ExoPlayer简单使用

leilifengxingmw 关注

4 2018.09.25 21:40:31 字数 1,928 阅读 92,977

ExoPlayer Library 概述

> ExoPlayer是运行在YouTube app Android版本上的视频播放器

ExoPlayer是构建在Android低水平媒体API之上的一个应用层媒体播放器。和Android内置的媒体播放器相比，ExoPlayer有许多优点。ExoPlayer支持内置的媒体播放器支持的所有格式外加自适应格式DASH和SmoothStreaming。ExoPlayer可以被高度定制和扩展以适应不同的使用场景。

ExoPlayer库的核心是ExoPlayer接口。ExoPlayer公开了传统的高水平媒体播放器的功能，例如媒体缓冲，播放，暂停和快进功能。ExoPlayer实现旨在对正在播放的媒体类型，存储方式和位置以及渲染方式做出一些假设（因此几乎没有限制）。ExoPlayer没有直接实现媒体文件的加载和渲染，而是把这些工作委托给了在创建播放器或者播放器准备好播放的时候注入的组件。所有ExoPlayer实现的通用组件是：

- `MediaSource`： 媒体资源，用于定义要播放的媒体，加载媒体，以及从哪里加载媒体。简单的说， `MediaSource` 就是代表我们要播放的媒体文件，可以是本地资源，可以是网络资源。`MediaSource` 在播放开始的时候，通过 `ExoPlayer.prepare` 方法注入。
- `Renderer`： 渲染器，用于渲染媒体文件。当创建播放器的时候， `Renderers` 被注入。
- `TrackSelector`： 轨道选择器，用于选择 `MediaSource` 提供的轨道（tracks），供每个可用的渲染器使用。
- `LoadControl`： 用于控制 `MediaSource` 何时缓冲更多的媒体资源以及缓冲多少媒体资源。`LoadControl` 在创建播放器的时候被注入。

ExoPlayer库提供了在普通使用场景下上述组件的默认实现。ExoPlayer可以使用这些默认的组件，也可以使用自定义组件。例如可以注入一个自定义的 `LoadControl` 用来改变播放器的缓存策略，或者可以注入一个自定义渲染器以使用Android本身不支持的视频解码器。

优点和缺点

优点

- 支持HTTP上的动态自适应流DASH和SmoothStreaming。更多详情请参看 Supported formats。
- 支持高级的HLS特点,例如正确的处理 `#EXT-X-DISCONTINUITY` 标签。
- 能够无缝的合并，串联，循环播放媒体文件。
- 能够被高度扩展和定制，以适用不同的场景。

缺点

- 在某些设备上播放音频，ExoPlayer可能会比MediaPlayer消耗更多的电量。

使用

1. 添加依赖

```
implementation 'com.google.android.exoplayer:exoplayer:2.X.X'
```

为了省事，我们依赖了整个ExoPlayer库。你也可以只依赖你真正需要的库。例如果你要播放DASH类型的媒体资源，你可以只依赖 `Core` , `DASH` , `UI` 这三个库。

```
implementation 'com.google.android.exoplayer:exoplayer-core:2.X.X'
implementation 'com.google.android.exoplayer:exoplayer-dash:2.X.X'
implementation 'com.google.android.exoplayer:exoplayer-ui:2.X.X'
```

整个ExoPlayer库包括5个子库，依赖了整个ExoPlayer库和依赖5个子库是等效的。

- `exoplayer-core`： 核心功能 (必要)
- `exoplayer-dash`： 支持DASH内容
- `exoplayer-hls`： 支持HLS内容
- `exoplayer-smoothstreaming`： 支持SmoothStreaming内容
- `exoplayer-ui`： 用于ExoPlayer的UI组件和相关的资源。

2. 在布局文件中加入PlayerView

```
<com.google.android.exoplayer2.ui.PlayerView
   android:id="@+id/video_view"
   android:layout_width="match_parent"
   android:layout_height="match_parent"/>
```

```
private PlayerView playerView;

@Override
protected void onCreate(Bundle savedInstanceState) {
    [...]
   playerView = findViewById(R.id.video_view);
}
```

3. 创建一个SimpleExoPlayer实例，SimpleExoPlayer是ExoPlayer接口的一个默认的通用实现。

```
private ExoPlayer player;
private boolean playWhenReady;
private int currentWindow;
private long playbackPosition;

private void initializePlayer() {
     player = ExoPlayerFactory.newSimpleInstance(
         new DefaultRenderersFactory(this),
         new DefaultTrackSelector(), new DefaultLoadControl());

     playerView.setPlayer(player);

     player.setPlayWhenReady(playWhenReady);
     player.seekTo(currentWindow, playbackPosition);
}
```

在上面的代码中，我们传入了默认的渲染工厂（DefaultRenderersFactory），默认的轨道选择器（DefaultTrackSelector）和默认的加载控制器（DefaultLoadControl），然后把返回的播放器实例赋值给成员变量player。

4. 创建一个MediaSource。

```
private void initializePlayer() {
     [...]
     //创建一个mp4媒体文件
     Uri uri = Uri.parse(getString(R.string.media_url_mp4));
     MediaSource mediaSource = buildMediaSource(uri);
     player.prepare(mediaSource, true, false);
}
```

```
private MediaSource buildMediaSource(Uri uri) {
  return new ExtractorMediaSource.Factory(
      new DefaultHttpDataSourceFactory("exoplayer-codelab")).
      createMediaSource(uri);
}
```

这时候就可以播放我们的媒体文件了。上个图。

11537585473_.pic_thumb.jpg

完整的initializePlayer方法

```
private void initializePlayer() {
        if (player==null){
            player = ExoPlayerFactory.newSimpleInstance(
                    new DefaultRenderersFactory(this),
                    new DefaultTrackSelector(), new DefaultLoadControl());

            playerView.setPlayer(player);

            player.setPlayWhenReady(playWhenReady);
            player.seekTo(currentWindow, playbackPosition);
        }

        Uri uri = Uri.parse(getString(R.string.media_url_mp4));
        MediaSource mediaSource = buildMediaSource(uri);
        player.prepare(mediaSource, false, true);
    }
```

如果我们要播放一个音频文件呢？我们只要在创建MediaSource的时候传入一个音频文件的路径就可以了，其他的都交给PlayerView即可，真是很爽。

```
Uri uri = Uri.parse(getString(R.string.media_url_mp3));
```

另外必须注意，我们要在合适的时机释放资源

```
private void releasePlayer() {
  if (player != null) {
    playbackPosition = player.getCurrentPosition();
    currentWindow = player.getCurrentWindowIndex();
    playWhenReady = player.getPlayWhenReady();
    player.release();
    player = null;
  }
}
```

组合媒体资源

ExoPlayer库提供了ConcatenatingMediaSource和DynamicConcatenatingMediaSource可以用来无缝的合并播放多个媒体资源。

使用ConcatenatingMediaSource

下面的方法构建了一个音频文件和视频文件组合的媒体文件

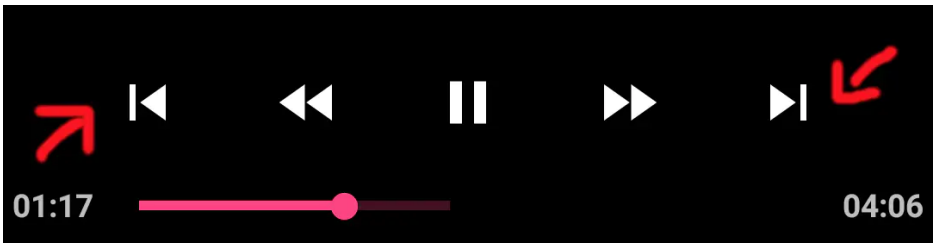
```
private MediaSource buildMediaSource(Uri uri) {
        DefaultHttpDataSourceFactory dataSourceFactory =
                new DefaultHttpDataSourceFactory("user-agent");

```

```
        ExtractorMediaSource videoSource =
                new ExtractorMediaSource.Factory(dataSourceFactory).
                        createMediaSource(uri);

        Uri audioUri = Uri.parse(getString(R.string.media_url_mp3));
        ExtractorMediaSource audioSource =
                new ExtractorMediaSource.Factory(dataSourceFactory).
                        createMediaSource(audioUri);

        return new ConcatenatingMediaSource(audioSource, videoSource);
    }
```



6b5e3f6d1570358a.png

你可以点击上一个和下一个按钮来播放上一个或者下一个媒体文件。

自适应流

ExtractorMediaSource适合常规的媒体文件（mp4，webm，mkv等等）。ExoPlayer也提供了自适应流媒体文件的实现，像DASH(DashMediaSource), SmoothStreaming (SsMediaSource)和HLS(HlsMediaSource)。

简而言之，自适应播放将视频或者音频文件切割成给定持续时间的多个块。这些块有不同的质量（尺寸或者比特率）。视频播放器可以根据当前可用的网络带宽选择不同质量的块。例如开始播放的时候选择低质量的块，可以更快的渲染第一帧，然后带宽足够的情况下，第二块可以选择更高的质量。

自适应轨道选择

注意：英文是Adaptive track selection，不知道怎么翻译才好，我的理解是就是选择上面所说的块，如果有大佬知道，还望不吝赐教。

自适应流的核心就是选择最合适当前播放环境的轨道。让我们启用自适应轨道选择。

自适应播放根据测量的下载速度来估计网络带宽。我们来定义一个DefaultBandwidthMeter常量。

```
private static final DefaultBandwidthMeter BANDWIDTH_METER =
    new DefaultBandwidthMeter();
```

```
private void initializePlayer() {
  if (player == null) {
    // 用来创建AdaptiveVideoTrackSelection的工厂
    TrackSelection.Factory adaptiveTrackSelectionFactory =
        new AdaptiveTrackSelection.Factory(BANDWIDTH_METER);

    player = ExoPlayerFactory.newSimpleInstance(
        new DefaultRenderersFactory(this),
        new DefaultTrackSelector(adaptiveTrackSelectionFactory),
        new DefaultLoadControl());
    [...]
}
```

DefaultTrackSelector用来选择轨道，我们把AdaptiveTrackSelection.Factory传入DefaultTrackSelector的构造函数，这样DefaultTrackSelector就可以选择自适应的轨道了。

创建一个自适应的媒体资源

DASH是一个广泛应用的自适应流格式。使用ExoPlayer播放DASH格式的媒体，需要创建一个自适应媒体资源。

```
private MediaSource buildMediaSource(Uri uri) {
  DataSource.Factory manifestDataSourceFactory =
      new DefaultHttpDataSourceFactory("ua");
  DashChunkSource.Factory dashChunkSourceFactory =
      new DefaultDashChunkSource.Factory(
      new DefaultHttpDataSourceFactory("ua", BANDWIDTH_METER));
  return new DashMediaSource.Factory(dashChunkSourceFactory,
      manifestDataSourceFactory).createMediaSource(uri);
}
```

传入一个DASH格式的uri

```
 Uri uri = Uri.parse(getString(R.string.media_url_dash));
```

这样就可以愉快的播放DASH格式的媒体文件了。

监听ExoPlayer的事件

```
private ComponentListener componentListener;

private class ComponentListener extends Player.DefaultEventListener {

        @Override
        public void onPlayerStateChanged(boolean playWhenReady, int playbackState) {
            String stateString;
            // actually playing media
            if (playWhenReady && playbackState == Player.STATE_READY) {
                Log.d(TAG, "onPlayerStateChanged: actually playing media");
            }
            switch (playbackState) {
                case Player.STATE_IDLE:
                    stateString = "ExoPlayer.STATE_IDLE      -";
                    break;
                case Player.STATE_BUFFERING:
                    stateString = "ExoPlayer.STATE_BUFFERING -";
                    break;
                case Player.STATE_READY:
                    stateString = "ExoPlayer.STATE_READY     -";
                    break;
                case Player.STATE_ENDED:
                    stateString = "ExoPlayer.STATE_ENDED     -";
                    break;
                default:
                    stateString = "UNKNOWN_STATE             -";
                    break;
            }
            Log.d(TAG, "changed state to " + stateString + " playWhenReady: " + playWhenRec
        }
    }
```

新建一个类ComponentListener继承Player.DefaultEventListener然后覆盖自己感兴趣事件方法。

然后注册监听

```
private void initializePlayer() {
 if (player == null) {
   [...]
   player = ExoPlayerFactory.newSimpleInstance(
       new DefaultRenderersFactory(this),
       new DefaultTrackSelector(adaptiveTrackSelectionFactory),
       new DefaultLoadControl());
   //注册监听
   player.addListener(componentListener);
   [...]
}
```

要记得在释放资源的时候，也移除掉监听器。

```
private void releasePlayer() {
 if (player != null) {
   [...]
   player.removeListener(componentListener);
   player.release();
   player = null;
 }
}
```

使用VideoRendererEventListener and AudioRendererEventListener

我们可以自定义MyVideoRendererEventListener，MyAudioRendererEventListener分别实现上面两个接口。

```
private MyVideoRendererEventListener videoRendererEventListener;
private MyAudioRendererEventListener audioRendererEventListener;
```

```
 private void initializePlayer() {
  [...]
  player.addListener(componentListener);
  player.addVideoDebugListener(videoRendererEventListener);
  player.addAudioDebugListener(audioRendererEventListener);
  [...]
}
```

释放资源的时候也一样，移除相应的监听器。

自定义用户界面

如果我们不设置的话，ExoPlayer 默认使用的播放控制界面是 `PlayerControlView`
如果我们完全不想使用这个控制界面，可以在布局文件里面修改

```
<com.google.android.exoplayer2.ui.PlayerView
   [...]
   app:use_controller="false"/>
```

这样控制界面就不显示了。

自定义PlayerControlView的行为

```
<com.google.android.exoplayer2.ui.PlayerView
   android:id="@+id/video_view"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:show_timeout="10000"
   app:fastforward_increment="30000"
   app:rewind_increment="30000"/>
```

上面的例子中，快进和快退都改成了30秒。控制界面自动消失时间是10秒。

自定义PlayerControlView界面的外观

1. 你可以自定义控制界面，然后在布局文件里更改属性 `controller_layout_id`

```
<com.google.android.exoplayer2.ui.PlayerView
   android:id="@+id/video_view"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:controller_layout_id="@layout/custom_playback_control"/>
```

> PlayerControlView通过id来识别它使用的所有UI元素。当你自定义布局文件的时候，必须保持标准元素的id，例如@id/exo_play 或 @id/exo_pause，以便让PlayerControlView有机会找到它们。

默认的PlayerControlView的控制界面是 `R.layout.exo_playback_control_view.xml` 。你也可以直接从ExoPlayer库中复制到app的res目录下面，然后做相应的更改即可。

参考链接：

- exoplayer-intro
- 完整项目源码
- guide