第 5 章

地图与定位

地图是地理学的第二语言，是展现地理空间数据的核心工具。在 Android Maps SDK 中提供了高效、易用的地图控件（MapView）及三维地图控件（SceneView）。本章仅介绍 MapView 的用法，读者可以类比学习 SceneView，两者的用法较为相近。

在地图控件之上，可以显示经过渲染的地图和图层，方便用户进行浏览、分析和处理。另外，移动 GIS 的突出优势就是绝大多数设备具有定位能力，可以将定位信息实时显示在地图上。

本章围绕地图控件，介绍地图、图层、定位等常见用法，核心知识点如下：

- 地图控件及其交互
- 使用常见的业务图层
- GPS 定位

5.1 地图控件、地图和图层

地图控件（MapView）、地图（ArcGISMap）和图层（Layer）的概念的联系非常紧密，其关系如图 5-1 所示。

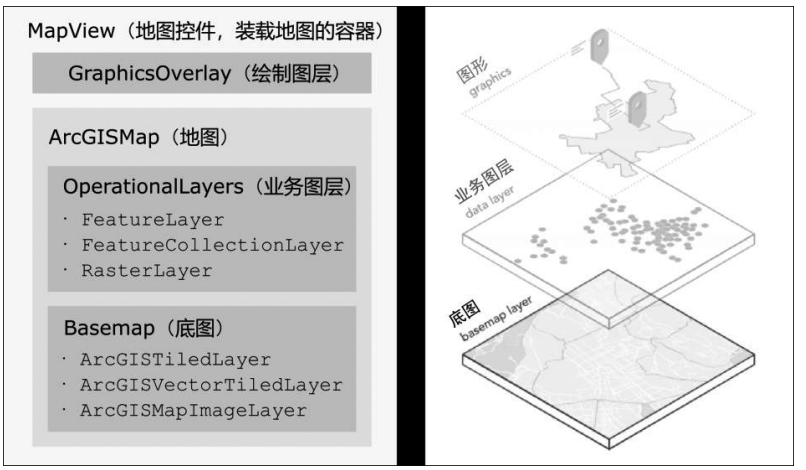

图 5-1　地图控件、地图和图层

地图控件是装载地图的容器。地图控件其本身并不包含数据，但可以容纳各种各样的图层，包括底图、业务图层和绘制图层等。

注意　底图、业务图层和绘制图层都可以认为是一般意义上的图层(Layer)。在 ArcGIS Maps SDK 中，之所以将这些图层分类，是为了方便开发者管理。在早期的 ArcGISRuntimeSDK 中，这些图层都是在同一个图层列表中管理的。

地图包含了底图和业务图层。底图(Basemap)是用于呈现基础地理信息数据的图层，通常展示土地类型、交通、水系、地名地址、POI 等数据，不需要和用户交互且图层较为稳定，不需要经常更新变化。业务图层(Operational Layer)也称为数据图层(Data Layer)，通常是需要和用户交互且可能经常变化的图层。底图和业务图层都是以列表的方式进行管理的，并且这些图层在列表中的顺序就是地图上的叠加顺序。

没有地图和图层的地图控件是没有灵魂的。通常，创建地图控件之前需要将地图和所需要的业务图层提前准备好。创建地图控件的基本流程如下：

(1) 确定底图类型或者创建底图图层。

(2) 创建地图对象。

(3) 确定业务图层列表。

(4) 创建地图控件，并加载地图。

本节介绍地图控件和底图的基本用法。

5.1.1　地图控件

地图控件地属于视图，可以放置在 Activity 的布局中，是展示和操作 GIS 数据的终端入口。与 SceneView 类似，它们都继承于 GeoView 类，并位于 com.arcgismaps.mapping.view 包。MapView 的构造函数如下：

- fun MapView(context: Context)
- fun MapView(context: Context, attrs: AttributeSet?)

通常，MapView 并不是在代码中创建的，而是在 XML 布局文件中声明的，典型代码如下：

```
<com.arcgismaps.mapping.view.MapView
    android:id="@+id/mapview"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

由于地图控件非常重要，所以在本书后面的例子中，将地图控件对象 mapView 定义为成员变量，代码如下：

```
class MainActivity : AppCompatActivity() {
    //地图控件
```

```
        private lateinit var mapView : MapView

        override fun onCreate(savedInstanceState: Bundle?) {
            super.onCreate(savedInstanceState)
            setContentView(R.layout.activity_main)
            …
            //获取 MapView 地图控件
            mapView =findViewById<MapView>(R.id.mapview)
            …
        }
    }
```

由于对象 mapView 无法在 MainActivity 的构造方法中初始化，所以按照之前的解决方案需要为其设置可空类型，即 MapView?，但是，如此一来就会非常麻烦，毕竟 mapView 对象在之后的代码中一定会被初始化，因此，这里使用 lateinit 延迟加载关键字进行设置，实则是向编译器保证该对象一定会被初始化。这是一种惯用用法，通常 Android 的视图对象均会采用 lateinit 关键字。

注意 lateinit 关键字只能用于 var 关键字前，不能用于 val 关键字定义的变量。

本节介绍地图控件的交互、视点、地图位置等控制方式。本节所介绍的内容均可以在 MapView 工程中找到，在该工程中，onCreate 函数包含了 3 个函数调用，分别为用于初始化地图控件的 initMapView 函数，用于监听地图控件变化的 listenMapViewChanges 函数，以及用于初始化基本视图的 initView 函数。后文中的代码基本可以在这 3 个函数中找到。

1. 地图控件交互

地图控件支持常见的手势交互，包括平移、缩放、旋转等，如表 2-3 所示。通过 MapView 对象的 interactionOptions 属性可以设置交互选项，包括以下几种。

(1) isEnabled：Boolean 类型，是否可以交互，默认值为 true。

(2) isPanEnabled：Boolean 类型，是否允许平移地图，默认值为 true。

(3) isZoomEnabled：Boolean 类型，是否允许缩放地图，默认值为 true。

(4) isRotateEnabled：Boolean 类型，是否允许旋转地图，默认值为 true。

(5) zoomFactor：Double 类型，单指双击(双指单击)屏幕缩放比例，默认为 2.0。当该系数大于 1.0 时，单指双击放大屏幕，双指点击缩小屏幕；反之单指双击缩小屏幕，双指点击放大屏幕。当该系数为 1.0 时，单指双击(双指单击)无法缩放屏幕。

(6) isFlingEnabled：Boolean 类型，手指平移屏幕且离开屏幕后，地图是否随平移方向滑动一定的距离，默认值为 true。

(7) snapToNorthThreshold：Double 类型，旋转屏幕时当向上方向偏离正北的角度小于该值时将地图向上方向设置为正北，默认为 7.5。

(8) isMagnifierEnabled：Boolean 类型，是否允许使用放大镜，默认值为 true。

(9) allowMagnifierToPan：Boolean 类型，是否允许使用放大镜平移，默认值为 true。

除了 snapToNorthThreshold、isMagnifierEnabled 和 allowMagnifierToPan 以外，其他选项也支持 SceneView。

例如，允许地图缩放，但是不允许地图平移和旋转，代码如下：

```
//允许地图缩放
mapView.interactionOptions.isZoomEnabled =true
//不允许地图平移
mapView.interactionOptions.isPanEnabled =false
//不允许地图旋转
mapView.interactionOptions.isRotateEnabled =false
```

2. 视点与地图范围

视点(Viewpoint)原本是影视的概念，是指聚焦者感知事件时所处的角度和位置。在地图控件中，表示地图的位置和缩放情况。通过视点的改变，可以改变当前的地图范围(Extent)。

创建视点的构造函数共有 10 个，可以通过经纬度坐标、中心点位置、图形位置等方式创建视点。常用的 Viewpoint 构造函数如下。

(1) fun Viewpoint(latitude：Double，longitude：Double，scale：Double)：通过经度(longitude)、纬度(latitude)和比例尺(scale)创建视点对象。

(2) fun Viewpoint(latitude：Double，longitude：Double，scale：Double，camera：Camera)：通过经纬度、比例尺和相机参数创建视点对象。

(3) fun Viewpoint(center：Point，scale：Double)：通过中心点几何体(center)和比例尺创建视点对象。

(4) fun Viewpoint(center：Point，scale：Double，rotation：Double)：通过中心点几何体(center)、比例尺和旋转角度(rotation)创建视点对象。

创建视点时，需要注意以下几个问题：

(1) 通过经纬度坐标创建视点时，应当使用 WGS84 坐标系的坐标值。

(2) 带有相机(camera)参数的构造函数可以用于场景控件(SceneView)。相机参数指定了经度(longitude)、纬度(latitude)、高度(altitude)、航向角(heading)、俯仰角(pitch)和横滚角(roll)值，其构造函数如下：

```
fun Camera (latitude: Double, longitude: Double, altitude: Double, heading:
Double, pitch: Double, roll: Double)
```

(3) ArcGIS Maps SDK 中，比例尺值(scale)采用数字比例尺的倒数描述。例如，对于 1：100000 比例尺，那么 scale 的值应当为 100000，而不是 1/100000。

定义 Viewpoint 对象后，即可通过地图控件的 setViewpoint 函数设置当前的视点位置，其函数签名如下：

```
fun setViewpoint(viewpoint: Viewpoint)
```

例如，定位到河北师范大学校园中心，将比例尺设置为 1∶3000，代码如下：

```
mapView.setViewpoint(Viewpoint(37.997048, 114.516049, 3000.0))
```

这种设置方法不带有任何动画，通常用于设置地图的初始位置。如果地图已经加载，则可以使用异步方法进行定位，带有地图切换位置的动画，可以提高用户体验，相关函数的签名如下：

```
suspend fun setViewpointAnimated(viewpoint: Viewpoint): Result<Boolean>
suspend fun setViewpointAnimated(viewpoint: Viewpoint, durationSeconds:
Float): Result<Boolean>
suspend fun setViewpointAnimated(viewpoint: Viewpoint, durationSeconds: Float,
curve: AnimationCurve): Result<Boolean>
```

如果没有现成的 Viewpoint 对象，则可以通过 setViewpointCenter 和 setViewpointGeometry 函数定位到中心点或者指定几何体。

注意 通过 setViewpointRotation 和 setViewpointScale 函数可以异步改变地图视点的旋转角度和比例尺。

例如，定位到北京城区的代码如下：

```
mapView.setViewpointAnimated(Viewpoint(39.915599, 116.402257, 500000.0))
```

由于该函数为挂起函数，所以需要异步使用。此时，使用 Lifecycle 模块提供的 lifecycleScope 创建协程作用域，并在其中调用上述函数，代码如下：

```
//定位到北京城区
lifecycleScope.launch {
    mapView.setViewpointAnimated(
                Viewpoint(39.915599, 116.402257, 500000.0))
}
```

执行上述代码，地图控件就可以以顺滑的动画过渡到指定的视点。

3. 监听视点变化

通过 MapView 的 isNavigating 属性可以判断当前地图的视点是否正在发生变化（包括用户互动操作或通过上述代码改变 Viewpoint），但是这种方式只能单次判断，不能实时监听。MapView 类中定义了 SharedFlow 类型的 navigationChanged 属性。这是一种热流类型，可以通过其 collect 函数监听值的变化。另外，该技术也用到了 Kotlin 的协程，所以仍然需要 lifecycleScope 创建协程作用域才可以正常使用。

当检测到地图发生变化时，在 Logcat 中会输出相应的日志，代码如下：

```
//监听地图变化
lifecycleScope.launch {
    mapView.navigationChanged.collect { value ->
        Log.d("mapView", "navigationChanged")
    }
}
```

此时，每当用户平移、缩放、旋转地图或者通过 setViewpointAnimated 等异步函数改变地图视点时，都会在 Logcat 中提示 navigationChanged。

那么，如何获取当前的视点呢？开发者可以调用 MapView 的 getCurrentViewpoint 函数获取视点对象，其签名如下：

```
fun getCurrentViewpoint(viewpointType: ViewpointType): Viewpoint?
```

这里的 viewpointType 参数用来指定视点类型，包括中心点和比例尺类型(CenterAndScale)及四至边界类型(BoundingGeometry)。监听地图视点变化，并将相关信息输出至控制台，代码如下：

```
//监听地图变化
lifecycleScope.launch {
  mapView.navigationChanged.collect { value ->
    Log.d("mapView", "navigationChanged")
    //获取中心点和比例尺
    val f =DecimalFormat("0.##")          //保留两位小数
    val vp1 =mapView.getCurrentViewpoint(ViewpointType.CenterAndScale)
    val center =vp1?.targetGeometry as Point
    Log.d("mapView", "坐标 X: ${f.format(center.x)}, Y: "
+"${f.format(center.y)}")
    Log.d("mapView", "比例尺: ${vp1.targetScale.toULong()}")
    //获取边界几何体(四至范围)
    val vp2 =mapView.getCurrentViewpoint(ViewpointType.BoundingGeometry)
    val bounding =vp2?.targetGeometry as Envelope
    Log.d("mapView", "xmin: ${f.format(bounding.xMin)}")
    Log.d("mapView", "xmax: ${f.format(bounding.xMax)}")
    Log.d("mapView", "ymin: ${f.format(bounding.yMin)}")
    Log.d("mapView", "ymax: ${f.format(bounding.yMax)}")
  }
}
```

在上述代码中，通过 DecimalFormat 对象对所有的浮点型数值进行格式化操作，保留其两位小数，以便更易观察其数值。在 onCreate 声明周期函数增加上述代码，编译并运行程序，这样每次用户改变地图视点时都会在 Logcat 中输出类似如下信息：

```
2023-04-05 … D/mapView: navigationChanged
2023-04-05 … D/mapView: 坐标 X: 12747816.66, Y: 4578890.75
```

```
2023-04-05 … D/mapView: 比例尺: 3000
2023-04-05 … D/mapView: xmin: 12747653.38
2023-04-05 … D/mapView: xmax: 12747979.95
2023-04-05 … D/mapView: ymin: 4578576.2
2023-04-05 … D/mapView: ymax: 4579205.3
```

类似地，通过 SharedFlow 类型的 mapScale 属性和 mapRotation 属性，可以监听地图控件的比例尺变化和旋转变化，代码如下：

```
//监听地图缩放
lifecycleScope.launch {
    mapView.mapScale.collect { value ->
        Log.d("mapView", "mapScale : ${value.toULong()}")
    }
}
//监听地图旋转
lifecycleScope.launch {
    mapView.mapRotation.collect { value ->
        Log.d("mapView", "mapRotation : ${value}")
    }
}
```

如果开发者并不需要持续监听，而仅需要当前的比例尺或缩放角度等信息，则可通过 SharedFlow 类型的 value 属性得到当前值，代码如下：

```
//提示视点信息
val info ="是否正在改变视点?${mapView.isNavigating}\n" +
"比例尺 1:${mapView.mapScale.value.toULong()}\n" +
"缩放角度 ${mapView.mapRotation.value}°"
showToast(info)
```

由于获取 value 值不是异步的，所以不需要使用协程技术。函数 showToast 定义了显示气泡信息功能，代码如下：

```
//气泡提示信息
fun showToast(str : String) {
    Toast.makeText(this, str, Toast.LENGTH_SHORT)
        .show()
}
```

在后文中均使用该 showToast 函数弹出气泡信息，不再赘述。执行上述代码，结果如图 5-2 所示。

4. 监听用户交互

除了可以监听地图控件视点以外，还可以通过 onSingleTapConfirmed、onLongPress 等 SharedFlow 类型的属性监听用户交互事件，如表 5-1 所示。

图 5-2　弹出视点信息

表 5-1　监听用户交互事件

属　　性	返 回 类 型	描　　述
onDown	SharedFlow(DownEvent)	用于监听按下事件
onUp	SharedFlow(UpEvent)	用于监听松开事件
onSingleTapConfirmed	SharedFlow(SingleTapConfirmedEvent)	用于监听单击事件
onPan	SharedFlow(PanChangeEvent)	用于监听地图平移事件
onDoubleTap	SharedFlow(DoubleTapEvent)	用于监听双击事件
onLongPress	SharedFlow(LongPressEvent)	用于监听长按事件
onTowPointerTap	SharedFlow(TwoPointerTapEvent)	用于监听双指单击事件

监听双击地图事件,代码如下:

```
lifecycleScope.launch {
    mapView.onDoubleTap.collect { event ->
        showToast("双击位置 X: ${event.mapPoint?.x}, Y: "
                +"${event.mapPoint?.y}")
    }
}
```

在 DoubleTapEvent 事件对象(其他事件对象与此类似)中,可以通过其 mapPoint 属性获取其双击位置。在上述代码中,将双击位置通过气泡弹出以提示用户,效果如图 5-3 所示。

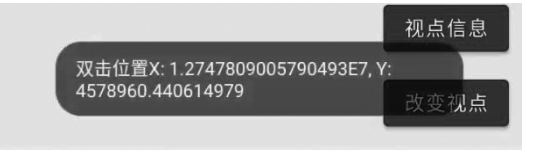

图 5-3　弹出双击位置信息

5. 其他常用属性和函数

下面通过列表的方式介绍地图控件另外一些比较常用的属性和函数,有些会在后文用到,但也有一些因篇幅限制不再详细说明。地图控件的常用属性如表 5-2 所示。

表 5-2 地图控件的常用属性

属 性	类 型	描 述
spatialReference	SpatialReference?	地图控件的空间参考
graphicOverlays	MutableList<GraphicsOverlay>	绘制图层列表
visibleArea	Polygon?	地图可见区域
unitsPerDip	Double	地图中每个像素上的单位长度
wrapAroundMode	WrapAroundMode	地图扩展模式,包括禁用(Disabled)模式和支持时启用(EnabledWhenSupported)模式。当启用时,地图控件坐标系在横向上是没有边界的
backgroundGrid	BackgroundGrid	背景格网,可以通过 BackgroundGrid 对象设置背景网格的颜色、大小等
grid	Grid?	坐标格网,可以设置 Grid 的子类,包括经纬格网(LatitudeLongitudeGrid) UTM 格网(UTMGrid)等

地图控件的常用函数如表 5-3 所示。

表 5-3 地图控件的常用函数

函 数 签 名	描 述
fun locationToScreen(mapPoint: Point): ScreenCoordinate	将地理坐标转换为屏幕坐标
fun screenToLocation(screenCoordinate: ScreenCoordinate): Point?	将屏幕坐标转换为地理坐标
suspend fun exportImage(): Result<BitmapDrawable>	将当前地图输出为二进制图像

地理坐标和屏幕坐标的转换是非常实用的。特别是在空间查询时,通过这些函数可以将用户点选地图的位置和真实的地理坐标进行转换。屏幕坐标类型 ScreenCoordinate 实际上是 DoubleXY 类的别名,包含了 x 和 y 两个 Double 类型的属性。

5.1.2 地图

地图是按照一定空间信息和渲染方式组合地理数据的综合体。完整的地图主要包括以下组成部分。

(1) 图层:经过渲染的并且按照一定顺序排列的图层,可以分为底图图层和业务图层。

(2) 地图属性:包括空间参考、包络矩形、书签、比例尺等。

本节介绍地图类的基本用法,并加载在线图层。

1. 地图

地图(ArcGISMap)类处于 com.arcgismaps.mapping 包中,其构造方法如下:

- fun ArcGISMap()

- fun ArcGISMap(basemap: Basemap)
- fun ArcGISMap(basemapStyle: BasemapStyle)
- fun ArcGISMap(spatialReference: SpatialReference?)
- fun ArcGISMap(uri: String)
- fun ArcGISMap(item: Item)

可以发现，除了可以直接创建空的地图对象以外，还可以通过底图、底图样式、URI 地址、项(Item)、空间参考等方式创建地图对象。

注意　这里的项(Item)是指 LocalItem(本地项)或 PortalItem(云端项)，详情可参考 5.2.2 节的相关内容。

在上文中都是通过初始化底图样式(BasemapStyle)的方式创建地图的，代码如下：

```
mapView.map =ArcGISMap(BasemapStyle.ArcGISTopographic)
```

通过 URI 加载发布在 ArcGIS Online 的地图，代码如下：

```
val url ="https://www.arcgis.com/home/item.html?id=
acc027394bc84c2fb04d1ed317aac674"
mapView.map =ArcGISMap(url)
```

对于 ArcGIS Online，也可以通过创建云端项的方式加载地图，代码如下：

```
/定义 ID
val portalItemId ="acc027394bc84c2fb04d1ed317aac674"
//定义 Portal 地址
val portal =Portal("https://www.arcgis.com")
//创建 PortalItem 对象
val portalItem =PortalItem(portal, portalItemId)
//加载地图
mapView.map =ArcGISMap(portalItem)
```

地图常见的属性如表 5-4 所示。

表 5-4　地图常见的属性

属　性	类　型	描　述
initialViewpoint	Viewpoint?	初始视点。如果没有为地图控件设置视点，则默认采用地图的初始视点
referenceScale	Double	参考比例尺，参考的基准比例尺
minScale	Double	地图最小比例尺，制约地图缩小
maxScale	Double	地图最大比例尺，制约地图放大
maxExtent	Envelope?	最大包络矩形，制约地图缩小