

讀書會

- 組員：龍品勳、張耀文、時顯文
- 討論時間：2024/12/11 10:00 AM
- 地點：圖書館

照片：



心得報告

- 心得：地圖確實是非常實用的應用軟體工具
- ChatGPT：

1. 請問這段程式碼的意思

```
private fun loadMap() {  
    val mapFragment = supportFragmentManager  
        .findFragmentById(R.id.mapFragment) as SupportMapFragment  
    mapFragment.getMapAsync(this)  
}
```



這段程式碼是用於 Android 的 Kotlin 程式，目的是載入地圖並準備好後續的地圖操作。讓我們逐行解析：

1. `private fun loadMap() {`

- 這是一個 `private` 函數，名稱為 `loadMap`，沒有回傳值。
- `private` 表示這個函數只能在其所在的類別內部使用。

2. `val mapFragment =`

```
supportFragmentManager.findFragmentById(R.id.mapFragment) as  
SupportMapFragment
```

- 這行的目的是透過 `supportFragmentManager` 找到 XML 中定義的一個地圖片段（`Fragment`）。
- `supportFragmentManager` 是用於管理和互動片段的工具。
- `findFragmentById(R.id.mapFragment)` 是用 ID（`R.id.mapFragment`）查找片段。
- `as SupportMapFragment` 表示將查找到的片段轉換為 `SupportMapFragment` 類型。
 - `SupportMapFragment` 是 Google Maps API 提供的一種片段類型，用於顯示地圖。

3. `mapFragment.getMapAsync(this)`

- `getMapAsync` 是 `SupportMapFragment` 的方法，用於非同步地載入地圖。
 - 它需要傳入一個實現了 `OnMapReadyCallback` 介面的對象（這裡是 `this`，即當前類別）。
 - 當地圖載入完成時，`OnMapReadyCallback` 介面的 `onMapReady(GoogleMap map)` 方法會被呼叫，讓開發者可以對地圖進行操作。



總結

這段程式碼的作用是：

1. 從 XML 的佈局檔案中查找一個 `SupportMapFragment`。
2. 使用 `getMapAsync` 非同步地載入地圖。
3. 當地圖載入完成時，觸發回呼方法（`onMapReady`），方便進一步設定地圖。

使用範例

假設有一個地圖的佈局：

xml

複製程式碼

```
<fragment
    android:id="@+id/mapFragment"
    android:name="com.google.android.gms.maps.SupportMapFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

實現 `OnMapReadyCallback` 的類別應包含類似的程式碼：

kotlin

複製程式碼

```
class MapsActivity : AppCompatActivity(), OnMapReadyCallback {

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

        // 呼叫 loadMap
        loadMap()
    }

    private fun loadMap() {
        val mapFragment = supportFragmentManager
            .findFragmentById(R.id.mapFragment) as SupportMapFragment
        mapFragment.getMapAsync(this)
    }

    override fun onMapReady(googleMap: GoogleMap) {
        // 地圖載入完成，這裡可以對地圖進行操作
        googleMap.moveCamera(CameraUpdateFactory.newLatLng(LatLng(37.7749, -122.4194)))
    }
}
```

2. 請問 `ACCESS_FINE_LOCATION` 和 `ACCESS_COARSE_LOCATION` 的區別



在 Android 開發中，`ACCESS_FINE_LOCATION` 和 `ACCESS_COARSE_LOCATION` 是兩種不同的權限，用於存取設備的位置信息。它們的區別在於精度和數據來源：

1. ACCESS_FINE_LOCATION

- 高精度定位。
- 使用 GPS、Wi-Fi、藍牙 和 行動網路 提供的最準確位置數據。
- 適合需要精確地理位置的應用，例如導航、地圖應用等。
- 精度範圍：一般可以達到幾米到幾十米的誤差。
- 權限聲明（在 `AndroidManifest.xml` 中）：

xml

複製程式碼

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
```

- 適用場景：
 - 導航應用（需要即時且準確的位置更新）。
 - 室外活動應用（例如跑步或騎行追蹤）。

2. ACCESS_COARSE_LOCATION

- 低精度定位。
- 使用 Wi-Fi 和 行動網路 提供的大致位置數據，不依賴 GPS。
- 無法提供精確位置，只能得到一個大致的區域範圍。
- 精度範圍：通常在數百米到幾公里之間。
- 權限聲明（在 `AndroidManifest.xml` 中）：

xml

複製程式碼

```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

- 適用場景：
 - 僅需要大概地理位置的應用，例如天氣應用（顯示城市級別天氣）。
 - 不需要過於精確的地理位置資料，減少對電量的影響。

3. 權限的關係

- `ACCESS_FINE_LOCATION` 隱含了 `ACCESS_COARSE_LOCATION`：
 - 如果您請求了 `ACCESS_FINE_LOCATION`，您也可以同時獲得 `ACCESS_COARSE_LOCATION` 的權限。
 - 但反之不成立，只有 `ACCESS_COARSE_LOCATION` 的權限時，您無法獲取高精度的位置。

4. 用法的注意事項

在使用這些權限時，需注意：

- **Android 6.0+**：這些權限是 **危險權限**，必須在執行時請求，且需要用戶同意。
- **請求權限**：

kotlin

複製程式碼

```
ActivityCompat.requestPermissions(  
    this,  
    arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),  
    LOCATION_PERMISSION_REQUEST_CODE  
)
```

- **檢查權限**：

kotlin

複製程式碼

```
if (ContextCompat.checkSelfPermission(  
    this,  
    Manifest.permission.ACCESS_FINE_LOCATION  
) == PackageManager.PERMISSION_GRANTED  
) {  
    // 權限已授予  
}
```

5. 選擇哪種權限

- 如果您的應用需要高精度位置（例如導航或精確的地理標記），應選擇 `ACCESS_FINE_LOCATION`。
- 如果僅需粗略位置（例如區域性服務或天氣更新），選擇 `ACCESS_COARSE_LOCATION` 即可，因為它對設備電量的消耗較低。

希望這能幫助您了解它們的區別與應用場景！

3. 請問 PERMISSION_GRANTED 和 PERMISSION_DENIED 的區別



在 Android 中，`PERMISSION_GRANTED` 和 `PERMISSION_DENIED` 是常數值，用於表示應用是否擁有某個權限。這兩個值主要在檢查和請求權限時使用。

1. `PERMISSION_GRANTED`

- 含義：權限已經被授予。
- 當用戶允許應用使用某項權限時，檢查權限的結果會是 `PERMISSION_GRANTED`。
- 常數值：`0`。
- 使用場景：
 - 通常與 `ContextCompat.checkSelfPermission` 或權限請求結果一起使用，判斷是否可以進行需要權限的操作。
 - 範例：

kotlin

複製程式碼

```
if (ContextCompat.checkSelfPermission(
    this,
    Manifest.permission.ACCESS_FINE_LOCATION
) == PackageManager.PERMISSION_GRANTED
) {
    // 權限已授予，可以執行相關操作
}
```

2. PERMISSION_DENIED

- 含義：權限未被授予。
- 當用戶拒絕授予某項權限時，檢查權限的結果會是 `PERMISSION_DENIED`。
- 常數值：`-1`。
- 使用場景：
 - 用來判斷當前應用是否缺少某項權限。
 - 如果權限未被授予，可以向用戶請求該權限。
 - 範例：

kotlin

複製程式碼

```
if (ContextCompat.checkSelfPermission(
    this,
    Manifest.permission.ACCESS_FINE_LOCATION
) == PackageManager.PERMISSION_DENIED
) {
    // 權限未授予，需要向用戶請求
    ActivityCompat.requestPermissions(
        this,
        arrayOf(Manifest.permission.ACCESS_FINE_LOCATION),
        LOCATION_PERMISSION_REQUEST_CODE
    )
}
```

3. 應用場景

權限檢查

使用 `ContextCompat.checkSelfPermission` 返回 `PERMISSION_GRANTED` 或 `PERMISSION_DENIED`：

kotlin


複製程式碼

```
val permissionStatus = ContextCompat.checkSelfPermission(
    this,
    Manifest.permission.CAMERA
)
if (permissionStatus == PackageManager.PERMISSION_GRANTED) {
    // 權限已授予
} else {
    // 權限未授予，需要請求
}
```

權限請求回調

當您請求權限後，會在 `onRequestPermissionsResult` 回調中檢查權限結果，結果會是 `PERMISSION_GRANTED` 或 `PERMISSION_DENIED`：

kotlin

 複製程式碼

```
override fun onRequestPermissionsResult(  
    requestCode: Int,  
    permissions: Array<out String>,  
    grantResults: IntArray  
) {  
    if (requestCode == LOCATION_PERMISSION_REQUEST_CODE) {  
        if (grantResults.isNotEmpty() && grantResults[0] == PackageManager.PERMISSION_GRANTED  
            // 用戶同意了權限  
        ) else {  
            // 用戶拒絕了權限  
        }  
    }  
}
```

4. 區別總結

狀態	PERMISSION_GRANTED	PERMISSION_DENIED
含義	權限已被授予，應用可以執行相關操作	權限未被授予，應用無法執行相關操作
值	0	-1
處理方式	直接執行需要權限的操作	向用戶請求權限

5. 注意事項

- 當用戶拒絕授予權限後，可以通過 `shouldShowRequestPermissionRationale` 判斷是否需要向用戶展示為什麼需要這個權限的解釋，提升授權成功的概率。
- 如果權限被永久拒絕（用戶選擇「不再詢問」），應引導用戶到系統設置中手動開啟權限。

這樣能清楚區分 `PERMISSION_GRANTED` 和 `PERMISSION_DENIED` 的用法及意義！

- GitHub 程式連結：
- GitHub 的截圖：
-