ドコモ AI エージェント API

サンプルアプリから音声制御ライブラリへの移行実装

第 1.0 版

改定日：2020.03.16

改訂履歴

版数	発行日	改訂履歴
1.0 版	2020.03.16	新規作成

サンプルアプリから音声制御ライブラリへの移行実装

概要

SpeakSDK から音声制御ライブラリへの移行手順をサンプルアプリの移行を例に解説する。

ライブラリ

¥(プロジェクト)¥app¥libs 以下のファイルを入れ替える。

SpeakSDK 利用時は以下のファイルを利用する。

・speak-debug.aar

・speak-release.aar

音声制御ライブラリ利用時は上記のファイルを削除し、以下のファイルを格納する。

・flow.jar

・vgasrmw.jar

・VoiceControlLibrary_debug_on.jar

※ ログ出力が不要な場合は「VoiceControlLibrary_debug_on.jar」の代わりに「VoiceControlLibrary.jar」を利用する

JNI ライブラリ

¥(プロジェクト)¥app¥src¥main¥jniLibs ディレクトリを作成し、以下のディレクトリごと格納する。

・armeabi-v7a

・arm64-v8a

Gradle

各 gradle ファイルの修正を行う。

¥(プロジェクト)¥settings.gradle

```
include ':app',':lsec'
```

¥(プロジェクト)¥app¥build.gradle

dependencies に音声制御ライブラリで利用するライブラリを追加する

```
dependencies {
    ...
    compile 'com.google.code.gson:gson:2.3.1'
    compile 'com.fasterxml.jackson.core:jackson-core:2.8.8'
    compile 'com.fasterxml.jackson.core:jackson-databind:2.8.8'
}
```

dependencies から SpeakSDK を削除する

```
dependencies {
    ...
    releaseCompile(name:'speak-release', ext:'aar')
    debugCompile(name:'speak-debug', ext:'aar')
}
```

AndroidManifest.xml

¥(プロジェクト)¥app¥src¥main¥AndroidManifest.xml

AndroidManifest.xml にパーミッションを追加する。

```
<!-- 音声録音処理、サーバ通信処理に使用 -->
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<!-- IMEI, 電話番号の取得に使用 -->
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<!-- Bluetooth 音声入力に使用 -->
<uses-permission android:name="android.permission.BLUETOOTH"/>
<!-- WiFi 状態取得に使用 -->
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

ソース修正

修正が必要なソースは以下のファイルです。

¥(プロジェクト)¥app¥src¥main¥java¥com¥nttdocomo¥trial_app¥ChatApplication.java

ChatApplication クラスより変数 SpeakSDK sdk をコメントアウトする。

```
    // private final Speak sdk = new Speak();
```

ChatApplication クラスに音声制御ライブラリ関連の変数を追加する。

```
    private NluMetaData sendTextModeMeta;

    // private static final String DEF_ModelFilePath = "wordSpottingModel.fuet";/* WordSpotting 用モデル（キーワードモデル）ファイル */

    // エンジン
    private VoiceRecognitionController mVoiceRecognitionController = null;
    // Entity
    // XMLParams.ParamDataClass mParamDataClass = null;
    private RecognitionParameterEntity mRecognitionParameterEntity = null;
    private CommunicationParameterEntity mCommunicationParameterEntity = null;
    private DebugParameterEntity mDebugParameterEntity = null;
    // private ModelBaseEntity mModelParameterEntity = null;
    // private MelodyFileParameterEntity mMelodyFileParameterEntity = null;
    private RoamingParameterEntity mRoamingParameterEntity = null;
    private SebastienParameterEntity mSebastienParameterEntityStart = null;
    private SebastienParameterEntity mSebastienParameterEntitySend = null;
```

public void onCreate() に音声制御ライブラリの初期化処理を追加する。

```
        mVoiceRecognitionController = new VoiceRecognitionController(getApplicationContext(), new VoiceRecognitionControllerEventListenerImpl());

        mRecognitionParameterEntity = new RecognitionParameterEntity();
        mCommunicationParameterEntity = new CommunicationParameterEntity("HA_APP", "mobile", 443, "VoiceAgent");
        // mDebugParameterEntity = new DebugParameterEntity();
        // mModelParameterEntity = new ModelBaseEntity();
        // mMelodyFileParameterEntity = new MelodyFileParameterEntity(null);
        mRoamingParameterEntity = new RoamingParameterEntity();
        mSebastienParameterEntityStart = new SebastienParameterEntity();
        // mSebastienParameterEntitySend = new SebastienParameterEntity();
```

ChatApplication.init()メソッドの SpeakSDK 初期化処理をコメントアウトする。

下記イベント発生時の処理は VoiceRecognitionControllerEventListener クラスにて記述する。

・メタデータ受信時の処理

・合成音声再生開始時の処理

・合成音声再生終了時の処理

```
public void init(final MainActivity activity) {
    // WebViewキャッシュの削除
    new WebView(activity).clearCache(true);
    chatView = activity.findViewById(R.id.chat_area);
    chatView.setOnScrollListener(balloonAdapter);
    chatView.setAdapter(balloonAdapter);
    chatView.setFriction(ViewConfiguration.getScrollFriction() * SCROLL_WEIGHT);
    chat.init(activity);

    /*
    sdk.set("EnableOCSP", true);
    sdk.set("OutputGain", 1.00);
    // メタデータ受信時の処理
    sdk.setOnMetaOut(new StringEventHandler() {
            …
    });
    // 合成音声再生開始時の処理
    sdk.setOnPlayStart(new StringEventHandler() {
            …
    });
    // 合成音声再生終了時の処理
    sdk.setOnPlayEnd(new StringEventHandler() {
            …
    });
    sdk.setContext(getApplicationContext());
    */
    if (Config.getInstance().getDeviceToken() == null) {
        // ユーザダッシュボードのログイン画面を表示
        FragmentTransaction ft = activity.getSupportFragmentManager().beginTransaction();
        ft.replace(R.id.base_layout, new Login()).commitAllowingStateLoss();
    }

}
```

SpeakSDK を利用している処理をコメントアウトする。

```
/**
 * NLU メタデータを送信
 *
 * @param metaData NLU メタデータ
 */
public void putMeta(NluMetaData metaData) {
    // sdk.putMeta(metaData);
}
```

```
/**
 * 音声入力 OFF
 */
public void mute() {
    // sdk.mute();
}

/**
 * 音声入力 ON
 */
public void unmute() {
    // sdk.unmute();
}

/**
 * 合成音声再生キャンセル
 */
public void cancelPlay() {
    // sdk.cancelPlay();
}
```

ChatApplication.start() から SpeakSDK を利用する処理をコメントアウトし、音声制御ライブラリの初期化処理を追加する。

```
/**
 * SDK の開始
 *
 * @param onStart 開始時の処理
 */
public void start(ChatStartHandler onStart) {
    chat.setStatus(ChatStatus.STARTING);
    /*
    sdk.setMicMute(onStart.mode == ChatMode.TEXT);
    Config config = Config.getInstance();
    sdk.setURL(config.getUrl());
    sdk.setDeviceToken(config.getDeviceToken());
    sdk.start(onStart, new ChatErrorHandler(onStart));
    */
    loadParam(true, true);
    if(onStart.mode == ChatController.ChatMode.VOICE) {
        initSebastien(RecognitionMode.VOICE_RECOGNITION_MODE);
    } else {
        initSebastien(RecognitionMode.SEND_TEXT_MODE);
        sendTextModeMeta = onStart.meta;
    }
}
```

音声制御ライブラリの設定を行う。

```
private void loadParam (final boolean load, final boolean isSebas) {

    Config config = Config.getInstance();

    mRecognitionParameterEntity.setAntiClipping(false);
    mRecognitionParameterEntity.setAutomaticEndByVoicelessMeasure(false);
    mRecognitionParameterEntity.setAutomaticStartingByVoicedMeasure(false);
    mRecognitionParameterEntity.setAutomaticUtteranceStartingTimeoutTime(20000);
    mRecognitionParameterEntity.setCharacterCode(CharacterSet.SHIFT_JIS);
    mRecognitionParameterEntity.setContinueCheck(false);
    mRecognitionParameterEntity.setImmediateDetect(false);
    mRecognitionParameterEntity.setInputMicType(InputMicType.MIC); // mike or bluetooth
    mRecognitionParameterEntity.setMaxRecognitionResultAcquisitionTime(0);
    mRecognitionParameterEntity.setModelData(null);
    mRecognitionParameterEntity.setParameterFileType(ParameterFileType.DEFAULT);
    mRecognitionParameterEntity.setRecordSizeOfVoiceInputBuffer(10000);
    mRecognitionParameterEntity.setServerVad(true);
    mRecognitionParameterEntity.setVoicelessJudgmentTime(300);
    mRecognitionParameterEntity.setWaitEchoCanceller(false);

    mCommunicationParameterEntity.setAccessPoint(CommunicationParameterEntity.AccessPointType.SEBASTIEN);
    mCommunicationParameterEntity.setAccessToken(config.getDeviceToken());
    mCommunicationParameterEntity.setApplicationName("HA_APP");
    mCommunicationParameterEntity.setBackendPort(443);
    mCommunicationParameterEntity.setBackendServiceName("VoiceAgent");
    mCommunicationParameterEntity.setBackendType("dospf.aiplat.jp");
    mCommunicationParameterEntity.setBackendTypeURLPath("/ciel");
    mCommunicationParameterEntity.setConnectLimit(30000);
    mCommunicationParameterEntity.setExtendField(null);
    mCommunicationParameterEntity.setInclusionApplicationPackageName("");
    mCommunicationParameterEntity.setInclusionApplicationVersionCode(null);
    mCommunicationParameterEntity.setLanguage("ja");
    mCommunicationParameterEntity.setLastRecognitionWaitingTime(10000);
    mCommunicationParameterEntity.setPackageNameOfInvokerApplication(null);
    mCommunicationParameterEntity.setSeqRecognition(false);
    mCommunicationParameterEntity.setTerminalType("mobile");
    mCommunicationParameterEntity.setValidateCert(false);
}
```

音声制御ライブラリの初期化を行う。

```
      private void initSebastien (RecognitionMode mode) {
       // outputLog("init - " + getModeString(mode, true) + " Sebastien");
       switch (mode) {
           case SW_ALWAYS_RECOGNITION_MODE:
               mVoiceRecognitionController.init(RecognitionMode.SW_ALWAYS_RECOGNITION_MODE,
mRecognitionParameterEntity, mCommunicationParameterEntity);
               break;
           case VOICE_RECOGNITION_MODE:
               mVoiceRecognitionController.init(RecognitionMode.VOICE_RECOGNITION_MODE,
mRecognitionParameterEntity, mCommunicationParameterEntity);
               break;
           case SEND_TEXT_MODE:
               // DcmLog.specTest(TAG, "H1-28/H2-28", "START");
               mVoiceRecognitionController.init(RecognitionMode.SEND_TEXT_MODE,
mCommunicationParameterEntity);
               break;
           case RECORD_RECOGNITION_MODE:
           case IDLE_MODE:
           default:
               // outputLog("ERROR MODE " + mode);
       }
   }
```

ChatApplication.stop()から SpeakSDK を利用した処理をコメントアウトし、音声制御ライブラリの停止処理を追加する。

```
   /**
    * SDK の停止
    */
   public void stop() {
       clearPlayAfterUtt();
       chat.setStatus(ChatStatus.STOP);
       chat.setSubtitle(R.string.stop);
       /*
       sdk.stop(new EventHandler() {
           @Override
           public void run() {
           }
       });
       */
       SebastienInternalState st =
mVoiceRecognitionController.getControllersState().getSebastienInternalState();
       // UNINITIALIZED, STOPPED, STARTING, STARTED, STOPPING;
       if((st == SebastienInternalState.STARTING) || (st == SebastienInternalState.STARTED)) {
           mVoiceRecognitionController.stop();
       }
   }
```

ChatApplication クラスに音声制御ライブラリのイベントリスナークラスを追加する。

```
    private class VoiceRecognitionControllerEventListenerImpl implements
VoiceRecognitionControllerEventListener {

        private ChatController chat;

        @Override
        public void notifyEvent (EventType eventType, Object eventData) {

            chat = ChatController.getInstance();

            switch (eventType) {
                case INIT_COMPLETE:
                    if(chat.isVoiceMode()){
                        mSebastienParameterEntityStart.setMeta(null);
                        mVoiceRecognitionController.start(mRoamingParameterEntity,
mSebastienParameterEntityStart);
                    } else {
                        mSebastienParameterEntityStart.setMeta(sendTextModeMeta);
                        mVoiceRecognitionController.start(mSebastienParameterEntityStart);
                    }
                    break;
                case NOTIFY_INPUT_MIC:
                    break;
                case NOTIFY_START_RECORD:
                    break;
                case START_COMPLETE:
                    chat.setStatus(ChatStatus.START);
                    if (chat.isVoiceMode()) {
                        chat.setAutoStop();
                        if (AudioAdapter.getInstance().isPlaying()) {
                            ChatApplication.getInstance().mute();
                        } else {
                            chat.setSubtitle(R.string.ready_to_talk);
                        }
                    } else {
                        ChatApplication.getInstance().putMeta(sendTextModeMeta);
                        chat.setWaiting();
                    }
                    break;
                case NOTIFY_DEVICE_STATE_CHANGED:
                    break;
                case NOTIFY_START_SPEECH:
                    if (chat.isVoiceMode()) {
                        chat.clearAutoStop();
                    }
                    handler.post(new Runnable() {
                        @Override
                        public void run() {
```

```
                        audioAdapter.pause();
                        chat.setSubtitle(R.string.playing_voice);
                    }
                });
                break;
            case NOTIFY_RECOGNIZED_DATA:
                ResultRecognitionInfoEntity resultRecognitionInfoEntity = (ResultRecognitionInfoEntity)
eventData;
                List<SpeechRecognitionResultEntity> resultInfoEntity =
resultRecognitionInfoEntity.getResultInfoEntity();
                if (!resultInfoEntity.isEmpty()) {
                    for (SpeechRecognitionResultEntity resultList : resultInfoEntity) {
                        // outputLog(resultList.getText());
                        String s = resultList.getText();
                        MetaData meta = new MetaData(MetaData.MetaDataType.SPEECHREC_RESULT);
                        meta.balloons.add(new Balloon(Balloon.BalloonType.USER_VOICE, s));
                        onMetaOut(meta);
                    }
                }
                break;
            case NOTIFY_RECOGNITION_SERVER:
                break;
            case NOTIFY_END_SPEECH:
                break;
            case NOTIFY_META_DATA:
                final MetaData meta = parser.parse(String.valueOf(eventData));
                if (meta != null) {
                    handler.post(new Runnable() {
                        @Override
                        public void run() {
                            onMetaOut(meta);
                        }
                    });
                }
                break;
            case NOTIFY_TTS_PLAY_START:
                if (chat.isVoiceMode()) {
                    chat.clearAutoStop();
                }
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        audioAdapter.pause();
                        chat.setSubtitle(R.string.playing_voice);
                    }
                });
                break;
            case NOTIFY_TTS_PLAY_END:
                // 合成音声再生後のエージェント切り替え
```

```
                AgentType agentType = switchAfterUtt.get();
                if (agentType != null) {
                    switchAfterUtt.set(null);
                    onSwitchAgent(agentType);
                }
                // 合成音声再生後の postback 送信
                Postback postback = postBackAfterUtt.get();
                if (postback != null) {
                    postBackAfterUtt.set(null);
                    putPostback(postback.payload, postback.clientData);
                }
                // 合成音声再生後のメディア再生
                Balloon balloon = playAfterUtt.poll();
                if (balloon != null && playAfterUtt.isEmpty()) {
                    audioAdapter.playAfterUtt(balloon);
                } else if (!audioAdapter.isPlaying()) {
                    chat.setSubtitle(R.string.ready_to_talk);
                    initSebastien(RecognitionMode.VOICE_RECOGNITION_MODE);
                }
                break;
            case STOP_COMPLETE:
                break;
            case DESTROY_COMPLETE:
                break;
            case NOTIFY_SOUND_LEVEL:
                break;
            case NOTIFY_DETECT_KEYWORD:
                break;
            case NOTIFY_SERVER_CONNECTED:
                break;
            case NOTIFY_NOT_DETECT_KEYWORD:
                break;
            default:
                break;
        }
    }

    @Override
    public void notifyError (ErrorInfoEntity errorInfo) {
        // outputLog("ERROR Type = " + errorInfo.getErrorType() + ", Code = " + errorInfo.getErrorCode() +
", Detail = " + errorInfo.getErrorDetail() + "¥n¥n¥n");
        final int WEB_SOCKET_ERR_FROM = 1000;
        final int WEB_SOCKET_ERR_TO = 5000;
        final int TOKEN_EXPIRED = 40102;

        // 音声対話中に WebSocket エラーが発生した場合は自動接続を行う
        int errCode = errorInfo.getErrorCode();
        boolean isWebSocketErr = WEB_SOCKET_ERR_FROM <= errCode && errCode <= WEB_SOCKET_ERR_TO;
        if (chat.isVoiceMode() && isWebSocketErr && chat.setAutoStart()) {
```

```
            return;
        }
        chat.setStatus(ChatStatus.STOP);
        chat.setSubtitle(R.string.stop);
        if (chat.isVoiceMode()) {
            chat.stopVoice();
        } else {
            chat.stopText();
        }
        if (errCode == TOKEN_EXPIRED) {
            DeviceTokenHelper.update(null);
        } else {
            Alert.newInstance(chat.getString(R.string.failed),
                    "ErrorCode : " + errorInfo.getErrorCode()
                            + "¥nErrortype : " + errorInfo.getErrorType()
                            + "¥nErrorDetail : " + errorInfo.getErrorDetail())
                    .show(chat.getFragmentManager(), null);
        }
    }
}
```