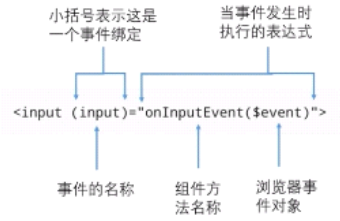
Event binding数据绑定

2018年11月21日 20:41

0. 数据绑定：
   数据绑定常见的有事件绑定、dom属性绑定、html属性绑定、css绑定、样式绑定等等
   a. 简单的绑定，一下两种是比较简单的绑定
   html中用[]来表示属性绑定
   <img [src]="imgUrl">
   <!--这两种方式都可以，插值表达式和属性绑定是一样的-->
   <img src="{{imgUrl}}">
   在component.ts中只要定义imgUrl就行
   imgUrl: string = 'http://placehold.it/150x100';

1. 事件绑定：



   <button(Click)= "saved = true">
   这表示点击后saved 的值为true

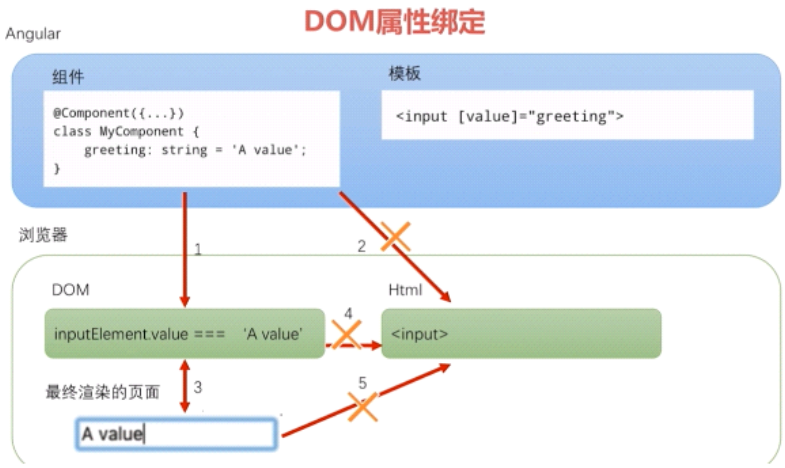
   也可以自定义事件绑定：
   在html中一般的事件绑定如下形式
   <!--事件绑定-->
   <button (click)="doOnClick($event)">点我</button>
   <input value="Tom" (input)="doOnInput($event)">
   $event是传递当前的浏览器事件对象传到.ts中，方便后台调用处理
   而在component.ts中需要定义他们的触发事件像这样

```
// event就是前端传过来的代表当前控件的$event
doOnClick(event: any) {
  console.log(event);
}
// event就是前端传过来的代表当前控件的$event
doOnInput(event: any) {
  // 这是dom属性，是对应输入的值，可以改变
  console.log(event.target.value);
  // 这是html属性，指定初始值，不可改变，
  // 即此例子指的是<input>空间中的value属性，固定就是定义的Tom
  console.log(event.target.getAttribute( qualifiedName: "value"));
}
```

   Note：注意dom和html属性的区别，下文还会做区别说明

   另外有个特殊的空间属性：
   <!--对于html而言无法设置disabled来决定这个按钮是否被禁用，只要在button中出现了disabled这个botton就会被禁用，无论disabled为False还是true-->
   <button disabled>点我</button>

2. dom属性绑定与html绑定的关系：
   a. 少量HTML属性和DOM属性之间有着1:1的映时。如id
   b. 有些HTML属性没有对应的DOM属性，如colspan.
   c. 有些DOM属性没有对应的HTML属性，如textContent.
   d. 就算名字相同，HTML属性和DOM属性也不是同一样东西
   e. HTML属性的值指定了初始值，DOM属性的值表示当前值。DOM属性的值可以改变；HTML属性的值不能改变，
   f. 模版绑定是过DOM属性和事件来工作的，而不是HTML属性
   后三点更重要

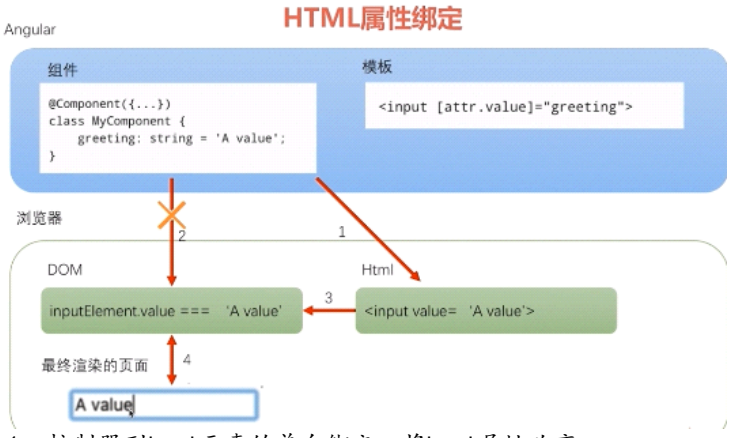


1. 组件绑定的是dom属性
2. html属性没有被更改过
3. 渲染的页面也是由dom绑定渲染的
4. 即使dom属性的值被改变，html属性也不会变
5. html属性只是初始值，不会随别的元素的改变而改变

Note:

有dom属性，优先用dom属性绑定

[]是属性的绑定语法。属性外面有[]，右侧会把引号内作为变量，通过后台查询到的值赋值给左边。如果没有方括号，就会当表达式用，相当于把一个字符串复制给左边。



1. 控制器到html元素的单向绑定，将html属性改变
2. angular不会改变dom元素的属性
3. 浏览器同步了html元素，将dom元素赋予了新的值
4. dom与页面也是同步的，从而渲染了最后的页面

3. dom属性绑定和0.简单绑定的例子一样，就不在赘述了
4. html属性绑定：

   但某一标签中没有dom属性，则考虑HTML属性绑定如下例：

   html中：

```
<tr>
    <td colspan="{{1+1}}">慕课网</td>
    <!--由于td中没有dom的colspan属性即出现dom元素没有对应属性的情况，故上面那段代码控制台会报错-->
    <!--这时候用到html属性绑定，而html绑定需要在原有的属性(colspan)前加上attr作为html绑定-->
    <!--attr是html属性绑定的标志，[attr.colspan]="size"就是把size绑定到td的html的colspan属性-->
    <td [attr.colspan]="size">慕课网</td>
</tr>
```

   在component.ts中只要定义一个size传过来就行

```
size: number = 2;
```

5. css绑定：

   css文件中先定义好a b c的样式

```css
.a {
  background: yellow;
}

.b {
  color:red;
}

.c {
  font-size: 60px;
}
```

html中：

```
<!--原始状态-->
<div class="a b c">慕课网</div>

<!--css绑定全替换-->
<div [class]="divClass">慕课网</div>

<!--改变某一个样式-->
<!--当对应的表达式(isBig)为true时，anguler就会把绑定的样式添加到class里，反之则移除-->
<div class="a b" [class.c]="isBig">慕课网</div>

<!--管理多个类名的样式,[ngClass]要求后面为k-V型的对象 -->
<div [ngClass]="divNgClass">慕课网</div>
```

对应的component.ts

```ts
export class BindComponent implements OnInit {

  divClass:string;

  isBig: boolean = false;

  divNgClass: any = {
    a: false,
    b: false,
    c: false,
  };

  constructor() {
    setTimeout( callback: () => {
      // 全改变
      this.divClass = 'a b c;'
      // 单一样式改变
      this.isBig = true;
      // css统一管理
      this.divNgClass = {
        a: true,
        b: true,
        c: true
      }
    }, ms: 3000);
  }
```

6. 样式绑定：

Html:

```
<!--样式绑定-->
<!--isDev为ts返回的值，如果是true选冒号前面的，反之选后面的-->
<!--样式绑定就不能像css一样在文件中定义标签属性的样式，故需要直接写出来-->
<!--样式的格式一般是[style.样式名]="..."-->
<div [style.color]="isDev?'red':'blue'">慕课网</div>
<!--类似于font-size这种有大小或者像素的单位差别时，.em就是他的单位，即在font-size后面加单位-->
<div [style.font-size.em]="isDev?3:1">慕课网</div>

<!--管理多个类名的style，[ngstyle]也要求后面为k-V型的对象 -->
<div [ngStyle]="divNgStyle">慕课网</div>
```

对应的component.ts：

```ts
export class BindComponent implements OnInit {

  isDev: boolean = false;

  divNgStyle: any = {
    color: 'red',
    background: 'yellow'
  };

  constructor() {
    setTimeout( callback: () => {
      // 改变样式
      this.isDev = true;
      // 批量更改style
      this.divNgStyle = {
        color: 'blue',
        background: 'red'
      };
    }, ms: 3000);
  }
```

7. 双向绑定：

双向绑定是视图和模型保持同步，无论哪方改变，另一方都将同步，前面学的都是单向绑定，事件绑定是模板到控制器，DOM是控制器到模板。

在html中

<!--比较繁琐的的双向绑定-->

<input [value]="name" (input)="doOnInput($event)">

也就是说这边输入了，通过doOnInput($event)就去同步dom元素的属性，将其更改为输入的值。然后再通过[value]="name"去更改html的初始value属性，从而达到同步的效果，这是其原理。当然Angular提供了个简单的方式如下所示

<!--双向绑定,先是[]后是()可记作盒子里面装香蕉，ngModel为双向绑定的标志，用其声明双向绑定-->

<input [(ngModel)]="name">

而component.ts只需要定义一个name和一个doOnInput($event){…}就好函数名可以任意取

```ts
export class BindComponent implements OnInit {

  name: string;

  constructor() {

    setInterval( handler: () =>{
    this.name = "Tom"
    }, timeout: 3000)
  }

  ngOnInit() { }

  doOnInput(event: any) {
    // 这是dom属性，是对应输入的值，可以改变
    console.log(event.target.value);
    // 这是html属性，指定初始值，不可改变，
    // 即此例子指的是<input>空间中的value属性，固定就是定义的Tom
    console.log(event.target.getAttribute( qualifiedName: "value"));

    this.name = event.target.value()
  }
}
```

这样就能在控制台看到dom和html打印出来的数据是相同的，即完成双向绑定

8. 响应式编程：

响应式编程利用的是观察者模式的原理实现的

观察者模式：两个对象组成，可观察对象和观察者组成。一般在初始化可观察对象的时候，会向可观察对象注册一些观察者对象。然后注册完之后，可观察对象，发生一些变化的时候，调用一些观察者里相应的一些方法，来把自己的变化告诉观察者，观察者去做一些处理。

响应式编程就是异步数据流编程。单击事件就是一个流。能够创建很多有事件的流。任何事物都能当作一个流。变量，输入，事件，属性，缓存，数据结构。有很多方法和函数创建流，过滤元素，连接，转换元素。

下面看两个例子：

案例一：后台自定义一个数组，在控制台打印这个数组中的偶数的平方

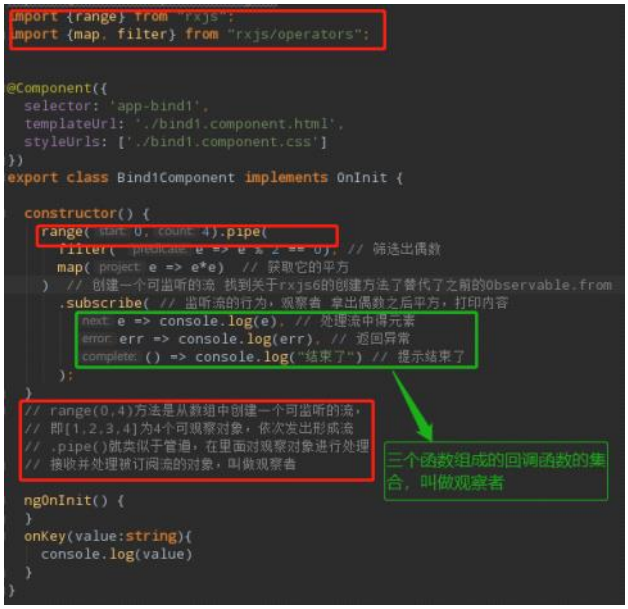
在html文件中：

<!--myField就是这个input的标签，后面的就是把value直接传到后台-->

<!--#myField成为模板本地变量，而#作为声明模板本地变量的标志-->

<input #myField (keyup)="onKey(myField.value)">

在component.ts中



onKey(value:string){…}中的value就是代表html的myField.value

案例2：假设，搜索输入框输入，搜索股票的信息，当输入ibm有的话返回信息
输入一个i，向服务器发送请求，最后调用了三次。
html中：
<!--formControl是ng指令-->
<!--当前input标签页面的formControl是后台searchInput的属性，通过这个属性，到input值改变时后面的searchInput会发射一个valuechanges事件，我们就需要在后台订阅这个事件-->
<input [formControl]="searchInput">

component.ts中

9. 管道：
Angular自带的管道：
html中：
<!--| 管道操作符，多个管道可以连起来作用-->
我的生日{{birthday | date:'yyyy-MM-dd HH:mm:ss' | uppercase}}
<!--number管道 格式：{最少整数位数}.{最少小数位数}-{最多小数位数}-->
圆周率{{pi | number: '1.1-4'}}

component.ts中只需要声明

```ts
export class PipingComponent implements OnInit {

  birthday: Date = new Date();
  pi: number = 3.1415926;

  size: number =4;
  constructor() { }

  ngOnInit() {
  }

}
```

提供几个网址作为Angular常用的自带管道：

Angular4 自带常用管道0： https://blog.csdn.net/HaiJing1995/article/details/71404350

Angular4 自带常用管道1： https://www.cnblogs.com/leiting/p/8880641.html

Angular4 自带常用管道2： https://blog.csdn.net/sky_sunshine_x/article/details/82840168

自定义管道：

生成管道服务 ng g s pipe/multiple 实现乘的操作

在multiple.pipe.ts中这样定义

```ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  // 管道名字，在html调用的时候也要写这个名字，名字可以自己定义
  name: 'multiple'
})
export class MultiplePipe implements PipeTransform {

  // value接受的一个输入值原始值， args：可选参数
  transform(value: any, args?: any): any {
    if(!args){
      args = 0;
    }
    return value * args;
  }

}
```

在html中：

```
<!--不传值则交给管道后台处理-->
<p>自定义管道{{size | multiple}}</p>
<!--传值为3，管道后台处理-->
<p>自定义管道{{size | multiple: 3}}</p>
```

在component.ts中只需要定义一个传入html的值就行

```
size: number =4;
```

自定义携带多个参数的例子：

pipe/xxx.ts中：

```ts
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({
  name: 'filter'   html直接用filter调用就行
})
export class FilterPipe implements PipeTransform {

  // 需要三个字段：商品列表,过滤的商品字段,输入的关键字
  transform(list: any[], filterField: string, keyword: string): any {
    // 判断传没传过滤的商品字段或者输入的关键字
    if(!filterField || !keyword) {
      return list;
    }
    return list.filter(
      item => {
        // 拿到商品的值fieldValue
        let fieldValue = item[filterField];
        // fieldValue.indexOf(keyword)中的indexOf是指fieldValue包含keyword这个字段的个数
        // 如果>0则返回true, 则页面就进行过滤,
        return fieldValue.indexOf(keyword) >= 0;
      }
    )
    return null;
  }

}
```

html中调用：

```
<!--products | filter: 'title': keyword 对products进行过滤-->
<div *ngFor="let product of products | filter: 'title': keyword"
```

多个参数时用冒号把各个参数隔开