

Android Tutorial using Kotlin 第三堂

（1）為ListView元件建立自定畫面

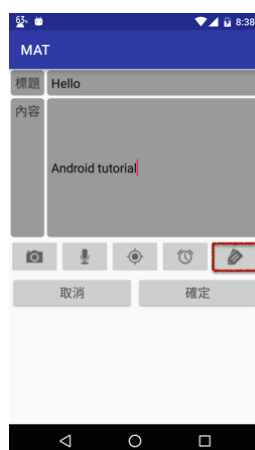
[Android Tutorial using Kotlin 第二堂（4）建立與使用Activity元件 << 前情](#)

ListView是Android應用程式很常使用的畫面元件，它可以顯示多筆資料項目讓使用者瀏覽、選擇與執行後續的操作。目前完成的事應用程式，只有把簡單的文字資料設定給ListView元件使用，其實這個元件有非常多不同的用途，它可以顯示比較複雜的資料，讓使用者勾選和執行後續的功能。

這一章會加強ListView元件的使用，為它設計專用的畫面，讓一個項目可以顯示比較多的資料：

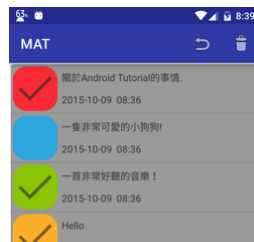


為了讓記事資料可以清楚的分類，所以在新增與修改記事加入設定顏色的功能：





在瀏覽記事資料的主畫面，提供使用者勾選項目的功能，在未選擇與已選擇項目的狀態，需要顯示不同的功能表項目：



如果使用者選擇記事項目，為應用程式加入刪除記事的功能：



9-1 記事資料的封裝

不論是開發一般或Android應用程式，應用程式的功能越寫越多，程式碼也會更複雜，一般的物件封裝作法可以讓程式碼比較些。這個應用程式需要管理所有的記事資料，所以應該為應用程式新增一個封裝記事資料的類別。因為希望可以為每一個記事加入顏色設定的功能，所以先建立一個封裝顏色資料的類別。在「net.macdidi.atk」套件上按滑鼠右鍵，選擇「New -> Kotlin File/class」，在Create New Kotlin File/class對話框的Name輸入「Colors」，Kind選擇「Enum」後選擇「OK」。參考下面的成這個程式碼：

```
package net.macdidi.atk

import android.graphics.Color

enum class Colors private constructor(val code: String) {

    LIGHTGREY("#BDBDBD"), BLUE("#33B5E5"), PURPLE("#AA66CC"),
    GREEN("#99CC00"), ORANGE("#FFBB33"), RED("#FF4444");

    fun parseColor(): Int {
        return Color.parseColor(code)
    }
}
```

在「net.macdidi.atk」套件上按滑鼠右鍵，選擇「New -> Kotlin File/class」，在New Kotlin File/class對話框的Name輸入「Item」後選擇「OK」。參考下面的內容完成這個程式碼：

```
package net.macdidi.atk

import java.util.*

class Item : java.io.Serializable {

    // 編號、日期時間、顏色、標題、內容、照相檔案名稱、錄音檔案名稱、經度、緯度、修改、已選擇
    var id: Long = 0
```

```

var datetime: Long = 0
var color: Colors
var title: String
var content: String
var fileName: String? = null
var recFileName: String? = null
var latitude: Double = 0.toDouble()
var longitude: Double = 0.toDouble()
var lastModify: Long = 0
var isSelected: Boolean = false

// 裝置區域的日期時間
val localeDatetime: String
    get() = String.format(Locale.getDefault(), "%tF %<tR", Date(datetime))

// 裝置區域的日期
val localeDate: String
    get() = String.format(Locale.getDefault(), "%tF", Date(datetime))

// 裝置區域的時間
val localeTime: String
    get() = String.format(Locale.getDefault(), "%tR", Date(datetime))

constructor() {
    title = ""
    content = ""
    color = Colors.LIGHTGREY
}

constructor(id: Long, datetime: Long, color: Colors, title: String,
            content: String, fileName: String, recFileName: String,
            latitude: Double, longitude: Double, lastModify: Long) {
    this.id = id
    this.datetime = datetime
    this.color = color
    this.title = title
    this.content = content
    this.fileName = fileName
    this.recFileName = recFileName
    this.latitude = latitude
    this.longitude = longitude
    this.lastModify = lastModify
}

}

```

為了讓記事資料項目可以使用不同的顏色分類，所以新增一個繪圖資源。在「res/drawable」目錄上按滑鼠右鍵，選擇「New Drawable resource file」。在「File name」輸入「item_drawable」後選擇「OK」。參考下面的內容完成這個繪圖資源：

```

<?xml version="1.0" encoding="utf-8"?>
<shape xmlns:android="http://schemas.android.com/apk/res/android"
    android:shape="rectangle" >

    <corners
        android:topLeftRadius="5sp"
        android:topRightRadius="5sp"
        android:bottomLeftRadius="5sp"
        android:bottomRightRadius="5sp" />

    <solid android:color="#AAAAAA"/>

</shape>

```

為了讓ListView元件的每一個項目可以顯示比較多的資料，你可以為項目建立一個畫面配置檔。這個畫面配置檔需要使用一個圖示（selected_icon.png），用來顯示使用者已經選擇一個項目，你可以在GitHub這一章的範例程式專案找到這個圖檔，把它拖到「res/drawable」目錄。現在準備新增一個給ListView元件項目使用的畫面資源，在「res/layout」目錄上按滑鼠右鍵，選擇「New -> Layout resource file」。在「File name」輸入「single_item」，Root element選擇「LinearLayout」，最後選擇「Create」參考下面的內容完成這個畫面資源：

```

<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >

    <!-- 顏色分類 -->
    <RelativeLayout
        android:id="@+id/type_color"
        android:layout_width="64dp"
        android:layout_height="64dp"
        android:layout_margin="3sp"
        android:background="@drawable/item_drawable" >

        <!-- 勾選 -->
        <ImageView
            android:id="@+id/selected_item"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_centerInParent="true"
            android:src="@drawable/selected_icon"
            android:visibility="invisible" />

    </RelativeLayout>

    <LinearLayout
        android:layout_width="match_parent"

```

```

        android:layout_height="match_parent"
        android:layout_margin="3sp"
        android:gravity="center_vertical"
        android:orientation="vertical" >

        <!-- 標題 -->
        <TextView
            android:id="@+id/title_text"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:gravity="center_vertical" />

        <!-- 日期時間 -->
        <TextView
            android:id="@+id/date_text"
            android:layout_width="match_parent"
            android:layout_height="0dp"
            android:layout_weight="1"
            android:gravity="center_vertical" />

    </LinearLayout>

</LinearLayout>

```

需要在ListView元件中顯示比較複雜的畫面，就不能使用一般的Adapter物件，你可以依照自己的需求，撰寫一個自定的Adapter給ListView元件使用。在「net.macdidi.atk」套件上按滑鼠右鍵，選擇「New -> Kotlin File/Class」，在New Kotlin File/Class對話框的Name輸入「ItemAdapter」後選擇「OK」。參考下面的內容完成這個程式碼：

```

package net.macdidi.atk

import android.content.Context
import android.graphics.drawable.GradientDrawable
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.*

class ItemAdapter(context: Context,
                  private val resource: Int,
                  private val items: MutableList<Item>)
    : ArrayAdapter<Item>(context, resource, items) {

    override fun getView(position: Int,
                          convertView: View?,
                          parent: ViewGroup): View {
        val itemView: LinearLayout
        // 讀取目前位置的記事物件
    }
}

```

```

        val item = getItem(position)

        if (convertView == null) {
            // 建立項目畫面元件
            itemView = LinearLayout(context)
            val inflater = Context.LAYOUT_INFLATER_SERVICE
            val li = context.getSystemService(inflater) as LayoutInflater
            li.inflate(resource, itemView, true)
        } else {
            itemView = convertView as LinearLayout
        }

        // 讀取記事顏色、已選擇、標題與日期時間元件
        val typeColor : RelativeLayout = itemView.findViewById(R.id.type_color)
        val selectedItem : ImageView = itemView.findViewById(R.id.selected_item)
        val titleView : TextView = itemView.findViewById(R.id.title_text)
        val dateView : TextView = itemView.findViewById(R.id.date_text)

        // 設定記事顏色
        val background = typeColor.background as GradientDrawable
        background.setColor(item.color.parseColor())

        // 設定標題與日期時間
        titleView.text = item.title
        dateView.text = item.localeDatetime

        // 設定是否已選擇
        selectedItem.visibility = if (item.isSelected) View.VISIBLE else View.INVISIBLE

        return itemView
    }

    // 設定指定編號の記事資料
    operator fun set(index: Int, item: Item) {
        if (index >= 0 && index < items.size) {
            items[index] = item
            notifyDataSetChanged()
        }
    }

    // 讀取指定編號の記事資料
    operator fun get(index: Int): Item {
        return items[index]
    }
}

```

完成這些程式碼與畫面配置檔以後，就準備好基本的工作了。

9-2 使用自定畫面的ListView元件

為了讓ListView元件使用已經準備好的程式碼與資源，之前已經寫好的主畫面元件，就要執行比較大幅度的修改。開啟「net.macdidi.atk」套件下的「MainActivity.kt」，修改欄位變數的宣告：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    private val item_list : ListView by bind(R.id.item_list)
    private val show_app_name: TextView by bind(R.id.show_app_name)

    // 刪除原來的宣告
    //private val data = ArrayList<String>()
    //private val adapter : ArrayAdapter<String>
    //    by lazy {ArrayAdapter(this, android.R.layout.simple_list_item_1, data)}

    // ListView使用的自定Adapter物件
    private val itemAdapter: ItemAdapter
        by lazy { ItemAdapter(this, R.layout.single_item, items) }
    // 儲存所有記事本的List物件
    private val items: ArrayList<Item> = ArrayList()

    // 選單項目物件
    private lateinit var add_item: MenuItem
    private lateinit var search_item: MenuItem
    private lateinit var revert_item: MenuItem
    private lateinit var delete_item: MenuItem

    // 已選擇項目數量
    private var selectedCount = 0

    ...

}
```

同樣在「MainActivity.kt」，參考下列的說明，修改「onCreate」函式的程式碼：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

}
```



```

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

    processControllers()

    // 刪除原來的程式碼
    //data.add("關於Android Tutorial的事情");
    //data.add("一隻非常可愛的小狗狗!");
    //data.add("一首非常好聽的音樂!");
    //item_list.adapter = adapter

    // 加入範例資料
    items.add(Item(1, Date().getTime(), Colors.RED,
        "關於Android Tutorial的事情.", "Hello content", "", "", 0.0, 0.0, 0))
    items.add(Item(2, Date().getTime(), Colors.BLUE,
        "一隻非常可愛的小狗狗!", "她的名字叫「大熱狗」，又叫\n作「奶嘴」，是一隻非常可愛\n的", 0.0, 0.0, 0))
    items.add(Item(3, Date().getTime(), Colors.GREEN,
        "一首非常好聽的音樂!", "Hello content", "", "", 0.0, 0.0, 0))

    item_list.adapter = itemAdapter
}

...
}

```

執行上面的修改以後，會發現這個程式碼出現一些錯誤，這些錯誤會在「onActivityResult」與「processControllers」這兩個函式，你可以參考下列的作法，先把這兩的函式的所有程式碼加上註解，後面再慢慢修改它們：

```

override fun onActivityResult(requestCode: Int,
                                resultCode: Int,
                                data: Intent) {

    /*
    ...
    */
}

private fun processControllers() {
    /*
    ...
    */
}

```

使用上面介紹的函式處理程式碼以後，錯誤的情況就會消失了，先執行這個應用程式，看看是否可以正常的顯示應用程式畫面。

9-3 新增記事的資料傳送與接收

改用目前的方式處理記事資料以後，新增記事的作法就要執行一些必要的修改。開啟「net.macdidi.atk」套件下的「ItemActivity.kt」，加入這些新的欄位變數宣告：

```
package net.macdidi.atk

...

class ItemActivity : AppCompatActivity() {

    private val title_text : EditText by bind(R.id.title_text)
    private val content_text : EditText by bind(R.id.content_text)

    // 啟動功能用的請求代碼
    enum class ItemAction {
        CAMERA, RECORD, LOCATION, ALARM, COLOR
    }

    // 記事物件
    private var item : Item = Item()

    ...

}
```

同樣在「ItemActivity.kt」，參考下列的說明，修改「onCreate」函式的程式碼：

```
package net.macdidi.atk

...

class ItemActivity : AppCompatActivity() {

    ...

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

        // 如果是修改記事
        if (intent.action == "net.macdidi.atk.EDIT_ITEM") {
            // 接收與設定記事標題
            val titleText = intent.getStringExtra("titleText")
            title_text.setText(titleText)
        }
    }
}
```

```
}  
  
...  
  
}
```

同樣在「ItemActivity.kt」，參考下列的說明，修改「onSubmit」函式的程式碼，調整確認新增記事以後要執行的工作：

```
package net.macdidi.atk  
  
...  
  
class ItemActivity : AppCompatActivity() {  
  
    ...  
  
    // 點擊確定與取消按鈕都會呼叫這個函式  
    fun onSubmit(view: View) {  
        // 確定按鈕  
        if (view.id == R.id.ok_item) {  
            // 讀取使用者輸入的標題與內容  
            val titleText = title_text.text.toString()  
            val contentText = content_text.text.toString()  
  
            // 設定記事物件的標題與內容  
            item.title = titleText  
            item.content = contentText  
  
            // 如果是修改記事  
            if (intent.action == "net.macdidi.atk.EDIT_ITEM") {  
                item.lastModify = Date().time  
            }  
            // 新增記事  
            else {  
                item.datetime = Date().time  
            }  
  
            // 設定回傳的記事物件  
            intent.putExtra("net.macdidi.atk.Item", item)  
            setResult(Activity.RESULT_OK, intent)  
        } else {  
            // 設定回應結果為取消  
            setResult(Activity.RESULT_CANCELED, intent)  
        }  
  
        // 結束  
        finish()  
    }  
}
```

```
...  
  
}
```

回到「net.macdidi.atk」套件下的「MainActivity.kt」，找到「onActivityResult」函式，移除之前加入的註解，參考下列的程式碼，修改新增記事後需要處理的工作。因為修改記事的部分還沒有完成，所以先把它們設定為註解。

```
package net.macdidi.atk  
  
...  
  
class MainActivity : AppCompatActivity() {  
  
    ...  
  
    override fun onActivityResult(requestCode: Int,  
                                   resultCode: Int,  
                                   data: Intent) {  
        // 如果被啟動的Activity元件傳回確定的結果  
        if (resultCode == Activity.RESULT_OK) {  
            // 讀取記事物件  
            val item = data.extras.getSerializable(  
                "net.macdidi.atk.Item") as Item  
  
            // 如果是新增記事  
            if (requestCode === 0) {  
                // 設定記事物件的編號與日期時間  
                item.id = items.size + 1L  
                item.datetime = Date().time  
  
                // 加入新增的記事物件  
                items.add(item)  
  
                // 通知資料改變  
                itemAdapter.notifyDataSetChanged()  
            }  
            // 如果是修改記事  
            /*  
            else if (requestCode == 1) {  
                ...  
            }  
            */  
        }  
    }  
  
    ...  
  
}
```

完成上面的工作以後，執行這個應用程式，測試新增記式資料的功能是否正確。

9-4 修改記事的資料傳送與接收

完成新增記事功能以後，接下來處理工作比較多一些的修改記事功能。開啟「net.macdidi.atk」套件下的「MainActivity.kt」
「processControllers」函式，移除之前加入的註解。在這個函式中找到處理ListView項目長按事件的程式碼，先把它們設定為
解：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

    private fun processControllers() {
        ...
        // 建立選單項目長按監聽物件
        /*
        val itemLongListener = AdapterView.OnItemLongClickListener {
            // position: 使用者選擇的項目編號，第一個是0
            _, _, position, _ ->
            ...
        }

        // 註冊選單項目長按監聽物件
        item_list.onItemLongClickListener = itemLongListener
        */

        ...
    }

    ...

}
```

接下來參考下列的程式碼，修改處理ListView項目點擊事件的程式碼：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...
```

```

private fun processControllers() {
    // 建立選單項目點擊監聽物件
    val itemListener = AdapterView.OnItemClickListener {
        // position: 使用者選擇的項目編號，第一個是0
        _, _, position, _ ->

        // 讀取選擇的記事物件
        val item = itemAdapter.getItem(position)

        val intent = Intent(
            "net.macdidi.atk.EDIT_ITEM")

        // 設定記事編號與記事物件
        intent.putExtra("position", position)
        intent.putExtra("net.macdidi.atk.Item", item)

        startActivityForResult(intent, 1)
    }

    ...
}

...
}

```

你可以注意到在點擊一個記事項目以後，傳送的資料已經修改為**Item**物件，所以修改記事元件也要執行對應的調整。開啟「net.macdidi.atk」套件下的「ItemActivity.kt」，修改「onCreate」函式裡面的程式碼：

```

package net.macdidi.atk

...

class ItemActivity : AppCompatActivity() {

    ...

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

        // 讀取Action名稱
        val action = intent.action

        // 如果是修改記事
        if (action == "net.macdidi.atk.EDIT_ITEM") {
            // 接收記事物件與設定標題、內容
            item = intent.extras.getSerializable(
                "net.macdidi.atk.Item") as Item

```

```

        title_text.setText(item.title)
        content_text.setText(item.content)

        // 根據記事物件的顏色設定畫面的背景顏色
        findViewById<TableLayout>(R.id.item_container)
            .setBackgroundColor(item.color.parseColor())
    }
}

...

}

```

修改記事物件在使用者確認內容以後，回到主畫面元件處理修改後的工作。開啟「net.macdidi.atk」套件下的「MainActivity」，修改「onActivityResult」函式裡面的程式碼：

```

package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

    override fun onActivityResult(requestCode: Int,
                                    resultCode: Int,
                                    data: Intent) {
        // 如果被啟動的Activity元件傳回確定的結果
        if (resultCode == Activity.RESULT_OK) {
            // 讀取記事物件
            val item = data.extras.getSerializable(
                "net.macdidi.atk.Item") as Item

            // 如果是新增記事
            if (requestCode === 0) {
                ...
            }
            // 如果是修改記事
            else if (requestCode == 1) {
                // 讀取記事編號
                val position = data.getIntExtra("position", -1)

                if (position != -1) {
                    // 設定修改的記事物件
                    items.set(position, item)
                    itemAdapter.notifyDataSetChanged()
                }
            }
        }
    }
}

```

```
}  
  
...  
  
}
```

完成修改記事功能的調整工作，執行應用程式，點選一筆記事項目，修改內容並確定以後，看看功能是否正確。

9-5 設定記事顏色

像記事這類應用程式，使用一段時間以後，通常會儲存很多資料，為了讓使用者可以清楚的分類與查詢這些記事資料，所以在應用程式加入顏色分類的功能。使用者在新增或修改記事資料的時候，可以依照自己的需求為它設定一個顏色，為設定顏色的功能新增一個Activity元件，元件的名稱是「ColorActivity」，畫面配置檔的名稱是「activity_color」。在最頂端的「app」目錄按滑鼠右鍵選擇「New -> Activity -> Empty Activity」，元件與畫面配置檔名稱依照上面的規劃。建立元件以後，開啟應用程式設定檔「AndroidManifest.xml」，參考下列的內容，加入對話框樣式的設定：

```
<activity  
    android:name=".ColorActivity"  
    android:theme="@android:style/Theme.Dialog" />
```

選擇顏色功能的畫面設計比較簡單一些，開啟在「res/layout」目錄下的「activity_color.xml」，把它修改為下面的內容：

```
<HorizontalScrollView  
    xmlns:android="http://schemas.android.com/apk/res/android"  
    xmlns:tools="http://schemas.android.com/tools"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:padding="6sp"  
    android:spacing="3sp"  
    tools:context="net.macdidi.atk.ColorActivity">  
  
    <LinearLayout  
        android:id="@+id/color_gallery"  
        android:layout_width="match_parent"  
        android:layout_height="match_parent"  
        android:orientation="horizontal" />  
  
</HorizontalScrollView>
```

開啟在「net.macdidi.atk」套件下的「ColorActivity.kt」，把它修改為下面的內容：


```

package net.macdidi.atk

import android.app.Activity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.LinearLayout

class ColorActivity : Activity() {

    private val color_gallery: LinearLayout by bind(R.id.color_gallery)

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

        val listener = ColorListener()

        for (c in Colors.values()) {
            val button = Button(this)
            button.setId(c.parseColor())
            val layout = LinearLayout.LayoutParams(128, 128)
            layout.setMargins(6, 6, 6, 6)
            button.setLayoutParams(layout)
            button.setBackgroundColor(c.parseColor())

            button.setOnClickListener(listener)

            color_gallery.addView(button)
        }
    }

    private inner class ColorListener : View.OnClickListener {

        override fun onClick(view: View) {
            intent.putExtra("colorId", view.getId())
            setResult(Activity.RESULT_OK, intent)
            finish()
        }
    }
}

```

完成準備工作以後，就可以回到記事元件加入需要的程式碼。開啟在「net.macdidi.atk」套件下的「ItemActivity.kt」，參考[說明加入啟動元件的程式碼：

```

package net.macdidi.atk

```

```

...

class ItemActivity : AppCompatActivity() {

    ...

    fun clickFunction(view: View) {
        when (view.id) {
            R.id.take_picture -> {
            }
            R.id.record_sound -> {
            }
            R.id.set_location -> {
            }
            R.id.set_alarm -> {
            }
            //選擇設定顏色功能
            R.id.select_color -> {
                // 啟動設定顏色的Activity元件
                startActivityForResult(Intent(this, ColorActivity::class.java),
                    ItemAction.COLOR.ordinal)
            }
        }
    }

    ...

}

```

同樣在ItemActivity.kt，參考下列的程式碼，執行選擇顏色後的設定工作：

```

package net.macdidi.atk

...

class ItemActivity : AppCompatActivity() {

    ...

    // 更改參數data的型態為Intent?
    override fun onActivityResult(requestCode: Int,
                                    resultCode: Int,
                                    data: Intent?) {
        if (resultCode == Activity.RESULT_OK) {

            val actionRequest = ItemAction.values()[requestCode]

            when (actionRequest) {
                ItemAction.CAMERA -> {

```

```

    }
    ItemAction.RECORD -> {
    }
    ItemAction.LOCATION -> {
    }
    ItemAction.ALARM -> {
    }
    // 設定顏色
    ItemAction.COLOR -> {
        if (data != null) {
            val colorId = data.getIntExtra(
                "colorId", Colors.LIGHTGREY.parseColor())
            item.color = getColors(colorId)
            // 根據選擇的顏色設定畫面的背景顏色
            findViewById<TableLayout>(R.id.item_container)
                .setBackgroundColor(item.color.parseColor())
        }
    }
}

// 轉換顏色值為Colors型態
private fun getColors(color: Int): Colors {
    var result = Colors.LIGHTGREY

    if (color == Colors.BLUE.parseColor()) {
        result = Colors.BLUE
    } else if (color == Colors.PURPLE.parseColor()) {
        result = Colors.PURPLE
    } else if (color == Colors.GREEN.parseColor()) {
        result = Colors.GREEN
    } else if (color == Colors.ORANGE.parseColor()) {
        result = Colors.ORANGE
    } else if (color == Colors.RED.parseColor()) {
        result = Colors.RED
    }

    return result
}

...
}

```

執行應用程式，在新增或修改記事資料的時候，執行設定顏色的測試。

9-6 選擇記事資料與主功能表

這一章最後的工作是完成讓使用者勾選記事資料、控制主功能表的顯示與刪除記事的功能。開啟在「net.macdidi.atk」套件下「MainActivity.kt」，找到「processControllers」函式，修改記事項目長按事件的程式碼，原來的點擊事件也要執行相關的修因為在使用者勾選事件項目以後，主功能表就要根據選擇的情況調整，所以也增加控制功能表顯示的函式processMenu：

```
package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

    private fun processControllers() {
        val itemListener = AdapterView.OnItemClickListener {
            // position: 使用者選擇的項目編號，第一個是0
            _, _, position, _ ->

            // 讀取選擇的記事物件
            val item = itemAdapter.getItem(position)

            // 如果已經有勾選的項目
            if (selectedCount > 0) {
                // 處理是否顯示已選擇項目
                processMenu(item)
                // 重新設定記事項目
                itemAdapter[position] = item
            } else {
                val intent = Intent(
                    "net.macdidi.atk.EDIT_ITEM")

                // 設定記事編號與記事物件
                intent.putExtra("position", position)
                intent.putExtra("net.macdidi.atk.Item", item)

                startActivityForResult(intent, 1)
            }
        }

        item_list.setOnItemClickListener = itemListener

        val itemLongListener = AdapterView.OnItemLongClickListener {
            // position: 使用者選擇的項目編號，第一個是0
            _, _, position, _ ->
            // 讀取選擇的記事物件
            val item = itemAdapter.getItem(position)
            // 處理是否顯示已選擇項目
            processMenu(item)
            // 重新設定記事項目
            itemAdapter[position] = item
            true
        }
    }
}
```

```

    }

    item_list.onItemLongClickListener = itemLongListener

    ...
}

// 處理是否顯示已選擇項目
private fun processMenu(item: Item?) {
    // 如果需要設定記事項目
    if (item != null) {
        // 設定已勾選的狀態
        item.isSelected = !item.isSelected

        // 計算已勾選數量
        if (item.isSelected) {
            selectedCount++
        } else {
            selectedCount--
        }
    }
}

// 根據選擇的狀況，設定是否顯示選單項目
add_item.setVisible(selectedCount == 0)
search_item.setVisible(selectedCount == 0)
revert_item.setVisible(selectedCount > 0)
delete_item.setVisible(selectedCount > 0)
}

...
}

```

同樣在「MainActivity.kt」，找到「onCreateOptionsMenu」函式，為了控制主功能表的顯示，參考下列的程式碼執行必要的

```

package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

    override fun onCreateOptionsMenu(menu: Menu): Boolean {
        menuInflater.inflate(R.menu.menu_main, menu)

        // 取得選單項目物件
        add_item = menu.findItem(R.id.add_item);
        search_item = menu.findItem(R.id.search_item);
    }
}

```

```

        revert_item = menu.findItem(R.id.revert_item);
        delete_item = menu.findItem(R.id.delete_item);

        // 設定選單項目
        processMenu(null);

        return true
    }

    ...

}

```

開啟「res/values/strings.xml」，加入下列需要的文字資源：

```

<string name="delete">刪除</string>
<string name="delete_item">確定要刪除 %1$d 個項目？</string>

```

開啟「MainActivity.kt」，找到「clickMenuItem」函式，加入取消勾選與刪除記事資料的程式碼：

```

package net.macdidi.atk

...

class MainActivity : AppCompatActivity() {

    ...

    fun clickMenuItem(item: MenuItem) {
        // 判斷該執行什麼工作
        when (item.itemId) {
            R.id.search_item -> {
            }
            // 使用者選擇新增選單項目
            R.id.add_item -> {
                // 使用Action名稱建立啟動另一個Activity元件需要的Intent物件
                val intent = Intent("net.macdidi.atk.ADD_ITEM")
                // 呼叫「startActivityForResult」，第二個參數「0」表示執行新增
                startActivityForResult(intent, 0)
            }
            // 取消所有已勾選的項目
            R.id.revert_item -> {
                for (i in 0 until itemAdapter.count) {
                    val ri = itemAdapter.getItem(i)

                    if (ri.isSelected) {

```

```

        ri.isSelected = false
        itemAdapter[i] = ri
    }
}

selectedCount = 0
processMenu(null)
}
// 刪除
R.id.delete_item -> {
    // 沒有選擇
    if (selectedCount == 0) {
        return
    }

    // 建立與顯示詢問是否刪除的對話框
    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 ->
        // 刪除所有已勾選的項目
        var index = itemAdapter.count - 1

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

            if (item.isSelected) {
                itemAdapter.remove(item)
            }

            index--
        }

        // 通知資料改變
        itemAdapter.notifyDataSetChanged()
        selectedCount = 0
        processMenu(null)
    }
    d.setNegativeButton(android.R.string.no, null)
    d.show()
}

...
}

```

完成這個階段的工作了，執行應用程式，看看加入的功能是不是都可以正常的運作。

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 第三堂（2）儲存與讀取應用程式資訊](#)

Does Clearly work fine?



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

 Give us feedback

Build upon ❤️ with Clearly