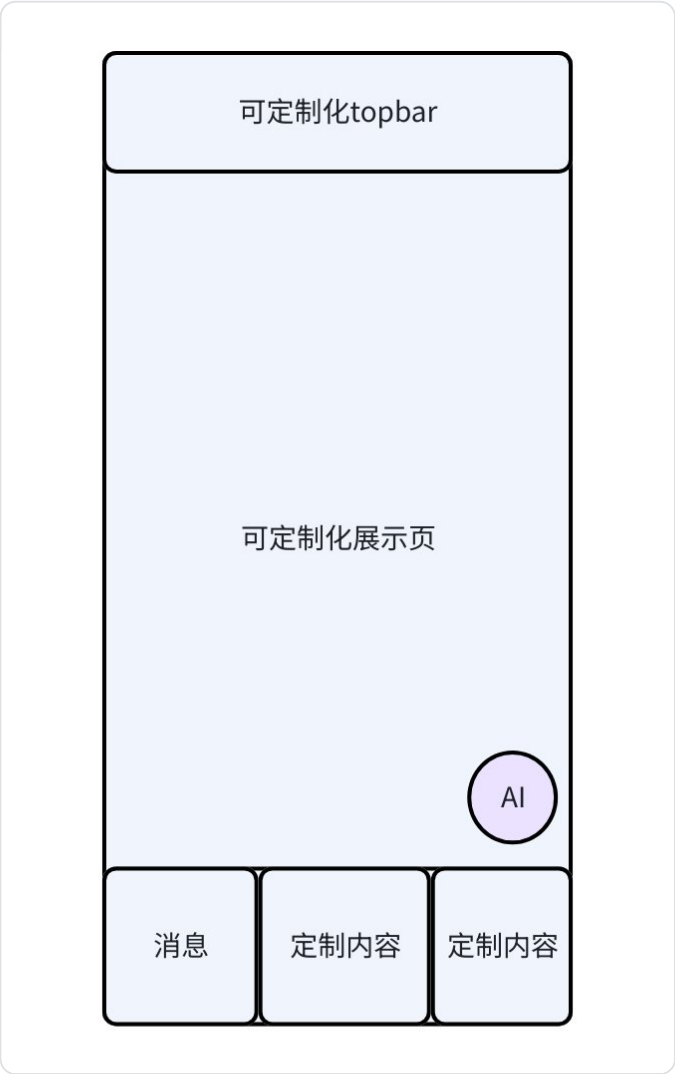


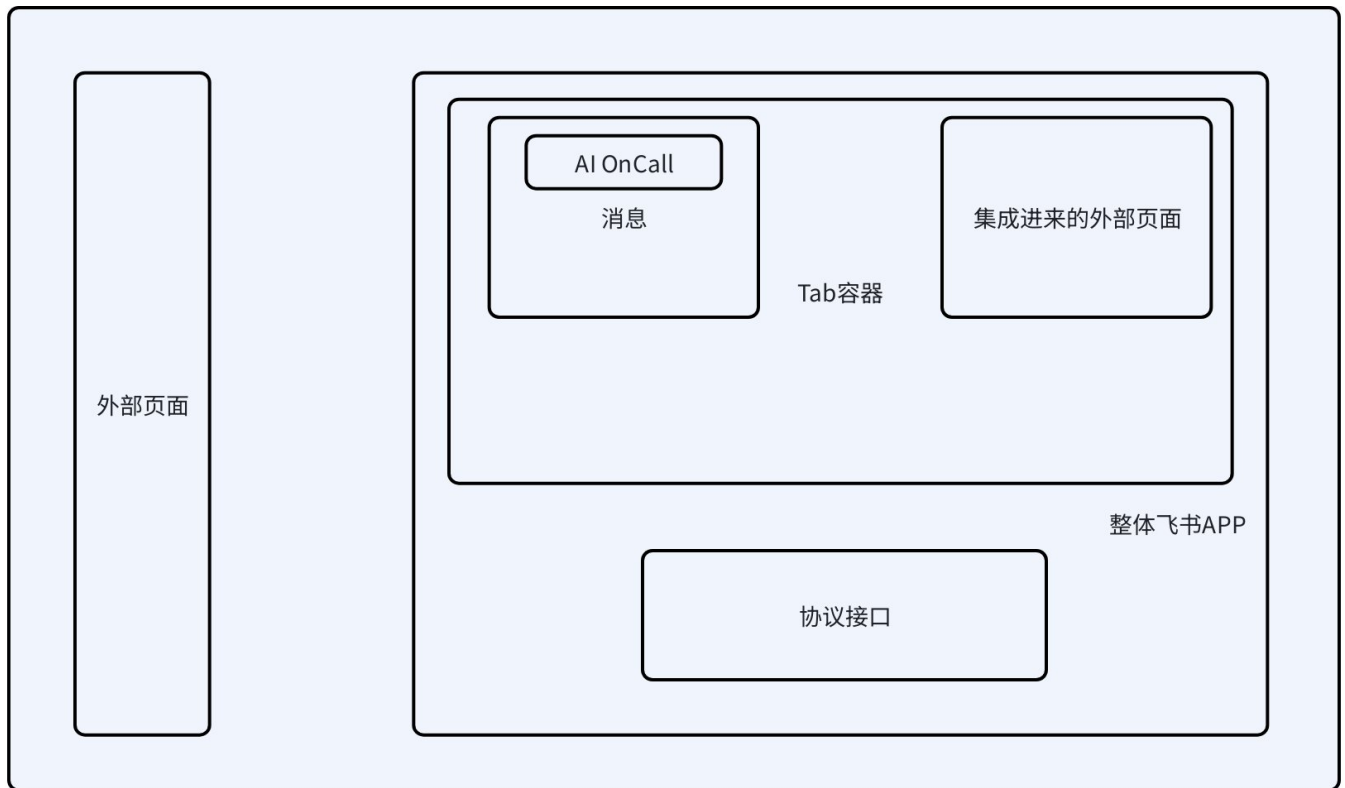
# 仿飞书Tab系统设计文档

## 1.需求分析

设计一个tab容器框架，用户可以通过容器对外提供的接口来将自己的页面注册到导航栏中，要实现高度定制化，用户可以自己去设计顶部的topbar，中间的展示页以及底部导航的图标和文字，我们默认提供一个消息的导航栏，在消息导航栏中添加一个按钮，跳转到AI对话页面，接入者可以通过与AI对话获取在Tab中集成自定义页面的相关信息，包括接口文档，参数说明等。



## 2.模块设计



设计四个模块

一个空项目（com.android.application）

只需要引入定制化之后的AAR

代码块

```
1 implementation(files("libs/toBTabAAR.aar"))
```

然后启动即可。

```
class MainActivity : ComponentActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        enableEdgeToEdge()
        setContent {
            FeiShuTabTheme {
                FeiShuTabApp(modifier = Modifier.fillMaxSize())
            }
        }
    }
}
```

一个Tab模块（com.android.library）

负责Tab功能的实现，在这个模块中实现Tab的数据存储层和页面展示层

在这个模块中引入AI对话模块，点击AI对话按钮时使用

代码块

```
1 implementation project(':lib-AIChat')
```

一个AI对话模块（com.android.library）

负责AI对话功能的实现

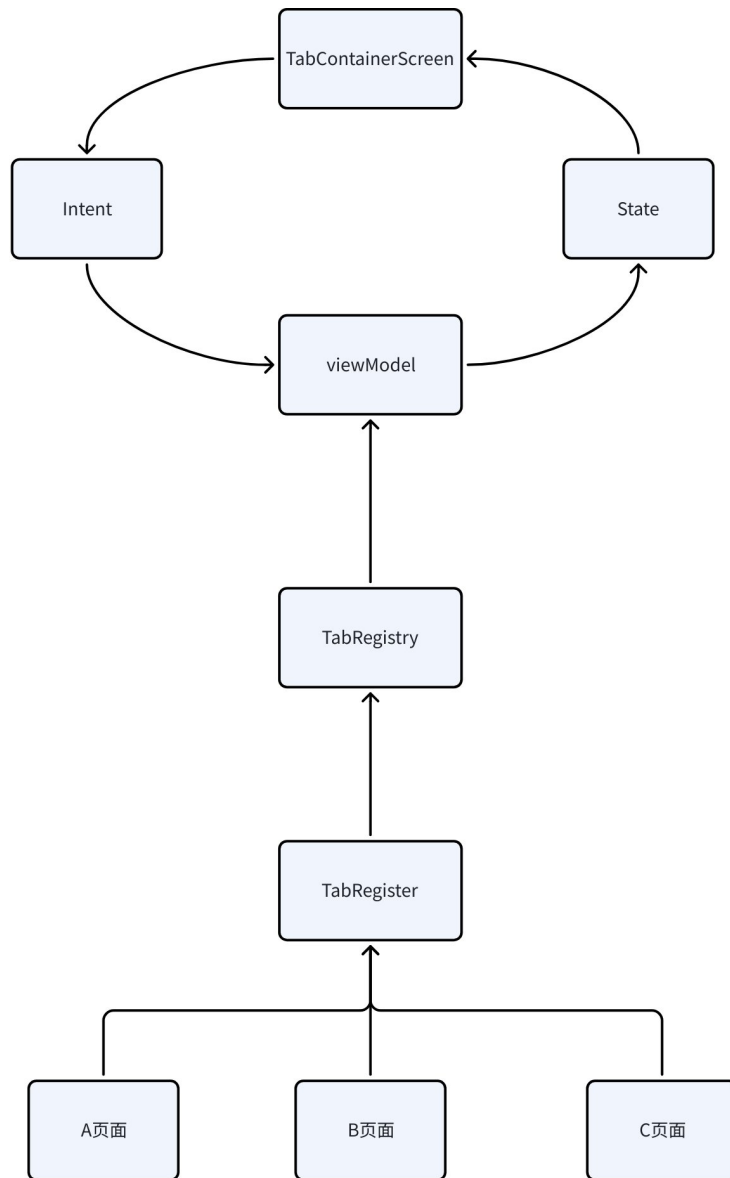
一个协议接口模块（com.android.library）

负责接口的定义和对外提供

## 3.架构设计与接口定义

### Tab架构和接口定义

对于Tab模块，主要为三个层级，协议接口层，数据存储层，页面展示层



对于协议接口层，定义TabRegister抽象接口，所有要定制的页面要实现抽象接口  
抽象接口定义

代码块

```
1  data class TabDescriptor(  
2      val id: String,  
3      val title: String,  
4      val icon: ImageVector,  
5      val route: String = "",  
6  )  
7  
8  interface TabRegister {  
9      val descriptor: TabDescriptor  
10     @Composable  
11     fun TopBar(navController: NavController)  
12     @Composable
```

```
13     fun Content(navController: NavController)
14 }
```

id：页面的顺序以及定位

title：底部导航栏要展示的文字内容

icon：底部导航中的图标

route：定制化页面的导航路由

TopBar：定制化TopBar内容

Content：定制化展示内容

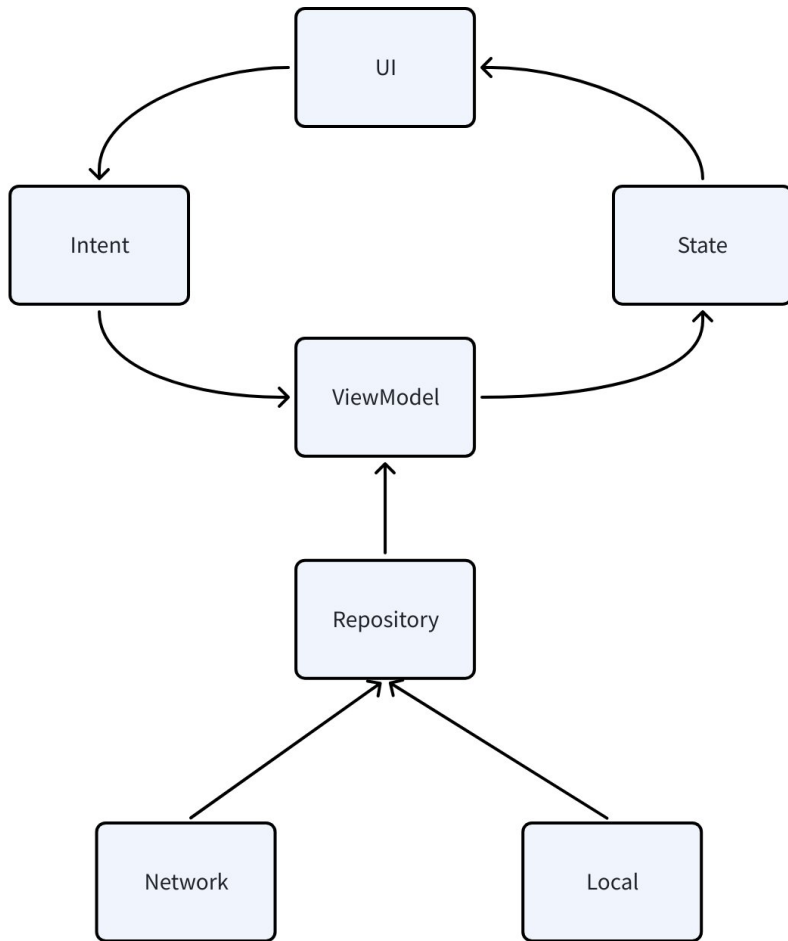
对于数据存储层，定义TabRegistry类存储注册的页面，负责管理所有注册的页面信息

内部用一个HashMap存储TabRegister，对外提供register，getById，navigateToTab，toList等接口

对于表示层，定义一个TabContainerScreen这个compose函数，从TabRegistry中获取Tab的名称，icon，Content等信息，进展示。

## AI onCall架构设计和接口定义

AI onCall计划使用MVI的架构模式，项目结构上大体上分为四层，UI层，ViewModel层，数据存储层和数据源层，因为使用MVI架构，要在UI层和ViewModel层中间添加State和Intent模块，分别用来表示数据状态和用户意图，UI通过Intent改变ViewModel，ViewModel通过State改变UI。在数据源层中，因为可能要保留对话，所以数据源层分成网络数据层和本地数据层。总体是这四个模块包，Util，Bean等跨层共享的数据方法等单独放在不同的包中。

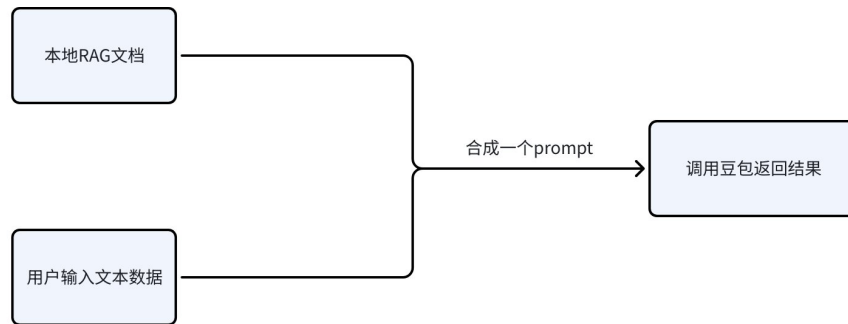


为了代码的健壮性，如果时间允许建议为发送对话请求等基础功能添加单测，通过Mock Intent和 State可以方便的去观察状态变化是否符合预期。

## 接口定义

AI onCall主要功能就是问答，询问框架相关信息AI给予回答，因此我理解只要实现一个POST请求接口就可以，无需鉴权，具体请求格式根据豆包API设置即可，对于响应体，因为要实现AI对话的流式数据交互，所以请求大模型的流式接口，根据官方文档定义相关响应体接收数据，在业务中进行处理实现流式交互效果

AI助手要针对我们的框架以及文档给出结论，因此要携带框架基本信息构建prompt



## 4.技术栈

Kotlin, Compose, Retrofit, OKHttp, Room