表单创建一个有机、有效、引人注目的数据输入体验。Angular 表单协调一组数据绑定控件，跟踪变更，验证输入的有效性，并且显示错误信息。

表单是商业应用的支柱，我们用它来执行登录、求助、下单、预订机票、安排会议，以及不计其数的其它数据录入任务。

任何经验丰富的 Web 开发人员都能使用适当的标签拼凑出 HTML 表单。 但是，要想做出具有贴心的数据输入体验的表单， 引导用户明晰、高效地完成表单背后的工作流程，挑战就大多了。

坦白地讲，*这当中*所需要的设计技能超出了本章的范围。

*That* takes design skills that are, to be frank, well out of scope for this guide.

**双向数据绑定、变更跟踪、有效性验证和错误处理**等功能离不开框架的支持。 本章将介绍 Angular 表单相关的内容。

下面，从零开始，一步一步构建出一个简单的表单。在这个过程中，我们将学会如何：

* 使用组件和模板构建 Angular 表单
* [(ngModel)]语法实现双向数据绑定，用于读取和写入输入控件的值
* 结合表单来使用ngModel，可以跟踪表单控件的状态变化和有效性
* 使用特殊的 CSS 类来跟踪控件状态，并提供强烈的视觉反馈
* 向用户显示验证错误提示，以及启用/禁用表单控件
* 利用[模板引用变量](https://angular.cn/docs/ts/latest/guide/template-syntax.html#ref-vars)在 HTML 元素之间共享信息

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

Run the [live example](https://angular.cn/resources/live-examples/forms/ts/plnkr.html).

**模板驱动的表单**

通常，使用 Angular [模板语法](https://angular.cn/docs/ts/latest/guide/template-syntax.html)编写模板，结合本章所描述的表单专用指令和技术来构建表单。

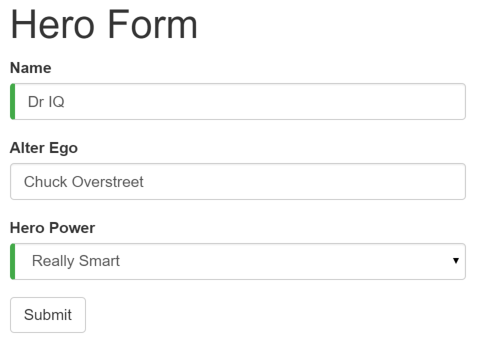
这不是创建表单的唯一方式，本章中只介绍模板驱动的表单。

利用 Angular 模板，可以构建几乎所有表单 — 登录表单、联系人表单…… 以及任何的商务表单。 可以创造性的摆放各种控件、把它们绑定到数据、指定校验规则、显示校验错误、有条件的禁用或 启用特定的控件、触发内置的视觉反馈等等，不胜枚举。

We can build almost any form we need with an Angular template — login forms, contact forms ... pretty much any business forms. We can lay out the controls creatively, bind them to data, specify validation rules and display validation errors, conditionally enable or disable specific controls, trigger built-in visual feedback, and much more.

它用起来很简单，这是因为 Angular 处理了大多数重复、单调的任务，这让我们可以不必亲自操刀、身陷其中。

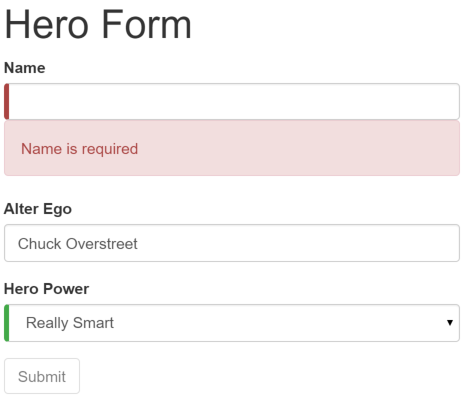
我们将讨论和学习构建如下的“模板驱动”表单：



这里是*英雄职业介绍所*，使用这个表单来维护候选英雄们的个人信息。每个英雄都需要一份工作。 公司的任务就是让适当的英雄去解决它/她所擅长应对的危机！

表单中的三个字段，其中两个是必填的。必填的字段在左侧有个绿色的竖条，方便用户分辨哪些是必填项。

如果删除了英雄的名字，表单就会用醒目的样式把验证错误显示出来。



注意，提交按钮被禁用了，而且输入控件左侧的“必填”条从绿色变为了红色。

稍后，会使用标准 CSS 来定制“必填”条的颜色和位置。

按下面的步骤，一点一点构建此表单：

1. 创建Hero模型类
2. 创建控制此表单的组件
3. 创建具有初始表单布局的模板
4. 使用ngModel双向数据绑定语法把数据属性绑定到每个表单输入控件
5. 往每个表单输入控件上添加name属性 (attribute)
6. 添加自定义 CSS 来提供视觉反馈
7. 显示和隐藏有效性验证的错误信息
8. 使用 **ngSubmit** 处理表单提交
9. 禁用此表单的提交按钮，直到表单变为有效

**搭建**

按照[搭建本地开发环境](https://angular.cn/docs/ts/latest/guide/setup.html)的说明，创建一个名为angular-forms的新项目。

**创建 Hero 模型类**

当用户输入表单数据时，需要捕获它们的变化，并更新到模型的实例中。 除非知道模型里有什么，否则无法设计表单的布局。

最简单的模型是个“属性包”，用来存放应用中一件事物的事实。 这里使用三个必备字段 (id、name、power)，和一个可选字段 (alterEgo，译注：中文含义是第二人格，例如 X 战警中的 Jean / 黑凤凰)。

在应用文件夹中创建hero.ts文件，并且写入下列类定义内容：

**app/hero.ts**

1. export class Hero {
2. constructor(
3. public id: number,
4. public name: string,
5. public power: string,
6. public alterEgo?: string
7. ) { }
8. }

这是一个少量需求和零行为的贫血模型。对演示来说很完美。

TypeScript 编译器为每个public构造函数参数生成一个公共字段，在创建新的英雄实例时，自动把参数值赋给这些公共字段。

alterEgo是可选的，调用构造函数时可省略，注意alterEgo?中的问号 (?)。

可以像这样创建新英雄：

let myHero = new Hero(42, 'SkyDog',

'Fetch any object at any distance',

'Leslie Rollover');

console.log('My hero is called ' + myHero.name); // "My hero is called SkyDog"

**创建表单组件**

Angular 表单分为两部分：基于 HTML 的*模板*和组件*类*，用来程序处理数据和用户交互。

先从组件类开始，是因为它可以简要说明英雄编辑器能做什么。

创建名叫hero-form.component.ts的文件，放进下列定义：

**app/hero-form.component.ts**

1. import { Component } from '@angular/core';
2. import { Hero } from './hero';
3. @Component({
4. moduleId: module.id,
5. selector: 'hero-form',
6. templateUrl: 'hero-form.component.html'
7. })
8. export class HeroFormComponent {
9. powers = ['Really Smart', 'Super Flexible',
10. 'Super Hot', 'Weather Changer'];
11. model = new Hero(18, 'Dr IQ', this.powers[0], 'Chuck Overstreet');
12. submitted = false;
13. onSubmit() { this.submitted = true; }
14. // TODO: Remove this when we're done
15. get diagnostic() { return JSON.stringify(this.model); }
16. }

这个组件没有什么特别的地方，没有表单相关的东西，与之前写过的组件没什么不同。

只需要前面章节中学过的概念，就可以完全理解这个组件：

1. 像往常一样，从 Angular 库中导入Component装饰器。
2. 导入刚刚创建的Hero模型。
3. @Component选择器"hero-form"表示可以用<hero-form>标签把这个表单放进父模板。
4. moduleId: module.id属性设置了基地址，用于从相对模块路径加载templateUrl。
5. templateUrl属性指向一个独立的 HTML 模板文件，名叫hero-form.component.html。
6. 为model和powers定义了供演示用的假数据。 将来，可以注入服务来获取和保存真实数据， 或者暴露这些属性为[输入与输出属性](https://angular.cn/docs/ts/latest/guide/template-syntax.html#inputs-outputs)，绑定到父组件上。 先不关心这些，因为这些未来的变化不会影响到表单。
7. 在最后增加diagnostic属性，它返回这个模型的 JSON 形式。 在开发过程中，它用于调试，最后清理时会丢弃它。

**为何分离模板文件？**

为什么不与我们在其他地方常常做的那样，以内联的方式把模板写在组件文件中呢？

没有什么答案在所有场合都总是“正确”的。当模板足够短的时候，内联形式更招人喜欢。 但大多数的表单模板都不短。通常，TypeScript 和 JavaScript 文件不是写（读）大型 HTML 的好地方， 而且没有几个编辑器能对混写的 HTML 和代码提供足够的帮助。 我们还是喜欢内容清晰、目标明确的短文件，像这个一样。

就算是在仅仅显示少数表单项目时，表单模板一般都比较庞大。所以通常最好的方式是将 HTML 模板放到单独的文件中。 一会儿将编写这个模板文件。在这之前，先退一步，再看看app.module.ts和app.component.ts，让它们使用新的HeroFormComponent。

**修改 *app.module.ts***

app.module.ts定义了应用的根模块。其中标识即将用到的外部模块，以及声明属于本模块中的组件，例如HeroFormComponent。

因为模板驱动的表单位于它们自己的模块，所以在使用表单之前，需要将FormsModule添加到应用模块的imports数组中。

把“快速起步”版的文件替换为如下内容：

**app/app.module.ts**

1. import { NgModule } from '@angular/core';
2. import { BrowserModule } from '@angular/platform-browser';
3. import { FormsModule } from '@angular/forms';
4. import { AppComponent } from './app.component';
5. import { HeroFormComponent } from './hero-form.component';
6. @NgModule({
7. imports: [
8. BrowserModule,
9. FormsModule
10. ],
11. declarations: [
12. AppComponent,
13. HeroFormComponent
14. ],
15. bootstrap: [ AppComponent ]
16. })
17. export class AppModule { }

有三处更改：

1. 导入FormsModule和新组件HeroFormComponent。
2. 把FormsModule添加到ngModule装饰器的imports列表中，这样应用就能访问模板驱动表单的所有特性，包括ngModel。
3. 把HeroFormComponent添加到ngModule装饰器的declarations列表中，使HeroFormComponent组件在整个模块中可见。

如果组件、指令或管道出现在模块的imports数组中，*不要*把它声明在declarations数组中。 如果它是你自己写的，并且属于当前模块，*就要*把它声明在declarations数组中。

**修改 *app.component.ts***

**Revise the *app.component.ts***

app.component.ts是应用的根组件，HeroFormComponent将被放在其中。

把"快速起步"的版本内容替换成下列代码：

**app/app.component.ts**

1. import { Component } from '@angular/core';
2. @Component({
3. selector: 'my-app',
4. template: '<hero-form></hero-form>'
5. })
6. export class AppComponent { }

仅有的一处修改。

1. template中只有新元素标签，即组件的selector属性。当应用组件被加载时，将显示这个英雄表单。

**创建初始 HTML 表单模板**

新建模板文件，命名为hero-form.component.html，并且填写如下内容：

Create a new template file called hero-form.component.html and give it the following definition:

**app/hero-form.component.html**

1. <div class="container">
2. <h1>Hero Form</h1>
3. <form>
4. <div class="form-group">
5. <label for="name">Name</label>
6. <input type="text" class="form-control" id="name" required>
7. </div>
8. <div class="form-group">
9. <label for="alterEgo">Alter Ego</label>
10. <input type="text" class="form-control" id="alterEgo">
11. </div>
12. <button type="submit" class="btn btn-default">Submit</button>
13. </form>
14. </div>

这只是一段普通的旧式 HTML 5 代码。这里有两个Hero字段，name和alterEgo，供用户输入。

*Name* <input>控件具有 HTML5 的required属性；但 *Alter Ego* <input>控件没有，因为alterEgo字段是可选的。

底部有个 *Submit* 按钮，具有一些 CSS 样式类。

**还没有真正用到Angular**。没有绑定，没有额外的指令，只有布局。

container、form-group、form-control和btn类来自 [Twitter Bootstrap](http://getbootstrap.com/css/)。纯粹是装饰。 我们使用 Bootstrap 来美化表单。嘿，一点样式都没有的表单算个啥！

The container, form-group, form-control, and btn classes come from [Twitter Bootstrap](http://getbootstrap.com/css/). Purely cosmetic. We're using Bootstrap to give the form a little style!

Angular 表单不需要任何样式库

Angular 不需要container、form-group、form-control和btn类， 或者外部库的任何样式。Angular 应用可以使用任何 CSS 库…… ，或者啥都不用。

添加样式表。

1. 在应用的根目录下打开终端窗口，输入如下命令：

 npm install bootstrap --save

 打开index.html文件并且把下列链接添加到<head>中。

1. <link rel="stylesheet"
2. href="https://unpkg.com/bootstrap@3.3.7/dist/css/bootstrap.min.css">

**用 *ngFor* 添加超能力**

英雄可以从认证过的固定列表中选择一项超能力。 这个列表位于HeroFormComponent中。

在表单中添加select，用ngFor把powers列表绑定到列表选项。 我们在之前的[显示数据](https://angular.cn/docs/ts/latest/guide/displaying-data.html)一章中见过ngFor。

在 *Alter Ego* 的紧下方添加如下 HTML：

Add the following HTML *immediately below* the *Alter Ego* group.

**app/hero-form.component.html (excerpt)**

<div class="form-group">

<label for="power">Hero Power</label>

<select class="form-control" id="power" required>

<option \*ngFor="let pow of powers" [value]="pow">{{pow}}</option>

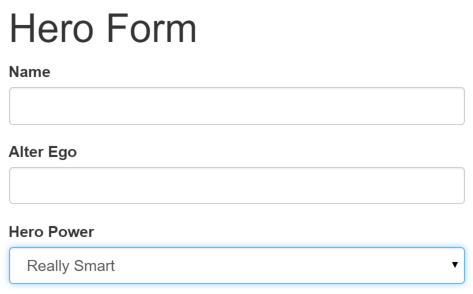
</select>

</div>

列表中的每一项超能力都会渲染成<option>标签。 模板输入变量p在每个迭代指向不同的超能力，使用双花括号插值表达式语法来显示它的名称。

**使用 *ngModel* 进行双向数据绑定**

如果立即运行此应用，你将会失望。



因为还没有绑定到某个英雄，所以看不到任何数据。 解决方案见前面的章节。 [显示数据](https://angular.cn/docs/ts/latest/guide/displaying-data.html)介绍了属性绑定。 [用户输入](https://angular.cn/docs/ts/latest/guide/user-input.html)介绍了如何通过事件绑定来监听 DOM 事件，以及如何用显示值更新组件的属性。

现在，需要同时进行显示、监听和提取。

虽然可以在表单中再次使用这些技术。 但是，这里将介绍个新东西，[(ngModel)]语法，使表单绑定到模型的工作变得超级简单。

找到 “Name” 对应的<input>标签，并且像这样修改它：

**app/hero-form.component.html (节选)**

<input type="text" class="form-control" id="name"

required

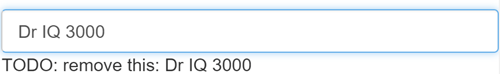
[(ngModel)]="model.name" name="name">

TODO: remove this: {{model.name}}

在 input 标签后添加用于诊断的插值表达式，以看清正在发生什么事。 给自己留个备注，提醒我们完成后移除它。

聚焦到绑定语法[(ngModel)]="..."上。

如果现在运行这个应用，开始在*姓名*输入框中键入，添加和删除字符，将看到它们从插值结果中显示和消失。 某一瞬间，它看起来可能是这样：



诊断信息可以证明，数据确实从输入框流动到模型，再反向流动回来。**这就是双向数据绑定！**

注意，<input>标签还添加了name属性 (attribute)，并设置为 "name"，表示英雄的名字。 使用任何唯一的值都可以，但使用具有描述性的名字会更有帮助。 当在表单中使用[(ngModel)]时，必须要定义name属性。

在内部，Angular 创建了一些FormControl，并把它们注册到NgForm指令，再将该指令附加到<form>标签。 注册每个FormControl时，使用name属性值作为键值。[本章后面](https://angular.cn/docs/ts/latest/guide/forms.html#ngForm)会讨论NgForm。

为*第二人格*和*超能力*属性添加类似的[(ngModel)]绑定和name属性。 抛弃输入框的绑定消息，在组件顶部添加到diagnostic属性的新绑定。 这样就能确认双向数据绑定*在整个 Hero 模型上*都能正常工作了。

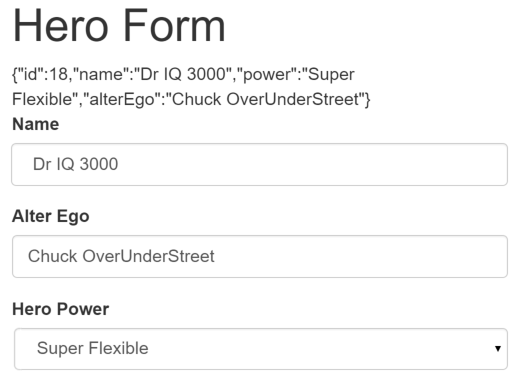
修改之后的表单，其核心是三个[(ngModel)]绑定，看起来像这样：

**app/hero-form.component.html (节选)**

1. {{diagnostic}}
2. <div class="form-group">
3. <label for="name">Name</label>
4. <input type="text" class="form-control" id="name"
5. required
6. [(ngModel)]="model.name" name="name">
7. </div>
8. <div class="form-group">
9. <label for="alterEgo">Alter Ego</label>
10. <input type="text" class="form-control" id="alterEgo"
11. [(ngModel)]="model.alterEgo" name="alterEgo">
12. </div>
13. <div class="form-group">
14. <label for="power">Hero Power</label>
15. <select class="form-control" id="power"
16. required
17. [(ngModel)]="model.power" name="power">
18. <option \*ngFor="let pow of powers" [value]="pow">{{pow}}</option>
19. </select>
20. </div>

* 每个 input 元素都有id属性，label元素的for属性用它来匹配到对应的输入控件。
* 每个 input 元素都有name属性，Angular 表单用它注册控件。

如果现在运行本应用，修改 Hero 模型的每个属性，表单看起来像这样：



表单顶部的诊断信息反映出所做的一切更改。

表单顶部的{{diagnostic}}绑定已经完成了它的使命，**删除**它。

**[(ngModel)]内幕**

*本节是对[(ngModel)]的深入剖析，它是可选的。不感兴趣？跳过它！*

绑定语法中的**[()]**是很好的线索。

在属性绑定中，值从模型中流动到屏幕上的目标属性 (property)。 通过把属性名括在方括号中来标记出目标属性，**[]**。 这是**从模型到视图**的单向数据绑定。

在事件绑定中，值从屏幕上的目标属性流动到模型。 通过把属性名括在圆括号中来标记出目标属性，**()**。 这是**从视图到模型**的反向单向数据绑定。

不出所料，Angular 选择了组合标点 **[()]** 来标记出双向数据绑定和**双向数据流**。

事实上，可以把NgModel绑定拆成两个独立的绑定，就像下面重写的 “Name” <input>绑定一样：

**app/hero-form.component.html (excerpt)**

<input type="text" class="form-control" id="name"

required

[ngModel]="model.name" name="name"

(ngModelChange)="model.name = $event" >

TODO: remove this: {{model.name}}

这个属性绑定看起来很眼熟，但事件绑定看起来有点怪。

ngModelChange并不是<input>元素的事件。 它实际上是来自NgModel指令的事件属性。 当 Angular 在表单中看到[(x)]的绑定目标时， 它会期待这个x指令有一个名为x的输入属性，和一个名为xChange的输出属性。

模板表达式中的另一个古怪之处是model.name = $event。 之前看到的$event对象来自 DOM 事件。 但ngModelChange属性不会生成 DOM 事件 —— 它是Angular EventEmitter类型的属性，当它触发时， 它返回的是输入框的值 —— 也正是希望赋给模型name属性的值。

很高兴知道这些，但是这样现实吗？实践上中，几乎总是优先使用[(ngModel)]形式的双向绑定。 只有当需要在事件处理函数中做一些特别的事情（例如合并或限制按键频率）时，才会拆分出独立的事件处理函数。

要学习ngModel和其它模板语法的更多知识，见[模板语法](https://angular.cn/docs/ts/latest/guide/template-syntax.html)。

**通过 ngModel 跟踪修改状态与有效性验证**

表单不仅是关于数据绑定的。我们还想知道表单中各个控件的状态。

在表单中使用ngModel可以获得比仅使用双向数据绑定更多的控制权。它还会告诉我们很多信息：用户碰过此控件吗？它的值变化了吗？数据变得无效了吗？

*NgModel* 指令不仅仅跟踪状态。它还使用特定的 Angular CSS 类来更新控件，以反映当前状态。 可以利用这些 CSS 类来修改控件的外观，显示或隐藏消息。

|  |  |  |
| --- | --- | --- |
| **状态**  **State** | **为真时的 CSS 类** | **为假时的 CSS 类** |
| 控件已经被访问过 | ng-touched  ng-touched | ng-untouched  ng-untouched |
| 控件值已经变化 | ng-dirty  ng-dirty | ng-pristine  ng-pristine |
| 控件值是有效的 | ng-valid  ng-valid | ng-invalid  ng-invalid |

往姓名<input>标签上添加名叫 **spy** 的临时[模板引用变量](https://angular.cn/docs/ts/latest/guide/template-syntax.html#local-vars)， 然后用这个 spy 来显示它上面的所有 CSS 类。

**app/hero-form.component.html (excerpt)**

<input type="text" class="form-control" id="name"

required

[(ngModel)]="model.name" name="name"

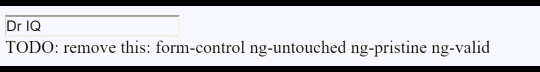
#spy >

<br>TODO: remove this: {{spy.className}}

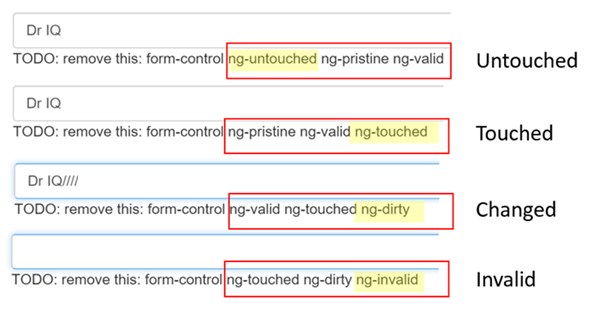
现在，运行本应用，并让*姓名*输入框获得焦点。 然后严格按照下面四个步骤来做：

1. 查看输入框，但别碰它
2. 点击输入框，然后点击输入框外面
3. 在名字的末尾添加些斜杠
4. 删除名字

动作和它对应的效果如下：



应该能看到下列四组类名以及它们的变迁：

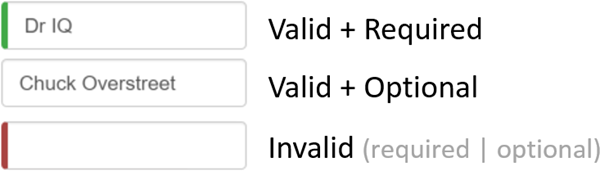


(ng-valid | ng-invalid)这一对是我们最感兴趣的。当数据变得无效时，我们希望发出强力的视觉信号， 还想要标记出必填字段。可以通过加入自定义 CSS 来提供视觉反馈。

**删除**模板引用变量#spy和TODO，因为它们已经完成了使命。

**添加用于视觉反馈的自定义 CSS**

可以在输入框的左侧添加带颜色的竖条，用于标记必填字段和无效输入：



在新建的forms.css文件中，添加两个样式来实现这一效果。把这个文件添加到项目中，与index.html相邻。

**forms.css**

.ng-valid[required], .ng-valid.required {

border-left: 5px solid #42A948; /\* green \*/

}

.ng-invalid:not(form) {

border-left: 5px solid #a94442; /\* red \*/

}

这些样式的作用于两个 Angular 有效性类和 HTML 5 的 “required” 属性。

更新index.html中的<head>来包含这个样式表。

**index.html (节选)**

<link rel="stylesheet" href="styles.css">

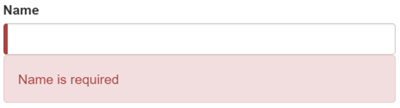
<link rel="stylesheet" href="forms.css">

**显示和隐藏验证错误信息**

我们能做的更好。

“Name” 输入框是必填的，清空它会让左侧的条变红。这表示*某些东西*是错的，但我们不知道错在哪里，或者如何纠正。 可以借助ng-invalid类来给出有用的提示。

当用户删除姓名时，看起来应该是这样的：



要达到这个效果，在<input>标签中添加：

1. [模板引用变量](https://angular.cn/docs/ts/latest/guide/template-syntax.html#ref-vars)
2. “is required”消息，放在邻近的<div>元素中，只有当控件无效时，才显示它。

对 *Name* 输入框做如下修改：

**app/hero-form.component.html (节选)**

<label for="name">Name</label>

<input type="text" class="form-control" id="name"

required

[(ngModel)]="model.name" name="name"

#name="ngModel" >

<div [hidden]="name.valid || name.pristine"

class="alert alert-danger">

Name is required

</div>

模板引用变量可以访问模板中输入框的 Angular 控件。 这里，创建了名叫name的变量，并且赋值为 "ngModel"。

为什么是 “ngModel”？ 指令的 [exportAs](https://angular.cn/docs/ts/latest/api/core/index/DirectiveMetadata-class.html#%21#exportAs) 属性告诉 Angular 如何链接模板引用变量到指令。 这里把name设置为ngModel是因为ngModel指令的exportAs属性设置成了 “ngModel”。

现在，把div元素的hidden属性绑定到name控件的属性，这样就可以控制“姓名”字段错误信息的可见性了。

**app/hero-form.component.html (节选)**

<div [hidden]="name.valid || name.pristine"

class="alert alert-danger">

上例中，当控件是有效的 (valid) 或全新的 (pristine) 时，隐藏消息。 “全新的”意味着从它被显示在表单中开始，用户还从未修改过它的值。

这种用户体验取决于开发人员的选择。有些人会希望任何时候都显示这条消息。 如果忽略了pristine状态，就会只在值有效时隐藏此消息。 如果往这个组件中传入全新（空）的英雄，或者无效的英雄，将立刻看到错误信息 —— 虽然我们还啥都没做。

有些人会为这种行为感到不安。它们希望只有在用户做出无效的更改时才显示这个消息。 如果当控件是“全新”状态时也隐藏消息，就能达到这个目的。 在往表单中[添加新英雄](https://angular.cn/docs/ts/latest/guide/forms.html#new-hero)时，将看到这种选择的重要性。

英雄的*第二人格*是可选项，所以不用改它。

英雄的*超能力*选项是必填的。 只要愿意，可以往<select>上添加相同的错误处理。 但没有必要，这个选择框已经限制了“超能力”只能选有效值。

**添加英雄及重置表单**

我们希望在这个表单中添加新的英雄。 在表单的底部放置“New Hero（新增英雄）”按钮，并把它的点击事件绑定到newHero组件。

**app/hero-form.component.html (新增英雄按钮)**

<button type="button" class="btn btn-default" (click)="newHero()">New Hero</button>

**app/hero-form.component.ts (新英雄方法)**

newHero() {

this.model = new Hero(42, '', '');

}

再次运行应用，点击 *New Hero* 按钮，表单被清空了。 输入框左侧的*必填项*竖条是红色的，表示name和power属性是无效的。 这可以理解，因为有一些必填字段。 错误信息是隐藏的，因为表单还是全新的，还没有修改任何东西。

输入名字，再次点击 *New Hero* 按钮。 这次，出现了错误信息！为什么？我们不希望显示新（空）的英雄时，出现错误信息。

使用浏览器工具审查这个元素就会发现，这个 *name* 输入框并不是全新的。 表单记得我们在点击 *New Hero* 前输入的名字。 更换了英雄*并不会重置控件的“全新”状态*。

我们必须清除所有标记，在调用newHero()方法后调用表单的reset()方法即可。

**app/hero-form.component.html (Reset the form)**

<button type="button" class="btn btn-default" (click)="newHero(); heroForm.reset()">New Hero</button>

现在点击“New Hero”重设表单和它的控制标记。

**使用 ngSubmit 提交表单**

在填表完成之后，用户还应该能提交这个表单。 “Submit（提交）”按钮位于表单的底部，它自己不做任何事，但因为有特殊的 type 值 (type="submit")，所以会触发表单提交。

仅仅触发“表单提交”在目前是没用的。 要让它有用，还要用另外的 Angular 指令更新<form>标签 —— NgSubmit， 并且通过事件绑定把它绑定到HeroFormComponent.submit()方法。

<form (ngSubmit)="onSubmit()" #heroForm="ngForm">

上面代码的最后出现一些额外的东西！定义了模板引用变量**#heroForm**，并初始化为 "ngForm"。

现在heroForm变量引用的是NgForm指令，它代表的是表单的整体。

**NgForm指令**

什么NgForm指令？之前没有添加过 [NgForm](https://angular.cn/docs/ts/latest/api/common/index/NgForm-directive.html) 指令啊！

是 Angular 干的。Angular 自动创建了NgForm指令，并把它附加到<form>标签。

NgForm指令为form元素扩充了额外的特性。 它持有通过ngModel指令和name属性为各个元素创建的那些控件，并且监视它们的属性变化，包括有效性。 它还有自己的valid属性，只有当*其中所有控件*都有效时，它才有效。

模板中稍后的部分，通过heroForm变量把按钮的disabled属性绑定到表单的整体有效性。这里是那点 HTML：

<button type="submit" class="btn btn-default" [disabled]="!heroForm.form.valid">Submit</button>

重新运行应用。表单打开时，状态是有效的，按钮是可用的。

现在，删除*姓名*。我们违反了“必填姓名”规则，它还是像以前那样显示出错误信息。同时，Submit 按钮也被禁用了。

没感动吗？再想一会儿。如果没有 Angular NgForm的帮助，又该怎么让按钮的禁用/启用状态和表单的有效性关联起来呢？

有了 Angular，它就是这么简单：

1. 定义模板引用变量，放在（强化过的）form 元素上
2. 从50行之外的按钮上引用这个变量。

**切换两个表单区域（额外的奖励）**

提交表单还是不够激动人心。

对演示来说，这个收场很平淡的。老实说，即使让它更出彩，也无法教给我们任何关于表单的新知识。 但这是练习新学到的绑定技能的好机会。 如果你不感兴趣，可以跳到本章的总结部分，不用担心错失任何东西。

来实现一些更炫的视觉效果吧。 隐藏掉数据输入框，显示一些其它东西。

先把表单包裹进<div>中，再把它的hidden属性绑定到HeroFormComponent.submitted属性。

**app/hero-form.component.html (节选)**

<div [hidden]="submitted">

<h1>Hero Form</h1>

<form (ngSubmit)="onSubmit()" #heroForm="ngForm">

<!-- ... all of the form ... -->

</form>

</div>

主表单从一开始就是可见的，因为submitted属性是 false，直到提交了这个表单。 来自HeroFormComponent的代码片段告诉了我们这一点：

submitted = false;

onSubmit() { this.submitted = true; }

当点击 Submit 按钮时，submitted标志会变成 true，并且表单像预想中一样消失了。

现在，当表单处于已提交状态时，需要显示一些别的东西。 在刚刚写的<div>包装下方，添加下列 HTML 块：

**app/hero-form.component.html (节选)**

1. <div [hidden]="!submitted">
2. <h2>You submitted the following:</h2>
3. <div class="row">
4. <div class="col-xs-3">Name</div>
5. <div class="col-xs-9 pull-left">{{ model.name }}</div>
6. </div>
7. <div class="row">
8. <div class="col-xs-3">Alter Ego</div>
9. <div class="col-xs-9 pull-left">{{ model.alterEgo }}</div>
10. </div>
11. <div class="row">
12. <div class="col-xs-3">Power</div>
13. <div class="col-xs-9 pull-left">{{ model.power }}</div>
14. </div>
15. <br>
16. <button class="btn btn-default" (click)="submitted=false">Edit</button>
17. </div>

英雄又出现了，它通过插值表达式绑定显示为只读内容。 这一小段 HTML 只在组件处于已提交状态时才会显示。

There's our hero again, displayed read-only with interpolation bindings. This slug of HTML only appears while the component is in the submitted state.

添加了 “Edit（编辑）”按钮，将 click 事件绑定到表达式，用于清除submitted标志。

当点它时，这个只读块消失了，可编辑的表单重新出现了。

够炫吗？好吧，我们已经尽力了！

**结论**

本章讨论的 Angular 表单技术利用了下列框架特性来支持数据修改、验证和更多操作：

* Angular HTML 表单模板。
* 带有Component装饰器的表单组件类。
* 用来处理表单提交的ngSubmit指令。
* 模板引用变量，例如#heroForm、#name和#power。
* 用于双向数据绑定、数据验证和变化追踪的[(ngModel)]语法和name属性。
* 指向 input 控件的引用变量上的valid属性，可用于检查控件是否有效、是否显示/隐藏错误信息。
* 通过绑定到NgForm的有效性状态，控制提交按钮的禁用状态。
* 定制 CSS 类来给用户提供无效控件的视觉反馈。

最终的项目目录结构看起来是这样：

angular-forms

app

app.component.ts

app.module.ts

hero.ts

hero-form.component.html

hero-form.component.ts

main.ts

node\_modules ...

index.html

package.json

tsconfig.json

这里是源码的最终版本：