用户输入触发 DOM 事件。我们通过事件绑定来监听它们，把更新过的数据导入回我们的组件和 model。

当用户点击链接、按下按钮或者输入文字时，这些用户动作都会产生 DOM 事件。 本章解释如何使用 Angular 事件绑定语法把这些事件绑定到事件处理器。

运行[在线例子](https://angular.cn/resources/live-examples/user-input/ts/plnkr.html)

**绑定到用户输入事件**

你可以使用 [Angular 事件绑定](https://angular.cn/docs/ts/latest/guide/template-syntax.html#event-binding)机制来响应任何 [DOM 事件](https://developer.mozilla.org/en-US/docs/Web/Events)。 许多 DOM 事件是由用户输入触发的。绑定这些事件可以获取用户输入。

要绑定 DOM 事件，只要把 DOM 事件的名字包裹在圆括号中，然后用放在引号中的[模板语句](https://angular.cn/docs/ts/latest/guide/template-syntax.html#template-statements)对它赋值就可以了。

下例展示了一个事件绑定，它实现了一个点击事件处理器：

<button (click)="onClickMe()">Click me!</button>

等号左边的(click)表示把按钮的点击事件作为**绑定目标**。 等号右边引号中的文本是**模板语句**，通过调用组件的onClickMe方法来响应这个点击事件。

写绑定时，需要知道模板语句的**执行上下文**。 出现在模板语句中的每个标识符都属于特定的上下文对象。 这个对象通常都是控制此模板的 Angular 组件。 上例中只显示了一行 HTML，那段 HTML 片段属于下面这个组件：

**app/click-me.component.ts**

@Component({

selector: 'click-me',

template: `

<button (click)="onClickMe()">Click me!</button>

{{clickMessage}}`

})

export class ClickMeComponent {

clickMessage = '';

onClickMe() {

this.clickMessage = 'You are my hero!';

}

}

当用户点击按钮时，Angular 调用ClickMeComponent的onClickMe方法。

**通过 $event 对象取得用户输入**

DOM 事件可以携带可能对组件有用的信息。 本节将展示如何绑定输入框的keyup事件，在每个敲击键盘时获取用户输入。

下面的代码监听keyup事件，并将整个事件载荷 ($event) 传递给组件的事件处理器。

**app/keyup.components.ts (template v.1)**

template: `

<input (keyup)="onKey($event)">

<p>{{values}}</p>

`

当用户按下并释放一个按键时，触发keyup事件，Angular 在$event变量提供一个相应的 DOM 事件对象，上面的代码将它作为参数传递给onKey()方法。

**app/keyup.components.ts (类 v.1)**

export class KeyUpComponent\_v1 {

values = '';

onKey(event:any) { // without type info

this.values += event.target.value + ' | ';

}

}

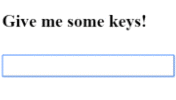
$event对象的属性取决于 DOM 事件的类型。例如，鼠标事件与输入框编辑事件包含了不同的信息。

所有[标准 DOM 事件对象](https://developer.mozilla.org/en-US/docs/Web/API/Event)都有一个target属性， 引用触发该事件的元素。 在本例中，target是[<input>元素](https://developer.mozilla.org/en-US/docs/Web/API/HTMLInputElement)， value.target.value返回该元素的当前内容。

在组件的onKey()方法中，把输入框的值和分隔符 (|) 追加组件的values属性。 使用[插值表达式](https://angular.cn/docs/ts/latest/guide/template-syntax.html#interpolation)来把存放累加结果的values属性回显到屏幕上。

假设用户输入字母"abc"，然后用退格键一个一个删除它们。 用户界面将显示：

a | ab | abc | ab | a | |



或者，你可以用event.key替代event.target.value，积累各个按键本身，这样同样的用户输入可以产生：

a | b | c | backspace | backspace | backspace |

***$event*的类型**

上例将$event转换为any类型。 这样简化了代码，但是有成本。 没有任何类型信息能够揭示事件对象的属性，防止简单的错误。

下面的例子，使用了带类型方法：

**app/keyup.components.ts (class v.1 - typed )**

export class KeyUpComponent\_v1 {

values = '';

onKey(event: KeyboardEvent) { // with type info

this.values += (<HTMLInputElement>event.target).value + ' | ';

}

}

$event的类型现在是KeyboardEvent。 不是所有的元素都有value属性，所以它将target转换为输入元素。 OnKey方法更加清晰的表达了它期望从模板得到什么，以及它是如何解析事件的。

**传入 *$event* 是靠不住的做法**

类型化事件对象揭露了重要的一点，即反对把整个 DOM 事件传到方法中，因为这样组件会知道太多模板的信息。 只有当它知道更多它本不应了解的 HTML 实现细节时，它才能提取信息。 这就违反了模板（*用户看到的*）和组件（*应用如何处理用户数据*）之间的分离关注原则。

下面将介绍如何用模板引用变量来解决这个问题。

**从一个模板引用变量中获得用户输入**

还有另一种获取用户数据的方式：使用 Angular 的[**模板引用变量**](https://angular.cn/docs/ts/latest/guide/template-syntax.html#ref-vars)。 这些变量提供了从模块中直接访问元素的能力。 在标识符前加上井号 (#) 就能声明一个模板引用变量。

下面的例子使用了局部模板变量，在一个超简单的模板中实现按键反馈功能。

**app/loop-back.component.ts**

@Component({

selector: 'loop-back',

template: `

<input #box (keyup)="0">

<p>{{box.value}}</p>

`

})

export class LoopbackComponent { }

这个模板引用变量名叫box，在<input>元素声明，它引用<input>元素本身。 代码使用box获得输入元素的value值，并通过插值表达式把它显示在<p>标签中。

这个模板完全是完全自包含的。它没有绑定到组件，组件也没做任何事情。

在输入框中输入，就会看到每次按键时，显示也随之更新了。



**除非你绑定一个事件，否则这将完全无法工作。**

只有在应用做了些异步事件（如击键），Angular 才更新绑定（并最终影响到屏幕）。

本例代码将keyup事件绑定到了数字0，这是可能是最短的模板语句。 虽然这个语句不做什么，但它满足 Angular 的要求，所以 Angular 将更新屏幕。

从模板变量获得输入框比通过$event对象更加简单。 下面的代码重写了之前keyup示例，它使用变量来获得用户输入。

**app/keyup.components.ts (v2)**

@Component({

selector: 'key-up2',

template: `

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

<p>{{values}}</p>

`

})

export class KeyUpComponent\_v2 {

values = '';

onKey(value: string) {

this.values += value + ' | ';

}

}

这个方法最漂亮的一点是：组件代码从视图中获得了干净的数据值。再也不用了解$event变量及其结构了。

**按键事件过滤（通过key.enter）**

(keyup)事件处理器监听*每一次按键*。 有时只在意*回车*键，因为它标志着用户结束输入。 解决这个问题的一种方法是检查每个$event.keyCode，只有键值是*回车*键时才采取行动。

更简单的方法是：绑定到 Angular 的keyup.enter 模拟事件。 然后，只有当用户敲*回车*键时，Angular 才会调用事件处理器。

**app/keyup.components.ts (v3)**

@Component({

selector: 'key-up3',

template: `

<input #box (keyup.enter)="onEnter(box.value)">

<p>{{value}}</p>

`

})

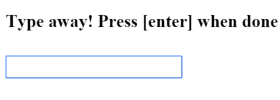
export class KeyUpComponent\_v3 {

value = '';

onEnter(value: string) { this.value = value; }

}

下面展示了它是如何工作的。



**失去焦点事件 (blur)**

前上例中，如果用户没有先按回车键，而是移开了鼠标，点击了页面中其它地方，输入框的当前值就会丢失。 只有当用户按下了回车键候，组件的values属性才能更新。

下面通过同时监听输入框的回车键和失去焦点事件来修正这个问题。

**app/keyup.components.ts (v4)**

@Component({

selector: 'key-up4',

template: `

<input #box

(keyup.enter)="update(box.value)"

(blur)="update(box.value)">

<p>{{value}}</p>

`

})

export class KeyUpComponent\_v4 {

value = '';

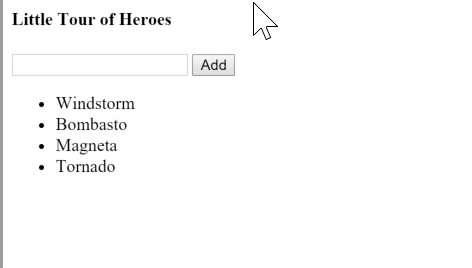
update(value: string) { this.value = value; }

}

**把它们放在一起**

上一章介绍了如何[显示数据](https://angular.cn/docs/ts/latest/guide/displaying-data.html)。 本章展示了事件绑定技术。

现在，在一个微型应用中一起使用它们，应用能显示一个英雄列表，并把新的英雄加到列表中。 用户可以通过输入英雄名和点击“添加”按钮来添加英雄。



下面就是“简版英雄指南”组件。

**app/little-tour.component.ts**

@Component({

selector: 'little-tour',

template: `

<input #newHero

(keyup.enter)="addHero(newHero.value)"

(blur)="addHero(newHero.value); newHero.value='' ">

<button (click)=addHero(newHero.value)>Add</button>

<ul><li \*ngFor="let hero of heroes">{{hero}}</li></ul>

`

})

export class LittleTourComponent {

heroes = ['Windstorm', 'Bombasto', 'Magneta', 'Tornado'];

addHero(newHero: string) {

if (newHero) {

this.heroes.push(newHero);

}

}

}

**小结**

* **使用模板变量来引用元素** — newHero模板变量引用了<input>元素。 你可以在<input>的任何兄弟或子级元素中引用newHero。
* **传递数值，而非元素** — 获取输入框的值并将*它*传递给组件的addHero，而不要传递newHero。
* **保持模板语句简单** — (blur)事件被绑定到两个 JavaScript 语句。 第一句调用addHero。第二句newHero.value=''在添加新英雄到列表中后清除输入框。

**源代码**

下面是本章讨论过的所有源码。

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'click-me',
4. template: `
5. <button (click)="onClickMe()">Click me!</button>
6. {{clickMessage}}`
7. })
8. export class ClickMeComponent {
9. clickMessage = '';
10. onClickMe() {
11. this.clickMessage = 'You are my hero!';
12. }
13. }

**小结**

你已经掌握了响应用户输入和操作的基础技术。

这些技术对小规模演示很实用，但是在处理大量用户输入时，很容易变得累赘和笨拙。 要在数据录入字段和模型属性之间传递数据，双向数据绑定是更加优雅和简洁的方式。 下一章表单解释了如何用NgModel来进行双向绑定。