

Android Tutorial using Kotlin 第三堂

（3）使用Android內建的SQLite資料庫

[Android Tutorial using Kotlin 第三堂（2）儲存與讀取應用程式資訊 << 前情](#)

Android系統內建「SQLite」資料庫，它是一個開放的小型資料庫，它跟一般商用的大型資料庫有類似的架構與用法，例如MySQL資料庫。應用程式可以建立自己需要的資料庫，在資料庫中使用Android API執行資料的管理和查詢的工作。儲存資料的數量是裝置的儲存空間決定的，所以如果空間足夠的話，應用程式可以儲存比較大量的資料，在需要的時候隨時可以執行資料庫的管理和查詢的工作。

一般商用的大型資料庫，可以提供快速存取與儲存非常大量的資料，也包含網路通訊和複雜的存取權限管理，不過它們都會使用一種共通的語言「SQL」，不同的資料庫產品都可以使用SQL這種資料庫語言，執行資料的管理和查詢的工作。SQLite資料庫雖然是個小型資料庫，不過它跟一般大型資料庫的架構與用法也差不多，同樣可以使用SQL執行需要的工作，Android另外提供許多API，讓開發人員使用API執行資料庫的工作。

這一章會從瞭解應用程式資料庫的需求開始，介紹如何建立資料庫與表格，在應用程式運作的過程中，如何執行資料庫的新增、刪除與查詢的工作。

11-1 設計資料庫表格

在資料庫的技術中，一個資料庫（Database）表示應用程式儲存與管理資料的單位，應用程式可能需要儲存很多不同的資料。例如一個購物網站的資料庫，就需要儲存與管理會員、商品和訂單資料。每一種在資料庫中的資料稱為表格（Table），例如會員表可以儲存所有的會員資料。

SQLite 資料庫的架構也跟一般資料庫的概念類似，所以應用程式需要先建立好需要的資料庫與表格後，才可以執行儲存與管理的工作。建立表格是在Android應用程式中，唯一需要使用SQL執行的工作。其它執行資料庫管理與查詢的工作，Android都提供各種功能的API，使用這些API就不需要瞭解太多SQL這種資料庫語言。

建立資料庫表格使用SQL的「CREATE TABLE」指令，這個指令需要指定表格的名稱，還有這個表格用來儲存每一筆資料的欄位（Column）。這些需要的表格欄位可以對應到主要類別中的欄位變數，不過SQLite資料庫的資料型態只有下面這幾種，使用這些欄位決定表格欄位可以儲存的資料型態：

- INTEGER – 整數，對應Byte、Short、Int 和Long。
- REAL – 小數，對應Float和Double。
- TEXT – 字串，對應String。

在設計表格欄位的時候，需要設定欄位名稱和型態，表格欄位的名稱建議就使用主要類別中的欄位變數名稱。表格欄位的型態對應欄位變數的型態，把它們轉換為SQLite提供的資料型態。通常在表格欄位中還會加入「NOT NULL」的指令，表示這個表格欄位不允許空值，可以避免資料發生問題。

表格的名稱可以使用主要類別的類別名稱，一個SQLite表格建議一定要包含一個可以自動為資料編號的欄位，欄位名稱固定為「_id」，型態為「INTEGER」，後面加上「PRIMARY KEY AUTOINCREMENT」的設定，就可以讓SQLite自動為每一筆資料編號。

儲存在這個欄位。

11-2 建立SQLiteOpenHelper類別

Android 提供許多方便與簡單的資料庫API，可以簡化應用程式處理資料庫的工作。這些API都在「android.database.sqlite」套件，它們可以用來執行資料庫的管理和查詢的工作。在這個套件中的「SQLiteOpenHelper」類別，可以在應用程式中執行建立資料庫與表格的工作，應用程式第一次在裝置執行的時候，由它負責建立應用程式需要的資料庫與表格，後續執行的時候開啟已經建立的資料庫讓應用程式使用。還有應用程式在運作一段時間以後，如果增加或修改功能，資料庫的表格也增加或修改了，它也可以讓應用程式執行資料庫的修改工作，讓新的應用程式可以正常的運作。

接下來設計建立資料庫與表格的類別，在「net.macdidi.atk」套件按滑鼠右鍵，選擇「New -> Kotlin File/Class」，在Name輸入「MyDBHelper」後選擇「OK」。參考下列的內容先完成部份的程式碼：

```
package net.macdidi.atk

import android.content.Context
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteDatabase.CursorFactory
import android.database.sqlite.SQLiteOpenHelper

class MyDBHelper(context: Context, name: String,
                 factory: CursorFactory?, version: Int)
    : SQLiteOpenHelper(context, name, factory, version) {

    override fun onCreate(db: SQLiteDatabase) {
        // 建立應用程式需要的表格
        // 待會再回來完成它
    }

    override fun onUpgrade(db: SQLiteDatabase,
                          oldVersion:
                          Int, newVersion: Int) {

        // 刪除原有的表格
        // 待會再回來完成它

        // 呼叫onCreate建立新版的表格
        onCreate(db)
    }

    companion object {

        // 資料庫名稱
        val DATABASE_NAME = "mydata.db"
        // 資料庫版本，資料結構改變的時候要更改這個數字，通常是加一
        val VERSION = 1

        // 需要資料庫的元件呼叫這個函式，這個函式在一般的應用都不需要修改
        fun getDatabase(context: Context): SQLiteDatabase {
            return MyDBHelper(context, DATABASE_NAME, null, VERSION).writableDatabase
        }
    }
}
```

```

    }

    }

}

```

11-3 資料庫功能類別

在Android應用程式中使用資料庫功能通常會有一種狀況，就是Activity或其它元件的程式碼，會因為加入處理資料庫的工作，碼變得又多、又複雜。一般程式設計的概念，一個元件中的程式碼如果很多的話，在撰寫或修改的時候，都會比較容易出錯。這裡說明的作法，會採用在一般應用程式中執行資料庫工作的設計方式，把執行資料庫工作的部份寫在一個獨立類別。

接下來設計應用程式需要的資料庫功能類別，提供應用程式與資料庫相關功能。在「net.macdidi.atk」套件按滑鼠右鍵，選擇「New -> Kotlin File/Class」，在Name輸入「ItemDAO」後選擇「OK」。參考下列的內容完成這個類別：

```

package net.macdidi.atk

import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import java.util.*

// 資料功能類別
class ItemDAO(context: Context) {

    companion object {
        // 表格名稱
        val TABLE_NAME = "item"

        // 編號表格欄位名稱，固定不變
        val KEY_ID = "_id"

        // 其它表格欄位名稱
        val DATETIME_COLUMN = "datetime"
        val COLOR_COLUMN = "color"
        val TITLE_COLUMN = "title"
        val CONTENT_COLUMN = "content"
        val FILENAME_COLUMN = "filename"
        val RECFILENAME_COLUMN = "recfilename"
        val LATITUDE_COLUMN = "latitude"
        val LONGITUDE_COLUMN = "longitude"
        val LASTMODIFY_COLUMN = "lastmodify"

        // 使用上面宣告的變數建立表格的SQL敘述
        val CREATE_TABLE = "CREATE TABLE " + TABLE_NAME + " (" +
            KEY_ID + " INTEGER PRIMARY KEY AUTOINCREMENT, " +
            DATETIME_COLUMN + " INTEGER NOT NULL, " +

```

```

        COLOR_COLUMN + " INTEGER NOT NULL, " +
        TITLE_COLUMN + " TEXT NOT NULL, " +
        CONTENT_COLUMN + " TEXT NOT NULL, " +
        FILENAME_COLUMN + " TEXT, " +
        RECFILENAME_COLUMN + " TEXT, " +
        LATITUDE_COLUMN + " REAL, " +
        LONGITUDE_COLUMN + " REAL, " +
        LASTMODIFY_COLUMN + " INTEGER)"
    }

    // 資料庫物件
    private val db: SQLiteDatabase = MyDBHelper.getDatabase(context)

    // 讀取所有記事資料
    val all: ArrayList<Item>
    get() {
        val result = ArrayList<Item>()
        val cursor = db.query(
            TABLE_NAME, null, null, null, null, null, null)

        while (cursor.moveToNext()) {
            result.add(getRecord(cursor))
        }

        cursor.close()
        return result
    }

    // 取得資料數量
    val count: Int
    get() {
        var result = 0
        val cursor = db.rawQuery("SELECT COUNT(*) FROM " + TABLE_NAME, null)

        if (cursor.moveToNext()) {
            result = cursor.getInt(0)
        }

        return result
    }

    // 關閉資料庫，一般的應用都不需要修改
    fun close() {
        db.close()
    }

    // 新增參數指定的物件
    fun insert(item: Item): Item {
        // 建立準備新增資料的ContentValues物件
        val cv = ContentValues()

        // 加入ContentValues物件包裝的新增資料
    }

```

```

        itemToContentValues(item, cv)

        // 新增一筆資料並取得編號
        // 第一個參數是表格名稱
        // 第二個參數是沒有指定欄位值的預設值
        // 第三個參數是包裝新增資料的ContentValues物件
        val id = db.insert(TABLE_NAME, null, cv)

        // 設定編號
        item.id = id
        // 回傳結果
        return item
    }

    // 修改參數指定的物件
    fun update(item: Item): Boolean {
        // 建立準備修改資料的ContentValues物件
        val cv = ContentValues()

        // 加入ContentValues物件包裝的修改資料
        itemToContentValues(item, cv)

        // 設定修改資料的條件為編號
        // 格式為「欄位名稱=資料」
        val where = KEY_ID + "=" + item.id

        // 執行修改資料並回傳修改的資料數量是否成功
        return db.update(TABLE_NAME, cv, where, null) > 0
    }

    private fun itemToContentValues(item : Item, cv : ContentValues) {
        // 第一個參數是欄位名稱， 第二個參數是欄位的資料
        cv.put(DATETIME_COLUMN, item.datetime)
        cv.put(COLOR_COLUMN, item.color.parseColor())
        cv.put(TITLE_COLUMN, item.title)
        cv.put(CONTENT_COLUMN, item.content)
        cv.put(FILENAME_COLUMN, item.fileName)
        cv.put(RECFILENAME_COLUMN, item.recFileName)
        cv.put(LATITUDE_COLUMN, item.latitude)
        cv.put(LONGITUDE_COLUMN, item.longitude)
        cv.put(LASTMODIFY_COLUMN, item.lastModify)
    }

    // 刪除參數指定編號的資料
    fun delete(id: Long): Boolean {
        // 設定條件為編號，格式為「欄位名稱=資料」
        val where = KEY_ID + "=" + id
        // 刪除指定編號資料並回傳刪除是否成功
        return db.delete(TABLE_NAME, where, null) > 0
    }

    // 取得指定編號的資料物件

```

```

operator fun get(id: Long): Item? {
    // 準備回傳結果用的物件
    var item: Item? = null
    // 使用編號為查詢條件
    val where = KEY_ID + "=" + id
    // 執行查詢
    val result = db.query(
        TABLE_NAME, null, where, null, null, null, null, null)

    // 如果有查詢結果
    if (result.moveToFirst()) {
        // 讀取包裝一筆資料的物件
        item = getRecord(result)
    }

    // 關閉Cursor物件
    result.close()
    // 回傳結果
    return item
}

// 把Cursor目前的資料包裝為物件
fun getRecord(cursor: Cursor): Item {
    // 準備回傳結果用的物件
    val result = Item()

    result.id = cursor.getLong(0)
    result.datetime = cursor.getLong(1)
    result.color = ItemActivity.getColors(cursor.getInt(2))
    result.title = cursor.getString(3)
    result.content = cursor.getString(4)
    result.fileName = cursor.getString(5)
    result.recFileName = cursor.getString(6)
    result.latitude = cursor.getDouble(7)
    result.longitude = cursor.getDouble(8)
    result.lastModify = cursor.getLong(9)

    // 回傳結果
    return result
}

// 建立範例資料
fun createSampleData() {
    val item = Item(0, Date().time, Colors.RED, "關於Android Tutorial的事情.", "Hello
    val item2 = Item(0, Date().time, Colors.BLUE, "一隻非常可愛的小狗狗!", "她的名字叫「大
    val item3 = Item(0, Date().time, Colors.GREEN, "一首非常好聽的音樂!", "Hello conten
    val item4 = Item(0, Date().time, Colors.ORANGE, "儲存在資料庫的資料", "Hello conten

    insert(item)
    insert(item2)
    insert(item3)
    insert(item4)
}

```

```

    }

}

```

完成資料庫功能類別以後，裡面也宣告了一些SQLiteOpenHelper類別會使用到的資料，開啟「MyDBHelper」類別，完成建立除表格的工作：

```

package net.macdidi.atk

...

class MyDBHelper(context: Context, name: String,
                 factory: CursorFactory?, version: Int)
    : SQLiteOpenHelper(context, name, factory, version) {

    override fun onCreate(db: SQLiteDatabase) {
        // 建立應用程式需要的表格
        db.execSQL(ItemDAO.CREATE_TABLE)
    }

    override fun onUpgrade(db: SQLiteDatabase,
                           oldVersion:
                               Int, newVersion: Int) {

        // 刪除原有的表格
        db.execSQL("DROP TABLE IF EXISTS " + ItemDAO.TABLE_NAME)

        // 呼叫onCreate建立新版的表格
        onCreate(db)
    }

    ...

}

```

11-4 使用資料庫中的記事資料

完成與資料庫相關的類別以後，其它的部份就簡單多了，Activity元件也可以保持比較簡潔的程式架構。開啟在「net.macdidi」套件下的「MainActivity」類別，修改原來自己建立資料的作法，改由資料庫提供記事資料並顯示在畫面。由於所有執行資料的程式碼都寫在「ItemDAO」類別，所以要宣告一個ItemDAO的欄位變數，「onCreate」函式也要執行相關的修改：

```

package net.macdidi.atk

...

```

```

class MainActivity : AppCompatActivity() {

    ...

    // 宣告資料庫功能類別欄位變數
    private val itemDAO : ItemDAO by lazy { ItemDAO(applicationContext) }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        processControllers()

        // 如果資料庫是空的，就建立一些範例資料
        // 這是為了方便測試用的，完成應用程式以後可以拿掉
        if (itemDAO.count == 0) {
            itemDAO.createSampleData()
        }

        // 取得所有記事資料
        items.addAll(itemDAO.all)
        item_list.adapter = itemAdapter
    }

    ...

}

```

完成這個部份的修改以後，執行應用程式，如果畫面上顯示像這樣的畫面，比原來預設的記事資料多了一筆「儲存在資料庫的資料」，就表示目前這些是儲存在資料庫中的記事資料。

接下來需要處理新增與修改的部份，同樣在「MainActivity」類別，找到「onActivityResult」函式，參考下列的內容修改程式

```

package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

    override fun onActivityResult(requestCode: Int,
                                   resultCode: Int,
                                   data: Intent) {
        if (resultCode == Activity.RESULT_OK) {
            val item = data.extras.getSerializable(
                "net.macdidi.atk.Item") as Item

            if (requestCode === 0) {

```



```

        // 新增記事資料到資料庫
        val itemNew : Item = itemDAO.insert(item)
        // 設定記事物件的編號
        item.id = itemNew.id

        items.add(item)
        itemAdapter.notifyDataSetChanged()
    }
    else if (requestCode == 1) {
        val position = data.getIntExtra("position", -1)

        if (position != -1) {
            // 修改資料庫中的記事資料
            itemDAO.update(item)

            items.set(position, item)
            itemAdapter.notifyDataSetChanged()
        }
    }
}

...

}

```

最後是刪除記事資料的部份，同樣在「MainActivity」類別，找到「clickMenuItem」函式，參考下列的內容修改程式碼：

```

package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

    fun clickMenuItem(item: MenuItem) {
        when (item.itemId) {
            ...
            // 刪除
            R.id.delete_item -> {
                if (selectedCount == 0) {
                    return
                }

                val d = AlertDialog.Builder(this)
                val message = getString(R.string.delete_item)
                d.setTitle(R.string.delete)
                    .setMessage(String.format(message, selectedCount))

```

```

        d.setPositiveButton(android.R.string.yes) { dialog, which ->
            var index = itemAdapter.count - 1

            while (index > -1) {
                val item = itemAdapter[index]

                if (item.isSelected) {
                    itemAdapter.remove(item)
                    // 刪除資料庫中的記事資料
                    itemDAO.delete(item.id)
                }

                index--
            }

            itemAdapter.notifyDataSetChanged()
            selectedCount = 0
            processMenu(null)
        }
        d.setNegativeButton(android.R.string.no, null)
        d.show()
    }
}

...
}

```

完成這一章所有的工作了，執行應用程式，試試看新增、修改和刪除記事資料的功能。因為記事資料都保存在資料庫，完成後，關閉應用程式再重新啟動，記事資料還是會顯示在畫面。

相關的檔案都可以在[GitHub](#)瀏覽與下載：

GitHub


<https://github.com/macddi5/Android-Tutorial-Kotlin>

後續 >> Android Tutorial using Kotlin 第四堂 (1) 使用照相機與麥克風

Does Clearly work fine?



Shortcuts: **SHIFT+CTRL+C** to Toggle, **ESC** to Close.

 Give us feedback

Build upon  with Clearly

