

Kotlin 5

111360138 電子三甲 蔣安聖

讀書會

組員：

111360127 林煒哲

111360137 許鎧晏

111360138 蔣安聖

討論時間：2024/12/06 下午 3 點 30

地點：共同科館



詢問 GPT 問題

1. Array<out String>的 out 是做什麼用的？



在 Kotlin 中，`Array<out String>` 中的 `out` 是 型別投影 (type projection) 的一部分，用於聲明泛型型別的協變性 (covariance)。這個機制允許我們將子型別的集合安全地傳遞到父型別的集合中。

協變性 (out) 的作用

1. `out` 關鍵字：

- `out` 表示此泛型型別是協變的，可以安全地將該泛型型別的子類型用作泛型的類型參數。
- 協變的泛型類型只能用於輸出位置（例如作為函數的返回值），不能用於修改或消耗（即作為函數參數）。
- 因此，`Array<out String>` 可以讀取 `String` 或 `String` 的子類型，但不能寫入新的元素。

2. 為什麼使用 `out`：

- `onRequestPermissionsResult` 方法的參數 `permissions` 是系統傳遞的，並且該參數不需要在方法中被修改，因此我們可以安全地使用協變。
- `Array<out String>` 保證了我們只能讀取元素，不能嘗試向 `Array` 添加其他型別的元素，這樣可以確保型別安全。

簡單例子：協變性

假設有一個 `Animal` 類型和它的子類型 `Dog`：

```
kotlin 複製程式碼

open class Animal
class Dog : Animal()

fun processAnimals(animals: Array<out Animal>) {
    // 只能讀取元素，例如打印
    for (animal in animals) {
        println(animal)
    }
    // 不能添加新的元素，會導致編譯錯誤
    // animals[0] = Dog() // Error
}
```

這裡的 `Array<out Animal>` 表示可以接受一個 `Array<Dog>` 或 `Array<Animal>`，但保證只能讀取而不能修改。

在 onRequestPermissionsResult 的使用情境

在 onRequestPermissionsResult 方法中，permissions 是一個 Array<out String>，這是因為：

- 系統傳遞的資料只需要讀取，不應修改。
- Kotlin 使用 out 來保證型別安全，即允許我們將子類型（例如 Array<String>）傳遞到協變型別參數中。

這樣的設計使得程式更加安全和靈活。

補充：in 關鍵字

與 out 相反，in 用於聲明逆變性（contravariance），即泛型型別只用於輸入位置，常用於處理函數參數。

例子：

```
kotlin 複製程式碼  
  
fun acceptAnimals(animals: Array<in Dog>) {  
    animals[0] = Dog() // 可以新增  
    // val animal: Animal = animals[0] // 無法保證類型，會出錯  
}
```

🔊 📄 👍 🔖 ↺

2. 請問 as? 是什麼意思?



在 Kotlin 中，`as?` 是一種安全型別轉換操作符（safe cast operator）。它的功能是嘗試將一個對象轉換為指定的型別，如果轉換成功，則返回該對象的轉換後型別；如果轉換失敗，則返回 `null`，而不會拋出類型轉換異常。

`as?` 的作用

1. 嘗試轉型：

- 如果對象是目標型別，則返回轉型後的對象。
- 如果對象不是目標型別，則返回 `null`。

2. 安全性：

- 使用 `as?` 可以避免因為類型不匹配而導致的運行時異常（`ClassCastException`）。

語法

kotlin

複製程式碼

```
val result = obj as? TargetType
```


- `obj`：需要進行轉型的對象。
- `TargetType`：想要轉型的目標類型。
- `result`：
 - 如果轉型成功，`result` 的型別是 `TargetType`。
 - 如果轉型失敗，`result` 的值是 `null`。

範例

成功的安全型別轉換

```
kotlin


val number: Any = 42
val result = number as? Int
println(result) // 輸出: 42
```

 複製程式碼

失敗的安全型別轉換

```
kotlin

val text: Any = "Hello"
val result = text as? Int
println(result) // 輸出: null
```

 複製程式碼


和 `as` 的區別

`as` 是非安全型別轉換操作符。如果轉型失敗，會拋出 `ClassCastException`：

`as` 的例子

```
kotlin


val text: Any = "Hello"
val result = text as Int // 這裡會拋出 ClassCastException
```

 複製程式碼

`as?` 的對比

```
kotlin

val text: Any = "Hello"
val result = text as? Int // 不會拋出異常，返回 null
println(result) // 輸出: null
```

 複製程式碼

在你的程式中

```
kotlin 複製程式碼  
  
val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as? SupportMapFragment
```

1. `supportFragmentManager.findFragmentById(R.id.map)` :
 - 這個方法返回的是一個 `Fragment` 對象，但它的具體型別是 `Fragment?`。
2. `as? SupportMapFragment` :
 - 嘗試將該 `Fragment` 轉換為 `SupportMapFragment` 型別。
 - 如果這個 `Fragment` 不是 `SupportMapFragment`，則返回 `null` 而不會拋出異常。
3. 為什麼使用 `as?` :
 - 保證程式的安全性。如果 `R.id.map` 對應的不是一個 `SupportMapFragment`，程式不會崩潰，而是返回 `null`，以進一步進行處理。

3. fragment 是一種 layout 還是一個元件？



在 Android 中，`<fragment>` 是一種特殊的元件（Component），它的目的是用來嵌入和管理 Fragment 的視圖和行為，但它在 XML 文件中可以被定義為佈局的一部分。以下是它的雙重性：

1. 元件的角色

- `<fragment>` 是一個元件：
 - 它代表一個 Fragment 的實例。
 - 這個元件不僅負責顯示內容，還負責處理該 Fragment 的生命週期和行為。
 - 通過 `android:name` 屬性指定要加載的 Fragment 類型，例如這裡的 `com.google.android.gms.maps.SupportMapFragment`。
 - Fragment 的功能：
 - 它可以動態添加或移除。
 - 它具有自己的生命週期，並可以與 Activity 一起協作。
-

2. 佈局的角色

- `<fragment>` 是佈局的一部分：
 - 在 XML 中，`<fragment>` 和其他佈局元素（如 `<LinearLayout>`、`<TextView>`）一起使用，可以構成一個完整的畫面。
 - 它的視圖會被嵌入到佈局中，根據 Fragment 的內容來顯示相應的畫面。

結合例子

元件的角度：

```
xml 複製程式碼  
  
<fragment  
    android:id="@+id/map"  
    android:name="com.google.android.gms.maps.SupportMapFragment"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent" />
```

- `android:name` 指定了這個元件對應的 Fragment 類（ `SupportMapFragment` ）。
- 系統會在執行時實例化 `SupportMapFragment` 並將其嵌入到佈局中。

佈局的角度：

- 它的佈局屬性（如 `android:layout_width` 和 `android:layout_height`）會影響它在畫面中的佈局方式。
- 它在佈局中的位置會決定這個 Fragment 的視圖顯示在哪裡。

Fragment 是什麼？

- Fragment 是 Android 的一個模組化 UI 元件。
- 可以看成是 Activity 的一部分，允許開發者將 UI 和邏輯劃分為更小的單位。
- 它既可以靜態地嵌入佈局（像你的範例），也可以動態地通過程式碼添加。

你的範例用途

在你的範例中：

- `<fragment>` 是一個元件，負責嵌入 `SupportMapFragment`，並管理地圖顯示和互動。
- 它也是一個佈局的一部分，可以和其他 UI 元素（如按鈕或文字）共同構成完整的畫面。

簡單來說：它既是元件，也是佈局的一部分，主要取決於它的使用背景和對應的行為。

比較 Kotlin 與 JAVA

onRequestPermissionsResult

```
@Override
public void onRequestPermissionsResult(
    int requestCode,
    @NonNull String[] permission,
    @NonNull int[] grantResults
) {
    super.onRequestPermissionsResult(requestCode, permission, grantResults);
    if(requestCode == 1) {
        for (int result : grantResults)
            if (result != PackageManager.PERMISSION_GRANTED)
                finish();
            else
                initMap();
    }
}
```

Java

```
override fun onRequestPermissionsResult(requestCode: Int, permissions: Array<out String>, grantResults: IntArray) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults)
    if (requestCode == 1) {
        if (grantResults.all { it == PackageManager.PERMISSION_GRANTED }) {
            initMap()
        } else {
            finish()
        }
    }
}
```

Kotlin

兩者基本上只在宣告的地方有不同，我個人認為 Java 的版本比較好看懂。

onCreate

```
@Override
protected void onCreate(Bundle savedInstanceState){
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED)
        ActivityCompat.requestPermissions( activity: this, new String[] {Manifest.permission.ACCESS_FINE_LOCATION}, requestCode: 1);
    else
        initMap();
}
```

Java

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

    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        ActivityCompat.requestPermissions( activity: this, arrayOf(Manifest.permission.ACCESS_FINE_LOCATION), requestCode: 1)
    } else {
        initMap()
    }
}
```

Kotlin

這部分基本上兩者相同，唯一不同的地方在於 Java 使用 String 傳入，而 kotlin 則是使用 arrayOf 函數來實現。

initMap

```
private void initMap(){
    SupportMapFragment mapFragment = (SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map);
    if(mapFragment != null)
        mapFragment.getMapAsync( callback: this);
}
```

Java

```
private fun initMap() {
    val mapFragment = supportFragmentManager.findFragmentById(R.id.map) as? SupportMapFragment
    mapFragment?.getMapAsync( callback: this)
}
```

Kotlin

這部分 Kotlin 就展現了他的長處，大幅化簡重複的段落，但相較 Java 之下會變得比較難閱讀。

onMapReady

```
@Override
public void onMapReady(@NonNull GoogleMap googleMap){
    if(ActivityResultCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)
        != PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION)
        != PackageManager.PERMISSION_GRANTED)
        return;
    googleMap.setMyLocationEnabled(true);

    MarkerOptions markerOptions = new MarkerOptions();
    markerOptions.position(new LatLng( latitude: 25.033611, longitude: 121.565000));
    markerOptions.title("台北101");
    markerOptions.draggable(true);
    googleMap.addMarker(markerOptions);
    markerOptions.position(new LatLng( latitude: 25.047924, longitude: 121.517081));
    markerOptions.title("台北車站");
    markerOptions.draggable(true);
    googleMap.addMarker(markerOptions);

    PolylineOptions polylineOptions = new PolylineOptions();
    polylineOptions.add(new LatLng( latitude: 25.033611, longitude: 121.565000));
    polylineOptions.add(new LatLng( latitude: 25.032728, longitude: 121.564137));
    polylineOptions.add(new LatLng( latitude: 25.047924, longitude: 121.517081));
    polylineOptions.color(Color.BLUE);
    Polyline polyline = googleMap.addPolyline(polylineOptions);
    polyline.setWidth(10);

    googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(new LatLng( latitude: 25.034, longitude: 121.545), zoom: 13));
}
```

Java

```

override fun onMapReady(googleMap: GoogleMap) {
    if (ActivityCompat.checkSelfPermission(context: this, Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
        ActivityCompat.checkSelfPermission(context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED
    ) {
        return
    }
    googleMap.isMyLocationEnabled = true

    // Add markers
    googleMap.addMarker(
        MarkerOptions()
            .position(LatLng( latitude: 25.033611, longitude: 121.565000))
            .title("台北101")
            .draggable(true)
    )
    googleMap.addMarker(
        MarkerOptions()
            .position(LatLng( latitude: 25.047924, longitude: 121.517081))
            .title("台北車站")
            .draggable(true)
    )

    // Add polyline
    val polylineOptions = PolylineOptions()
        .add(LatLng( latitude: 25.033611, longitude: 121.565000))
        .add(LatLng( latitude: 25.032728, longitude: 121.564137))
        .add(LatLng( latitude: 25.047924, longitude: 121.517081))
        .color(Color.BLUE)
        .width(10f)

    googleMap.addPolyline(polylineOptions)

    // Move camera
    googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(LatLng( latitude: 25.034, longitude: 121.545), zoom: 13f))
}

```

Kotlin

在 Add Marker 之前的程式碼基本相同，在其之後的程式碼差異就很大了，我自己覺得這部分 Kotlin 真的寫得比較好閱讀，在程式碼本身的撰寫邏輯之下，使其變得非常具有結構性，一眼就能大致看出該段程式碼內部的關係，我很喜歡 Kotlin 在這種地方的發揮。

心得

在這次的 Lab 中，我們學到要如何使用 google Map 的 API，我覺得這個章節挺重要的，尤其是上課的時候，不是因為 google Map 重要，是因為連接 API 非常重要，一個大程式不可能全部都是自己寫的，光是程式本身就會需要多位開發人員協同開發，可想而知，一款 APP 不可能完全不用其他服務，因此學習使用 API 是非常有必要的，使用 API 串結各種服務能將 APP 的功能再往上提升一個檔次。

業師在授課的期間有說到一點我覺得很重要的概念，就是 API 與 SDK，業師說一個公司製作出一個好的服務後，其他人向該公司申請，請求提供一個接口，讓其他人可以製作 APP 連接到該服務，因此就產生了 API 接口以及 SDK，公司會將 SDK 給予開發者，在 SDK 中通常會含有範例程式碼以及文檔等等，充分理解這些文件就可以使用 API 了。

GitHub 程式連結與截圖

<https://github.com/DolphinBlast/JavaHW10>

