Mini声音管理

修改历史

Version	Contributor	Date	Change Log
v1.0	彭钉	18/08/10	init

概述

Mini声音管理服务是ROSE中一个服务应用，安装服务相应要安装ROSE中Master应用，master应用的安装参看《master用户指导》。用于管理Mini不同优先级下声音的播放控制，本文档简要说明这些接口及sdk集成。

SDK

- protobuf 结构

```
syntax = "proto3";
option java_package = "com.ubtrobot.mini.voice.protos";
option java_outer_classname = "VoiceProto";
import "google/protobuf/any.proto";

message Request {
    Cmd cmd = 1;
    string session = 2;
    google.protobuf.Any data = 3;
}
```

```
//MiniMediaPlayer 声音源类别
enum Source {
  RING = 0;//电话声音
  ALARM = 1;//闹钟
  SPEECH = 2; //语音闲聊
  MUSIC = 3; //音乐
  BEHAVIOR = 4; //表现力配音
}

enum Cmd {
  CREATE = 0;
  SET_DATA_SOURCE = 1;
  PREPARE = 2;
  START = 3;
  PAUSE = 4;
  STOP = 5;
  SEEKTO = 6;
  RESET = 7;
  RELEASE = 8;
  IS_PLAYING = 9;
  SET_VOLUME = 10;
  GET_DURATION = 11;
  GET_POSITION = 12;
}

enum State {
  PREPARED = 0;
  PERCENT = 1;
  SEEK = 2;
  COMPLETE = 3;
  ERROR = 4;
  RELEASED = 5;
}

message Error {
  int32 what = 1;
```

```
    int32 extra = 2;
}

message BriefVoice {
    string data = 1;
    Type type = 2;
    string priority = 3;
}

message VoiceProcess {
    int32 pid = 1;
    int32 uid = 2;
    Source source = 3;
}

enum Type {
    TTS = 0;
    FILE = 1;
}

message FakeEvent {
    Cmd cmd = 1;
    google.protobuf.Any data = 2;
}
```

- TTS 接口

```
/**
   * 播放 TTs
   *
   * @param text the text of TTs
   * @param priority 优先级 {@link Priority}
   * @param listener tts state callback
   */
 public void playTTs(String text, Priority priority, final VoiceListener listener);
```

```
/**
 * 播放本地TTs音频文件
 *
 * @param mp3FileName localTTs 下文件名
 * @param priority 优先级 {@link Priority}
 * @param listener tts state callback
 */
public void playLocalTTs(String mp3FileName, Priority priority, final VoiceListener listener) '

/**
 * 停止TTS
 *
 * @param priority 优先级 {@link Priority}
 * @param listener tts ResponseListener callback
 */
public void stopTTs(Priority priority, @Nullable final ResponseListener<Void> listener) ;

/**
 * 播放本地TTs音频文件, 没有参与VoiceManage内其他声音冲突管理
 *
 * @param mp3FileName localTTs 下文件名
 * @param listener tts state callback
 */
public void playUnsafeTTs(String mp3FileName, @Nullable final VoiceListener listener);
```

- MiniMediaPlayer

```
public final class MiniMediaPlayer {

 public static MiniMediaPlayer create(MasterContext context, VoiceProto.Source source)
     throws VoiceException {
   return new MiniMediaPlayer(context, source);
 }
```

```
/**
 * Sets the data source as a content Uri.
 *
 * @param path the Content URI of the data you want to play
 * @throws IOException if it is called in an invalid state
 * @throws IllegalArgumentException path argument
 * @throws SecurityException security
 * @throws IllegalStateException illegal state
 */
public void setDataSource(String path)
   throws IOException, IllegalArgumentException, SecurityException, IllegalStateException ;

/**
 * Prepares the player for playback, synchronously.
 * <p>
 * After setting the datasource and the display surface, you need to either
 * call prepare() or prepareAsync(). For files, it is OK to call prepare(),
 * which blocks until MediaPlayer is ready for playback.
 *
 * @throws IllegalStateException if it is called in an invalid state
 */
public void prepare() throws IOException, IllegalStateException ;

/**
 * Prepares the player for playback, synchronously.
 * <p>
 * After setting the datasource and the display surface, you need to either
 * call prepare() or prepareAsync(). For files, it is OK to call prepare(),
 * which blocks until MediaPlayer is ready for playback.
 *
 * @throws IllegalStateException if it is called in an invalid state
 */
public void prepareAsync() throws IllegalStateException ;

/**
 * Starts or resumes playback. If playback had previously been paused,
 * playback will continue from where it was paused. If playback had
 * been stopped, or never started before, playback will start at the
 * beginning. if playback source is low priority than other playback,
 * be rejected, return false.
 */
public void start();

/**
 * Pauses playback. Call start() to resume.
 *
 * @throws IllegalStateException if the internal player engine has not been
 * initialized.
 */
public void pause() throws IllegalStateException {
  mediaPlayer.pause();
  faker.pause();
}

/**
 * Stops playback after playback has been started or paused.
 *
 * @throws IllegalStateException if the internal player engine has not been
 * initialized.
 */
public void stop() ;

/**
 * Resets the MediaPlayer to its uninitialized state. After calling
 * this method, you will have to initialize it again by setting the
 * data source and calling prepare().
 */
public void reset() ;

/**
 * Releases resources associated with this MediaPlayer object.
 * It is considered good practice to call this method when you're
 * done using the MediaPlayer. In particular, whenever an Activity
 * of an application is paused (its onPause() method is called),
 * or stopped (its onStop() method is called), this method should be
 * invoked to release the MediaPlayer object, unless the application
 * has a special need to keep the object around. In addition to
 * unnecessary resources (such as memory and instances of codecs)
 * being held, failure to call this method immediately if a
 * MediaPlayer object is no longer needed may also lead to
 * continuous battery consumption for mobile devices, and playback
 * failure for other applications if no multiple instances of the
 * same codec are supported on a device. Even if multiple instances
 * of the same codec are supported, some performance degradation
 * may be expected when unnecessary multiple instances are used
 * at the same time.
 */
public void release() ;

/**
 * Seeks to specified time position.
 *
 * @param msec the offset in milliseconds from the start to seek to
 * @throws IllegalStateException if the internal player engine has not been
 * initialized
 */
public void seekTo(int msec) ;

/**
 * Checks whether the MediaPlayer is playing.
 *
 * @return true if currently playing, false otherwise
 * @throws IllegalStateException if the internal player engine has not been
 * initialized or has been released.
 */
public boolean isPlaying() ;

/**
 * Sets the volume on this player.
 * This API is recommended for balancing the output of audio streams
 * within an application. Unless you are writing an application to
 * control user settings, this API should be used in preference to
 * {@link AudioManager#setStreamVolume(int, int, int)} which sets the volume of ALL streams of
 * a particular type. Note that the passed volume values are raw scalars in range 0.0 to 1.0.
 * UI controls should be scaled logarithmically.
 *
 * @param volume left volume scalar
 */
public void setVolume(float volume) ;

/**
 * Gets the duration of the file.
 *
 * @return the duration in milliseconds, if no duration is available
 * (for example, if streaming live content), −1 is returned.
 */
public int getDuration() ;

/**
 * Gets the current playback position.
 *
 * @return the current position in milliseconds
 */
public int getCurrentPosition() ;

/**
 * Register a callback to be invoked when the media source is ready
 * for playback.
 *
 * @param listener the callback that will be run
 */
public void setOnPreparedListener(@Nullable MiniMediaPlayer.OnPreparedListener listener) ;
/**
 * Register a callback to be invoked when the end of a media source
 * has been reached during playback.
 *
 * @param listener the callback that will be run
 */
public void setOnCompletionListener(@Nullable MiniMediaPlayer.OnCompletionListener listener) ;

/**
 * Register a callback to be invoked when the status of a network
 * stream's buffer has changed.
 *
 * @param listener the callback that will be run.
 */
public void setOnBufferingUpdateListener(
    @Nullable MiniMediaPlayer.OnBufferingUpdateListener listener) ;

/**
 * Register a callback to be invoked when a seek operation has been
 * completed.
 *
 * @param listener the callback that will be run
 */
public void setOnSeekCompleteListener(@Nullable MiniMediaPlayer.OnSeekCompleteListener listener) ;

/**
 * Register a callback to be invoked when an error has happened
 * during an asynchronous operation.
 *
 * @param listener the callback that will be run
 */
public void setOnErrorListener(@Nullable MiniMediaPlayer.OnErrorListener listener) ;

public interface OnErrorListener {
  boolean onError(MiniMediaPlayer mp, int what, int extra);
}

public interface OnSeekCompleteListener {
  void onSeekComplete(MiniMediaPlayer mp);
}

public interface OnBufferingUpdateListener {
    void onBufferingUpdate(MiniMediaPlayer mp, int percent);
  }

  public interface OnCompletionListener {
    void onCompletion(MiniMediaPlayer mp);
  }

  public interface OnPreparedListener {
    void onPrepared(MiniMediaPlayer mp);
  }
}
```