Обработка аудио на стороне сервера

Описание

В сборке 5.2.795 добавлена возможность перехвата декодированного аудио из опубликованного потока в формате PCM 16 бит с последующей обработкой на стороне сервера. Например, звук из потока может быть записан в файл.

> **Attention**
>
> На опубликованный поток должен быть хотя бы один подписчик, чтобы декодировать аудио из этого потока.

Реализация обработчика аудио

Для захвата декодированного аудио из опубликованных потоков необходимо разработать класс на языке Java, реализующий интерфейс `IDecodedPcmInterceptor`. Функция этого класса `pcmDecoded()` будет получать декодированное аудио в формате PCM в виде последовательность байтов. Приведем пример реализации класса для записи декодированного аудио из потока в файл в формате WAV:

DecodedPcmInterceptorTest.java

```
package com.flashphoner.pcmInterceptor;

// Import Flashphoner SDK packages as needed
import com.flashphoner.media.rtp.recorder.OutputFileType;
import com.flashphoner.media.utils.FileNameUtils;
import com.flashphoner.media.utils.WaveUtil;
import com.flashphoner.sdk.media.IDecodedPcmInterceptor;
import com.flashphoner.sdk.setting.Settings;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
// Import standard Java packages as needed
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Timer;
import java.util.TimerTask;

/**
 * Custom decoded audio interceptor implementation example
 * The example records first 10 seconds of audio track from a stream
published to WAV file
 */
public class DecodedPcmInterceptorTest implements IDecodedPcmInterceptor{

    // Will log errors to server log.
    private static final Logger log =
LoggerFactory.getLogger("DecodedPcmInterceptorTest");

    // File object to write data
    protected RandomAccessFile incomingRecorder;
    // Last audio packet timestamp
    private volatile long lastTs;
    // Sampling rate to write to a file header
    private int samplingRate;
    // Number of cahnnels to write to a file header
    private int numChannels;
    // Timer to stop recording
    private Timer cancelTimer;
    // Stream name to form file name
    private String streamName;

    public DecodedPcmInterceptorTest() {
    }

    /**
     * Method to handle decoded audio
     * @param streamName - stream name
     * @param pcm - decoded audio data from packet as byte array
     * @param samplingRate - audio track sampling rate
     * @param numChannels - audio track number of channels
     * @param timestamp - audio packet timestamp
     */
    @Override
    public void pcmDecoded(String streamName, byte[] pcm, int
samplingRate, int numChannels, long timestamp) {
        // Remember data to write to the file header
        updateSr(samplingRate);
        updateNumChannels(numChannels);
        // Remember the stream name and create the file to write
```

```
            updateStreamName(streamName);
            // Start timeout to stop recording
            startCloseTimer();
            try {
                // Write audio data to the file
                recordIncoming(pcm, timestamp);
            } catch (IOException e) {
                e.printStackTrace();
            }
        }


        /**
         * Method to start timer task which should stop recording after 10
    seconds
         */
        public void startCloseTimer() {
            if (cancelTimer == null) {
                cancelTimer = new Timer();
                cancelTimer.schedule(new TimerTask() {
                    @Override
                    public void run() {
                        try {
                            close();
                        } catch (IOException e) {
                            e.printStackTrace();
                        }
                    }
                }, 10000);
            }
        }

        /**
         * Method to store audio sampling rate to write it to the file header
         */
        public void updateSr(int samplingRate) {
            if (this.samplingRate == 0) {
                this.samplingRate = samplingRate;
            }
        }

        /**
         * Method to store audio channels number to write it to the file
    header
         */
        public void updateNumChannels(int numChannels) {
            if (this.numChannels == 0) {
                this.numChannels = numChannels;
            }
        }

        /**
         * Method to store the stream name and create the file to write
         */
        public void updateStreamName(String streamName) {
            if (this.streamName == null || this.streamName.isEmpty()) {
                // Create the file name
                this.streamName =
    FileNameUtils.adaptRecordName(streamName+".wav");
                try {
                    // Create the file and reserve header space
                    incomingRecorder = new
```

```
RandomAccessFile(Settings.RECORD.getValue()+"/"+this.streamName, "rw");
                incomingRecorder.write(new
byte[OutputFileType.WAV_HEADER_OFFSET]);
            } catch (IOException e) {
                log.error("Can't create DecodedPcmInterceptorTest, " +
e.getMessage());
            }
            log.info("Create DecodedPcmInterceptorTest");
        }
    }

    /**
     * Method to write audio data to the file
     */
    public void recordIncoming(byte[] data, long ts) throws IOException {
        incomingRecorder.write(data);
        lastTs = ts;
    }

    /**
     * Method to close file
     */
    private void close() throws IOException {
        // Write header to the file
        writeHeader();
        try {
            // Close the file
            incomingRecorder.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
        log.info("Close DecodedPcmInterceptorTest");
    }

    /**
     * Method to write header to the beginning of the file
     */
    protected void writeHeader() throws IOException {
        // Get the file size
        int size = (int) incomingRecorder.length();
        // Form the header
        byte[] header = WaveUtil.getPcmWaveHeader((size -
OutputFileType.WAV_HEADER_OFFSET), samplingRate, numChannels);
        // Write the header to the beginning of the file
        incomingRecorder.seek(0);
        incomingRecorder.write(header);
    }
}
```

Затем следует скомпилировать класс в байт-код. Для этого создаем дерево каталогов, соответствующее названию пакета написанного класса

```
mkdir -p com/flashphoner/pcmInterceptor
```

и выполняем команду

```
javac -cp /usr/local/FlashphonerWebCallServer/lib/wcs-
core.jar:/usr/local/FlashphonerWebCallServer/lib/slf4j-api-1.6.4.jar
./com/flashphoner/pcmInterceptor/DecodedPcmInterceptorTest.java
```

Теперь упакуем скомпилированный код в jar-файл

```
jar -cf testPcmInterceptor.jar ./com/flashphoner/pcmInterceptor/*.class
```

и скопируем его в каталог, где размещены библиотеки WCS сервера

```
cp testPcmInterceptor.jar /usr/local/FlashphonerWebCallServer/lib
```

Для того, чтобы использовать разработанный класс, необходимо указать имя его пакета и расположение каталога для записей в файле flashphoner.properties

```
decoded_pcm_interceptor=com.flashphoner.pcmInterceptor.DecodedPcmInterceptorTest

record=/usr/local/FlashphonerWebCallServer/records
```

и перезапустить WCS.

Отдельный каталог для собственных Java библиотек

Начиная со сборки 5.2.1512, Java библиотеки (jar файлы) должны помещаться в каталог `/usr/local/FlashphonerWebCallServer/lib/custom`

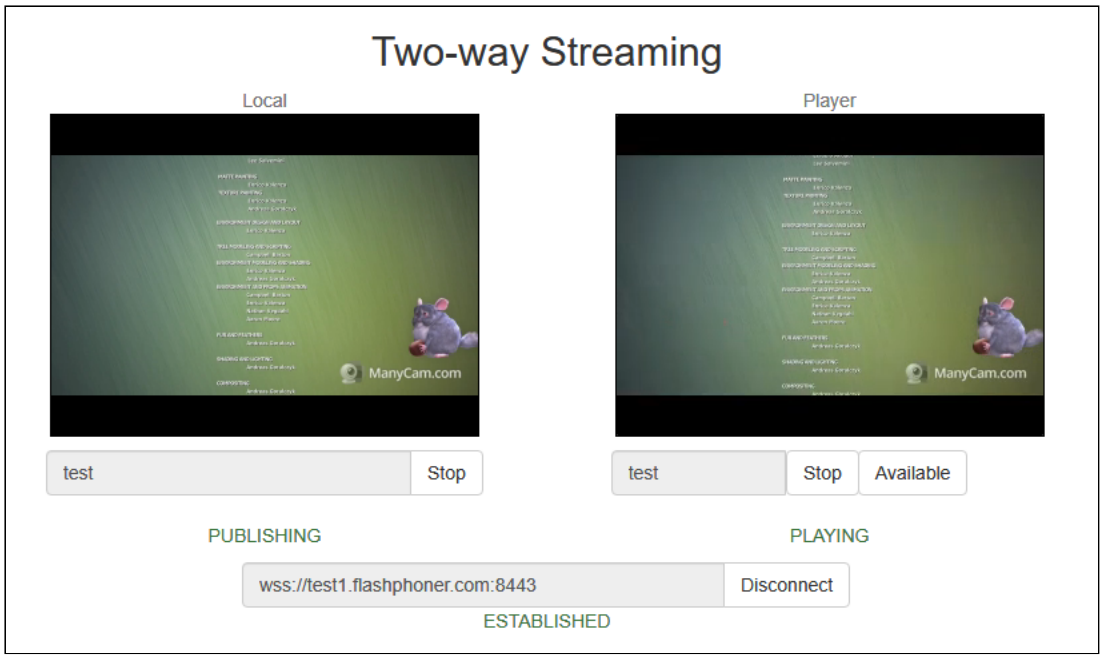
```
cp testPcmInterceptor.jar /usr/local/FlashphonerWebCallServer/lib/custom
```

Этот каталог сохраняется при дальнейших обновлениях сервера к более новым сборкам. Таким образом, нет необходимости снова копировать jar файлы после установки обновления.

Тестирование

1. Опубликуйте поток в примере Two Way Streaming `https://test1.flashphoner.com:8444/client2/examples/demo/streaming/two_way_streaming/two_way_streaming.html`, где `test1.flashphoner.com` - адрес WCS

сервера



2. Проверьте наличие файла в каталоге `/usr/local/FlashphonerWebCallServer/records/`

```
total 1812
-rw-r--r-- 1 root root 1854764 Oct 22 16:01 test-1badb540-1445-11eb-9dfd-614c13aa0eb7.wav
```