

Android データベース処理の便利（ヘルパー）クラスの利用

SQLiteOpenHelper クラス

必ずしもこのヘルパークラスは利用する必要はないのですが、利用すると以下のような利点があります。

このクラスを使うと発生する3つの利点)

- ① 指定したデータベースファイルが存在していないときは自動的にファイル生成を行う事ができる。
- ② DB オープン時、もしテーブルが存在していないときは自動的にテーブル生成を行う事ができる。
- ③ テーブルのバージョンアップが感知でき、既存のテーブルやデータの更新を指示通りにやってくれる。

サンプル)

●新規プロジェクト設定

プロジェクト名 : UseDBHelperAndroid

ターゲット : 1.5

アプリケーション名 : UseDBHelperAndroid

パッケージ名 : db.helper

Create Activity : MainActivity

バージョン : 3

●DB 管理ヘルパークラス (DatabaseHelper.java) を別途定義します。SQLiteOpenHelper を継承します。後は、このヘルパークラスを Activity 系クラス内で生成し利用するだけです。

```
package db.helper;
```

```
import android.content.Context;
```

```
import android.database.sqlite.SQLiteDatabase;
```

```
import android.database.sqlite.SQLiteOpenHelper;
```

```

import android.database.sqlite.SQLiteDatabase.CursorFactory;

public class DatabaseHelper extends SQLiteOpenHelper {

    /* データベース名 */
    private final static String DB_NAME = "androidstudydb";
    /* データベースのバージョン */
    private final static int DB_VER = 1;

    /*
     * バージョンアップした際に呼び出すコンストラクタ
     * 第2引数で、バージョン番号を指定できる。onUpgrade()の引数 newVersion に
渡される
     * 以前のバージョン番号より上がっていると、onUpgrade()が自動的に呼ばれる
     */
    DatabaseHelper(Context context, int version) {
        super(context, DB_NAME, null, version);
    }

    /*
     * 初回テーブル生成時に利用するコンストラクタ
     * バージョン番号は1で設定される
     */
    public DatabaseHelper(Context context) {
        super(context, DB_NAME, null, DB_VER);
    }

    /*
     * onCreate メソッド データベースが作成された時に呼ばれます。
     * テーブルの作成などを行います。
     * 今回は、MyTable という名前のテーブルを1つ作成しています。
     */
    @Override
    public void onCreate(SQLiteDatabase db) {
        //テーブル作成の SQL 文

```

```

        String sql = "";
        sql += "create table MyTable (";
        sql += " No integer primary key autoincrement";
        sql += ",Name text not null";
        sql += ",Tel text";
        sql += ",Age integer";
        sql += ")";

        db.execSQL(sql);//SQL 実行
    }

    /*
     * onUpgrade メソッド onUpgrade()メソッドはデータベースをバージョンアップした時に呼ばれます。
     * 現在のレコードを退避し、テーブルを再作成した後、退避したレコードを戻すなどの処理を行います。
     */
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        // 今回、更新処理内容は決めていないので、未実装
    }
}

```

●main.xml (レイアウトファイル)

```

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:id="@+id/buttonInsert"
        android:text="Insert">
    </Button>

```

```
        <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content"
android:id="@+id/buttonUpdate"
                android:text="Update">
        </Button>

        <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content"
android:id="@+id/buttonDelete"
                android:text="Delete">
        </Button>
        <Button android:layout_width="wrap_content"
                android:layout_height="wrap_content"
android:id="@+id/buttonSelect"
                android:text="Select">
        </Button>
</LinearLayout>
```

●データベースにデータを登録・更新・削除を行う画面系クラスです (MainActivty.java)

```
package db.helper;

import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends Activity {
    private DatabaseHelper dbHelper;
```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    // ボタンに Click リスナーを設定する。
    Button buttonInsert = (Button) this.findViewById(R.id.buttonInsert);
    buttonInsert.setOnClickListener(buttonInsert_ClickListener);
    Button buttonUpdate = (Button)
this.findViewById(R.id.buttonUpdate);
    buttonUpdate.setOnClickListener(buttonUpdate_ClickListener);
    Button buttonDelete = (Button) this.findViewById(R.id.buttonDelete);
    buttonDelete.setOnClickListener(buttonDelete_ClickListener);
    Button buttonSelect = (Button) this.findViewById(R.id.buttonSelect);
    buttonSelect.setOnClickListener(buttonSelect_ClickListener);

    // ここでヘルパークラスを生成
    dbHelper = new DatabaseHelper(this);
}

/* Insert ボタンの Click リスナー。クラス名のない無名内部クラス形式でイベント
処理を定義 */
private OnClickListener buttonInsert_ClickListener = new OnClickListener() {
    public void onClick(View v) {
        buttonInsert_Click(v); // 下記に定義した insert メソッドを呼ぶ
    }
};

/* Update ボタンの Click リスナー */
private OnClickListener buttonUpdate_ClickListener = new OnClickListener()
{
    public void onClick(View v) {
        buttonUpdate_Click(v);
    }
};

/* Delete ボタンの Click リスナー */
private OnClickListener buttonDelete_ClickListener = new OnClickListener() {
    public void onClick(View v) {

```

```

        buttonDelete_Click(v);
    }
};

/* Select ボタンの Click リスナー */
private OnClickListener buttonSelect_ClickListener = new OnClickListener() {
    public void onClick(View v) {
        buttonSelect_Click(v);
    }
};

/*
 * Insert ボタン Click 処理
 */
private void buttonInsert_Click(View v) {
    // 列名と値をペアで管理する ContentValues オブジェクトを利用
    // 名前、電話番号、年齢をセットする
    ContentValues values = new ContentValues();
    // 本来は以下の値をユーザに入力してもらうように作成するのだが。便
    宜的に値を用意
    values.put("Name", "yan");
    values.put("Tel", "0000-1234-5678");
    values.put("Age", 18);

    // 書き込みモードで DB に接続し、オープンする
    // 書き込みモードはデータの排他制御が自動的に行われる
    SQLiteDatabase db = dbHelper.getWritableDatabase();

    // 処理したレコード ID
    long ret;

    // 列名のスペルミス等によりインサートに失敗すると、-1 を返す仕様
    ret = db.insert("MyTable", null, values);

    db.close();// DB 接続を解除

```

```

        if (ret == -1) {
            Toast.makeText(this, "Insert 失敗",
                Toast.LENGTH_SHORT).show();
        } else {
            Toast.makeText(this, "Insert 成功 : " + ret + "件",
                Toast.LENGTH_SHORT).show();
        }
    }

}

/*
 * Update ボタン Click 処理
 */
private void buttonUpdate_Click(View v) {

    ContentValues values = new ContentValues();
    values.put("Age", 24); // set Age=24 に変更
    String whereClause = "No = ?"; // 更新する列名を指定
    String whereArgs[] = new String[1]; // 更新する列の値を指定。
    whereArgs[0] = "1"; // No が 1 の行を更新

    SQLiteDatabase db = dbHelper.getWritableDatabase();
    // 処理したレコード件数
    int ret;

    // update from MyTable set Age=24 where No=1; と同じ
    ret = db.update("MyTable", values, whereClause, whereArgs);

    db.close();

    if (ret == -1) {
        Toast.makeText(this, "Update 失敗",
            Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this, "Update 成功 : " + ret + "件",
            Toast.LENGTH_SHORT)
            .show();
    }
}

```

```

    }
}

/* Delete ボタン Click 処理 */
private void buttonDelete_Click(View v) {
    String whereClause = "No = ?";
    String whereArgs[] = new String[1];
    whereArgs[0] = "1";

    SQLiteDatabase db = dbHelper.getWritableDatabase();
    // 処理したレコード件数
    int ret;

    // delete from MyTable where No=1; と同じ
    ret = db.delete("MyTable", whereClause, whereArgs);

    db.close();

    if (ret == -1) {
        Toast.makeText(this, "Delete 失敗",
Toast.LENGTH_SHORT).show();
    } else {
        Toast.makeText(this, "Delete 成功 : " + ret + "件",
Toast.LENGTH_SHORT)
            .show();
    }
}

/* Select ボタン Click 処理 */
private void buttonSelect_Click(View v) {
    String[] columns = {"Name", "Tel", "Age"}; // 表示する列名を配列で指
定

    String where = "No = ?"; // 検索キーの列名を指定
    String[] param = {"1"}; // 検索キーの値を指定

    // 検索なので読み込み専用で DB を開く

```



```

        SQLiteDatabase db = dbHelper.getReadableDatabase();
        // 検索実行
        Cursor curs = db.query("MyTable", columns, where, param, null, null,
                                "No desc", "10");

        // 検索データ詰め込み用
        StringBuilder selectData = new StringBuilder(0); // 初期サイズは 0 文字
        while (curs.moveToNext()) {
            // 列データの取り出し
            String name = curs.getString(0);
            String tel = curs.getString(1);
            int age = curs.getInt(2);
            selectData.append(name + ":" + tel + ":" + age + "才" + "¥n");
        }

        if (selectData.length() != 0) {
            Toast.makeText(this, selectData,
                           Toast.LENGTH_LONG).show();
        } else { // 初期サイズの 0 文字だったら
            Toast.makeText(this, "検索結果はありませんでした。",
                           Toast.LENGTH_LONG).show();
        }
        db.close();
    }
}

```

■ポイント解説

コードを見てみましょう。例えば、以下のようなクラスを実装します。

```

class SubOpenHelper extends SQLiteOpenHelper{
    //コンストラクタ
    引数：Activity 系オブジェクト
    引数：データベースのファイル名
    引数：整数のバージョン番号

```

```

public SubOpenHelper(Context c,String dbname,int version){
    super(c,dbname,null,version);
}
//コールバックメソッド このクラスが利用されるとき自動的に呼ばれる
public void onCreate(SQLiteDatabase db){
    db.execSQL( "create table -省略-" );
}

//コンストラクタで新しいバージョン番号が指定されると自動的に呼ばれる
public void onUpgrade(
    SQLiteDatabase db,int oldVersion,int newVersion){
    実装例
    //現在のレコードを取得して、一旦メモリへ退避。
    //テーブルの削除
    //新しくテーブルを作り直して、
    //メモリへ退避させたレコードを挿入する etc
}
}

```

コンストラクタ

まずはコンストラクタを見ましょう。

スーパークラスのコンストラクタを呼び出してます。

第一引数は、Context 型インスタンスです。

自身のアプリを示す場合は、getApplicationContext()を使って生成します。

第二引数は、DB ファイル名です。

第四引数は、DB のバージョンで、作成するときに指定できます。

例えば、テーブルの仕様を変えた後であれば、この数値を上げてやる、という使い方が出来ます。

第 3 引数には CursorFactory 型インスタンスを渡すのですが、とりあえず普通に使う分には使わないので null でいいです。(私も良く知らないなので、もし余力があれば調べて、別記事で紹介します。)

onCreate メソッド

このインスタンスを使ってデータベースをオープンする時に、もし指定されたデータベー

スが無い場合に、自動的に実行されるメソッドです。

ここでは、このデータベースで使用するテーブルの **Create** 文の発行を行えばよいでしょう。その他、デフォルトで入れておきたいレコードがあれば挿入してやるなどの処理も行えます。

既にデータベースが存在しているときに、このインスタンスが生成されたとしても実行されません。

onUpgrade メソッド

データベースのバージョンが上がったときに自動的に実行されるメソッドです。

このインスタンスを使ってデータベースをオープンするときに、コンストラクタで渡したバージョンと、実際に存在しているデータベースのバージョンが違うときに呼び出され、古いバージョンの値と、新しいバージョンの値は引数としても渡されてきます。

サンプルコードでは、処理の実体を書いていませんが、例えば、テーブルの仕様を変えたという想定ですと、一旦データを取り出しておいてメモリに持たせておき、今のテーブルを消して、新しくテーブルを作り直して、取り出しておいたレコードをちゃんと挿入しなおす、という処理を行う事ができます。

不必要なら処理を書かなくてもいいですが、定義をしておかないとコンパイルが通りません。

このような形で、**SQLiteOpenHelper** のメソッドをオーバーライド実装しておきます。

データベースのオープン処理

では、次は、最初に行うべきデータベースのオープン処理をみてみましょう。

以下のようなコードになります。

```
SQLiteDatabase sdb;
```

```
SubOpenHelper helper =
```

```
    new SubOpenHelper(this," test.db" ,1);
```

```
    sdb = helper.getWritableDatabase();
```

```
    //もしくは、
```

```
    //sdb = helper.getReadableDatabase();
```

まずは、先ほど実装した **SQLiteOpenHelper** のサブクラスのインスタンスを生成します。

そして、そのインスタンスの、**getWritableDatabase()**メソッドを使って、データベースを

オープンします。

読み取り専用でいいときは、`getReadableDatabase()`でも使用できます。

`getWritableDatabase()`、もしくは `getReadableDatabase()` メソッドの戻り値

`SQLiteDatabase` 型インスタンスを使って、レコードの挿入や削除、検索を行っていきます。

■レコードの追加方法

それではまず、レコード追加時のサンプルコードを見てみましょう。

なお、テーブル定義は、テーブル名は「bookmarklist」で、`_id` 列と `bookmark` 列の 2 列が存在する事とします。

※`_id` 列については、別記事「Android アプリで使用する SQLite のテーブル作成時の注意点」を参照してください。

```
ContentValues values = new ContentValues();
values.put("bookmark", "http://android.roof-balcony.com");
long id = sdb.insert("bookmarklist", null, values);
if (id < 0){
    //処理失敗時の処理を行う。
}
```

レコードの追加は、`SQLiteDatabase.insert()` メソッドを使います。

第一引数はテーブル名です。

第二引数はちょっと飛ばして、第三引数は挿入するレコードの内容を格納した `ContentValues` 型インスタンスを渡します。

列名をキーとして、値を `put()` メソッドで渡します。

今回は一つですが、必要なカラム数分だけ `put()` メソッドを実行しましょう。

第二引数は、API リファレンスの英文をよくみましたが不明です・・・。

`insert()` メソッドの戻り値は、`id` に設定された数値となります。

-1 が返却された場合は、エラーを示します。

エラー時は、警告を出すなど、エラーに必要な処理を行って下さい。

■レコードの更新方法

次は、レコード更新のサンプルコードについてみてみましょう。

```
String id = "1";
ContentValues values = new ContentValues();
values.put("bookmark", "http://freegame.roof-balcony.com");
sdb.update("bookmarklist", values, "_id=?", new String[]{id});
```

レコードの更新は、`SQLiteDatabase.update()`メソッドを使います。

第一引数は、レコードの更新を行うテーブル名を指定します。

第二引数は、レコード追加時と同じく、`ContentValues` 型インスタンスを設定します。

詳しくは先ほどのレコード追加時の説明を見てください。

第三引数は、SQL の `where` 句、つまり更新するレコードの条件にするカラムを指定します。

サンプルコードのように、「?」を付ける事で、この後で指定する第四引数の配列にあたる条件が代入されて SQL が生成されます。

これらの処理を SQL で表現すると、以下のようになります。

```
update bookmarklist
set bookmark = "http://freegame.roof-balcony.com"
where _id = 1
```

`update()`メソッドの戻り値は、実際更新されたレコード数が返却されます。

■レコードの削除方法

次は、レコード削除のサンプルコードについてみてみましょう。

```
String id = "1";
sdb.delete("bookmarklist", "_id = ?", new String[]{id});
```

レコードの更新は、`SQLiteDatabase.delete()`メソッドを使います。

第一引数は、レコードの更新を行うテーブル名を指定します。

第二引数、第三引数は、先ほどの更新(`update`)と同じく、条件を指定します。

`delete()`の戻り値は、削除されたレコード数が戻ります。

これも `update()`同様、API リファレンスには、エラー時の動作については書かれていませ

んでした・・・。

なお、上記の処理を SQL 文にすると、以下のようになります。

```
delete from bookmarklist where _id = 1
```

SQL 文をそのまま実行する方法

これまでのように、レコードの追加は `insert()`、更新は `update()`、削除は `delete()` の各メソッドを使用する方法もありますが、`SQLiteDatabase` クラスには、SQL 文をそのまま実行してくれるメソッドもあります。

それが、`SQLiteDatabase.execSQL()` です。

SQL が得意な方は、この `execSQL()` の方が使いやすいと思います。

例えば、先ほどの例のレコード更新の場合だと、以下のようなコードになります。

```
String sqlstr = "update bookmarklist " +
    " set bookmark = 'http://freegame.roof-balcony.com' " +
    " where _id = 1"
sdb.execSQL(sqlstr);
```

これなら、SQL 文さえ理解しておけば、全て `execSQL()` で実装できます。

更新や削除の際、複雑な条件文を指定するのであれば、SQL 文の状態で行った方が簡単ではないかなと思います。

注意事項ですが、この `execSQL()` は、`select` 文は使えません。

レコードの追加、更新、削除や、テーブルの作成、削除にはこの `execSQL()` が使えます。`select` 文の場合は、`rowQuery()` や `query()` メソッドしか使えません。

■ レコードの検索

SQL でいう `select` 文、レコードの検索について説明していきます。

一概にはいえませんが、データベースを取り扱う上で、コード的には最も実装する場面が多いのが、この `select` ではないかと思います。

、既に `SQLiteDatabase` 型インスタンスを生成しているものとします。

変数名は、`sdb` とします。

なお、テーブル定義は、テーブル名は「`bookmarklist`」で、`_id` 列と `bookmark` 列の 2 列が存在する事とします。

レコードの検索を行った後、検索結果は、**Cursor** というインスタンスとして返されてきます。

この **Cursor** インスタンスの取得方法は、大きく 2 通りありますので、まずはその方法を紹介します。

SQLiteDatabase.query()の使い方

まずは一つ目の、**SQLiteDatabase.query()**メソッドの使い方を説明します。

```
final String[] columns = new String[]{"_id","bookmark"};
String where = "bookmark like ?";
String param = "%android%";
Cursor c =
    sdb.query( "bookmarklist" ,columns,where,new String[]{param},
               null,null," _id desc" ," 10" );
```

SQLiteDatabase.query()メソッドの第一引数はテーブル名です。

第二引数は、取得する列名(カラム名、フィールド名)の配列を指定します。

第三引数、第四引数は取得するレコードの条件を指定します。

今回は、**bookmark** 列の文字列に、「**android**」という文言が含まれる文字列、という条件にしています。

第五引数は、**group by** 句を指定します。

第六引数は、**Having** 句を指定します。

第七引数は、**order by** 句を指定します。

第八引数は、**limit** 句(取得するレコードの上限数)を指定します。

使わない場合は、**null** を指定します。

今回の処理を **SQL** 文で表現すると、以下ようになります。

```
select _id,bookmark
from bookmarklist
where bookmark like '%android%'
order by _id desc
limit 10
```

SQLiteDatabase.rawQuery()の使い方

それでは次は、`rowQuery()`メソッドの使い方を説明します。
`rowQuery()`メソッドは、文字通り、生のクエリを使用します。

```
String sqlstr = "select _id,bookmark " +  
                " from bookmarklist " +  
                " where bookmark like '%android%' " +  
                " order by _id desc " +  
                " limit 10" ;  
Cursor c = sdb.rowQuery(sqlstr);
```

`SQLiteDatabase.rowQuery()`メソッドの引数は、SQL 文です。
個人的には、先ほどの `query()`メソッドより使いやすいと思うんですが、どうでしょうか。
`query()`メソッドは、どの引数に何のパラメータを与えればよかったんだっけ？といちいち調べないといけなさそうなのですが、`rowQuery()`メソッドは明確です。
それに、`inner join` や `left outer join` 等を使って、複数テーブルを結合させるような SQL だったら、この `rowQuery()`の方がいいでしょう。
Android アプリで使われる程度の SQL は難しいとは思えませんが、生の SQL を実行できる `rowQuery()`の方が、可読性もいいのではないかと思います。

「初めての Android」 P.166～P.177 を見ながら実装せよ。

プロジェクト名 : Event2
アプリケーション名 : Event2
パッケージ : org.example.events
Create Activity : Events
ターゲット : 1.5
バージョン : 3

リソースファイル
res/layout に以下 2 ファイルを作成

●main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <!-- 'list'と'empty'のための組み込み ID に注意 -->
    <ListView
        android:id="@android:id/list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
    <TextView
        android:id="@android:id/empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/empty" />
</LinearLayout>
```

●item.xml （リストアイテムの表示レイアウトを別途定義したもの）

```
<?xml version="1.0" encoding="utf-8"?>

<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="horizontal"
    android:padding="10sp">
    <TextView
        android:id="@+id/rowid"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView
        android:id="@+id/rowidcolon"
        android:layout_width="wrap_content"
```

```

        android:layout_height="wrap_content"
        android:text=": "
        android:layout_toRightOf="@id/rowid" />
<TextView
    android:id="@+id/time"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_toRightOf="@id/rowidcolon" />
<TextView
    android:id="@+id/timecolon"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text=": "
    android:layout_toRightOf="@id/time" />
<TextView
    android:id="@+id/title"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:ellipsize="end"
    android:singleLine="true"
    android:textStyle="italic"
    android:layout_toRightOf="@id/timecolon" />
</RelativeLayout>

```

リソースファイル

res/values に以下 1 ファイルを作成

●strings.xml

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Events</string>
    <string name="empty">No events!</string>
</resources>

```

●Constants.java （定数宣言用インターフェース、BaseColumns はコンテンツプロバイダー使用時に必要だが、今回は使用せず。テキストサンプルの Event3 で利用されている）

```
package org.example.events;

import android.provider.BaseColumns;

public interface Constants extends BaseColumns {
    //テーブル名
    public static final String TABLE_NAME = "events";

    // Events データベースのカラム
    public static final String TIME = "time";//日時
    public static final String TITLE = "title";//イベントのタイトル
}
```

●EventsData.java （データベース ヘルパークラス）

```
package org.example.events;

import static android.provider.BaseColumns._ID;
import static org.example.events.Constants.TABLE_NAME;
import static org.example.events.Constants.TIME;
import static org.example.events.Constants.TITLE;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class EventsData extends SQLiteOpenHelper {
    private static final String DATABASE_NAME = "events.db";
    private static final int DATABASE_VERSION = 1;

    /** Events データベースのためのヘルパーオブジェクトを作る */
    public EventsData(Context ctx) {
        super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
    }
}
```

```

@Override
public void onCreate(SQLiteDatabase db) {
    db.execSQL("CREATE TABLE " + TABLE_NAME + " (" + _ID
        + " INTEGER PRIMARY KEY AUTOINCREMENT, " + TIME
        + " INTEGER," + TITLE + " TEXT NOT NULL);");
}

@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion,
    int newVersion) {
    //仮の更新処理：テーブルの削除と新規作成
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
    onCreate(db);
}
}

```

●Events.java (デフォルトでリストが組み込まれた画面用クラス)

```

package org.example.events;

import static android.provider.BaseColumns._ID;
import static org.example.events.Constants.TABLE_NAME;
import static org.example.events.Constants.TIME;
import static org.example.events.Constants.TITLE;
import android.app.ListActivity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.widget.SimpleCursorAdapter;

// ListActivity クラスを継承
public class Events extends ListActivity {

```

```

//表示対象のテーブルの列名
private static String[] FROM = { _ID, TIME, TITLE, };
//TIME で降順 （ソートのルールで使用）
private static String ORDER_BY = TIME + " DESC";

//表示するテキストビューの id を列挙
private static int[] TO = { R.id.rowid, R.id.time, R.id.title, };

private EventsData events;

@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    //データベースヘルパークラスを利用
    events = new EventsData(this);

    try {
        addEvent("Hello, Android!");//レコード追加
        Cursor cursor = getEvents();//検索結果取得
        showEvents(cursor);//画面にリスト表示
    } finally { //この try finally は安全に確実にクローズするための記述です！
        events.close();
    }
}

//データ追加用メソッド
private void addEvent(String string) {
    // Events データソースに新しいレコードを挿入する。
    // 削除、更新も同様の方法で実行できる
    SQLiteDatabase db = events.getWritableDatabase();
    ContentValues values = new ContentValues();
    values.put(TIME, System.currentTimeMillis());
    values.put(TITLE, string);
}

```

```

        db.insertOrThrow(TABLE_NAME, null, values);
    }

    //検索用メソッド
    private Cursor getEvents() {
        // 管理されたクエリーを実行する。Activity は、クローズの他、
        // 必要な場合は再クエリーを処理する
        SQLiteDatabase db = events.getReadableDatabase();
        Cursor cursor =
            db.query(TABLE_NAME, FROM, null, null, null, null, ORDER_BY);
        startManagingCursor(cursor);
        return cursor;
    }

    //リスト表示メソッド。item.xml を参照して表示形式を決定している
    private void showEvents(Cursor cursor) {
        // データバインド（リスト画面に検索結果を貼り付け）をセットアップする
        SimpleCursorAdapter adapter = new SimpleCursorAdapter(this,
            R.layout.item, cursor, FROM, TO);
        setListAdapter(adapter);
    }
}

```

■ListActivity とは？

ListActivity はリスト表示専用のアクティビティクラスで、アクティビティの内部に ListView オブジェクトを内包しています。

新たに ListView ウィジェットを作成して、アクティビティに配置する必要がありません。内部の ListView オブジェクトに Adapter オブジェクトを指定するには、setListAdapter メソッドを使います。

ListActivity にレイアウトファイルを指定する

ListActivity も、通常のアクティビティのようにレイアウトファイルを指定する事ができますが、レイアウトファイルには守らなければならないきまりがあります。

レイアウトファイルの ListView の ID には、固定の ID 「android:list」を指定します。

また、ListActivity にレイアウトファイルを指定すると、リストの表示項目が無い場合に、リストのかわりにメッセージを自動的に表示させる事ができます。

この、メッセージを表示するウィジェットの ID には、固定の ID 「android:empty」を指定します。

以下にその例を示します。

レイアウトファイル (main.xml)

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:orientation="vertical"
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
```

```
>
```

```
<TextView
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="リスト表示のサンプル"
```

```
/>
```

```
<ListView
```

```
    android:id="@+id/android:list"
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:background="#00FF00"
```

```
/>
```

```
<TextView
```

```
    android:id="@+id/android:empty"
```

```
    android:layout_width="fill_parent"
```

```
    android:layout_height="wrap_content"
```

```
    android:text="データが存在しません。"
```

```
        android:background="#FF0000"  
    />  
</LinearLayout>
```
