

Android Tutorial using Kotlin 第五堂

（1）廣播接收元件 - BroadcastReceiver 與 AlarmManager

[Android Tutorial using Kotlin 第四堂（3）讀取裝置目前的位置 – Google Services Location << 前情](#)

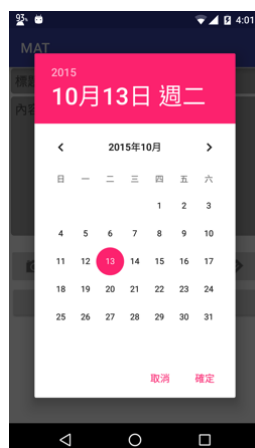
Android系統有一種特別的「廣播事件」，它可以在系統或其它應用程式發生一些事件的時候，通知需要的應用程式執行一些工作。例如裝置在接到來電的時候，系統會發出一個來電的廣播事件，如果應用程式需要在裝置來電的時候執行一些工作，可以設計一個接收來電廣播事件的「廣播接收元件」。

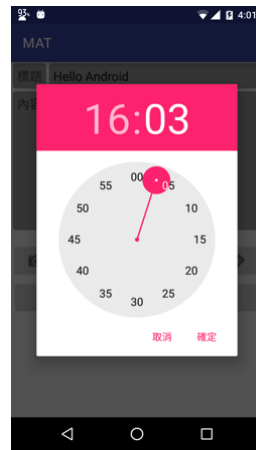
廣播接收元件是一個繼承自「`android.content.BroadcastReceiver`」的子類別，在這個類別中實作接收到廣播事件後需要執行的工作。Android系統在很多不同的情況都會發出廣播事件，你可以依照應用程式的需求，為廣播接收元件設定它要接收與處理哪種廣播事件。

如果需要的話，應用程式也可以發出自己定義的廣播事件，這樣的作法只有在需要與別的應用程式互動的時候，才會執行這動作。這一章會說明發出廣播與設計廣播接收元件的作法，為記事資料加入提醒的功能。選擇一個記事資料以後可以選擇設定提醒功能：



使用者可以依照自己的需求，選擇提醒的日期與時間：





使用者設定的日期與時間到了以後，會使用記事的標題顯示訊息框：



後續的內容會把訊息框改為系統的通知（Notification）。

15-1 發送與接收廣播事件

在一些特別的情況下，應用程式需要發送自己定義廣播事件，裝置中的其它應用程式可以接收與處理這個廣播事件。系統或自己定義的廣播事件，都是使用**Action**名稱來識別它是哪一種廣播事件，所以要為自己定義的廣播事件取一個**Action**名稱，用這個名稱發送廣播事件。呼叫**Activity**元件提供的「**sendBroadcast**」函式可以發送廣播事件，它需要一個設定好**Action**名稱的**Intent**物件，你也可以在**Intent**物件中設定一些資料，這些資料可以傳送給處理的廣播接收元件使用。下面這個程式片段示範自己定義的廣播事件作法：

```
// 發送廣播事件用的Action名稱
public static final String BROADCAST_ACTION =
    "net.macdidi.broadcast01.action.MYBROADCAST01";
...
// 建立準備發送廣播事件的Intent物件
Intent intent = new Intent(BROADCAST_ACTION);
// 如果需要的話，也可以設定資料到Intent物件
intent.putExtra("name", nameValue);
intent.putExtra("age", ageValue);
// 發送廣播事件
sendBroadcast(intent);
```

廣播接收元件是一個繼承自「`android.content.BroadcastReceiver`」的子類別，它的任務是在接收到廣播事件後執行一些工作。元件只需要實作「`onReceive`」函式，在函式中實作接收到指定廣播事件以後需要執行的工作。下面的程式片段示範基本的廣播元件作法：

```
// 繼承自BroadcastReceiver的廣播接收元件
public class MyBroadcastReceiver extends BroadcastReceiver {
    // 接收廣播後執行這個函式
    // 第一個參數Context物件，用來顯示訊息框、啟動服務
    // 第二個參數是發出廣播事件的Intent物件，可以包含資料
    @Override
    public void onReceive(Context context, Intent intent) {
        // 讀取包含在Intent物件中的資料
        String name = intent.getStringExtra("name");
        int age = intent.getIntExtra("age", -1);
        ...
        // 因為這不是Activity元件，需要使用Context物件的時候，
        // 不可以使用「this」，要使用參數提供的Context物件
        Toast.makeText(context, message, Toast.LENGTH_SHORT).show();
    }
}
```

廣播接收元件在設計好以後，一定要在應用程式設定檔中使用「`receiver`」標籤加入設定，在標籤中設定**Action**名稱決定它接哪種廣播事件：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application ... >
        <!-- 使用receiver標籤，名稱設定廣播接收元件類別名稱 -->
        <receiver android:name="MyBroadcastReceiver">
            <intent-filter>
                <!-- 使用Action名稱設定接收的廣播事件 -->
                <action android:name="
                    net.macdidi.broadcast01.action.MYBROADCAST01" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

完成廣播接收元件和需要的設定，應用程式安裝到裝置以後，廣播接收元件就會在等待指定廣播事件的狀態，系統在偵測到廣播事件，就會呼叫這個廣播接收元件的`onReceive`函式。

在應用程式設定檔中執行廣播接收元件的設定，會讓這個廣播接收元件一直在等待接收的狀態。如果應用程式只需要在運作時接收廣播事件，就不要在應用程式設定檔中執行設定，應該在元件中使用程式碼執行註冊與移除廣播接收元件的工作。以**Activity**來說，在`onResume`函式中呼叫「`registerReceiver`」執行註冊的工作。在`onPause`函式中呼叫「`unregisterReceiver`」執行移除。下面這個程式片段示範使用程式碼註冊與移除廣播接收元件的作法：

```

public static final String BROADCAST_ACTION =
    "net.macdidi.broadcast01.action.MYBROADCAST01";

// 建立廣播接收元件物件
MyBroadcastReceiver receiver = new MyBroadcastReceiver();
...
@Override
protected void onResume() {
    super.onResume();
    // 準備註冊與移除廣播接收元件的IntentFilter物件
    IntentFilter filter = new IntentFilter(Intent.ACTION_TIME_TICK);
    // 註冊廣播接收元件
    registerReceiver(receiver, filter);
}

@Override
protected void onPause() {
    // 移除廣播接收元件
    unregisterReceiver(receiver);
    super.onPause();
}

```

15-2 系統廣播事件

廣播接收元件主要的應用是接收特定的系統廣播事件，可以在系統發出廣播的時候執行一些需要的工作。Android系統規劃很多的系統廣播事件，也都為它們取好Action名稱，這些名稱都分類宣告在Android API。這是一些宣告在不同類別或套件下的廣播

- **android.content.Intent** – 主要的系統廣播事件都宣告在這個類別，例如裝置開機完成的Action名稱變數是「ACTION_BOOT_COMPLETED」，實際的名稱是「android.intent.action.BOOT_COMPLETED」。
- **android.bluetooth** – 與藍牙設備相關的廣播事件。例如這個套件下的「BluetoothAdapter」類別，宣告接收藍牙設備狀態的廣播事件變數「ACTION_STATE_CHANGED」。
- **android.hardware.Camera** – 與相機設備相關的廣播事件。例如使用相機錄製影片與拍攝照片的廣播事件名稱，變數名稱「ACTION_NEW_VIDEO」與「ACTION_NEW_PICTURE」。
- **android.net.wifi.WifiManager** – 與Wifi網路設備相關的廣播事件。例如「NETWORK_STATE_CHANGED_ACTION」是網路狀態廣播事件的變數名稱。
- **android.media.AudioManager** – 與裝置音效相關的廣播事件。例如裝置音效模式改變的變數名稱「RINGER_MODE_CHANGED_ACTION」。

主要的系統廣播事件宣告在Intent類別中，變數的名稱都是以「ACTION」開始，這些是比較常見的廣播事件：

- **ACTION_BOOT_COMPLETED** – 系統裝置完成開機的工作後發送的廣播事件，如果要接收這個廣播事件，要加入「RECEIVE_BOOT_COMPLETED」的授權設定到應用程式設定檔中。
- **ACTION_TIME_TICK** – 系統固定每一分鐘發送一次這個廣播事件。
- **ACTION_DATE_CHANGED**、**ACTION_TIME_CHANGED** – 改變系統的日期與時間的時候發送的廣播事件。

如果應用程式需要在系統開機完成後執行一些特定的工作，使用在Intent類別宣告的「ACTION_BOOT_COMPLETED」廣播事件，它的Action名稱是「android.intent.action.BOOT_COMPLETED」。像這類在固定情況下送出的系統廣播事件，應該在應用程式設定中執行註冊的工作。下面這個程式片段示範接收系統開機完成事件的廣播元件作法：

```
//繼承自BroadcastReceiver的廣播接收元件
public class BootCompletedReceiver extends BroadcastReceiver {
    // 接收廣播後執行這個函式
    // 第一個參數Context物件，用來顯示訊息框、啟動服務
    // 第二個參數是發出廣播事件的Intent物件，可以包含資料
    @Override
    public void onReceive(Context context, Intent intent) {
        // 執行廣播元件的工作
    }
}
```

在應用程式設定檔使用「receiver」標籤加入設定，在標籤中設定Action名稱的時候，要使用廣播事件實際的Action名稱，它們都可以在Android API文件中查詢。下面這個片段示範在應用程式設定檔中執行設定的作法：

```
<?xml version="1.0" encoding="utf-8"?>
<manifest ... >
    <application ... >
        <!-- 使用receiver標籤，名稱設定廣播接收元件類別名稱 -->
        <receiver android:name="BootCompletedReceiver">
            <intent-filter>
                <!-- 設定系統啟動完成的Action名稱 -->
                <action android:name=
                    "android.intent.action.BOOT_COMPLETED" />
            </intent-filter>
        </receiver>
    </application>
</manifest>
```

15-3 修改記事類別與資料庫

瞭解廣播事件與廣播接收元件基本的概念以後，就可以為記事應用程式加入提醒的功能。目前因為記事沒有儲存提醒日期時間資料，所以需要加入相關的修改，包含記事類別與資料庫。

開啟「Item.kt」，依照下列的程式片段加入提醒的日期時間資料：

```
package net.macdidi.atk

import java.util.*

class Item : java.io.Serializable {
```

```

...

// 提醒日期時間
var alarmDatetime : Long = 0

...

}

```

為了讓提醒的日期時間資料也可以儲存在資料庫，開啟「ItemDAO.kt」，參考下列的程式片段，加入新的欄位定義變數與修表格的敘述：

```

package net.macdidi.atk

...

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

    companion object {
        ...

        // 提醒日期時間
        val ALARMDATETIME_COLUMN = "alarmdatetime"

        // 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, " +
            ALARMDATETIME_COLUMN + " INTEGER)"

    }

    ...

}

```

同樣在「ItemDAO.kt」，修改負責在新增與修改記事設定資料的「itemToContentValues」函式：

```

package net.macdidi.atk

...

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

    ...

    private fun itemToContentValues(item : Item, cv : ContentValues) {
        ...

        // 提醒日期時間
        cv.put(ALARMDATETIME_COLUMN, item.alarmDatetime)
    }

    ...

}

```

同樣在「ItemDAO.kt」，修改負責讀取記事資料的「getRecord」函式：

```

package net.macdidi.atk

...

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

    ...

    // 把Cursor目前的資料包裝為物件
    fun getRecord(cursor: Cursor): Item {
        ...

        // 提醒日期時間
        result.alarmDatetime = cursor.getLong(10)

        return result
    }

    ...

}

```

因為已經修改資料表的架構，所以要修改資料庫的版本編號，開啟「MyDBHelper.kt」，參考下面的程式片段，把資料庫的版本修改為2：

```
package net.macdidi.atk

...

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

    ...

    companion object {

        ...

        // 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 = 2

        ...

    }

}
```

15-4 記事鬧鈴提醒功能

完成基本類別與資料庫的修改後，接下來就可以為應用程式加入提醒的操作功能。為了接收系統的提醒廣播事件，依照下列步驟，為應用程式新增一個廣播接收元件：

1. 在「app」目錄上按滑鼠右鍵 -> 選擇「Other」 -> 選擇「Broadcast Receiver」：
2. 在「Class Name」欄位輸入「AlarmReceiver」。
3. 選擇「Finish」。

參考下列的程式片段，在建立好的廣播接收元件類別，修改「onReceive」函式的程式碼：

```
package net.macdidi.atk

import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import android.widget.Toast

class AlarmReceiver : BroadcastReceiver() {
```



```

        override fun onReceive(context: Context, intent: Intent) {
            // 讀取記事標題
            val title = intent.getStringExtra("title")
            // 顯示訊息框
            Toast.makeText(context, title, Toast.LENGTH_LONG).show()
        }
    }
}

```

開啟應用程式設定檔「**AndroidManifest.xml**」，找到**Android Studio**自動加入的廣播接收元件設定，參考下面的片段修改設定

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.macdidi.atk">

    ...

    <application ...>
        ...

        <!-- 提醒廣播接收元件 -->
        <receiver
            android:name=".AlarmReceiver"
            android:enabled="true"
            android:exported="true" />

    </application>

</manifest>

```

開啟「**ItemActivity.kt**」，加入下列設定提醒日期時間的函式宣告：

```

package net.macdidi.atk

...

class ItemActivity : AppCompatActivity() {

    ...

    private fun processSetAlarm() {
        val calendar = Calendar.getInstance()

        if (item.alarmDatetime != 0L) {
            // 設定為已經儲存的提醒日期時間
            calendar.timeInMillis = item.alarmDatetime
        }
    }
}

```

```

    }

    // 讀取年、月、日、時、分
    val yearNow = calendar.get(Calendar.YEAR)
    val monthNow = calendar.get(Calendar.MONTH)
    val dayNow = calendar.get(Calendar.DAY_OF_MONTH)
    val hourNow = calendar.get(Calendar.HOUR_OF_DAY)
    val minuteNow = calendar.get(Calendar.MINUTE)

    // 儲存設定的提醒日期時間
    val alarm = Calendar.getInstance()

    // 設定提醒時間
    val timeSetListener = TimePickerDialog.OnTimeSetListener {
        _, hourOfDay, minute ->
        alarm.set(Calendar.HOUR_OF_DAY, hourOfDay);
        alarm.set(Calendar.MINUTE, minute);

        item.alarmDatetime = alarm.getTimeInMillis()
    }

    // 選擇時間對話框
    val tpd = TimePickerDialog(this, timeSetListener, hourNow, minuteNow, true)

    val dateSetListener = DatePickerDialog.OnDateSetListener {
        _, year, monthOfYear, dayOfMonth ->
        alarm.set(Calendar.YEAR, year);
        alarm.set(Calendar.MONTH, monthOfYear);
        alarm.set(Calendar.DAY_OF_MONTH, dayOfMonth);

        // 繼續選擇提醒時間
        tpd.show()
    }

    val dpd = DatePickerDialog(this, dateSetListener, yearNow, monthNow, dayNow)
    dpd.show()
}

}

```

同樣在「ItemActivity.kt」，在「clickFunction」函式加入啟動設定日期時間功能的敘述：

```

package net.macdidi.atk

...

class ItemActivity : AppCompatActivity() {

    ...

```

```

        fun clickFunction(view: View) {
            when (view.id) {
                ...
                R.id.set_alarm -> {
                    // 設定提醒日期時間
                    processSetAlarm()
                }
                ...
            }
        }
    }

    ...
}

```

開啟「MainActivity.kt」，找到「onActivityResult」函式，加入使用「AlarmManager」執行提醒功能的程式碼：

```

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

            // 是否修改提醒設定
            var updateAlarm = false

            if (requestCode === 0) {
                val itemNew : Item = itemDAO.insert(item)
                item.id = itemNew.id
                items.add(item)
                itemAdapter.notifyDataSetChanged()

                // 設定為已修改提醒
                updateAlarm = true
            }
            else if (requestCode == 1) {
                val position = data.getIntExtra("position", -1)

                if (position != -1) {

```

```

        // 讀取原來的提醒設定
        val ori = itemDAO[item.id]
        // 判斷是否需要設定提醒
        updateAlarm = item.alarmDatetime != ori?.alarmDatetime

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

// 設定提醒
if (item.alarmDatetime != 0L && updateAlarm) {
    val intent = Intent(this, AlarmReceiver::class.java)
    intent.putExtra("title", item.title)

    val pi = PendingIntent.getBroadcast(
        this, item.id.toInt(),
        intent, PendingIntent.FLAG_ONE_SHOT)

    val am = getSystemService(Context.ALARM_SERVICE) as AlarmManager
    am.set(AlarmManager.RTC_WAKEUP, item.alarmDatetime, pi)
}

}

}

...

}

```

15-5 開機完成廣播事件

依照上面的說明已經完成記事提醒的功能，不過使用「AlarmManager」執行提醒的工作，在Android系統重新開機以後就會失效，所以需要設計接收開機完成的廣播接收元件，重新執行設定提醒的工作。依照下列的步驟，為應用程式新增一個廣播接收元件。

1. 在「app」目錄上按滑鼠右鍵 -> 選擇「Other」 -> 選擇「Broadcast Receiver」：
2. 在「Class Name」欄位輸入「InitAlarmReceiver」。
3. 選擇「Finish」。

參考下列的程式片段，在建立好的廣播接收元件類別，修改「onReceive」函式的程式碼：

```

package net.macdidi.atk

import android.app.AlarmManager
import android.app.PendingIntent

```

```

import android.content.BroadcastReceiver
import android.content.Context
import android.content.Intent
import java.util.*

class InitAlarmReceiver : BroadcastReceiver() {

    override fun onReceive(context: Context, intent: Intent) {
        // 建立資料庫物件
        val itemDAO = ItemDAO(context.applicationContext)
        // 讀取資料庫所有記事資料
        val items = itemDAO.all
        // 讀取目前時間
        val current = Calendar.getInstance().timeInMillis

        val am = context.getSystemService(Context.ALARM_SERVICE) as AlarmManager

        for (item in items) {
            // 如果有設定提醒而且提醒還沒有過期
            if (item.alarmDatetime != 0L && item.alarmDatetime > current) {
                // 設定提醒
                val alarmIntent = Intent(context, AlarmReceiver::class.java)
                alarmIntent.putExtra("title", item.title)
                val pi = PendingIntent.getBroadcast(
                    context, item.id.toInt(),
                    alarmIntent, PendingIntent.FLAG_ONE_SHOT)
                am.set(AlarmManager.RTC_WAKEUP, item.alarmDatetime, pi)
            }
        }
    }
}

```

開啟應用程式的設定檔「**AndroidManifest.xml**」，在「**manifest**」標籤下加入接收開機完成的廣播事件授權與修改元件的設定

```

<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.macdidi.atk">

    ...

    <!-- 接收開機完成廣播事件 -->
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />

    <application ...>
        ...

        <!-- 開機完成廣播接收元件 -->
        <receiver

```

```
        android:name=".InitAlarmReceiver"
        android:enabled="true"
        android:exported="true"
        android:permission="android.permission.RECEIVE_BOOT_COMPLETED">
        <intent-filter>
            <category android:name="android.intent.category.DEFAULT"/>
            <action android:name="android.intent.action.BOOT_COMPLETED" />
        </intent-filter>
    </receiver>

</application>

</manifest>
```

完成這一章所有的工作了，這個部份的功能可以在模擬或實體裝置測試。開啟記事資料，為它設定一分鐘後的提醒，看看是否以正確的顯示訊息框。

相關的檔案都可以在[GitHub](#)瀏覽與下載：

GitHub


<https://github.com/macdidi5/Android-Tutorial-Kotlin>

[後續 >> Android Tutorial using Kotlin 第五堂（2）系統通知服務 – Notification](#)

Does Clearly work fine?



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

 Give us feedback

Build upon  with Clearly

