mediasession分析

Last edited by **caoquanli** 1 month ago

Android8.1 Mediasession 分析

概念性简述

- MediaSession 框架是Google推出的专门解决媒体播放时界面和服务的通讯问题，涉及的功能较多，本文只针对MediaSession部分进行分析。

核心类

- MediaSessionManager
- MediaSessionService
- MediaSessionRecord
- MediaSessionStack

一、MediaSession使用

- 按照APK的开发流程的话，应该是使用的MediaSessionCompat，但是本文不涉及这一块，如有兴趣请自行研究，MediaSessionCompat实质上最后也是去使用MediaSession。

- 根据demo代码我们来一个个的来分析下。

1.1 初始化配置

- 初始化并激活MediaSession

```
public class TestMediaSession{
    private void initMediaSession() {
        //第一个参数是上下文对象，第二个是用与调试使用的，可以填写任意的字符串
        mMediaSession = new MediaSession(context, "TestMediaSession");
        //FLAG_HANDLES_MEDIA_BUTTONS 控制媒体按钮
        //FLAG_HANDLES_TRANSPORT_CONTROLS 控制传输命令
        //FLAG_EXCLUSIVE_GLOBAL_PRIORITY 优先级最高的，会在activity处理之前先处理，
        //且不需要配合MediaPlayer使用。（注：Android8.1开始MediaSession必须配合
        //MediaPlayer使用，之后会有说明）
        mMediaSession.setFlags(MediaSession.FLAG_HANDLES_MEDIA_BUTTONS |
                MediaSession.FLAG_HANDLES_TRANSPORT_CONTROLS);
        //MediaSession必须激活才能去使用，当你不希望使用MediaSession的是，可以设置false
        mMediaSession.setActive(true);
    }
    ...
    ...
    ...
}
```

1.2 设置监听回调

```
public class TestMediaSession{
    ...
    ...
    ...
    public void setCallBack(MediaSession.Callback callback){
        mMediaSession.setCallback(callback);
    }
    //Callback有多个函数可以重写，但本文只针对MediaKey的接收，故只重写此函数
    class MediaSessionCallback extends MediaSession.Callback {
        @override
        public boolean onMediaButtonEvent(Intent mediaButtonEvent) {
            KeyEvent event = (KeyEvent) mediaButtonEvent.getParcelableExtra(Intent.EXTRA_KE
            return true;
        }
    }
    ...
    ...
    ...

}
```

- 以上步骤完成就可以去接收MediaKey了。

二、MediaSession源码分析

类图

- 首先我们先看一下大体的一个类图，看一下MediaSession这个模块涉及到了哪些(其中我省去了一些,例如MediaController)可以看出去主要就是我们之前提出来的几个核心类以及它们的内部类。通过类图来看的话，我们可以这样来梳理一下关系，MediaSession通过MediaSessionManager与MediaSessionService中产生了关联，然后create了一个ISession对象，也可以理解为MediaSessionRecord，之后的回调监听处理，都是通过MediaSessionRecord去完成的，而MediaSession是可以设置多个的，例如多个apk都去设置，那么怎么去管理这么多的MediaSessionRecord呢，这个时候就使用了MediaSessionStack，顾名思义就是一个类似栈的方式去管理MediaSessionRecord，先进后出，这个我会在之后详细说一下，

2.1 MediaSeesion的设置

- 根据第一节的代码我们知道了使用MediaSession需要进行哪些准备工作，看上去只是设置了几个变量，其实内在是做了不少事情的，我们先来看一下简单的一个时序图：

- 我们挑重要的说一下：
  - 在MediaSession的构造函数中获取了MediaSessionManager，然后通过MediaSessionManager去创建了MediaSessionRecord，将自己的 CallbackStub设置给MediaSessionRecord ，并将其add进了MediaSessionStack，通过MediaSessionStack统一的去管理 MediaSessionRecord，内部使用的是List去存储的，至于是怎么处理这个List的，之后会详细看一下。
  - setCallBack，MediaSession的监听回调得通过此函数设置，将重写的MediaSession.Callback，设置给了Mediasession，最后创建了 CallbackMessageHandler，CallbackStub与CallbackMessageHandler是有关联的，至此MediaSession与MediaSessionRecord就关联上 了。
  - setFlags,这个有必要说一下，在MediaSessionCompat中flag有三种，FLAG_HANDLES_MEDIA_BUTTONS、FLAG_HANDLES_TRANSPORT_CONTROLS和FLAG_HANDLES_QUEUE_COMMANDS，常用的是前两种，前者是控制媒体按钮，后者是控制传输命令，而在MediaSession中的flag与MediaSessionCompat略有不同，分别是FLAG_HANDLES_MEDIA_BUTTONS、FLAG_HANDLES_TRANSPORT_CONTROLS 和FLAG_EXCLUSIVE_GLOBAL_PRIORITY，前两者与MediaSessionCompat功能一样，后者是hide的，Android官方不公开去使用的，优先级最高 的，会在activity处理之前先处理，且不需要配合MediaPlayer使用，而前两者是必须去配合Mediaplayer去使用的，在下文中会有提及，此处不再 多说。

2.2 MediaSeesion的管理

- MediaSession是可以存在多个的，那么它们的使用、管理是怎么做的呢，，其实很简单，首先每当有MediaSession（有MEdiaSession就会生成MediaSessionRecord）产生的时候，最终都会调用MediaSessionStack的addSession函数，将MediaSessionRecord使用list保存下来，如果MediaSession死亡的话，会从list列表中remove掉。但是保存下来了，不代表就去使用了，MediaSessionStack中有一个MediaSessionRecord全局变量 `mMediaButtonSession` ,真正起作用，对外使用的是它，也就是说，保存下来后，还需要将自身赋值给mMediaButtonSession，才能让MediaSession起作用。而这个赋值操作需要AudioPlaybackMonitor的回调才能激活，MediaSession才能起作用（前提是MediaSession已经setActive为true了），可以看一下具体的代码如下：

```
//frameworks/base/services/core/java/com/android/server/media/MeediaSessionService.java
public void onStart() {
    ...
```

```
    ...
    //如代码所示，获取了一个AudioService,然后设置了一个monitor去进行回调监听，一旦有audio发生变化，就会
    mAudioService = getAudioService();
    mAudioPlaybackMonitor = AudioPlaybackMonitor.getInstance(getContext(), mAudioService);
    mAudioPlaybackMonitor.registerOnAudioPlaybackStartedListener(
            new AudioPlaybackMonitor.OnAudioPlaybackStartedListener() {
        @Override
        public void onAudioPlaybackStarted(int uid) {
            synchronized (mLock) {
                FullUserRecord user =
                        getFullUserRecordLocked(UserHandle.getUserId(uid));
                if (user != null) {
                    user.mPriorityStack.updateMediaButtonSessionIfNeeded();
                }
            }
        }
    });
    ...
    ...
}

//frameworks/base/services/core/java/com/android/server/media/MediaSessionStack.java
public void updateMediaButtonSessionIfNeeded() {
    if (DEBUG) {
        Log.d(TAG, "updateMediaButtonSessionIfNeeded, callers=" + Debug.getCallers(2));
    }
    //获取audio中的uid,通过uid,去find已经add进来的MediaSession，如果发现当前正在使用的MediaSession
    //的uid跟audio中正在播放的不符的话，就会将MediaSession进行更新，没有正在使用的MediaSession的话，就
    //将mMediaButtonSession更新了。
    IntArray audioPlaybackUids = mAudioPlaybackMonitor.getSortedAudioPlaybackClientUids();
    for (int i = 0; i < audioPlaybackUids.size(); i++) {
        MediaSessionRecord mediaButtonSession =
                findMediaButtonSession(audioPlaybackUids.get(i));
        if (mediaButtonSession != null) {
            Log.d("MediaSessionTest", "Found the media button session");
            // Found the media button session.
            mAudioPlaybackMonitor.cleanUpAudioPlaybackUids(mediaButtonSession.getUid());
            if (mMediaButtonSession != mediaButtonSession) {
                updateMediaButtonSession(mediaButtonSession);
                 Log.d("MediaSessionTest", "updateMediaButtonSession");
            }
            return;
        }
    }
}
```

- 简单的概括成一张图的话，可能更容易理解，一个MediaSession创建的时候会经历两种情况：
  - ①->②->③->⑦（音频未播放）
  - ①->②->③->⑦->⑧（音频已播放）

- 而当音频文件播放的时候会按照如下顺序更新：
  - ④->⑤->⑥->⑦->⑧：

- 用一句话来总结的话，就是在创建MediaSession的时候会先去update一下，看此时MediaSession所在的进程是否已经开始播放音频，如果播放的话，就直接更新成创建的MediaSession,如果没有的话，就只是add进list。如果有音频播放的话，再去update一下，看是否有能匹配该进程的MediaSession（就是去匹配uid），有的话，就更新，没有就算了。