

能力介绍 & API

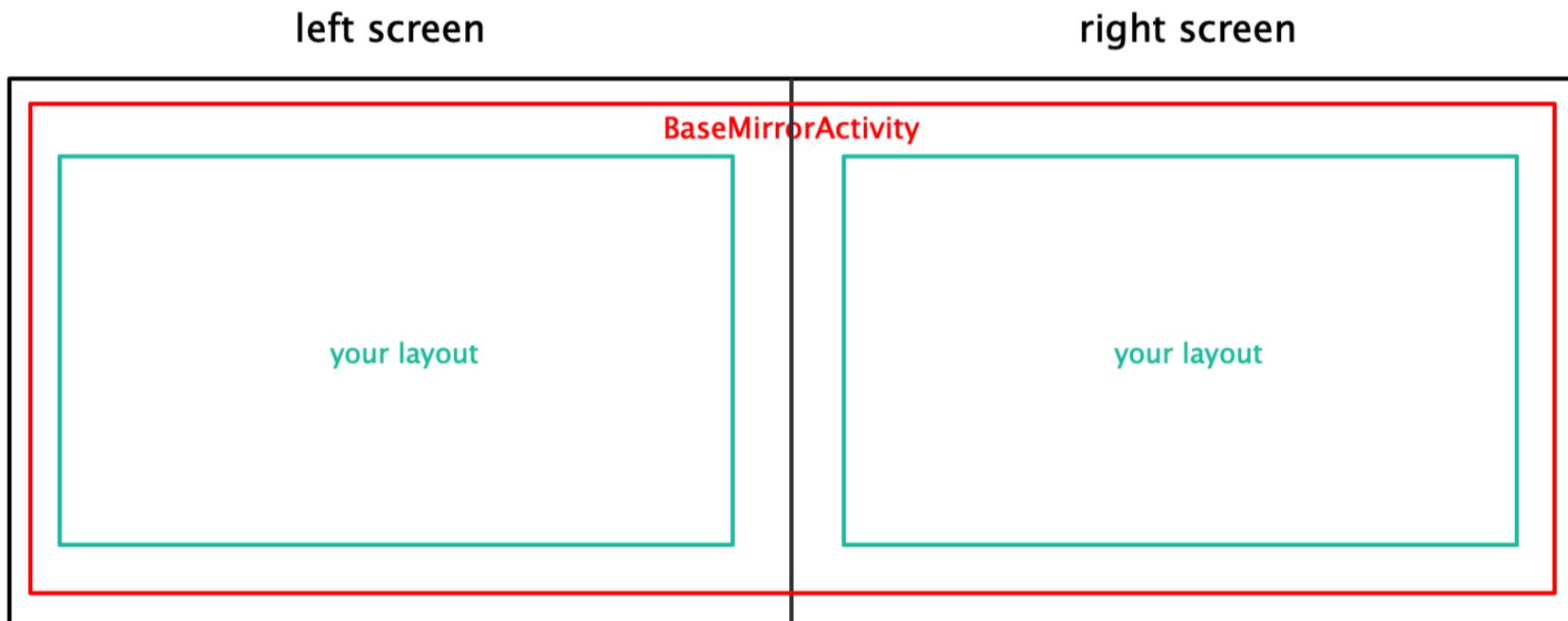
1. 合目

为了实现合目，除了需要相同的UI布局左右各放置一份外，每次对左边的view进行修改，同样的操作也需要在右边的布局也重新设置一遍，这两点都会引入不少繁琐的模版代码。为了解决这个问题，基于ViewBinding，SDK中提供了**BindingPair**工具类，将左右布局对应的ViewBinding对象传入它的构造函数后，通过**updateView**方法同时操作左右的布局，通过**setLeft**方法只操作左边的布局，通过**checkIsLeft**方法，在**updateView**的block中判断当前是左边的布局还是右边的布局。

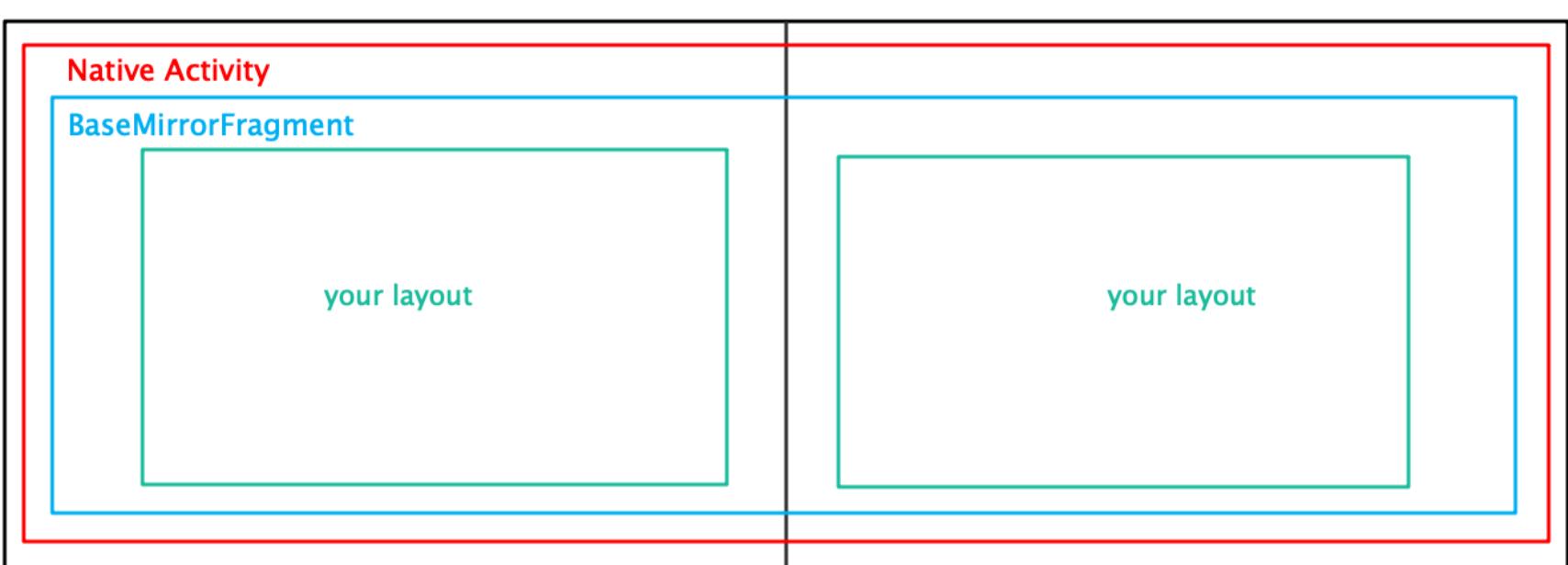
1.1 合目组件

合目镜像操作主要由基于ViewBinding的**BindingPair**工具类完成，基于BindingPair封装了以下UI组件：

- Activity级别：**BaseMirrorActivity**，实现了Activity自动镜像布局视图的逻辑，开发者只用关注业务的UI需求，按照原生的xml布局方式开发布局文件就能实现合目的效果，具体可以参考sample中的FusionVisionActivity或者所有的BaseMirrorActivity子类



- Fragment级别：**BaseMirrorFragment**，实现了Fragment级别的合目封装，添加方式与原生的Fragment一致，但是注意不能添加到BaseMirrorActivity中去,具体实现可参考sample中的FragmentDemoActivity



- View级别：基于组合的**MirrorContainerView** 和基于继承的**BaseMirrorContainerView**

MirrorContainerView的使用可参考**MirrorContainerViewActivity**，注意一定要实现**mirrorContainer.bindTo**方法，否则合目不生效；

BaseMirrorContainerView的使用可参考**com.ffalcon.mercury.android.sdk.demo.ui.wedget.TitleView**,需要注意的是**MirrorView**也同样不能添加到**BaseMirrorActivity**中

- Toast: **FToast**，支持合目的通用toast封装，具体使用可参考**FusionVisionHomeActivity**
- Dialog: **FDIALOG**，支持合目的通用Dialog封装，具体使用可参考**DialogActivity**

上述的合目组件除了**FToast**和**FDIALOG**是独立于页面的组件外，其他的合目组件可以根据业务需求合理的进行组装，以达到完美的双屏合目效果

1.2 updateView & setLeft

mBindingPair.updateView{}:

上述的合目组件均由**BindingPair**工具类实现了布局左右映射的效果，一份布局文件同时映射到左右两个屏幕，所以实质上在内存中还是同时存在两份布局的，为了处理繁琐的镜像操作，SDK提供了**mBindingPair.updateView**方法来同时处理左右布局同一个组件的更新操作。

eg:

代码块

```

1 //在updateView block中的所有操作均会映射到左右两个布局文件中
2 mBindingPair.updateView {
3     tvTitle.text = "my title"
4 }
5 //等价于
6 mBindingPair.left.tvTitle.text = "my title"
7 mBindingPair.right.tvTitle.text = "my title"

```

需要注意的是，由于该特性，推荐的做法是只在**updateView** block中做UI层面的更新操作，**如果在该方法中进行了对外部源数据的修改或者绑定了事件，可以预知的行为是源数据的修改和绑定的事件均会执行两次，造成不可预知的后果！！！**如果有外部数据更新的操作或者对具体组件进行事件绑定的逻辑，推荐的做法是使用**mBindingPair.setLeft**方法

mBindingPair.setLeft{}:

```

1 mBindingPair.setLeft {
2     focusHolder.addFocusTarget(
3         //一般的做法是将事件绑定在左边的视图组件上面
4         FocusInfo(
5             btnEvent,
6             eventHandler = { action -> handleAction(action) },
7             focusChangeHandler = { hasFocus ->
8                 //如果响应某个事件需要同步更新两边的组件，这个时候可以再次调用updateView的方法
9                 mBindingPair.updateView {
10                     triggerFocus(hasFocus, btnEvent, mBindingPair.checkIsLeft(this))
11                 }
12             }
13         )
14     )
15     focusHolder.currentFocus(mBindingPair.left.btnEvent)
16 }

```

当然如果复杂业务需要**updateView**中进行逻辑整合又不期望被调用两次的话，也可以在**updateView**的block中对当前执行区域进行判断，具体的实现如下：

代码块

```

1 mBindingPair.updateView {
2     //do something.....
3
4     //check isLeft
5     if (mBindingPair.checkIsLeft(this)){
6         // only left do something.....
7     }
8 }

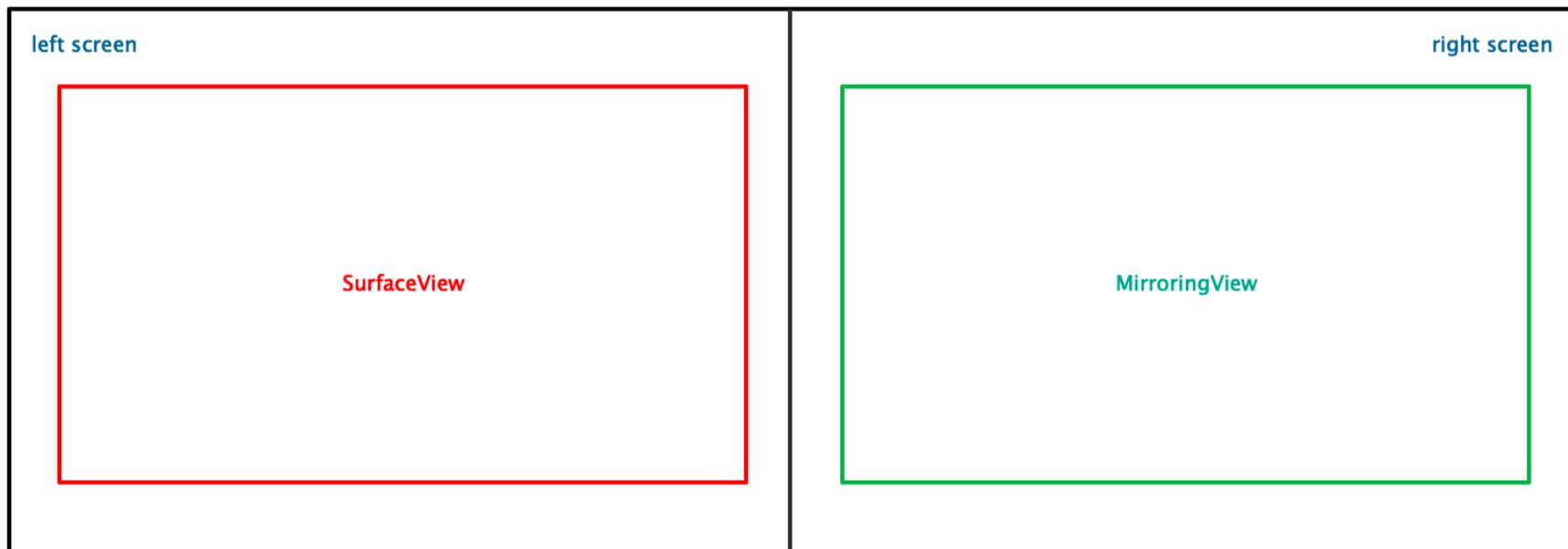
```

更多具体的场景实现可以参考sample中的代码

1.3 视频合目 (SurfaceView)

上文中的合目组件只能对于常规的UI组件进行镜像合目的操作，但对于视频播放需要用到SurfaceView这种类型的场景，还需要进行其他的特殊处理。rayneo对于这种合目场景提供了**MirroringView**组件进行支持，具体使用详情可参考sample中的VideoPlayActivity，主要的步骤如下：

1.确保布局中视频播放组件（SurfaceView）与镜像组件（MirroringView）分别处于不同的屏幕



2.将SurfaceView设置到MirroringView中，并开启Mirroring，此时surface开始渲染的同时也会将画面镜像到MirroringView中去

```

代码块
1     binding.textureView.let { leftTexture ->
2         binding.mirrorView.setSource(leftTexture)
3         binding.mirrorView.startMirroring()
4     }
5
6     .....
7
8     //在必要的地方停止
9     override fun onDestroy() {
10        super.onDestroy()
11        binding.mirrorView.stopMirroring()
12        mPlayer?.release()
13    }

```

2. 焦点管理

2.1 焦点说明

眼镜上的屏幕无法直接触控，无法像手机可以直接选中目标View，所以需要**自己维护焦点**的切换，一般**前滑和后滑手势**用来切换焦点，**单击手势**用来触发焦点View的事件响应，**双击手势**退出焦点返回上一级。

焦点管理最简单的实现，可以通过维护一个索引值index，不同的索引值对应不同的能获取焦点的view，前滑和后滑手势触发焦点位的切换，单击响应对应view的点击事件，双击释放焦点给上一级或退出页面。但对于复杂页面，这样简单粗暴的实现方式，并不容易维护。

代码块

```

1 //1. 构建FocusHolder对象
2 val focusHolder = FocusHolder(true)
3 ....
4 // 2. 以左边布局中的view, 作为focus对象的target,
5 mBindingPair.setLeft {
6     //2.1 构建focusInfo对象
7     val btn1Info = FocusInfo(
8         btn1,
9         eventHandler = { action ->
10            // 处理事件响应
11            when (action) {
12                is TempleAction.Click -> {
13                    FToast.show("bt1 click")
14                }
15                else -> Unit
16            }
17        },
18        focusChangeHandler = { hasFocus ->
19            // 处理焦点切换
20            mBindingPair.updateView {
21                btn1.setBackgroundColor(getColor(if(hasFocus) R.color.purple_200 else R.color.black))
22            }
23        }
24    )
25 }
26 //2.2. 添加同一层级的可以获取焦点的多个FocusInfo对象
27 focusHolder.addFocusTarget(
28     // 第一个焦点位
29     btn1Info,
30     // 第二个焦点位
31     FocusInfo(
32         btn2,

```

```

33     eventHandler = {action ->
34         when(action) {
35             is TempleAction.Click -> {
36                 FToast.show("bt2 click")
37             }
38             else -> Unit
39         }
40     },
41     focusChangeHandler = { hasFocus ->
42         mBindingPair.updateView {
43             btn2.setBackgroundColor(getColor(if(hasFocus) R.color.purple_200 else R.color.black))
44         }
45     }
46 ),
47 // 第xxx个焦点位
48 .....
49 )
50 // 2.3 设置默认的焦点位
51 focusHolder.currentFocus(mBindingPair.left.btn1)
52 }
53 // 3. 最终构建FixPosFocusTracker对象
54 fixPosFocusTracker = FixPosFocusTracker(focusHolder).apply {
55     //3.1 设置当前获取到焦点，如果没有获取焦点，则不会响应任何事件
56     focusObj.hasFocus = true
57 }
58 .....
59 //4. 与TempleAction事件流对接
60 lifecycleScope.launch {
61     repeatOnLifecycle(Lifecycle.State.RESUMED) {
62         templeActionViewModel.state.collect {
63             FLogger.i("DemoActivity", "action = $it")
64             when (it) {
65                 is TempleAction.DoubleClick -> {
66                     // 统一处理双击退出逻辑
67                     finish()
68                 }
69                 else -> fixPosFocusTracker?.handleFocusTargetEvent(it)
70             }
71         }
72     }
73 }

```

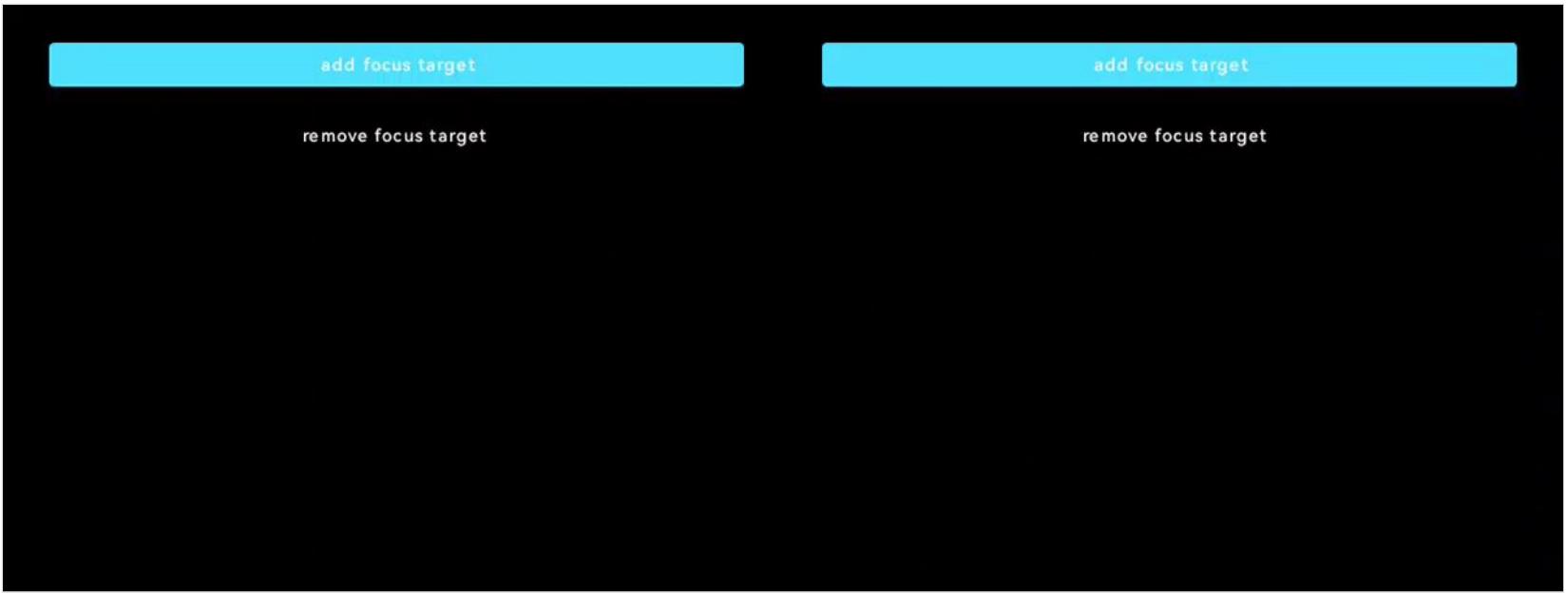
这样同一层级的焦点，就可以按照以上流程来处理。而FixPosFocusTracker本身又可以作为FocusInfo的target，层层嵌套后，即可应对复杂页面的焦点切换场景。

对于RecyclerView来说，焦点一般在某个Item View上，焦点的切换又是另外一种情况。根据焦点位是否固定，可以分为**列表+固定焦点位**和**列表+移动焦点位**两种情况，SDK分别提供了**RecyclerViewSlidingTracker**和**RecyclerViewFocusTracker**两个工具类来处理，具体代码参考SDKDemo中的FixedFocusPosRVActivity和MovedFocusPosRVActivity两个页面。

针对其他情况，比如自定义View中的整体的焦点切换，可以通过实现IFocusable接口来统一处理，只有在hasFocus属性为true时，才响应事件。

同时RecyclerViewSlidingTracker、RecyclerViewFocusTracker、IFocusable对象都可以作为FocusInfo的target。

2.2 动态焦点



根据用户交互、视线追踪、场景变化或设备姿态，**实时调整视图中“焦点视图”的位置、状态、显示内容或可交互优先级**——简单说就是让“用户当前需要关注 / 可操作的核心视图”随场景动态变化，而非固定在某个位置或状态

1. 用户转头看左侧菜单 → 焦点自动从右侧弹窗切换到左侧菜单；
2. 场景中出现新的交互目标（如虚拟按钮）→ 动态添加该按钮为新的焦点视图；

在眼镜合目的视图中动态调整焦点视图，需要强依赖视图组件，动态添加焦点视图示例代码如下（详情请参考sample中的DynamicFocusTargetActivity）：

代码块

```
1  private fun addDynamicFocus() {
2      // 扩展函数API动态添加焦点View
3      // 使用可变引用，因为handle需要在eventHandler中使用
4      var handle: FocusViewHandle<View>? = null
5      handle = mBindingPair.addFocusView(
6          parent = mBindingPair.left.llParent,
7          viewFactory = {
8              //这里动态添加一个button，实际以需求为准
9              Button(this@DynamicFocusTargetActivity).apply {
10                  text = "dynamic focus target"
11                  background = null
12              }
13          },
14          focusHolder = focusHolder,
15          focusConfig = {
16              // 设置布局参数 (eg: LinearLayout, 以实际视图为准)
17              layoutParamsFactory = { parent, view ->
18                  LinearLayout.LayoutParams(
19                      150.dp,
20                      ViewGroup.LayoutParams.WRAP_CONTENT
21                  ).apply {
22                      setMargins(
23                          20.dp,
24                          5.dp,
25                          20.dp,
26                          5.dp
27                      )
28                  }
29              }
30          }
31          //这里处理视图的响应事件
32          eventHandler = { action ->
33              when (action) {
34                  is TempleAction.Click -> {
35                      FToast.show("dynamic focus target click!!")
36                  }
37              }
38          }
39      }
40      //这里处理视图的响应事件
41      eventHandler = { action ->
42          when (action) {
43              is TempleAction.Click -> {
44                  FToast.show("dynamic focus target click!!")
45              }
46          }
47      }
48  }
```

```

37
38         else -> Unit
39     }
40 }
41 //这里处理视图焦点切换
42 onFocusChange = { view, hasFocus, isLeft ->
43     triggerFocus(hasFocus, view, isLeft)
44 }
45 }
46 )
47
48 // focusHandles是一个管理动态添加焦点视图的可变列表，可酌情添加
49 focusHandles.add(handle)
50
51 currentDynamicFocus = handle
52 }

```

移除动态焦点视图如下：

代码块

```
1 currentDynamicFocus.clearFocusView()
```

3. TP管理

目前**右镜腿**可以触发MotionEvent TP事件，相对于手机的二维TP事件，左右镜腿产生的TP事件，只有一维，即X坐标动态变化，Y坐标为固定值。除了左右镜腿外，还有**戒指配件**，配对连接后也会产生TP事件。

事件响应：TP事件分发至Activity或View级别后，在分发流程中会将原始MotionEvent传递给**TouchDispatcher**，TouchDispatcher会将原生TP事件识别为**单击、长按、双击、三击、前滑&后滑等手势**（部分手势带有系统默认音效），并以**CommonTouchCallback**接口暴露出去，调用方只需根据对应手势完成自己业务逻辑即可。

基于**TouchDispatcher**和**CommonTouchCallback**，针对Activity封装了BaseTouchActivity和BaseEventActivity。BaseTouchActivity自动注册了手势监听。BaseEventActivity继承自BaseTouchActivity，并将对应的手势转成了kotlin Flow事件流（BaseMirrorActivity继承自BaseEventActivity），将事件映射为了TempleAction子类对象，事件监听示例代码如下：

代码块

```

1 class TPEventActivity : BaseMirrorActivity<ActivityTpEventBinding>() {
2     private var fixPosFocusTracker: FixPosFocusTracker? = null
3     override fun onCreate(savedInstanceState: Bundle?) {
4         super.onCreate(savedInstanceState)
5         initFocusTarget()
6         initEvent()
7     }
8
9     private fun initFocusTarget() {
10         val focusHolder = FocusHolder(false)
11         mBindingPair.setLeft {
12             focusHolder.addFocusTarget(
13                 FocusInfo(
14                     btnEvent,
15                     eventHandler = { action -> handleAction(action) },
16                     focusChangeHandler = { hasFocus ->
17                         mBindingPair.updateView {
18                             triggerFocus(hasFocus, btnEvent, mBindingPair.checkIsLeft(this))
19                         }
20                     }
21                 )
22             )
23         }
24     }
25
26     private fun handleAction(action: Action) {
27         when (action) {
28             is Action.TP -> {
29                 if (action.isLeft) {
30                     fixPosFocusTracker?.setLeftFocus(
31                         FocusHolder(true).apply {
32                             addFocusTarget(FocusInfo(
33                                 btnEvent,
34                                 eventHandler = { action -> handleAction(action) },
35                                 focusChangeHandler = { hasFocus ->
36                                     mBindingPair.updateView {
37                                         triggerFocus(hasFocus, btnEvent, mBindingPair.checkIsLeft(this))
38                                     }
39                                 }
40                             ))
41                         })
42                 } else {
43                     fixPosFocusTracker?.setRightFocus(
44                         FocusHolder(true).apply {
45                             addFocusTarget(FocusInfo(
46                                 btnEvent,
47                                 eventHandler = { action -> handleAction(action) },
48                                 focusChangeHandler = { hasFocus ->
49                                     mBindingPair.updateView {
50                                         triggerFocus(hasFocus, btnEvent, mBindingPair.checkIsLeft(this))
51                                     }
52                                 }
53                             ))
54                         })
55                 }
56             }
57         }
58     }
59
60     private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
61         if (hasFocus) {
62             mBindingPair.updateView {
63                 triggerFocus(hasFocus, btnEvent, isLeft)
64             }
65         }
66     }
67
68     private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
69         if (hasFocus) {
70             mBindingPair.updateView {
71                 triggerFocus(hasFocus, btnEvent, isLeft)
72             }
73         }
74     }
75
76     private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
77         if (hasFocus) {
78             mBindingPair.updateView {
79                 triggerFocus(hasFocus, btnEvent, isLeft)
80             }
81         }
82     }
83
84     private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
85         if (hasFocus) {
86             mBindingPair.updateView {
87                 triggerFocus(hasFocus, btnEvent, isLeft)
88             }
89         }
90     }
91
92     private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
93         if (hasFocus) {
94             mBindingPair.updateView {
95                 triggerFocus(hasFocus, btnEvent, isLeft)
96             }
97         }
98     }
99
100    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
101        if (hasFocus) {
102            mBindingPair.updateView {
103                triggerFocus(hasFocus, btnEvent, isLeft)
104            }
105        }
106    }
107
108    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
109        if (hasFocus) {
110            mBindingPair.updateView {
111                triggerFocus(hasFocus, btnEvent, isLeft)
112            }
113        }
114    }
115
116    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
117        if (hasFocus) {
118            mBindingPair.updateView {
119                triggerFocus(hasFocus, btnEvent, isLeft)
120            }
121        }
122    }
123
124    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
125        if (hasFocus) {
126            mBindingPair.updateView {
127                triggerFocus(hasFocus, btnEvent, isLeft)
128            }
129        }
130    }
131
132    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
133        if (hasFocus) {
134            mBindingPair.updateView {
135                triggerFocus(hasFocus, btnEvent, isLeft)
136            }
137        }
138    }
139
140    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
141        if (hasFocus) {
142            mBindingPair.updateView {
143                triggerFocus(hasFocus, btnEvent, isLeft)
144            }
145        }
146    }
147
148    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
149        if (hasFocus) {
150            mBindingPair.updateView {
151                triggerFocus(hasFocus, btnEvent, isLeft)
152            }
153        }
154    }
155
156    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
157        if (hasFocus) {
158            mBindingPair.updateView {
159                triggerFocus(hasFocus, btnEvent, isLeft)
160            }
161        }
162    }
163
164    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
165        if (hasFocus) {
166            mBindingPair.updateView {
167                triggerFocus(hasFocus, btnEvent, isLeft)
168            }
169        }
170    }
171
172    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
173        if (hasFocus) {
174            mBindingPair.updateView {
175                triggerFocus(hasFocus, btnEvent, isLeft)
176            }
177        }
178    }
179
180    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
181        if (hasFocus) {
182            mBindingPair.updateView {
183                triggerFocus(hasFocus, btnEvent, isLeft)
184            }
185        }
186    }
187
188    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
189        if (hasFocus) {
190            mBindingPair.updateView {
191                triggerFocus(hasFocus, btnEvent, isLeft)
192            }
193        }
194    }
195
196    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
197        if (hasFocus) {
198            mBindingPair.updateView {
199                triggerFocus(hasFocus, btnEvent, isLeft)
200            }
201        }
202    }
203
204    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
205        if (hasFocus) {
206            mBindingPair.updateView {
207                triggerFocus(hasFocus, btnEvent, isLeft)
208            }
209        }
210    }
211
212    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
213        if (hasFocus) {
214            mBindingPair.updateView {
215                triggerFocus(hasFocus, btnEvent, isLeft)
216            }
217        }
218    }
219
220    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
221        if (hasFocus) {
222            mBindingPair.updateView {
223                triggerFocus(hasFocus, btnEvent, isLeft)
224            }
225        }
226    }
227
228    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
229        if (hasFocus) {
230            mBindingPair.updateView {
231                triggerFocus(hasFocus, btnEvent, isLeft)
232            }
233        }
234    }
235
236    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
237        if (hasFocus) {
238            mBindingPair.updateView {
239                triggerFocus(hasFocus, btnEvent, isLeft)
240            }
241        }
242    }
243
244    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
245        if (hasFocus) {
246            mBindingPair.updateView {
247                triggerFocus(hasFocus, btnEvent, isLeft)
248            }
249        }
250    }
251
252    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
253        if (hasFocus) {
254            mBindingPair.updateView {
255                triggerFocus(hasFocus, btnEvent, isLeft)
256            }
257        }
258    }
259
260    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
261        if (hasFocus) {
262            mBindingPair.updateView {
263                triggerFocus(hasFocus, btnEvent, isLeft)
264            }
265        }
266    }
267
268    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
269        if (hasFocus) {
270            mBindingPair.updateView {
271                triggerFocus(hasFocus, btnEvent, isLeft)
272            }
273        }
274    }
275
276    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
277        if (hasFocus) {
278            mBindingPair.updateView {
279                triggerFocus(hasFocus, btnEvent, isLeft)
280            }
281        }
282    }
283
284    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
285        if (hasFocus) {
286            mBindingPair.updateView {
287                triggerFocus(hasFocus, btnEvent, isLeft)
288            }
289        }
290    }
291
292    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
293        if (hasFocus) {
294            mBindingPair.updateView {
295                triggerFocus(hasFocus, btnEvent, isLeft)
296            }
297        }
298    }
299
300    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
301        if (hasFocus) {
302            mBindingPair.updateView {
303                triggerFocus(hasFocus, btnEvent, isLeft)
304            }
305        }
306    }
307
308    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
309        if (hasFocus) {
310            mBindingPair.updateView {
311                triggerFocus(hasFocus, btnEvent, isLeft)
312            }
313        }
314    }
315
316    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
317        if (hasFocus) {
318            mBindingPair.updateView {
319                triggerFocus(hasFocus, btnEvent, isLeft)
320            }
321        }
322    }
323
324    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
325        if (hasFocus) {
326            mBindingPair.updateView {
327                triggerFocus(hasFocus, btnEvent, isLeft)
328            }
329        }
330    }
331
332    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
333        if (hasFocus) {
334            mBindingPair.updateView {
335                triggerFocus(hasFocus, btnEvent, isLeft)
336            }
337        }
338    }
339
340    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
341        if (hasFocus) {
342            mBindingPair.updateView {
343                triggerFocus(hasFocus, btnEvent, isLeft)
344            }
345        }
346    }
347
348    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
349        if (hasFocus) {
350            mBindingPair.updateView {
351                triggerFocus(hasFocus, btnEvent, isLeft)
352            }
353        }
354    }
355
356    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
357        if (hasFocus) {
358            mBindingPair.updateView {
359                triggerFocus(hasFocus, btnEvent, isLeft)
360            }
361        }
362    }
363
364    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
365        if (hasFocus) {
366            mBindingPair.updateView {
367                triggerFocus(hasFocus, btnEvent, isLeft)
368            }
369        }
370    }
371
372    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
373        if (hasFocus) {
374            mBindingPair.updateView {
375                triggerFocus(hasFocus, btnEvent, isLeft)
376            }
377        }
378    }
379
380    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
381        if (hasFocus) {
382            mBindingPair.updateView {
383                triggerFocus(hasFocus, btnEvent, isLeft)
384            }
385        }
386    }
387
388    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
389        if (hasFocus) {
390            mBindingPair.updateView {
391                triggerFocus(hasFocus, btnEvent, isLeft)
392            }
393        }
394    }
395
396    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
397        if (hasFocus) {
398            mBindingPair.updateView {
399                triggerFocus(hasFocus, btnEvent, isLeft)
400            }
401        }
402    }
403
404    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
405        if (hasFocus) {
406            mBindingPair.updateView {
407                triggerFocus(hasFocus, btnEvent, isLeft)
408            }
409        }
410    }
411
412    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
413        if (hasFocus) {
414            mBindingPair.updateView {
415                triggerFocus(hasFocus, btnEvent, isLeft)
416            }
417        }
418    }
419
420    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
421        if (hasFocus) {
422            mBindingPair.updateView {
423                triggerFocus(hasFocus, btnEvent, isLeft)
424            }
425        }
426    }
427
428    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
429        if (hasFocus) {
430            mBindingPair.updateView {
431                triggerFocus(hasFocus, btnEvent, isLeft)
432            }
433        }
434    }
435
436    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
437        if (hasFocus) {
438            mBindingPair.updateView {
439                triggerFocus(hasFocus, btnEvent, isLeft)
440            }
441        }
442    }
443
444    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
445        if (hasFocus) {
446            mBindingPair.updateView {
447                triggerFocus(hasFocus, btnEvent, isLeft)
448            }
449        }
450    }
451
452    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
453        if (hasFocus) {
454            mBindingPair.updateView {
455                triggerFocus(hasFocus, btnEvent, isLeft)
456            }
457        }
458    }
459
460    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
461        if (hasFocus) {
462            mBindingPair.updateView {
463                triggerFocus(hasFocus, btnEvent, isLeft)
464            }
465        }
466    }
467
468    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
469        if (hasFocus) {
470            mBindingPair.updateView {
471                triggerFocus(hasFocus, btnEvent, isLeft)
472            }
473        }
474    }
475
476    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
477        if (hasFocus) {
478            mBindingPair.updateView {
479                triggerFocus(hasFocus, btnEvent, isLeft)
480            }
481        }
482    }
483
484    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
485        if (hasFocus) {
486            mBindingPair.updateView {
487                triggerFocus(hasFocus, btnEvent, isLeft)
488            }
489        }
490    }
491
492    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
493        if (hasFocus) {
494            mBindingPair.updateView {
495                triggerFocus(hasFocus, btnEvent, isLeft)
496            }
497        }
498    }
499
500    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
501        if (hasFocus) {
502            mBindingPair.updateView {
503                triggerFocus(hasFocus, btnEvent, isLeft)
504            }
505        }
506    }
507
508    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
509        if (hasFocus) {
510            mBindingPair.updateView {
511                triggerFocus(hasFocus, btnEvent, isLeft)
512            }
513        }
514    }
515
516    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
517        if (hasFocus) {
518            mBindingPair.updateView {
519                triggerFocus(hasFocus, btnEvent, isLeft)
520            }
521        }
522    }
523
524    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
525        if (hasFocus) {
526            mBindingPair.updateView {
527                triggerFocus(hasFocus, btnEvent, isLeft)
528            }
529        }
530    }
531
532    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
533        if (hasFocus) {
534            mBindingPair.updateView {
535                triggerFocus(hasFocus, btnEvent, isLeft)
536            }
537        }
538    }
539
540    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
541        if (hasFocus) {
542            mBindingPair.updateView {
543                triggerFocus(hasFocus, btnEvent, isLeft)
544            }
545        }
546    }
547
548    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
549        if (hasFocus) {
550            mBindingPair.updateView {
551                triggerFocus(hasFocus, btnEvent, isLeft)
552            }
553        }
554    }
555
556    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
557        if (hasFocus) {
558            mBindingPair.updateView {
559                triggerFocus(hasFocus, btnEvent, isLeft)
560            }
561        }
562    }
563
564    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
565        if (hasFocus) {
566            mBindingPair.updateView {
567                triggerFocus(hasFocus, btnEvent, isLeft)
568            }
569        }
570    }
571
572    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
573        if (hasFocus) {
574            mBindingPair.updateView {
575                triggerFocus(hasFocus, btnEvent, isLeft)
576            }
577        }
578    }
579
580    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
581        if (hasFocus) {
582            mBindingPair.updateView {
583                triggerFocus(hasFocus, btnEvent, isLeft)
584            }
585        }
586    }
587
588    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
589        if (hasFocus) {
590            mBindingPair.updateView {
591                triggerFocus(hasFocus, btnEvent, isLeft)
592            }
593        }
594    }
595
596    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
597        if (hasFocus) {
598            mBindingPair.updateView {
599                triggerFocus(hasFocus, btnEvent, isLeft)
600            }
601        }
602    }
603
604    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
605        if (hasFocus) {
606            mBindingPair.updateView {
607                triggerFocus(hasFocus, btnEvent, isLeft)
608            }
609        }
610    }
611
612    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
613        if (hasFocus) {
614            mBindingPair.updateView {
615                triggerFocus(hasFocus, btnEvent, isLeft)
616            }
617        }
618    }
619
620    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
621        if (hasFocus) {
622            mBindingPair.updateView {
623                triggerFocus(hasFocus, btnEvent, isLeft)
624            }
625        }
626    }
627
628    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
629        if (hasFocus) {
630            mBindingPair.updateView {
631                triggerFocus(hasFocus, btnEvent, isLeft)
632            }
633        }
634    }
635
636    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
637        if (hasFocus) {
638            mBindingPair.updateView {
639                triggerFocus(hasFocus, btnEvent, isLeft)
640            }
641        }
642    }
643
644    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
645        if (hasFocus) {
646            mBindingPair.updateView {
647                triggerFocus(hasFocus, btnEvent, isLeft)
648            }
649        }
650    }
651
652    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
653        if (hasFocus) {
654            mBindingPair.updateView {
655                triggerFocus(hasFocus, btnEvent, isLeft)
656            }
657        }
658    }
659
660    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
661        if (hasFocus) {
662            mBindingPair.updateView {
663                triggerFocus(hasFocus, btnEvent, isLeft)
664            }
665        }
666    }
667
668    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
669        if (hasFocus) {
670            mBindingPair.updateView {
671                triggerFocus(hasFocus, btnEvent, isLeft)
672            }
673        }
674    }
675
676    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
677        if (hasFocus) {
678            mBindingPair.updateView {
679                triggerFocus(hasFocus, btnEvent, isLeft)
680            }
681        }
682    }
683
684    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
685        if (hasFocus) {
686            mBindingPair.updateView {
687                triggerFocus(hasFocus, btnEvent, isLeft)
688            }
689        }
690    }
691
692    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
693        if (hasFocus) {
694            mBindingPair.updateView {
695                triggerFocus(hasFocus, btnEvent, isLeft)
696            }
697        }
698    }
699
700    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
701        if (hasFocus) {
702            mBindingPair.updateView {
703                triggerFocus(hasFocus, btnEvent, isLeft)
704            }
705        }
706    }
707
708    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
709        if (hasFocus) {
710            mBindingPair.updateView {
711                triggerFocus(hasFocus, btnEvent, isLeft)
712            }
713        }
714    }
715
716    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
717        if (hasFocus) {
718            mBindingPair.updateView {
719                triggerFocus(hasFocus, btnEvent, isLeft)
720            }
721        }
722    }
723
724    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
725        if (hasFocus) {
726            mBindingPair.updateView {
727                triggerFocus(hasFocus, btnEvent, isLeft)
728            }
729        }
730    }
731
732    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
733        if (hasFocus) {
734            mBindingPair.updateView {
735                triggerFocus(hasFocus, btnEvent, isLeft)
736            }
737        }
738    }
739
740    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
741        if (hasFocus) {
742            mBindingPair.updateView {
743                triggerFocus(hasFocus, btnEvent, isLeft)
744            }
745        }
746    }
747
748    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
749        if (hasFocus) {
750            mBindingPair.updateView {
751                triggerFocus(hasFocus, btnEvent, isLeft)
752            }
753        }
754    }
755
756    private fun updateView(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
757        if (hasFocus) {
758            mBindingPair.updateView {
759                triggerFocus(hasFocus, btnEvent, isLeft)
760            }
761        }
762    }
763
764    private fun triggerFocus(hasFocus: Boolean, btnEvent: ButtonEvent, isLeft: Boolean) {
765        if (hasFocus) {
766            mBindingPair.updateView {
767                triggerFocus(hasFocus, btnEvent, isLeft)
768            }
769        }
770    }
771
772   
```

```
19             }
20         )
21     )
22     focusHolder.currentFocus(mBindingPair.left.btnEvent)
23 }
24 fixPosFocusTracker = FixPosFocusTracker(focusHolder).apply {
25     focusObj.reqFocus()
26 }
27 }
28 }
29
30 private fun handleAction(action: TempleAction) {
31     when (action) {
32         is TempleAction.LongClick -> {
33             FToast.show("LongClick")
34         }
35
36         is TempleAction.Click -> {
37             FToast.show("Click")
38         }
39
40         is TempleAction.DoubleClick -> {
41             FToast.show("DoubleClick")
42             finish()
43         }
44
45         is TempleAction.TripleClick -> {
46             FToast.show("TripleClick")
47         }
48
49         is TempleAction.SlideBackward -> {
50             FToast.show("SlideBackward")
51         }
52
53         is TempleAction.SlideForward -> {
54             FToast.show("SlideForward")
55         }
56
57         is TempleAction.SlideUpwards -> {
58             FToast.show("SlideUpwards")
59         }
60
61         is TempleAction.SlideDownwards -> {
62             FToast.show("SlideDownwards")
63         }
64
65
66         else -> Unit
67     }
68 }
69
70 private fun initEvent() {
71     lifecycleScope.launch {
72         repeatOnLifecycle(Lifecycle.State.RESUMED) {
73             templeActionViewModel.state.collect {
74                 fixPosFocusTracker?.handleFocusTargetEvent(it)
75             }
76         }
77     }
78 }
79
80 private fun triggerFocus(hasFocus: Boolean, view: View, isLeft: Boolean) {
```

```

81         view.setBackgroundColor(getColor(if (hasFocus)
82             com.ffalcon.mercury.android.sdk.R.color.color_rayneo_theme_0 else R.color.black))
83             // 3D效果
84             make3DEffectForSide(view, isLeft, hasFocus)
85     }

```

如果当前视图没有具体的组件需要监听TP事件，只需要实现类似原生Activity返回的效果，那么只需要与TempleAction事件流对接，眼镜推荐的做法是双击退出页面，示例代码如下：

代码块

```

1  class DialogActivity : BaseMirrorActivity<LayoutDialogBinding>() {
2
3      override fun onCreate(savedInstanceState: Bundle?) {
4          super.onCreate(savedInstanceState)
5          initEvent()
6      }
7
8      private fun initEvent() {
9          lifecycleScope.launch {
10             repeatOnLifecycle(Lifecycle.State.RESUMED) {
11                 templeActionViewModel.state.collect {
12                     FLogger.i("DemoActivity", "action = $it")
13                     when (it) {
14                         is TempleAction.DoubleClick -> {
15                             finish()
16                         }
17                         is TempleAction.Click -> {
18                             showDialog()
19                         }
20                         else -> Unit
21                     }
22                 }
23             }
24         }
25     }

```

如果之前使用过X2的SDK，那么X2的应用可以迁移到X3上，X3 SDK相对X2新增了DeviceUtil的isX3Device函数，用于判断是否是RayneoX3

代码块

```

1  if(DeviceUtil.isX3Device()){
2      // in Rayneo X3
3  }else{
4      // in Rayneo X2
5  }

```

3.1 Rayneo X3 SDK中CommonTouchBack接口改动

增加了如下函数，在X2系列产品中不包含下面的事件

代码块

```

1  //上滑
2  open fun onTPSlideUpwards(args: FlingArgs): Boolean {
3      return false

```

```
4  }
5  //下滑
6  open fun onTPSlideDownwards(args: FlingArgs): Boolean {
7      return false
8  }
9  //双指点击
10 open fun onTPDoubleFingerClick() {
11 }
12 //双指长按
13 open fun onTPDoubleFingerLongClick() {
14 }
15
```

修改了如下函数，增加了vertical，表示是否是垂直方向上的滑动数据。

代码块

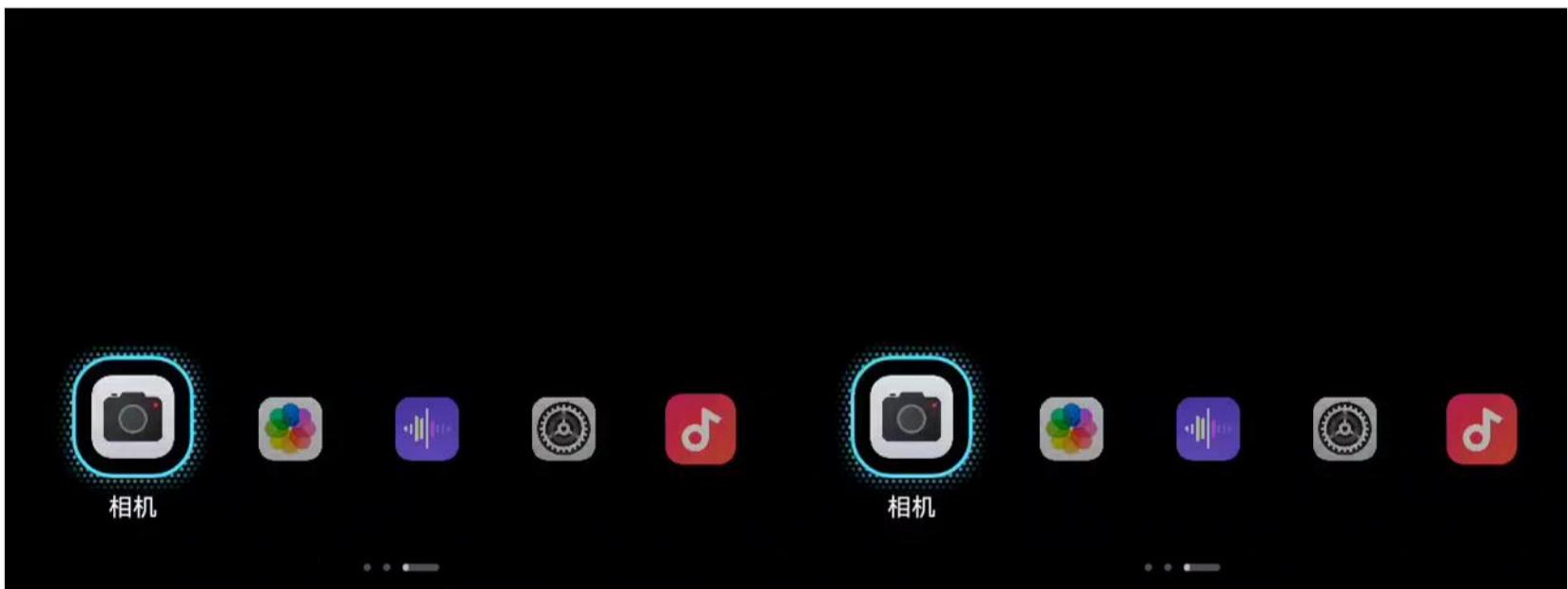
```
1  onTPSlideContinuous(delta: Float, longClick: Boolean = false, vertical:Boolean=false)
```

并且在CommonTouchBack中，增加了属性filterMode,当CommonTouchCallback设置为OnlyX的时候onTPSlideContinuous只返回X轴滑动数据，当CommonTouchCallback设置为OnlyY的时候，onTPSlideContinuous只返回Y轴滑动的数据。**只有RayneoX3支持此项设置。**

并且TempleAction.TpSlideContinuous也增加了滑动的方向是否是X轴还是Y轴。

3.2 SlideBackward & SlideForward

需要特别说明一下前滑事件TempleAction.SlideForward与后滑事件TempleAction.SlideBackward的处理，在Rayneo眼镜中触控板的滑动与UI的映射关系分为两种，**一种是自然模式，一种是非自然模式**，用户可以在设置界面里面进行设置，设置路径参考下方视频：



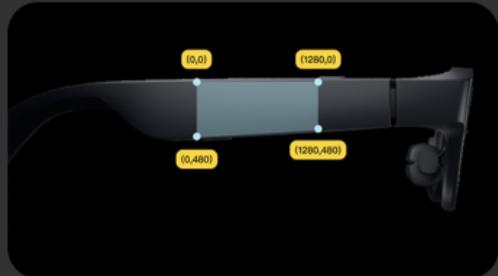
自然模式(系统默认)

触控板与UI映射关系如下图：

该模式下从镜腿方向往镜片方向滑动会被系统识别为TempleAction.SlideBackward事件，反过来会被识别为TempleAction.SlideForward事件

触控板与 UI 映射关系（自然）

右侧触控板



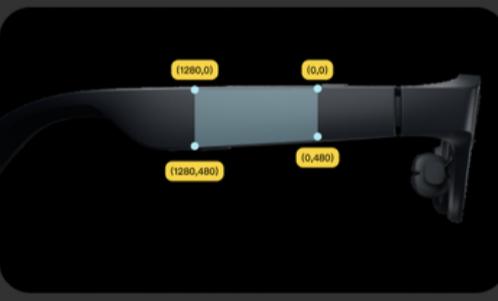
非自然模式下

触控板与UI映射关系如下图：

该模式下从镜腿方向往镜片方向滑动会被系统识别为TempleAction.SlideForward事件，反过来会被识别为TempleAction.SlideBackward事件

触控板与 UI 映射关系（非自然）

右侧触控板



4. 跟手效果

基于TouchDispatcher和CommonTouchListener，对于一般场景来说已经够用，但如果想实现手机端列表跟手滑动的效果，则无法实现，原因有2个：

- TouchDispatcher其实是对原始事件的二次处理，而列表跟手效果，需要将原始事件直接喂给对应的列表；
- 原始事件的Y坐标为固定值，已经写死，而目标列表可横可竖，且只响应落在列表区域内的事件。这意味着对于横向列表，需要根据列表中心位置修改Y坐标值；对于竖向列表，需要将TP事件对X、Y坐标对调，然后再固定X坐标值。

目前跟手效果的整体实现思路是：原始事件仍旧喂给TouchDispatcher，以触发各类手势，但只响应单击、双击、长按等操作，前滑&后滑操作不响应。同时RecyclerView固定焦点位，截获原始事件流，根据原始事件生成新的MotionEvent，修改Y坐标后，将新TP事件喂给RecyclerView，之后回收新生成的MotionEvent；新生成的TP事件流，只实现跟手效果，不做点击等事件响应。

按场景，可以分为以下3种情况：

- 固定焦点位+列表滚动跟手效果：核心工具类是RecyclerViewSlidingTracker
- 移动焦点位+列表 跟手效果：按滑动距离方式，核心工具类是RecyclerViewFocusTracker

- 移动焦点位 + 固定位置View 跟手效果：核心工具类是FixPosFocusTracker

其中，移动焦点位跟手效果的相关的核心逻辑是：滑动一定距离（比如50dp）后，根据滑动方向，触发焦点前后、上下切换逻辑。固定焦点位时，需要将TP事件修改坐标后，传给RecyclerView，示例代码如下：

代码块

```
1 // 监听原始事件，实现跟手效果
2 recyclerViewSlidingTracker.observeOriginMotionEventStream(
3     motionEventDispatcher
4 ) { event ->
5     // 生产新的MotionEvent，并修改坐标，X与Y坐标是否互换位置，根据使用场景，动态调整
6     MotionEvent.obtain(
7         event.downTime,
8         event.eventTime,
9         event.action,
10        320f,
11        event.x,
12        event.metaState
13    ).apply {
14        val e = this
15        FLogger.d(
16            "onReceiveEvent: x = ${e.x}, y = ${e.y},action = ${e.actionName()}, deviceId = ${e.deviceId}"
17        )
18    }
19 }
20
21 lifecycleScope.launchWhenResumed {
22     val templeActionViewModel =
23         ViewModelProvider(this@FixedFocusPosRVActivity).get<TempleActionViewModel>()
24     templeActionViewModel.state.collect {
25         if (!favoriteTracker.focusObj.hasFocus || !this.isActive || it.consumed) {
26             return@collect
27         }
28         recyclerViewSlidingTracker.handleActionEvent(it) { action ->
29             when (action) {
30                 is TempleAction.DoubleClick -> {
31                     finish()
32                 }
33                 is TempleAction.Click -> {
34                     if (!action.consumed) {
35                         (mBindingPair.left.recyclerView.adapter as FixedFocusPosAdapter)
36                             .getCurrentData()?.apply {
37                                 FToast.show(displayName)
38                             }
39                         }
40                     }
41                     else -> {}
42                 }
43             }
44         }
45 }
```

其中motionEventDispatcher是BaseTouchActivity成员变量，用于截获原始的事件流；observeOriginMotionEventStream函数，用于修改X、Y坐标，将原始事件流转为目标RecyclerView(左屏幕中)可以响应的事件流。完整代码，参考FixedFocusPosRVActivity。

5. 3D效果实现

目前在Android原生开发环境下，实现了基于双目视差的伪3D效果，基本原理是让左右布局中相同位置的View，**左边**布局的view向右偏移，**右边**布局的view向左偏移，从而产生视差，不同的偏移值会产生不同的景深感觉，合目后形成3D视觉效果。SDK提供了**make3DEffect()** 和**make3DEffectForSide()**，来实现该功能。

代码块

```
1 // 同时设置左右两个View
2 make3DEffect(left.slideView, right.slideView, true, 10f)
3 // 左右两边分别单独调用
4 make3DEffectForSide(this.ivSelectHover, isLeft, true)
```

6. 音频开发

在RayNeo眼镜上进行音频业务的开发，实现步骤与一般的android平台音频应用开发类似，如果需要调用MIC进行声音的收录，RayNeo眼镜上封装特有的MIC收音模式，在X2跟X3上略有不同：

X2支持的收音模式：

每个apk在调用mic的时候，去设置状态通知audioHAL; AudioManager.setparameter("audio_source=xxxx");

模式	描述	示例
audio record:	暂未配置	setParameters("audio_source_record=sound")
camera record:	调用镜腿2颗mic，声音全收进来，无降噪算法	setParameters("audio_source_record=camorder")
translation:	调用3颗麦克风，不收佩戴者声音，收外部声音	setParameters("audio_source_record=translation")
voice assistant:	调用镜腿2颗mic，主要收佩戴者声音	setParameters("audio_source_record=voice assistant")

为了确保各个应用都有自己的mic组合，请在退出record的时候，也设置下**setParameters("audio_source_record=off")**;

X3支持的收音模式：

模式	前置条件(AudioRecord)	setParameters	描述
OFF	退出录音	audioManager.setParameters("audio_source_record=off")	apk退出时记得释放麦克风，退出record的时候需要设置
RECORD_TRANSLATION	<code>device.id == SPEAKER_MIC;</code> <code>CHANNEL_IN_MONO;</code> 送翻译引擎的采样率一般设置 16k;	 ExampleRecordTranslation.java	调用前麦克风，录制周围人声，适合会议记录/现场翻译
CAMCORDER	<code>device.id == SPEAKER_MIC;</code> <code>CHANNEL_IN_STEREO;</code> 采样率根据需要设定，最高不超过48kHz;	 ExampleRecordCamcorder.java	左右镜腿麦克风，立体声录音，全向收音，适合录像场景使用
VOICE_RECOGNITION	<code>device.id == SPEAKER_MIC;</code> <code>CHANNEL_IN_MONO;</code> 识别引擎一般要求Sample rate为 16kHz，单通道;	 ExampleRecordRecognition.java	调用三颗麦克风，只收佩戴者声音，送识别引擎识别，不考虑音质，适合于语音助手，导航场景等
VOICE_COMMUNICATION	<code>device.id == SPEAKER_MIC</code>	 ExampleRecordCommunication.java	调用前麦和右方麦克风，只收佩戴者的声音，适用于网络通话、网络会议等

7. camera开发

在RayNeo眼镜上进行camera应用的开发与一般的android平台camera应用开发并无什么区别，详情请参考developer.android.com，同时也需要注意权限的申请以及资源的释放。

这里推荐使用camera2 API，sample中有详细的示例演示了如何在眼镜应用中调用camera，并在两个屏幕上进行预览以及调用camera获取一帧图片的逻辑。具体的实现方式可以参考CameraActivity。

如果需要查看眼镜camera支持的分辨率以及支持的预览参数，用户可自行枚举查看，示例中也有一个简单的camera枚举方法仅供参考：

代码块

```
1 private fun enumerateCameraResolutions() {
2     val cameraManager = getSystemService(CAMERA_SERVICE) as CameraManager
3     val cameraIdList = cameraManager.cameraIdList
4
5     for (cameraId in cameraIdList) {
6         val characteristics = cameraManager.getCameraCharacteristics(cameraId)
7         val map = characteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP)
8
9         if (map != null) {
10             val previewSizes = map.getOutputSizes(SurfaceTexture::class.java)
11             val pictureSizes = map.getOutputSizes(Format.JPEG)
12
13             FLogger.d("Camera ID: $cameraId")
14             FLogger.d("Supported Preview Sizes:")
15             for (size in previewSizes) {
16                 FLogger.d(" ${size.width}x${size.height}")
17             }
18
19             FLogger.d("Supported Picture Sizes:")
20             for (size in pictureSizes) {
21                 FLogger.d(" ${size.width}x${size.height}")
22             }
23         }
24     }
25 }
```

snapdragonCamera的源码示例 (AE收敛状态=2则可以正常取图) :

<https://github.com/Lh1600852534/snapdragonCamera/blob/master/src/com/android/camera/CaptureModule.java>

VGA camera

Rayneo x3眼镜上还有一个独立的VGA摄像头，专门用于空间定位，支持的参数如下：

代码块

1 2025-09-25 18:13:32.490 1583-1583 camera	com.ffalcon.mercury.android.sdk	D
Camera ID: 1		
2 2025-09-25 18:13:32.490 1583-1583 camera	com.ffalcon.mercury.android.sdk	D
Supported Preview Sizes:		
3 2025-09-25 18:13:32.491 1583-1583 camera	com.ffalcon.mercury.android.sdk	D
640x480		
4 2025-09-25 18:13:32.491 1583-1583 camera	com.ffalcon.mercury.android.sdk	D
640x400		
5 2025-09-25 18:13:32.491 1583-1583 camera	com.ffalcon.mercury.android.sdk	D
640x360		

6	2025-09-25 18:13:32.491	1583-1583	camera 480x360	com.ffalcon.mercury.android.sdk	D
7	2025-09-25 18:13:32.491	1583-1583	camera 352x288	com.ffalcon.mercury.android.sdk	D
8	2025-09-25 18:13:32.491	1583-1583	camera 320x240	com.ffalcon.mercury.android.sdk	D
9	2025-09-25 18:13:32.491	1583-1583	camera 320x180	com.ffalcon.mercury.android.sdk	D
10	2025-09-25 18:13:32.491	1583-1583	camera 176x144	com.ffalcon.mercury.android.sdk	D
11	2025-09-25 18:13:32.491	1583-1583	camera Supported Picture Sizes:	com.ffalcon.mercury.android.sdk	D
12	2025-09-25 18:13:32.491	1583-1583	camera 640x480	com.ffalcon.mercury.android.sdk	D
13	2025-09-25 18:13:32.491	1583-1583	camera 640x400	com.ffalcon.mercury.android.sdk	D
14	2025-09-25 18:13:32.491	1583-1583	camera 640x360	com.ffalcon.mercury.android.sdk	D
15	2025-09-25 18:13:32.492	1583-1583	camera 480x360	com.ffalcon.mercury.android.sdk	D
16	2025-09-25 18:13:32.492	1583-1583	camera 352x288	com.ffalcon.mercury.android.sdk	D
17	2025-09-25 18:13:32.492	1583-1583	camera 320x240	com.ffalcon.mercury.android.sdk	D
18	2025-09-25 18:13:32.492	1583-1583	camera 320x180	com.ffalcon.mercury.android.sdk	D
19	2025-09-25 18:13:32.492	1583-1583	camera 176x144	com.ffalcon.mercury.android.sdk	D

8. IMU数据获取

可通过Android原生的api获取并显示加速度计、陀螺仪和磁力计的数据，详情请参考demo中的IMUActivity

对于需要设备姿态（如旋转角度）的应用，直接使用原始传感器数据非常复杂。Android 提供了软件传感器，它自动融合了多个硬件传感器的数据，结果更平滑、更准确。

游戏旋转矢量传感器 (Game Rotation Vector)

这是最适合游戏和实时姿态跟踪的传感器，它不使用磁力计，因此不受磁场干扰，但无法提供绝对的方向（如指北）。

代码块

```

1 // 在 onCreate 中获取传感器
2 private var gameRotationVectorSensor: Sensor? = null
3
4 // 在 onCreate 方法内添加:
5 gameRotationVectorSensor = sensorManager.getDefaultSensor(Sensor.TYPE_GAME_ROTATION_VECTOR)
6
7 // 在 onResume 中注册
8 gameRotationVectorSensor?.let {
9     sensorManager.registerListener(this, it, SensorManager.SENSOR_DELAY_FASTEST)
10 }
11
12 // 在 onSensorEvent 中处理数据
13 Sensor.TYPE_GAME_ROTATION_VECTOR -> {
14     // event.values 包含4个分量: [x*sin(θ/2), y*sin(θ/2), z*sin(θ/2), cos(θ/2)]
15     // 这是一个四元数(Quaternion)，表示旋转
16     val x = event.values[0]
17     val y = event.values[1]
18     val z = event.values[2]
19     val w = event.values[3]
20     // 可以将四元数转换为欧拉角（俯仰角、横滚角、偏航角）以便于理解

```

```
21     // 注意：由于不使用磁力计，偏航角是相对值，会漂移。  
22 }
```

将四元数转换为欧拉角（简单示例）

代码块

```
1  private fun quaternionToEuler(x: Float, y: Float, z: Float, w: Float): FloatArray {  
2      val euler = FloatArray(3)  
3  
4      // 俯仰角 (pitch, X轴旋转)  
5      val sinP = 2.0f * (w * x + y * z)  
6      val cosP = 1.0f - 2.0f * (x * x + y * y)  
7      euler[0] = Math.atan2(sinP.toDouble(), cosP.toDouble()).toFloat()  
8  
9      // 横滚角 (roll, Y轴旋转)  
10     val sinR = 2.0f * (w * y - z * x)  
11     euler[1] = if (Math.abs(sinR) >= 1) {  
12         (Math.PI / 2).toFloat() * Math.signum(sinR)  
13     } else {  
14         Math.asin(sinR.toDouble()).toFloat()  
15     }  
16  
17      // 偏航角 (yaw, Z轴旋转)  
18      val sinY = 2.0f * (w * z + x * y)  
19      val cosY = 1.0f - 2.0f * (y * y + z * z)  
20      euler[2] = Math.atan2(sinY.toDouble(), cosY.toDouble()).toFloat()  
21  
22      // 将弧度转换为角度 (可选)  
23      euler[0] = Math.toDegrees(euler[0].toDouble()).toFloat()  
24      euler[1] = Math.toDegrees(euler[1].toDouble()).toFloat()  
25      euler[2] = Math.toDegrees(euler[2].toDouble()).toFloat()  
26  
27      return euler  
28 }
```

采样率常量

在 `registerListener` 时，你可以选择不同的采样率：

- `SENSOR_DELAY_FASTEST`：最快，尽可能高的频率。
- `SENSOR_DELAY_GAME`：适合游戏的速率。
- `SENSOR_DELAY_NORMAL`：默认速率，适合屏幕方向改变。
- `SENSOR_DELAY_UI`：适合用户界面变化。

最佳实践：总是记得在 `onPause()` 中调用 `sensorManager.unregisterListener(this)` 来避免后台耗电。

9. 手机连接状态 & GPS推流

手机连接状态主要是通过MobileState类来获取，`MobileState.isMobileConnected()`得到手机连接状态的数据流，可以实时监听手机的连接状态，实现方式如下：

代码块

```
1  private fun collectBleStatus() {  
2      MobileState.isMobileConnected().onEach {  
3          FLogger.d("isMobileConnected:$it")  
4          mBindingPair.updateView {
```

```
5         tvBleStatus.text = if (it) "connect" else "disconnect"
6     }
7 }.launchIn(lifecycleScope)
8 }
```

获取连接手机的GPS推流需要先集成IPC SDK，集成方式可参考IPCSDK for Android，具体实现方式可参考IPC SDK Sample或者上述Sample中的APIActivity，需要注意的是onResponse回调在异步线程，需要注意线程切换：

代码块

```
1 private val response =
2     OnResponseListener { response ->
3         if (response?.getData() == null) return@OnResponseListener
4         try {
5             val jo = JSONObject(response.getData())
6             if (jo.has("mLatitude") && jo.has("mLongitude") && jo.has("mAltitude")) { //GPS数据
7                 val mProvider = jo.getString("mProvider")
8                 val mTime = jo.getLong("mTime")
9                 val mElapsedRealtimeNanos = jo.getLong("mElapsedRealtimeNanos")
10                val mLatitude = jo.getDouble("mLatitude")
11                val mLongitude = jo.getDouble("mLongitude")
12                runOnUiThread {
13                    mBindingPair.updateView {
14                        tvTvLocationInfo.text = "$mLatitude,$mLongitude"
15                    }
16                }
17            }
18        } catch (e: JSONException) {
19            e.printStackTrace()
20        }
21    }
22 }
```