onSubmit()">
  <!--而formGroupName的值是个字符串，所以用字符串绑定就可以-->
  <div formGroupName="dateRange">
    <!--formControlName也是字符串，前端的逻辑和ts一样，都是在嵌套进去-->
```

但由于username不在formGroup里面，故onSubmit点击事件在控制台打印的信息，不会携带username的信息。

如果需要username的信息，则需要通过以下方式改变，在formGroup中定义一个单独的formControl

```
// 创建日历
formModel: FormGroup = new FormGroup( controls: {
  // 定义一个单独的FormControl
  username2: new FormControl( formState: "AAA"),
  // 用FormGroupName来连接FormGroup FormGroupName就是dateRange
  dateRange: new FormGroup( controls: {
    // FormControlName就是from和to，前端调用就用这两个
    from: new FormControl(), // 日期范围
    to: new FormControl()
  }),
  emails: new FormArray( controls: [
    new FormControl( formState: 'aa@df.com'),
    new FormControl( formState: 'BB@DD.COM')
  ])
});
```

效果：



可以看出来username并没有在控制台打印而username2在控制台打印出来了

注意：所有指令都是以form开头的。以name结尾，不需要使用[]，只需要指定名字；只能用在【formGroup】指令之内。不以name结尾则用[]。

3. 响应式表单重构

星期五，十一月 23，2018　7:03 下午

1. 首先写数据模型

```
export class ReactiveRegistComponent implements OnInit {

  formModel:FormGroup;
  constructor() {
    this.formModel = new FormGroup( controls: {
      username: new FormControl(),
      mobile: new FormControl(),
      passwordGroup: new FormGroup( controls: {
        password: new FormControl(),
        confirmp: new FormControl()
      })
    })
  }

  onSubmit() {
    console.log(this.formModel.value)
  }
```

2. 指令链接

```
<form [formGroup]="formModel" (submit)="onSubmit()">
  <div>用户名:<input type="text" formControlName="username"/></div>
  <div>手机号: <input type="text" formControlName="mobile"/></div>
  <div formGroupName="passwordGroup">
    <div>密码: <input type="text" formControlName="password"/></div>
    <div>确认密码:<input type="text" formControlName="confirmp"/></div>
  </div>
  <div><input type="submit" value="注册"/></div>
</form>
```

简化方法：用FormBulider来配置一个表单模型比用new关键字实例化类，代码少，fb.group相当于new 了一个group，可以接受另外的参数，校验，用一个数组实例化formcontrol实例，第一个元素初始值，校验方法，交互校验方法。多于三个，其他元素忽略。

```
// 用FormBuilder来定义上面的forModel来简化代码
// FormBuilder需要用依赖注入的方式定义
constructor(fb: FormBuilder) {
  this.formModel = fb.group( controlsConfig: {
        username: [''],
        mobile: [''],
        passwordGroup: fb.group( controlsConfig: {
          password: [''],
          confirmp: ['']
        })
      })
}
```

4. 表单校验

星期五，十一月 23，2018　　7:22 下午

Angular的校验器：普通的方法
校验响应式表单
校验模板式表单

一、使用表单校验数据

1、angular中自带了几个常见的表单校验的是在Validators中的required,minLength,maxLength，pattern等
2、自定义表单校验器：校验器就是一个方法，名字可以自定义，但是需要一个参数且参数必须和验证的那个Control类型相同，必须有一个返回值，返回值可以是任意结构的对象，但是这个对象有一个要求，就是他的key必须要的string类型的，value可以是任意类型

格式

（1）

```
/*校验器就是一个方法，名字可以自定义，但是需要一个参数且参数必须和验证的那个Control类型相同，
必须有一个返回值，返回值可以是任意结构的对象，但是这个对象有一个要求，
就是他的key必须要的string类型的，value可以是任意类型*/
//下面就是这个校验器方法的格式
xxx(control: AbstractControl): {[key: string]: any} {
  return null;
}
```

（2）

```
export function xxx(control:FormControl): any{

  const volid = test(); // 校验方法
  // key: 为html调用的，value可以是值或者对象
  return volid ? null: {key: value}
}
```

后面还会介绍返回是一个流的格式，供异步校验处理

3、响应式表单字段中可以写三个值，第一个是返显到页面上的输入值，第二个参数是校验器(可以是一数组)，第三个参数异步校验(常见判断手机号码，用户名是否重复注册)

一、自带表单校验演示例子：

用户注册的验证

```
// 用FormBuilder来定义上面的forModel来简化代码
// FormBuilder需要用依赖注入的方式定义
constructor(fb: FormBuilder) {
  this.formModel = fb.group( controlsConfig: {
        username: ['', [Validators.required, Validators.minLength( minLength: 6)]],/*必填，最短6维*/
        mobile: [''], // 将校验器放入手机号字段

        passwordGroup: fb.group( controlsConfig: {
          password: ['', Validators.minLength( minLength: 6)],
          confirmp: ['']
        },) // 校验的是group，需要声明到一个对象里key为validators
      })
}


onSubmit() {
  // 校验结果
  let isValid: boolean = this.formModel.get('username').valid;
  console.log('username的校验结果是: ' + isValid);
  // 校验错误信息
  let errors: any = this.formModel.get('username').errors;
  console.log('username的错误信息是' + JSON.stringify(errors));
  console.log(this.formModel.value)

}
```

二、自定义一个校验方法的步骤(手机号校验)

```
// 自定义校验器，手机号校验
mobileValidator(control: FormControl): any {
  // 手机号正则表达式
  const mobieReg = /^(((13[0-9]{1})|(15[0-9]{1})|(18[0-9]{1}))+\d{8})$/;
  // 校验输入的值
  const valid = mobieReg.test(control.value);
  console.log('mobile的校验结果是: ' + valid);
  /*返回空代表通过，valid为false的时候，返回对象，key,随便给一个值*/
  return valid ? null : {mobile : true};
}
```

调用校验方式：

```
// 用FormBuilder来定义上面的forModel来简化代码
// FormBuilder需要用依赖注入的方式定义
constructor(fb: FormBuilder) {
  this.formModel = fb.group( controlsConfig: {
        username: ['', [Validators.required, Validators.minLength( minLength: 6)]],
        mobile: ['', this.mobileValidator], // 将校验器放入手机号字段

        passwordGroup: fb.group( controlsConfig: {
          password: ['', Validators.minLength( minLength: 6)],
          confirmp: ['']
        },) // 全局引入
      })
}
```

为了满足同时校验几个字段(例如密码校验)

```
  equalValidator(group: FormGroup): any {
    const password: FormControl = group.get('password') as FormControl;
    const confirmp: FormControl = group.get('confirmpass') as FormControl;
    // 校验结果
    const valid: boolean = (confirmp.value === password.value);
    console.log('密码校验结果' + valid);
    return valid ? null : {equal: true};
  }
```

调用密码校验方法

```
// 用FormBuilder来定义上面的forModel来简化代码
// FormBuilder需要用依赖注入的方式定义
constructor(fb: FormBuilder) {
  this.formModel = fb.group( controlsConfig: {
        username: ['', [Validators.required, Validators.minLength( minLength: 6)]],
        mobile: ['', this.mobileValidator], // 将校验器放入手机号字段

        passwordGroup: fb.group( controlsConfig: {
          password: ['', Validators.minLength( minLength: 6)],
          confirmp: ['']
          // 校验的是group，需要声明到一个对象里key必须为validator
        },  extra: {validator: this.equalValidator})
      })
}
```

Note: 校验group的时候对象的key一定是validator

如果想声明一个外部的类作为全局通用的检验方式，下面将给出比较通用的例子

新建一个ts，把校验方法放进去，然后导出。移出的方法不是ts 的类的方法，而是全局的ts的函数，需要用function来声明，用export来暴露出去。然后在模板组件里直接引用它。

Validators.ts(密码验证，手机号验证)

```ts
import {FormControl, FormGroup} from "@angular/forms";

export function mobileValidator(control: FormControl): any {
  // 手机号正则表达式
  const mobieReg = /^(((13[0-9]{1})|(15[0-9]{1})|(18[0-9]{1}))+\d{8})$/;
  // 校验输入的值
  const valid = mobieReg.test(control.value);
  console.log('mobile的校验结果是: ' + valid);
  /*返回空代表通过，valid为false的时候，返回对象，key,随便给一个值*/
  return valid ? null : {mobile : true};
}


export function equalValidator(group: FormGroup): any {
  const password: FormControl = group.get('password') as FormControl;
  const confirmp: FormControl = group.get('confirmpass') as FormControl;
  // 校验结果
  const valid: boolean = (confirmp.value === password.value);
  console.log('密码校验结果' + valid);

  return valid ? null : {equal: {err: "密码和确认密码不匹配"}};
}
```

在组件中的调用方法：

```ts
// 用FormBuilder来定义上面的forModel来简化代码
// FormBuilder需要用依赖注入的方式定义
constructor(fb: FormBuilder) {
  this.formModel = fb.group( controlsConfig: {
        username: ['', [Validators.required, Validators.minLength( minLength: 6)]]
        mobile: ['', mobileValidator], // 全局引入，这样所有的组件都可以引用这些校验器

        passwordGroup: fb.group( controlsConfig: {
          password: ['', Validators.minLength( minLength: 6)],
          confirmp: ['']
          // 校验的是group，需要声明到一个对象里key必须为validator
        }, extra: {validator: equalValidator}) // 全局引入
      })
}
```

```ts
onSubmit() {
  // 校验结果
  let isValid: boolean = this.formModel.get('username').valid;
  console.log('username的校验结果是: ' + isValid);
  // 校验错误信息
  let errors: any = this.formModel.get('username').errors;
  console.log('username的错误信息是' + JSON.stringify(errors));
  console.log(this.formModel.value)

  // 只有这里实你的表单全都是合法的时候 this.forModel.valid才是true
  if (this.formModel.valid){
    console.log(this.formModel.value)
  }
}
```

模板

```html
<!--自定义error的错误信息-->
<form [formGroup]="formModel" (submit)="onSubmit()">
  <div>用户名:<input type="text" name="username" formControlName="username"/></div>
  <!--hasError两个参数，一是校验的是否必填，校验器失败返回的对象的key的值，有值就是失败的
    。二是检查的字段名字，hidden为true隐藏，所以取反。-->
  <div [hidden]="!formModel.hasError('required','username')">用户名是必填项</div>
  <div [hidden]="!formModel.hasError('minlength','username')">用户名最小长度为6</div>
  <div>电话: <input type="text" name="mobile" formControlName="mobile"/></div>
  <div [hidden]="!formModel.hasError('mobile','mobile')">请输入正确的手机号</div>
  <div formGroupName="passwordGroup">
    <div>密码: <input type="text" name="password" formControlName="password"/></div>
    <!--这里的hasError第二个参数是个数组，因为这个的passwordsGroup是一个二级属性，所以需要用数组来校验-->
    <div [hidden]="!formModel.hasError('minlength',['passwordGroup','password'])">密码最小长度是6</div>
    <div>确认密码:<input type="text" name="confirmp" formControlName="confirmp"/></div>
    <!--{{formModel.getError('equal','passwordGroup')?.err}} 可以直接引用后台ts返回的错误信息-->
    <div [hidden]="!formModel.hasError('equal','passwordGroup')">{{formModel.getError('equal','passwordGroup')?.err}}</div>
  </div>
  <div><input type="submit" value="提交"/></div>
</form>
```

异步校验器：可观测的流

Validators.ts

```
// 异步校验器
export function mobileAsyncValidator(control: FormControl): any {
  // 手机号正则表达式
  const mobieReg = /^(((13[0-9]{1})|(15[0-9]{1})|(18[0-9]{1}))+\d{8})$/;
  // 校验输入的值
  const valid = mobieReg.test(control.value);
  console.log('mobile的校验结果是: ' + valid);
  /*返回时把返回值放在一个流里返回的*/
  return of( a: valid ? null : {mobile : true});
}
```

组件调用：

```
// 用FormBuilder来定义上面的forModel来简化代码
// FormBuilder需要用依赖注入的方式定义
constructor(fb: FormBuilder) {
  this.formModel = fb.group( controlsConfig: {
        username: ['', [Validators.required, Validators.minLength( minLength: 6)]],
        mobile: ['', mobileValidator, mobileAsyncValidator], // 引入异步校验器

        passwordGroup: fb.group( controlsConfig: {
          password: ['', Validators.minLength( minLength: 6)],
          confirmp: ['']
          // 校验的是group，需要声明到一个对象里key必须为validator
        }, extra: {validator: equalValidator}) // 全局引入
      })
}
```

模板

```
<div>{{formModel.status}}</div><!--表单的状态-->
```

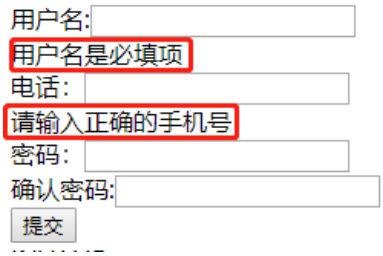
效果：

用户名:
用户名是必填项
电话:
请输入正确的手机号
密码:
确认密码:
提交
INVALID

5. 状态字段

星期一，十一月 26，2018　　9:04 上午

解决用户名初始化后，未进行输入就显示错误信息的问题。



Angular提供了五个验证字段：

1. touched和untouched：关注是否获取过焦点。获取过焦点就touched为true，untouched为false，反之即相反。这两个信息一般用来控制错误信息的显示
2. pristine（本来的，原来的）和dirty:如果一个字段的值从来没有被改变过，那么它的pristine是true,dirty为false;修改过，pristine是false，dirty是true
3. pending:当一个字段处于异步校验时，为true,显示图片或者文字让用户知道你正在异步校验。
4. 对于整个表单而言，只要有一个字段是touched，则都是touched的，只有所有表单的字段是untouched，整个表单才是untouched的；只要有一个字段是dirty，则都是dirty的，只有所有表单的字段是pristine，整个表单才是pristine的

根据状态添加样式.css文件：

```
.hasError{border:solid 1px red;}
```

html文件

```
<!--自定义error的错误信息-->
<form [formGroup]="formModel" (submit)="onSubmit()">
  <!--[class.hasError]="formModel.get('username').invalid && formModel.get('username').touched-->
  <!--如果用户名输入错误，且用户名被聚焦过，就给他一个hasError样式，然后展示css上定义的.hasError样式-->
  <div>用户名:<input [class.hasError]="formModel.get('username').invalid && formModel.get('username').touched"
                  type="text" name="username" formControlName="username"/></div>
  <!--控制整体的错误信息是显示还是不显示，用户信息通过或者用户名输入未框获取焦点-->
  <div [hidden]="formModel.get('username').valid || formModel.get('username').untouched">
    <!--hasError两个参数，一是校验的是否必填，校验器失败返回的对象的key的值，有值就是失败的
      。二是检查的字段名字，hidden为true隐藏，所以取反。-->
    <div [hidden]="!formModel.hasError('required','username')">用户名是必填项</div>
    <div [hidden]="!formModel.hasError('minlength','username')">用户名最小长度为6</div>
  </div>
  <div>电话: <input type="text" name="mobile" formControlName="mobile"/></div>
  <div [hidden]="formModel.get('mobile').valid || formModel.get('mobile').pristine">
    <div [hidden]="!formModel.hasError('mobile','mobile')">请输入正确的手机号</div>
  </div>
  <div [hidden]="!formModel.get('mobile').pending">
    正在校验手机号的合法性
  </div>
  <div formGroupName="passwordGroup">
    <div>密码: <input type="text" name="password" formControlName="password"/></div>
    <!--这里的hasError第二个参数是个数组，因为这个的passwordsGroup是一个二级属性，所以需要用数组来校验-->
    <div [hidden]="!formModel.hasError('minlength',['passwordGroup','password'])">密码最小长度是6</div>
    <div>确认密码:<input type="text" name="confirmp" formControlName="confirmp"/></div>
    <!--{{formModel.getError('equal','passwordGroup')?.err}} 可以直接引用后台ts返回的错误信息-->
    <div [hidden]="!formModel.hasError('equal','passwordGroup')">
      {{formModel.getError('equal','passwordGroup')?.err}}
    </div>
  </div>
  <div><input type="submit" value="提交"/></div>
</form>
```

效果图：

用户名:
用户名是必填项
电话:
密码:
确认密码:
提交

6. 模板式表单校验

星期一，十一月 26，2018　　10:16 上午

模板式表单里，指令是唯一可以用的东西，只能将校验方法包装成指令，然后才能在膜板使用。

自带的模板校验指令：

Angular提供了一些校验器有对应的指令，像之前的required、minlength等等写到模板上。但为了区分指令和属性，要在form标签上加上noValidator属性，不要启动游览器默认的表单校验。

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm.value,myForm.valid)" novalidator>
  <div>用户名:<input  ngModel required minlength="6" type="text" name="username"/></div>
```

自定义校验指令：

步骤1：生成两个指令(以手机号和密码校验为例)

```
ng g directive  directives/mobileValidator
ng g directive  directives/equalValidator
```

步骤2：

指令和组件基本上是一样的，区别在于指令没有模板(不需要膜板)即没有templateUrl，selector中的字符串就是在模板引用的指令(作为属性来用)，并支持用户自定义。为了让html调用这些指令，需要将其包装成一个指令，通过providers来实现(注入器的方式)。

mobileValidator.ts:

```
@Directive({
  selector: '[Mobile]',
  providers: [{provide: NG_VALIDATORS, useValue: mobileValidator, multi: true}]
  /*校验器的provide一定是NG_VALIDATORS,写死;useValue为校验方法的名字, multi等于true*/
})
export class MobileValidatorDirective {
  constructor() { }
}
```

Note：校验器的provide一定是NG_VALIDATORS,写死;useValue为校验方法的名字，multi等于true

appEqualValidator.ts

```
import { Directive } from '@angular/core';
import {NG_VALIDATORS} from '@angular/forms';
import {equalValidator, mobileValidator} from '../validator/Validators';

@Directive({
  selector: '[equal]',
  providers: [{provide: NG_VALIDATORS, useValue: equalValidator, multi: true}]
  /*multi:一个token挂多个值*/
})
export class EqualValidatorDirective {
  constructor() { }
}
```

步骤3：

指令为模板表单加上了校验，由于模板式表单没有数据模型，想要在控制器里了解信息，需要从模板将想要了解的信息传入控制器。比如提交表单，想知道表单是否有效，那提交方法里就要传入myForm.valid,在控制器的提交方法里判断这个属性。

html绑定：通过指令，将校验逻辑加入表单

```
                                    <!-- 将myForm.valid传入控制器 -->
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm.value,myForm.valid)" noValidator>
  <!-- 引入自带的Validators的校验方法 required minlength -->
  <div>用户名:<input  ngModel  required minlength="6" type="text" name="username"/></div>
  <!-- 引入自定义的校验方法mobile -->
  <div>手机号：<input ngModel mobile type="text" name="mobile"/></div>
  <!-- 引入自定义的校验方法equal -->
  <div ngModelGroup="passwordGroup" equal>
     <div>密码：<input ngModel type="password" minlength="6" name="password"/></div>
     <div>确认密码:<input ngModel type="password" name="confirmpass"/></div>
    </div>
  </div>
  <div><input type="submit" value="提交"/></div>
</form>
```

控制台接收这个值：

```
onSubmit(value: any, valid: boolean) {
    console.log(valid);
   console.log(value);
  }
```

效果图：

步骤4：

控制错误信息是否显示，要用 hasError，即用myForm.form.hasError来判断，然后再用状态属性控制它是否显示

因为在模板表单，状态属性模板表单和响应式不同。模型的值和状态的变更是异步的，用同步的方式访问这些属性就会报错，所以用onMobileInput(myForm)来把表单传入后台

```
<form #myForm="ngForm" (ngSubmit)="onSubmit(myForm.value,myForm.valid)" novalidator>
  <div>用户名:<input  ngModel  required minlength="6" type="text" name="username" (input)="onMobileInput(myForm)"/></div>
  <!--控制整体的错误信息是显示还是不显示，用户信息通过或者用户名输入框获取焦-->
  <div [hidden]="mobileValid || mobileUntouched">
    <!-- 都用!myForm.form来判断 -->
    <div [hidden]="!myForm.form.hasError('required','username')">用户名是必填项</div>
    <div [hidden]="!myForm.form.hasError('minlength','username')">用户名最新长度是6</div>
  </div>

  <div>手机号：<input ngModel appMobileValidator type="text" name="mobile"/></div>
   <div [hidden]="myForm.form.get('mobile').valid || myForm.form.get('mobile').pristine">
    <div [hidden]="!myForm.form.hasError('mobile','mobile')">请输入正确的手机号</div>
   </div>

  <div ngModelGroup="passwordGroup" appEqualValidator>
     <div>密码：<input ngModel type="password" minlength="6" name="password"/></div>
    <div [hidden] = "!myForm.form.hasError('minlength', ['passwordsGroup', 'password'])">密码最小长度是6</div>
     <div>确认密码:<input ngModel type="password" name="confirmpass"/></div>
    <div [hidden] = "!myForm.form.hasError('equal', 'passwordsGroup')">
      {{myForm.form.getError('equal', 'passwordsGroup')?.descxx}}
    </div>
  </div>
  <div><input type="submit" value="提交"/></div>
</form>
```

将状态都传入后台，然后再到前台调用

这两个都要改成上面的那个用户名的格式

后台ts：

```
mobileValid: boolean = true;
mobileUntouched: boolean = true;
onMobileInput(form: NgForm) {
    if (form) {
      this.mobileValid = form.form.get('mobile').valid;
      this.mobileUntouched = form.form.get('mobile').untouched;
    }
  }
```