<https://indepth.dev/posts/1117/the-shell-library-patterns-with-nx-and-monorepo-architectures>

源代码：

<https://github.com/NachoVazquez/nx-feature-shell-variation>

<https://github.com/NachoVazquez/composite-shell-variation>

**feature shell libraries** and **shell libraries 的区别**

*****feature-shell是一个特定应用程序的feature库*****

不是很清楚，对吧?那可能是因为我们漏掉了一些重要的部分。首先，让我们分而治之。

1 第一部分表明**feature shell**是特定于应用程序的。因此，不能在应用程序上下文之外使用它。

Nwrl书中使用的示例表示为其航空公司务开发不同应用程序。其中一个应用程序是booking应用程序。booking是两个实际应用程序(Booking Web Application and the Booking Mobile Application)在领域方面的封装。这样的封装是在两个应用程序共享相同功能和特性的前提下进行的。

在本例中，**feature**-shell就是booking-**feature**-shell。

现在我们知道**feature**-shell直接与应用程序的域级概念相连接

2 定义的第二部分，也是问题的最后一部分是，**feature**-shell是一个功能库(feature library)。

根据Nrwl，特性库包含一组文件，用于配置应用程序中的业务用例或页面。它包括容器组件、页面组件、路由组件和特定于用例的表示组件，通常包括与业务相关的每种类型的组件。

除了booking-feature-shell，预订应用程序示例中至少还有三个功能库:flight search(航班搜索)、passenger information(乘客信息)和seatmap(座位地图)。每一个都表示应用程序的不同页面。为什么不是不同的子域名呢?

但是，这两个概念到底有什么关系呢?Nx的书给了我们另一个线索。

libs\booking\feature-shell\src\lib\booking-feature-shell.module.ts

export const routes: Routes = [

{

path: '',

pathMatch: 'full',

component: FlightSearchComponent

},

{

path: 'passenger',

loadChildren: () =>

import('@nrwl-airlines/booking/feature-passenger-info').then(

m => m.BookingFeaturePassengerInfoModule

)

},

{

path: 'seatmap',

loadChildren: () =>

import('@nrwl-airlines/shared/seatmap/feature-seat-listing').then(

m => m.SharedSeatmapFeatureSeatListingModule

)

}

];

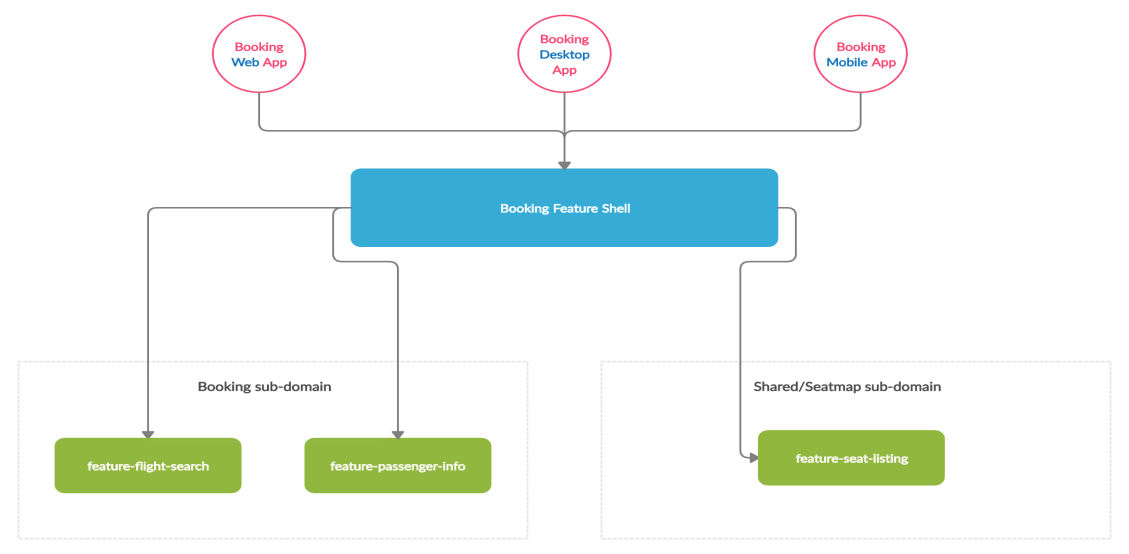
清单1向我们展示了如何在booking应用程序中配置导航。但是，这是一个具体的实现，我们将booking应用程序作为booking域的抽象来讨论。

这些路由属于两个应用程序中的哪一个?嗯,在这两个(Booking Web Application and the Booking Mobile Application)。

如果这些路由被放置在我们的一个booking应用程序中，我们将需要在另一个booking应用程序中复制它。这种复制有一个额外的缺点，因为我们需要维护两个或多个相同的实现并保持同步。换句话说，我们违反了DRY(不要重复自己)原则。

直观的解决方案是将这个“路由和初始化”逻辑提取到一个共享库中，由我们所有的booking应用程序导入。

请注意，只有在我们假设booking应用程序的行为方式和路由相同的情况下，上述结论才成立。



现在很明显，feature-shell的职责是充当应用程序(子域、页面)的顶级路由的协调器。对其本质的一种派生处理是，我们将为每个域方面的应用程序找到一个特性外壳库。

*每个面向领域的应用程序都有一个feature-shell库。*

面向领域的应用程序是由具有相同路由、行为和功能的应用程序组成的。在图1的示例中，Booking应用程序是Booking Web 应用程序、Booking Desktop应用程序和Booking Mobile 应用程序的联合。

在我看来，关于feature-shell库的一个最重要的混淆来源是，它们没有被当作一个单独的库，而是作为特性库的一个特殊情况，即使它们的职责不同。

Feature shell library use cases

现在我们已经更好地理解了什么是feature-shell以及它的职责是什么(至少根据Nrwl指南)，我们必须问自己:使用它们有真正的价值吗?它们适用于每个场景吗?

根据我的经验，这种feature-shell几乎没有适用的场景。让我们试着找出一个例子，看看它什么时候会有用。

使用Nrwl方法，我们在navigation/sub-domain/feature路由方面至少需要两个相同的应用程序。它们唯一的区别就是它们的平台，但请记住，我们正在创建基于javascript的库和应用程序。

因此，我们的应用程序必须在支持相同路由系统的JavaScript框架中实现。现在让我们关注Angular，但是下面的分析也适用于其他框架和类库，比如React。

如果我们想通过一个feature-shell来共享我们的实现，我们需要一个与Angular兼容的框架。我们有什么选择?

****Ionic****

我们可以将桌面应用程序构建为web应用程序，并将其部署在web上，同时为应用程序商店构建混合移动应用程序。

****NativeScript****

我们可以共享代码并使用NativeScript的构建过程来拥有桌面(web)和移动(本地)特定平台的模板。

****Electron****

我们可以在电子包装中有一个用于移动设备的常规web应用程序和一个混合桌面应用程序。

不过也有一些需要注意的地方。Nx不支持上述任何现成的技术。电子还没有被支持，但计划是加入它。

另外，还有一个开源扩展可以将Electron集成到Nx中。  
另外，电子、NativeScript和Ionic这三种语言都可以与Nx一起使用，这要归功于xplat。

正如我们所看到的，feature-shell并不是对每个场景都有意义的库模式。