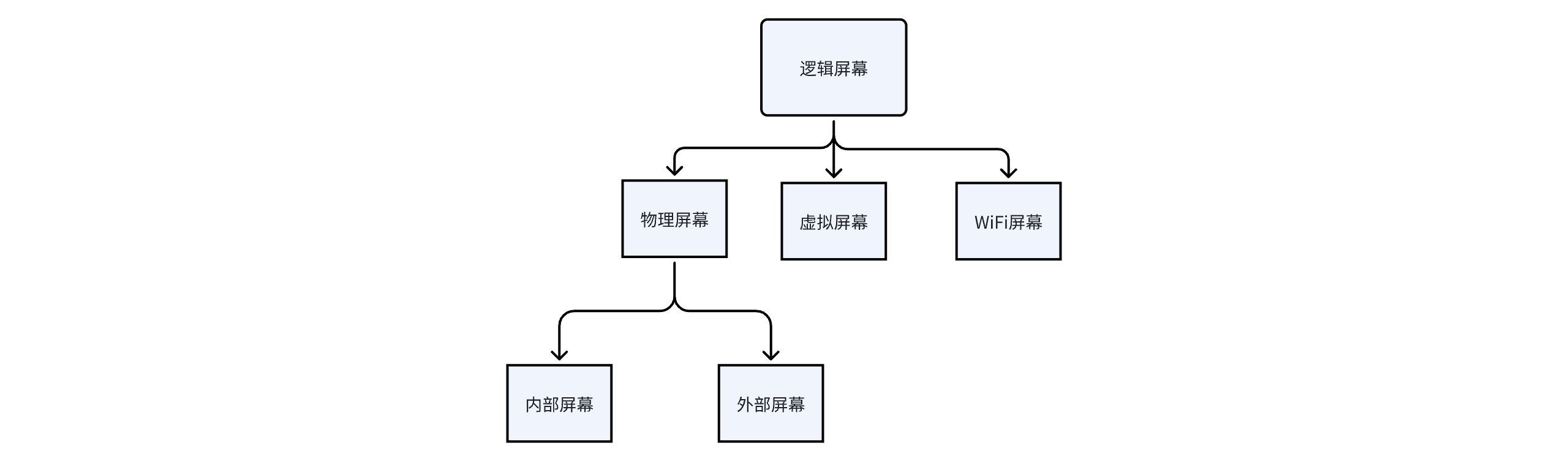
**地图投屏方案以及可能产生的问题**

以下文档是基于重汽潍柴项目和小康E3项目的实际投屏方案的总结和梳理

投屏方案

在进行投屏前需要了解一些概念：逻辑屏幕（Display），物理屏幕以及虚拟屏幕。逻辑屏幕是对物理屏幕的抽象，通过Display可以获取物理屏幕的一些属性信息。虚拟屏幕是对物理屏幕的模拟，但无法直接看到虚拟屏幕上的内容。



投屏，一般是将画面投到另一块屏幕上，也就是说有两块物理屏幕。

投屏画面来源：

静态画面：bitmap，比如投一张图片到屏幕上

可变画面：普通View视图，比如投一个布局到屏幕上。

如何获逻辑屏幕

不管是什么类型的屏幕，我们能拿到都是抽象的逻辑屏幕（Display）

|  |
| --- |
| Kotlin val displays: Array<Display> = context.getSystemService(DisplayManager::class.java).displays |

**如何向已有的逻辑屏幕上添加视图**

1. 将显示在主屏的视图显示到其它屏幕上

|  |
| --- |
| Shell adb shell am display move-stack id displayId |

这种方式是通过adb命令来切换，需手动在PC上输入，如果需要在Android代码中执行，可以使用以下方式切换：

|  |
| --- |
| Kotlin // id为当前activity所在的栈id，displayId为屏幕id // 由于是在Android应用中执行，不是在PC端执行，所以不需要以adb开头 Runtime.getRuntime().exec("am display move-stack id displayId") |

通过这种方式的切换会走activity的销毁重建流程，通过在AndroidManifest.xml中添加配置可避免

|  |
| --- |
| XML android:configChanges="uiMode|screenSize" |

**方案一**

通过使用Presentation来投屏

|  |
| --- |
| Kotlin val ctx: Context val display: Display val p = Presentation(ctx, display) p.setContentView(R.layout.main\_presentation\_dialog) p.show() |

presentation其实就是一个dialog，只是能够将这个dialog显示在对应的逻辑屏幕上，使用方式和普通dialog完全一致。

**方案二**

启动一个显示在其它屏幕上的activity

|  |
| --- |
| Kotlin val ctx: Context val display: Display val intent: Intent // 要跳转的activity val option = ActivityOptions.makeBasic() option.launchDisplayId = display.displayId ctx.startActivity(intent, option.toBundle()) |

该种方式和presentation的方式类似，但需要注意，启动后有两个activity同时处于resume状态。

**方案三**

通过windowManger直接添加View

|  |
| --- |
| Kotlin val ctx: Context val display: Display val view: View val wlp: WindowManager.LayoutParams // 创建对应屏幕的上下文对象 val displayContext = ctx.createDisplayContext(display) // 获取对应屏幕的windowManager val wm = displayContext.getSystemService(WindowManager::class.java) wm.addView(view, wlp) |

**避坑**

如果外部提供一个surface，地图的投屏需要显示在这个surface上该如何实现？

surface是用于交换图像数据的，用于连接数据生产者和消费者。

如何将View数据输出到surface？

将图像数据输出到surface，最熟悉的API是通过canvas来输出图形数据，但这种方式无法输出view显示的数据。

可以用Presentation显示View，但需要一个Display参数。

可以通过surface构造一个虚拟屏幕

|  |
| --- |
| Kotlin val dm = ctx.getSystemService(DisplayManager::class.java) // 构造对应大小的虚拟屏幕，需要和surface的大小相同 val virtualDisplay = dm.createVirtualDisplay("map-v-display", width, height, ctx.resources.displayMetrics.densityDpi, surface, DisplayManager.VIRTUAL\_DISPLAY\_FLAG\_PRESENTATION) // 构造Presentation val presentation = Presentation(ctx, virtualDisplay.display) presentation.setContentView(R.layout.layout\_map) // 输出图像数据到display（surface） presentation.show() |

**缺点**

该方式投屏无法触发点击事件，因为只提供了surface，而surface一般是由surfaceView等数据消费者提供。如果要响应点击事件，需要surface提供方对TouchEvent进行转发（转发给Presentation）。

潍柴早期投屏方案就是通过surface来输出地图视图，但在实际开发中，模拟器、手机均能正常投屏，但在车机上却无法投屏，系统会报一个关于surface错误。经过和外部沟通，确定无法通过surface的方式实现投屏，故潍柴项目采用了方案3进行投屏。

**副屏创建**

这里围绕单进程多地图的方式来实现：

创建副屏地图：

和创建主地图的方式一模一样

1. 创建MapDevice：MapService.createDevice(deviceId, attribute, observer);
2. 创建MapView：MapService.createMapView(mapViewParam，proxy， null， null，null）
3. 关联Surface

|  |
| --- |
| Kotlin MapSurfaceView mMapSurfaceView = new MapSurfaceView(group.getContext()); mMapSurfaceView.setMapService(mMapService); mMapSurfaceView.setDefaultDevice(mMapDevice); mMapSurfaceView.setDefaultMapView(mMapView); |

以上步骤为关键的几个步骤，具体步骤需参照主地图创建方式。

地图状态同步，投屏时，需要将主图的状态同步到副屏。

状态包括：

地图比例尺、车标、TMC、地图扎点、导航路线、地图视角等。

**实际存在的一些问题**

**1**

针对外部高频开关副屏的操作，导航需要考虑到多线程问题，即内部副屏还未创建完成，又接收到了关闭信号。

针对高频信号问题，推荐在完成关闭或者打开副屏地图后，再次对信号进行检测，防止当前表现和信号不一致的问题。

**2**

项目中，投屏是需要拉起导航到前台的。项目中是通过监听页面的onResume声明周期来实现，页面处于resume状态才进行投屏。

引申出的问题：当拉起导航时，如果用户此时主动将导航切到后台，那么存在不进入resume的情况，表现形式就是：导航收到了投屏信号，但是并未进行投屏。

**3**

项目中，投屏方案采用的方案三，通过windowManager向副屏添加地图视图，在此基础上，当收到取消投屏信号时，应当调用windowManager.removeView(view)来移除导航视图。但是当连续收到取消信号时，会重复执行windowManager.removeView(view)方法，导致崩溃。为解决这个问题，可能采用view.isAttachedToWindow方法来判断是否应该removeView，如下

|  |
| --- |
| Kotlin fun close() {  if (rootView.isAttachedToWindow) {  displayWindowManager.removeView(rootView)  } } |

采用这种方式看起来没问题，但当接收到高频投屏信号时，会导致视图移除失败。失败的原因就是isAttachedToWindow不可信。

|  |
| --- |
| Kotlin displayWindowManager.addView(rootView, mLayoutParams) val isAttached = rootView.isAttachedToWindow // 这里值为false |

当addView之后，isAttachedToWindow 并不是立即返回true，如果在这段时间内又收到了关闭信号，就会导致调整不执行，表现就是地图视图仍然显示。

解决方式就是用一个变量来判断是否需要removeView

|  |
| --- |
| Kotlin private var onShow = false fun show() {  //...  onShow = true } fun close() {  if (onShow) {  onShow = false  displayWindowManager.removeView(rootView)  } } |