ExoPlayer播放器自定义总结

ExoPlayer播放器是google主推的一款开源播放器，github上有其源码。

ExoPlayer库的核心是ExoPlayer接口，ExoPlayer 拥有传统高级播 放器功能，比如支持缓冲，播放，暂停和快进。这样设计的目的是对媒体播放有更多的思考，比如怎么缓存，缓存在哪里，怎么渲染UI，而不是直接实现多媒体的加载和UI渲染，当一个播放器被创建或者当它准备重放的时候，ExoPlayer 来完成这些播放组件。ExoPlayer 实现了以下功能：

MediaSource 定义了加载、播放媒体，并且可以读取已经加载的媒体，使用ExoPlayer.prepare 可在播放开始传入MediaSource 。

Renderers 是一个渲染媒体的独立组件，当播放器被创建时，Renderers 会被注入。

TrackSelector 用来选择磁道，由MediaSource 所提供，并会被任意的可用的Renderers 所使用，当播放器被创建时，TrackSelector 会被注入。

LoadControl 用来控制缓存的，当媒体需要更多缓存的时候，LoadControl 来控制缓存的数量，当播放器被创建时，LoadControl 会被注入。

这个库提供了一些组件的默认实现的常见例子，下面我们详介绍下。

ExoPlayer 可以用在这些组件上，如果有特殊需求也可以自定义实现，例如自定义LoadControl 可以改变缓冲需求，或者自定义Renderer 可以改变Android原生不支持的视频编解码器。

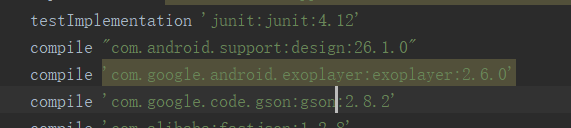
注入组件的概念在播放器库中多处使用，允许很多子组件去替换默认的实现，例如，需要一个默认的MediaSource实现，或者更多的DataSource 通过他们的构造方法去注入，通过提供一个自定义的factory 可以从非标准的数据源或网络加载数据。

Exoplayer播放器是比较复杂的，后续会慢慢分析，这一次先简单分析一下使用ExoPlayer来自定义一个需要的播放器。

定义一个播放器的流程如下：

1. 添加ExoPlayer库的依赖；
2. 创建一个SimpleExoPlayer 实例
3. 把player渲染在View上，为了展示播放器内容或者响应用户交互；
4. 使用MediaSource 准备播放；
5. 使用完成后，释放资源。

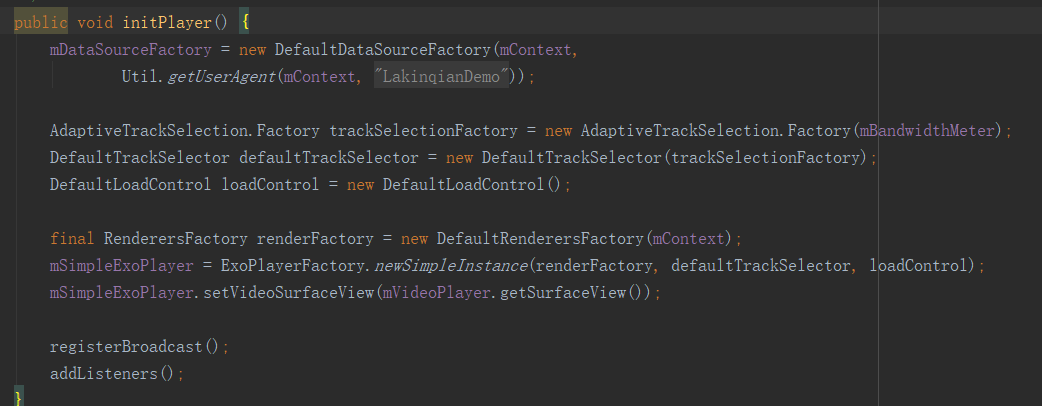
添加ExoPlayer依赖：



具体以来的版本可以看github上释放的最新版本。https://github.com/google/ExoPlayer

创建一个播放器：

使用ExoPlayerFactory创建一个ExoPlayer实例， 该Factory提供了一系列方法来创建各种定制ExoPlayer实例。 大部分情况下，使用ExoPlayerFactory.newSimpleInstance方法之一即可， 这些方法返回SimpleExoPlayer，它实现了ExoPlayer以添加额外的高级别播放器功能。



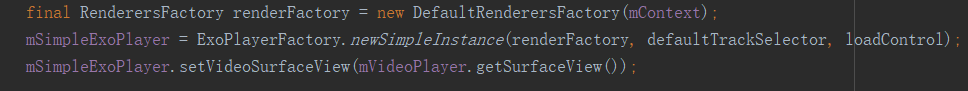
播放器准备好后，可以通过各种方法来控制播放。 例如，setPlayWhenReady启动和暂停播放，seekTo方法控制进度。

退出时通过ExoPlayer.release释放播放器，释放硬件资源。

把player渲染在View上

视频播放肯定是需要，所以，我们需要把player渲染到一个view上，ExoPlayer库提供了一个SimpleExoPlayerView 类，封装了PlaybackControlView 和Surface 用于展示视频，SimpleExoPlayerView 可以写在你的layout文件加下面的xml文件中，然后绑定player给这个view。

但是，我们也可以不用他提供的这个view类，使用AspectRatioFrameLayout + SurfaceView来实现更多的自定义内容。我现在用的就是后面这种方式。



使用MediaSource 准备播放

ExoPlayer库中每一帧都是通过MediaSource 来显示的，ExoPlayer 库提供的MediaSource 支持 DASH (DashMediaSource), SmoothStreaming (SsMediaSource), HLS (HlsMediaSource) 和 普通的媒体文件(ExtractorMediaSource)。在ExoPlayer demo中例举了4个PlayerActivity ，用户可自行查看。所以，我们播放视频的时候，都是通过传入MediaSource来进行视频源的播放的。

使用完成后，释放资源

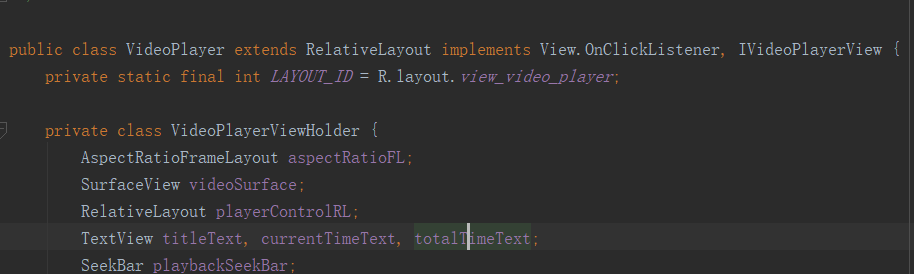
当player不在需要被使用时，一定要去释放player资源，比如视频解码器。这里可以调用ExoPlayer.release 完成操作。

这里，我主要记录的是player渲染到view这个过程，因为我们的页面会有很多操作，有很多手势，所以，我们需要去考虑播放页面的设计，之前的项目中，我也使用过exoplayer播放器，播放页面也会有比较多的操作，之前的做法是playerView没有去做定义，就是正常的AspectRatioFrameLayout + SurfaceView，然后对播放页面进行设计，添加很多的按钮和控件，这样的设计是没有什么问题的，只是会觉得这个压面很乱，很多暴露出来的各种view，尤其是当合作开发时，不负责这个页面的同事看到这个页面会觉得很杂乱。而且不同的地方复用起来极不方便。这一次就换了一种方法。我们自定义一个播放view，里面包含所有的点击以及手势操作，对外就暴露一个VideoPlayer这样一个用来播放的view，使用的时候及其方便，只需要在播放页面添加VideoPlayer这个view，然后把播放源传进去即可。这样就体现出来封装的优雅。

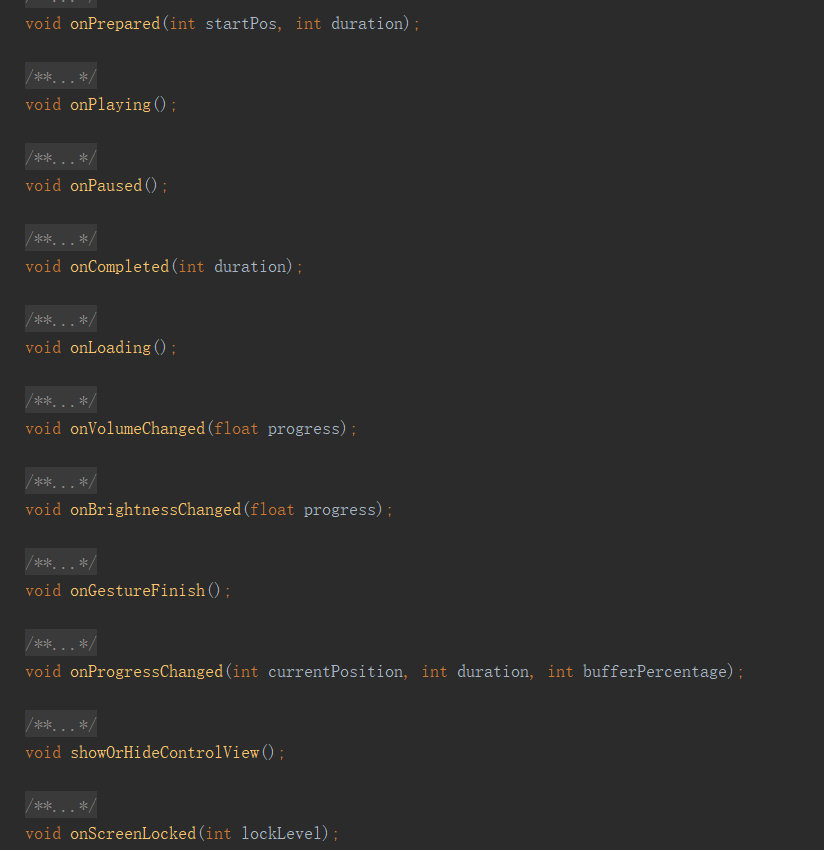
接下来我们分析一下实现过程吧。

由于最近接触MVP这个设计模式比较多，发现这个设计模式确实用起来比较方便，这个自定义播放view我也使用了这个模式，这个自定义的播放view里面包含了大量的逻辑和UI更新，所以我使用了V和P，将逻辑和页面更新分离开来。

VIEW层：VideoPlayer



VideoPlayer就是一个自定义view，继承RelativeLayout。就跟之前使用MVP中的Activity一样，他也会抽象出来一个View接口，也就是IVideoPlayerView接口，里面有很多待实现的方法，都是VideoPlayer中需要用到的，用来更新UI的一些方法，所有的view的更新都会在这里完成。其中包含了各种点击事件和手势操作。

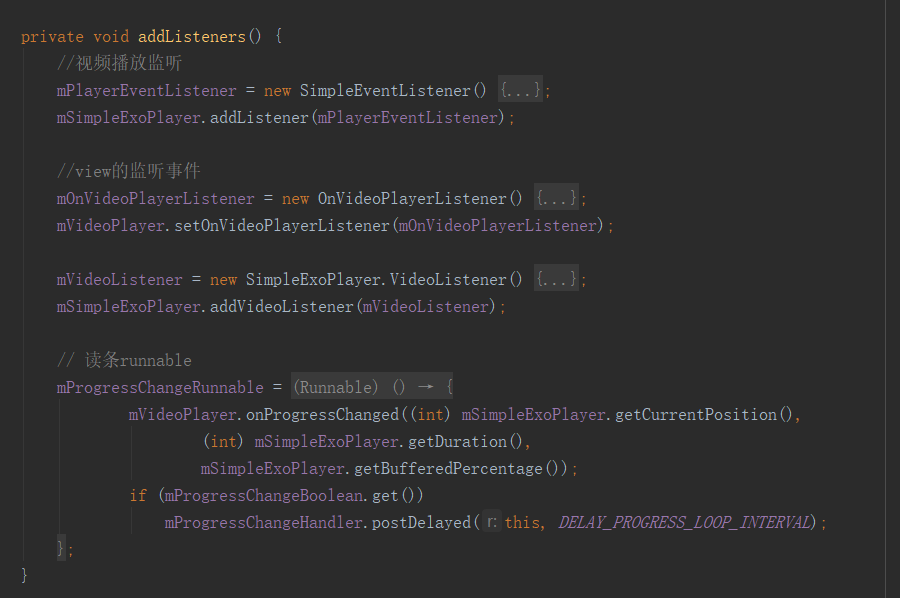


Presenter层：VideoPlayerHelper

这个VideoPlayerHelper就是Presenter层，里面包含了所有的逻辑处理，从创建播放器到添加视频源，到处理各种点击手势等的逻辑，都在这个Helper类里面完成。而外部使用的时候，就是通过这个Helper + VideoPlayer类实现播放的。

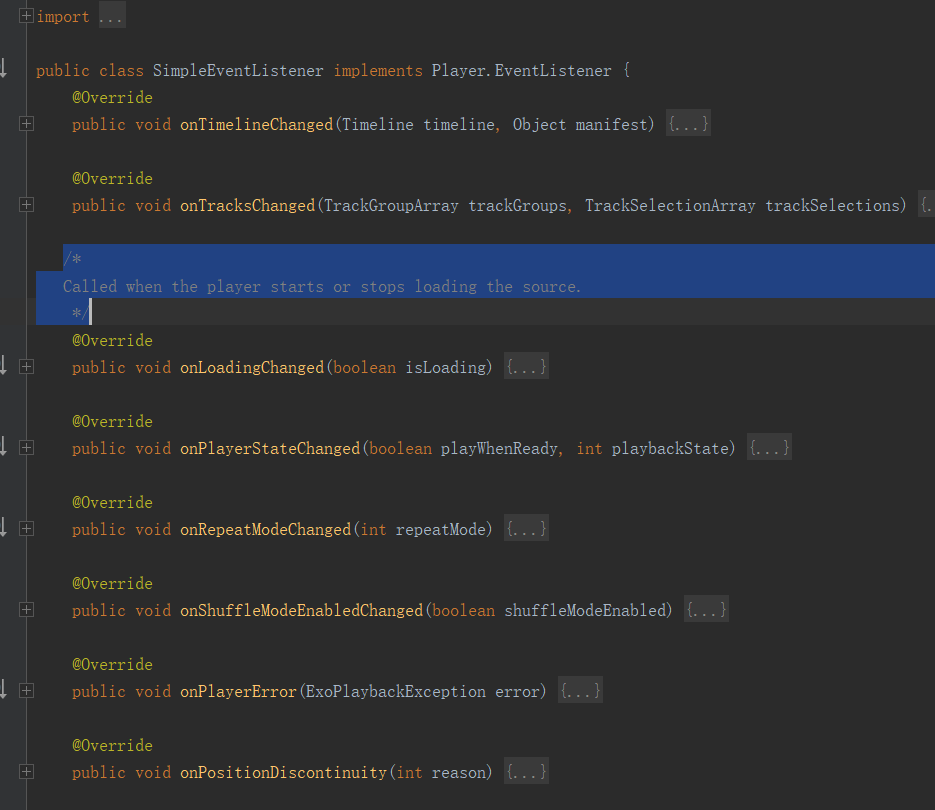


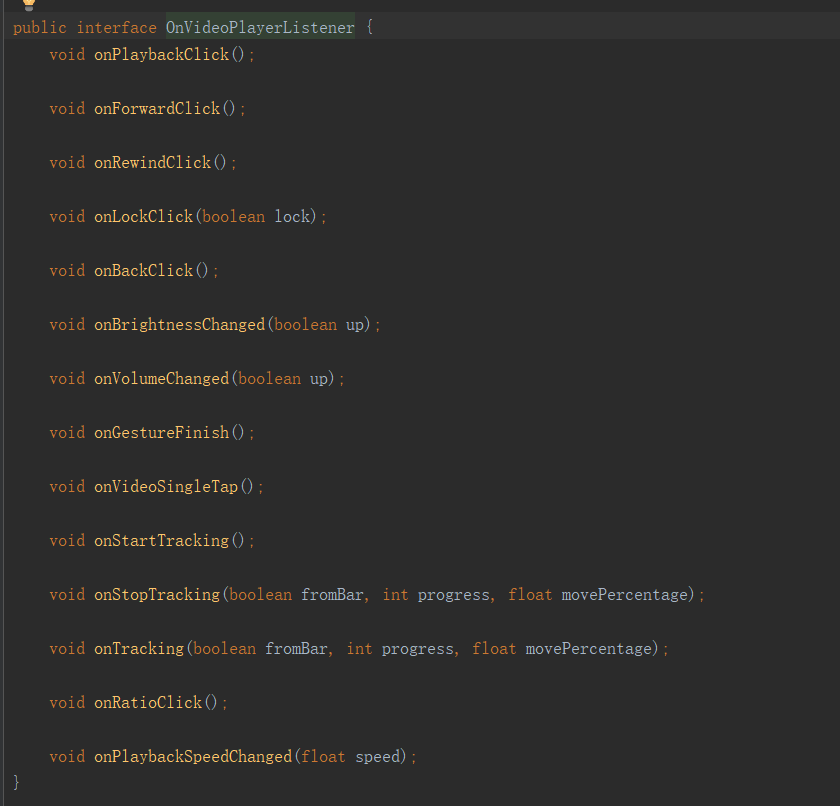
视频播放时，在Helper类中会添加很多监听，有播放器播放的过程的监听，也有一些用户行为的监听。



其中SimpleEventListener是继承自ExoPlayer.EventListener，是用来监听播放器播放状态的。

比如开始加载视频，开始播放，播放状态更改，播放出错等回调。



OnVideoPlayerListener是自定义的一个监听，用来监听我们自定义的操作。

包括暂停，下一个，上一个，快进，快退，亮度，音量手势等。这一部分是比较核心的，包含了大部分的自定义view和自定义的内容。

SimpleExoPlayer.VideoListener是监听真实播放页面的大小更改的。

这个Helper类会添加三个监听类到VideoPlayer类，分别是SimpleEventListener， OnVideoPlayerListener，SimpleExoPlayer.VideoListener。

当用户点击操作或者播放器播放状态改变时，根据点击的位置或者手势，首先响应的肯定是ViewPlayer类，因为VideoPlayer只处理view的更新，而且播放器是在Helper类中初始化的，所以这个时候需要通过这个监听来回调到Helper处理逻辑。

Helper类里面会持有一个VideoPlayer对象，当Helper类处理完逻辑后，往往需要更新UI，这个时候，往往需要回调到VideoPlayer类中进行UI更新。

通过这种MVP的方式来自定义一个播放VIEW，就会显得逻辑十分的清晰，代码接口也会比较优雅，易于复用。