# 鸿蒙开源第三方组件

——B站开源弹幕库引擎的迁移

上期文章发布之后，收到了很多朋友的关注，大家认为弹幕库能在鸿蒙系统上跑起来非常的炫酷。但同时也有一些朋友问我，像这样的鸿蒙第三方组件是如何迁移过来的。本期，我们将以弹幕库为例，让大家了解开源三方件是如何从安卓迁移到鸿蒙的！看完本篇内容，你不仅能够明白“我们是如何迁移弹幕库”的，还可以自己尝试迁移感兴趣的安卓开源三方件！

本次的内容包含三个部分，主要为大家讲解：开源三方件迁移的本质是什么、如何完成开源三方件的迁移。

1. 本质

三方件是一个向下使用安卓/鸿蒙的SDK方法，完成特定功能，向上为用户提供统一的接口，供用户调用的中间件。

想将一个安卓的三方件迁移至鸿蒙，要做的是：向上的接口保持不变（功能输出和接口相同，方便用户调用），向下调用的方法由安卓转为鸿蒙，也就是从使用安卓的底层SDK提供的方法变为使用鸿蒙的底层SDK提供的方法。下图以弹幕库为例子解释了这个过程：

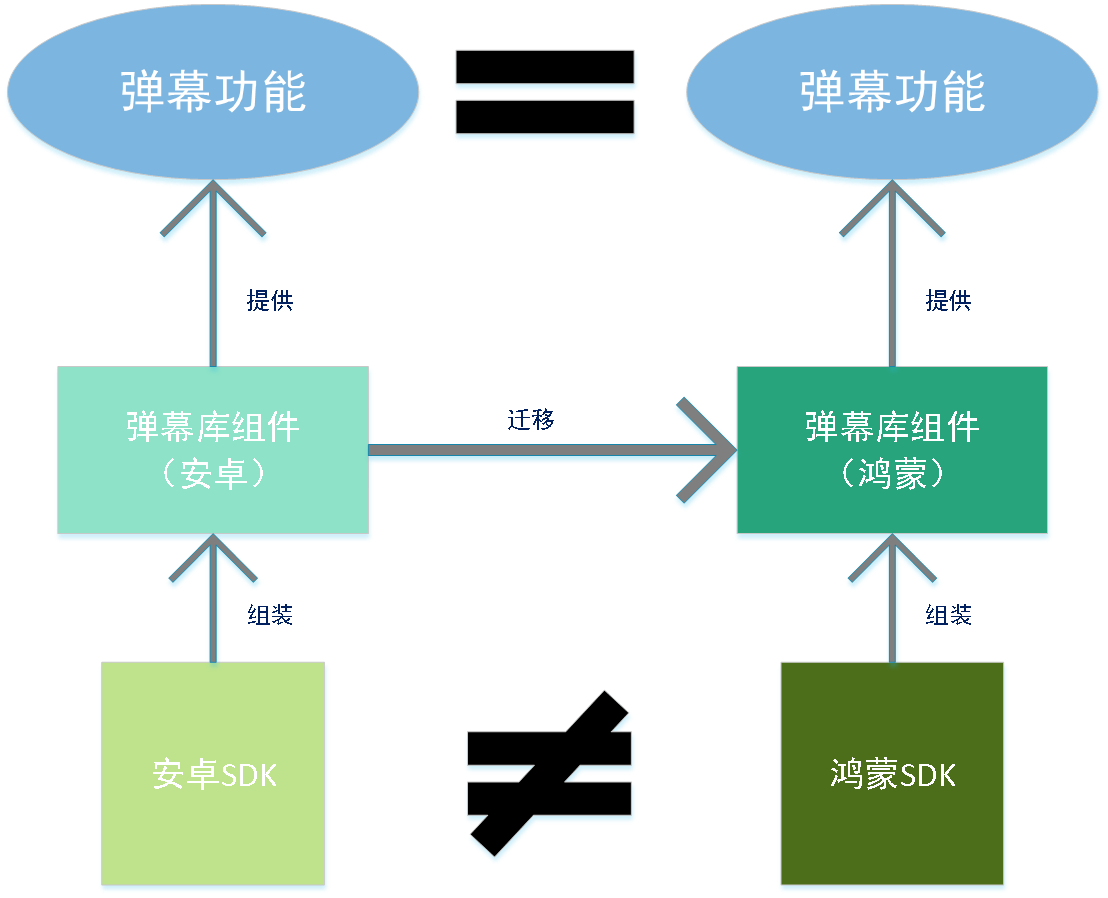


图1：弹幕库组件迁移示意图

原始弹幕库组件版本（<https://github.com/bilibili/DanmakuFlameMaster>）中，我们点开其中任意几个文件，查看其import信息：

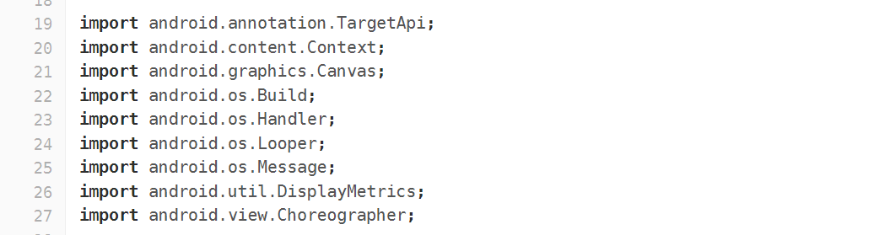


图2：DrawHandler.java

可以看到，文件中使用了大量的“import android.xxxx”的包名，表明组件包含了大量安卓自身的API调用，来实现特定功能，因此我们要做的是通过调用鸿蒙的API来实现相同的功能，由于安卓和鸿蒙底层架构及通信机制等各方面的不同，所以此处并不是简单地逐一替换关系。

1. 方法

在实际的迁移过程中，安卓和鸿蒙的包名、类名和方法名都可能会不同。根据迁移的难易程度，可以分为以下四种情况。

1. 包名不同，类名相同

有一些比较简单的类，它在安卓和鸿蒙中仅仅是包名发生了变化，类名、接口和方法没有变化，像这样的类，我们只要在文件中进行import后的包名替换即可，如下图所示。

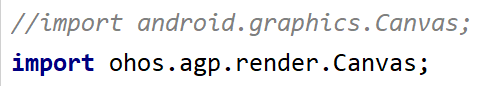


图3： 源码示例

1. 类名不同，方法名相同

有一些类，它在安卓和鸿蒙中包名和类名都发生了变化，接口没有变化，如下图所示。



图4：源码示例

此时需要翻阅安卓和鸿蒙的开发文档，以未变化的方法名为线索进行查找，对同功能的类进行匹配。

安卓：

<https://developer.android.google.cn/reference/packages>

鸿蒙：

<https://developer.harmonyos.com/cn/docs/documentation/doc-references/overview-0000001054518434>

像上面说的两种简单情况，在实际的迁移过程中几乎是不会出现的，在鸿蒙系统中也只有很少一部分基础库所提供的接口是与安卓相似的，其他部分的库与安卓相差甚远。

1. 包名、类名和方法完全不同

此时，对于安卓中调用的接口，无法直接在鸿蒙中找到与之对应的。需要我们具有一定的源码阅读能力，充分理解安卓接口的功能实现和模块划分（媒体、权限、视图...），再去鸿蒙相应的模块查找与之相似的使用方法。



图五：源码示例

这里就是一个典型的例子，注释方法为安卓方法，下面方法是鸿蒙方法。可以看到在RECT（Class：RectFloat）这个变量的初始化中就出现了不一样的使用方式。

1. 功能重写

前三种方法，接口在安卓和鸿蒙间存在对应关系，我们只需要理解、查找和匹配。在功能实现方式完全不同的情况下，需要我们具有一定的代码阅读能力和重写能力，在鸿蒙中对相应功能进行重写。以弹幕库为例：

---多线程通信：

安卓和鸿蒙的多线程通信机制类似，但是使用方法相差较大。

解决方法：功能重写。

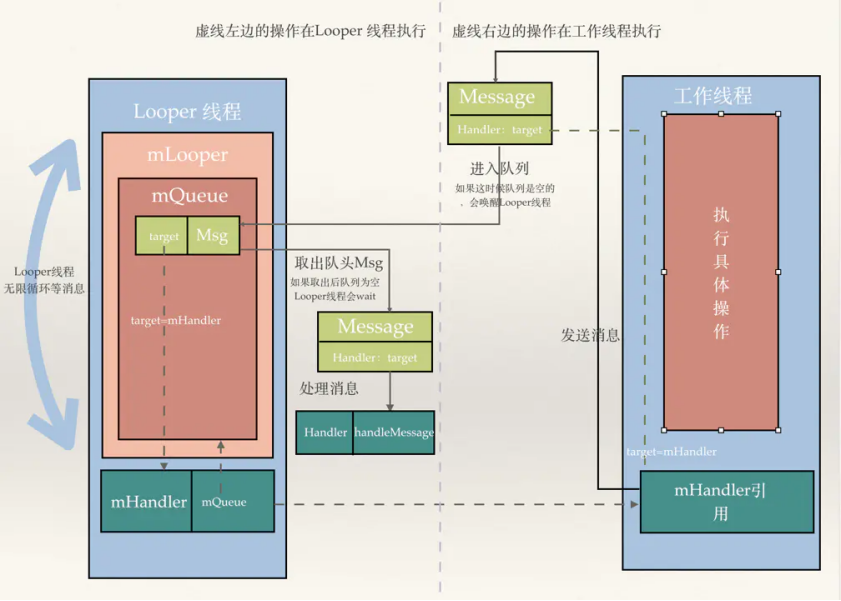


图6：安卓中的多线程通信逻辑

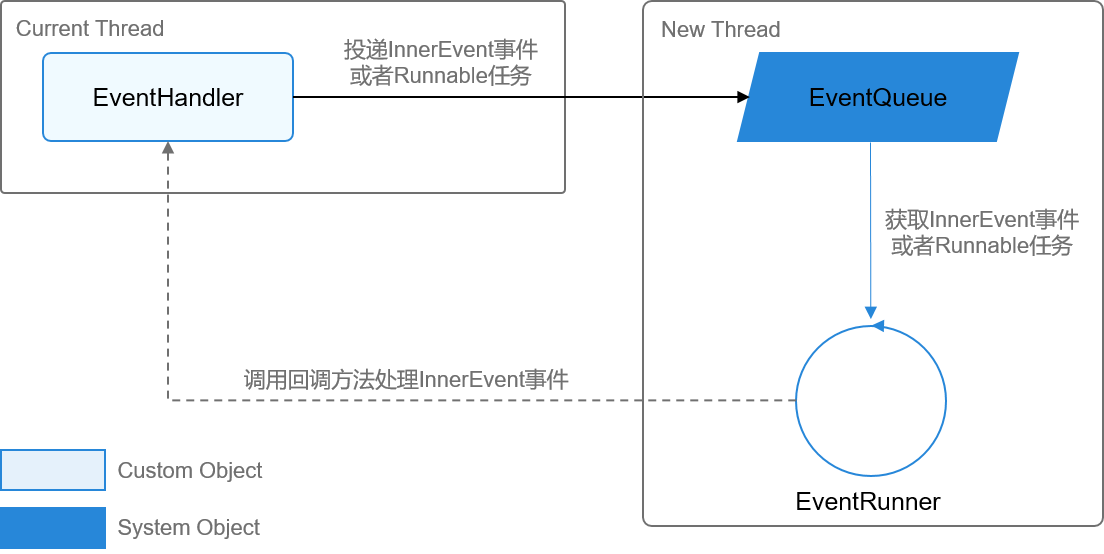


图7：鸿蒙中的多线程通信逻辑

可以看到两个版本的多线程通信的大体逻辑都是相同的，两个Thread想要进行通信，都需要四个步骤：

1.通信发起方需要构建（或实例化）一个结构体（或类）来进行事件的传递（并携带信息本身）。

2.这个携带着信息的结构化对象到了信息接收方后，会首先进入一个消息队列进行等待处理。

3.当结构化对象被从队列中取出后，会进入消息接收方的信息处理逻辑。

4.处理完成后消息接收方会生成一个反馈（回调），将处理结果反馈给消息发送方。

但是我们可以细心观察一下，安卓和鸿蒙的处理细节是有很大差异的。在安卓上使用这个机制的时候，需要先new一个HandlerThread，然后new一个looper，接着将他们绑定在一起，使用Thread.start才能让跑起来，实现消息队列发送及取出消息处理的功能。在鸿蒙上则不用如此麻烦，只需要new一个eventrunner就可以实现同样的功能，如下图所示。

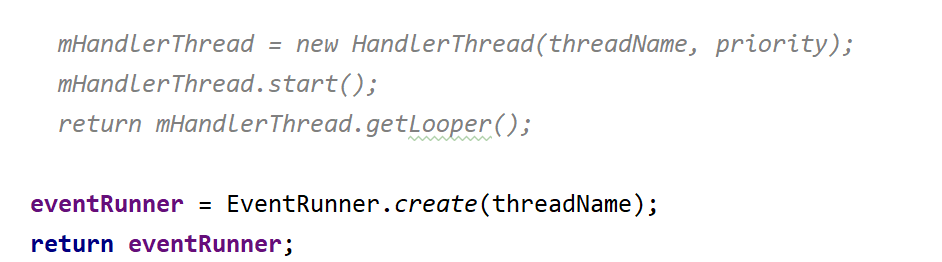


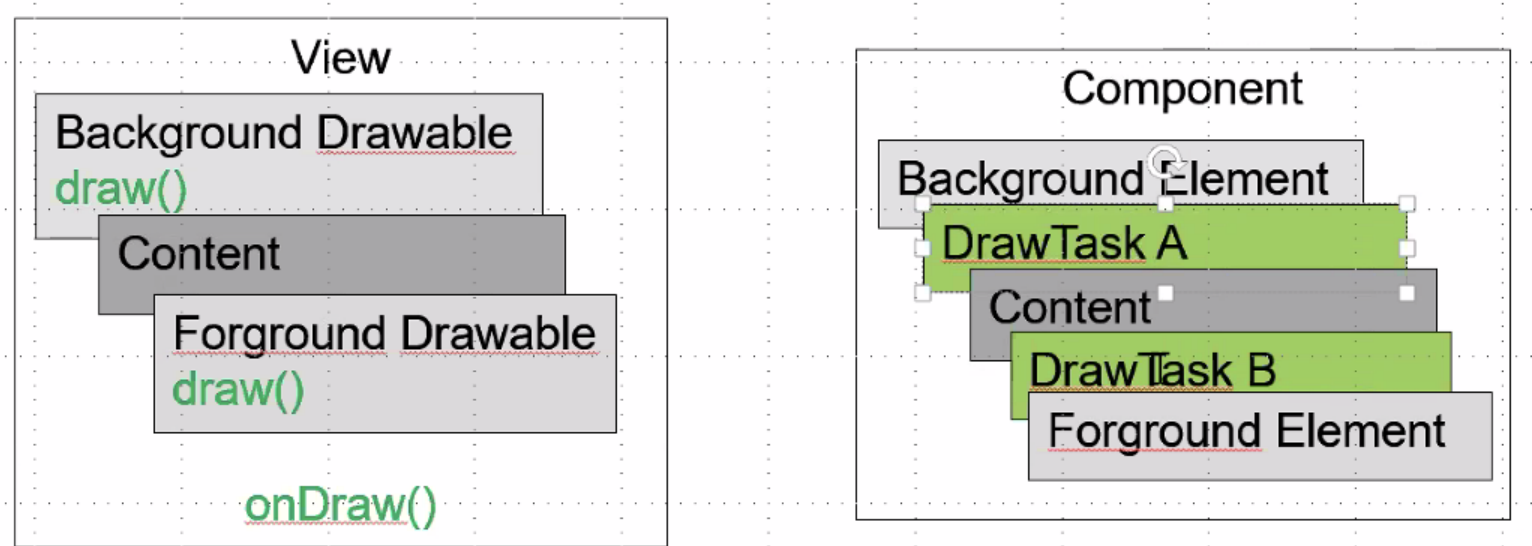
图8：源码示例

---UI部分

由于鸿蒙UI的构建方式和安卓存在很多差异，导致无法直接通过替换接口的方式来实现与安卓相对应的UI能力，这就需要进一步将对应的功能基于鸿蒙的构建方式进行重写。

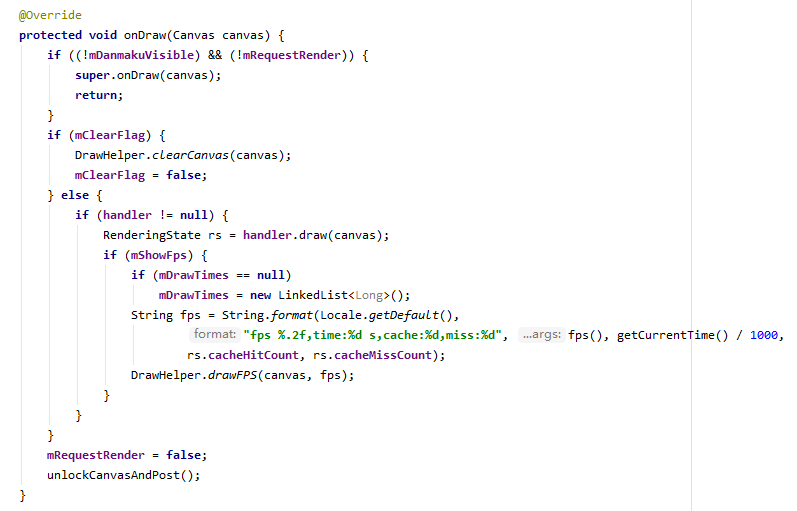
解决方法：功能重写

以OHOSP/AOSP自定义绘制能力对比为例：



安卓在绘制时，直接调用draw()方法，将canvas绘制到view上，鸿蒙则需对当前的component添加drawtask，重写drawtask里的ondraw方法，完成对canvas的绘制。

安卓实现如下：



鸿蒙：

