angular-university 课程

[https://blog.](https://blog.angular-university.io/angular-2-smart-components-vs-presentation-components-whats-the-difference-when-to-use-each-and-why/)[angular-university.io/angular-2-smart-components-vs-presentation-components-whats-the-difference-when-to-use-each-and-why/](https://blog.angular-university.io/angular-2-smart-components-vs-presentation-components-whats-the-difference-when-to-use-each-and-why/)

源代码库

https://github.com/angular-university/reactive-patterns-course

智能组件和演示组件之间的典型交互

我们在这里看到的示例非常频繁，我们让智能组件通过将数据注入到演示组件@Input，并通过接收演示组件可能触发的任何操作@Output。

在这种情况下，我们使用自定义lesson事件来表明我们已经在列表中选择了给定的课程。

@Output通过清晰定义的界面，使用演示组件仍与智能组件保持隔离：

课程列表演示组件仅知道它发出了一个事件，但不知道该事件的接收者是什么，或者接收者将如何响应该事件

主屏幕智能组件订阅了lesson自定义事件并对事件做出反应，但是它不知道是什么触发了该事件。用户是双击课程列表还是单击视图按钮？这对于智能组件是透明的。

这样就很简单了，这里可能出什么问题了？

拆分智能组件与演示组件的清晰方法？

这样，我们可能会得出结论，构建我们的应用程序就像使所有顶级组件成为智能组件一样简单，并通过使用本地表示组件树来构建它们。

但事实是，有时并不是那么简单，因为自定义事件lesson不会冒出来。因此，例如，如果您有一棵深的组件树，并且希望多个级别的组件知道该事件，则该事件将不会冒泡。

什么问题导致自定义事件不会冒泡呢？

可以说，我们不但在课程列表和home组件之间仅嵌套了一层，还具有以下几层：课程列表位于选项卡面板中的可折叠面板内部。

课程列表仍希望通知home组件已通过lesson事件选择了课程。但是中间的两个组件TabPanel和CollapsiblePanel非特定于应用程序的表示组件。

想象它们是Angular Material库的组件！

这些仅用于Presentation的组件不知道lesson事件，因此它们无法冒泡。那么，我们该如何实现呢？为什么自定义事件不能简单地冒出来呢？

为什么自定义事件不冒出来，就像点击之类的DOM事件那样?

这不是偶然的，这是为了避免类似于服务总线的解决方案，比如AngularJs中的$scope.$emit()和$scope.$broadcast()意外创建的事件soup场景而设计的。

这些类型的机制往往会在应用程序的不同位置之间创建紧密的依赖关系，而这些位置不应该相互意识到，而且事件会被多次触发，或者以只查看一个文件时不明显的顺序触发。

因此，表示组件的自定义事件将只对其父组件可见，而不会在树的更上层可见。

如果由于某种原因我们确实需要冒泡行为，我们仍然可以使用element.dispatchEvent()。但是大多数时候这不是我们想要实现的。

那么，如何解决选项卡面板方案中可折叠面板中课程列表的情况？

我们仍然应该为课程列表创建一个表示组件。显示课程的功能是独立的，所以LessonsListComponent的版本仍然适用，它只是在应用程序中随处可见的东西。但是这个列表如何通知home组件呢?

对此有几个解决方案。我们应该研究的一个解决方案，特别是在构建一个大规模应用程序时，是研究像ngrx/store这样的解决方案。

但是，即使使用存储解决方案，我们也可能不希望将存储注入表示组件中。因为选择课程的结果可能并不总是将事件分派到存储。

为了保持示例简单，让我们从创建一个专门的类似于商店的服务开始，以仅解决这个课程选择问题:

@Injectable()

export class LessonSelectedService {

private \_selected: BehaviorSubject<Lesson> = new BehaviorSubject(null);

public selected$ = this.\_selected.asObservable().filter(lesson => !!lesson);

select(lesson:Lesson) {

this.\_selected.next(lesson);

}

}

正如我们所看到的，LessonSelectedService公开了一个可观察的selected$，它将在每次选择新课程时发出一个值。

请注意，我们在服务内部创建了一个主题(BehaviorSubject)，但没有将其公开给外部。这是因为主题(BehaviorSubject)本质上是事件总线，所以我们希望保持对谁可以在服务中发出事件的控制。

如果我们公开主题，应用程序的任何其他部分都可以代表服务发出事件，这是要避免的。

如何使用这个服务，因为我们不能将它注入LessonsListComponent，对吗?我们将讨论这一部分，现在让我们首先看看如何在Home组件中使用这个新服务。

在Home组件中使用新服务

Home组件将做的是，它将在构造函数中注入新组件:

@Component({...})

export class HomeComponent implements OnInit {

lessons: Lesson[];

constructor(

private lessonsService: LessonsService,

private lessonSelectedService: LessonSelectedService) {

}

ngOnInit() {

this.lessonSelectedService.selected$.subscribe(

lesson => this.selectLesson(lesson));

}

selectLesson(lesson) { }

}

如我们所见，我们已经订阅了selected$ Observable，它会发出新的lesson，我们触发组件的特定逻辑来处理选择。

但是请注意，Home组件并不知道课程列表，它只知道应用程序的其他部分触发了课程选择。应用程序的两个部分仍然是隔离的:

1 selected$的发射器不知道Home组件，

2 Home组件不知道课程，

3 两个参与者都只知道LessonSelectedService

所以我们已经解决了这个问题，对吧?还没有，因为我们仍然不想在LessonListComponent中注入新服务，这将使它成为一个智能组件，而我们希望它保持一个表示组件。那么如何解决这个问题呢?

如何保持LessonsListComponent是一个演示组件?

实际上，解决这个问题的一种方法是使它成为一个智能组件;-)我们可能会得出这样的结论:在应用程序中存在该表的任何地方，我们总是希望触发对LessonSelectedService的调用。

这将使课程列表组件成为特定于应用程序的组件，而它可能已经是这样了。例如，我们可能不会交付该组件并在多个应用程序中使用它。

因此，这将解决问题，这意味着像Home组件这样的顶级应用程序组件可能由一棵组件树组成，而不仅仅是表示组件。

智能组件不仅仅是顶级组件

一个智能组件不需要仅仅是一个顶级路由器组件。我们可以看到，树的下面可能还有其他组件，它们也被注入了像LessonSelectedService这样的服务，并不一定只从@Input()获得它们的数据。

另一种将LessonsListComponent保持为表示组件的解决方案

解决这个问题的另一种方法是保持课程列表组件不变，并在任何需要的地方使用它。但在这种情况下，我们可以将其封装在一个智能组件中，它被注入LessonSelectedService:

@Component({

selector: 'custom-lessons-list',

template: `

<lessons-list [lessons]="lessons" (lesson)="selectLesson($event)"></lessons-list> `

})

export class CustomLessonsListComponent {

constructor(private lessonSelectedService: LessonSelectedService) { }

selectLesson(lesson) { this.lessonSelectedService.select(lesson); }

}

在这里，我们创建了一个包装器智能组件，并将其称为CustomLessonsListComponent。在本例中，我们包装了自己的表示组件，但也可以包装来自第三方库的组件。

假设有一个MyCustomCountrySelectDropdown，它包装了一个通用的下拉列表，并将来自具体服务的数据注入其中。

如何决定构建什么组件?

在开始构建我们的应用程序时，并不总是很清楚什么是组件，以及什么是智能组件vs .表示组件。

那么，如何将应用程序分割成许多组件呢?一个页面的标题应该是一个组件吗，即使它只被使用了一次?

组织和可读性是创建组件的唯一原因，即使组件只在一个地方使用。在小文件中分离内容有助于保持代码基础的可维护性，并且使用Angular CLI，在创建新组件时没有任何开销:只需一个命令，我们就有了一个工作组件，可以在几秒钟内粘贴头部。

如何进行组件设计

解决这个问题的一种方法是避免从一开始就定义什么是组件以及组件的类型:我们可以从只使用纯HTML和第三方组件构建顶级组件开始。

只有当模板开始变大时，我们才开始把它分解成组件。如果在屏幕的几个部分中使用了某些东西，并且总是触发一个给定的动作，比如调用存储分派，那么我们可以考虑将其重构为一个更小的智能组件。

如果后来我们意识到需要像刚才创建的智能组件那样表示相同的数据，我们可以将其表示部分提取到表示组件中。

获得一组设计良好的组件的最佳方法是通过持续重构，使用Angular CLI可以轻松完成重构。

结论  
在构建应用程序时，我们可以寻找机会将纯粹的表示逻辑提取到表示组件中:这些组件只使用@Input和Output，在我们需要隔离表示逻辑并重用它时非常有用。

如果我们想让两个组件解耦并且不知道彼此，那么可以使用共享服务或存储来实现组件树中不同级别的智能组件或兄弟组件之间的通信。

但我们也可能希望将组件完全注入彼此并创建紧密耦合，有时这是最好的解决方案。在这种情况下，通过@ViewChild将组件相互注入可能是最好的方法。

智能组件与表示组件是一个有用的区别

一般来说，记住智能组件和表示组件之间的区别是非常有用的，但它可能并不适用于应用程序的所有组件。

我们可以使用一个小组件，它既能识别服务，又能在树的更深处显示一些数据，比如在课程选择时调用存储的课程列表。

将该组件进一步拆分为智能组件和表示组件可能并不总是必要的。

智能组件vs演示组件更多的是一种心态，我们会问自己:

1 这种表示逻辑在应用程序的其他地方有用吗?

2 把事情进一步分开会有用吗?

3 我们是否在应用程序中创建了意外的紧密耦合?

我们不需要将构建的每个组件的所有呈现逻辑提取到单独的表示组件中。它更多地是关于在任何给定时间构建对我们的应用程序最有意义的组件，并在CLI简化的连续迭代过程中进行必要的重构。

通用组件设计(以及它的潜在问题)

Angular应用开发中一个非常重要的方面就是应用组件设计:如何将不同类型的组件组合在一起，何时使用组件vs .使用指令，如何以及何时将组件中的功能提取到指令中，等等。

Angular组件提供了许多种方式使用和组合的功能。这允许我们根据情况采用各种各样的应用程序设计。

在这篇文章中，我们将讨论一种我们经常听说的特殊类型的设计。

容器组件vs表示组件

我们将要讨论的设计场景是容器组件与表示组件之间的组件分离。这是一个很流行的设计，现在在Angular生态系统中被越来越多的人使用，因为现在Angular支持组件模型了。丹·阿布拉莫夫(Dan Abramov @dan\_abramov)在博客中展示了该设计:

表示组件和容器组件 ：   
这篇文章是写给React的，但其理念同样适用于任何允许基于组件的设计模型的生态系统，比如Angular。

容器与展示设计的一个例子

让我们举一个设计的例子，请记住不同的术语被用来命名不同类型的组件。  
设计的核心思想是有不同类型的组件。使用同样的术语，在上面的博客文章中，我们有:

1 容器组件——这些组件知道如何从服务层检索数据。请注意，路由的顶级组件通常是一个容器组件，这就是为什么这种类型的组件最初如此命名的原因

2 表示组件——这些组件只是将数据作为输入，并知道如何在屏幕上显示。它们还可以发出自定义事件