Android 13 返回导航大变更，返回键彻底废弃 + 可预见型返回手势

小虾米君 郭霖 2022-11-07 08:00 发表于江苏

/ 今日科技快讯 /

据外国媒体报道，马斯克计划裁撤推特约3700个岗位，这占到这家社交媒体公司员工总数的一半，目的是在其440亿美元的收购案之后降低成本。此外，马斯克还打算取消公司现行的远程办公政策，要求剩余员工前往公司上班。

/ 作者简介 /

大家周一好，新的一周继续加油努力吧！

本篇文章转自TechMerger的博客，文章主要分享了Android 13 返回导航重大变更与适配，相信会对大家有所帮助！

原文地址：

> https://juejin.cn/post/7105645114760331300

/ 前言 /

Android 10 首次引入了全局返回手势，但直到返回触发才能看到目标上层画面。13 针对该特性进行了优化，即返回触发之前可以预览上层画面。同时彻底废弃了返回键相关的 API，这将对现有的 App 逻辑产生巨大的影响！

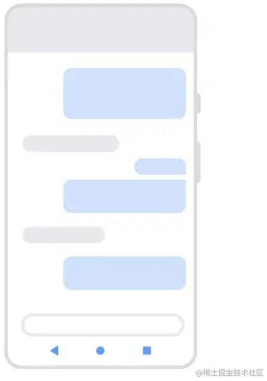
Android 13 针对包括手机、大屏、折叠屏等 Android 设备推出了可预见型返回手势（Predictive Back Gesture）特性。该特性将便于用户在返回完成之前可以先预览到目标画面或结果，这样的话可以允许他们决定是否要继续返回或者放弃并停留在当前画面。

另外引入关于 KEYCODE_BACK KeyEvent 相关的一系列变更。

为节省篇幅和统一认识，后续的相关描述将按照如下规则简称：

- 本次引入的可预见型返回手势 + KEYCODE_BACK 系列变更：统称为**新返回导航**
- KEYCODE_BACK KeyEvent：简称为 KEYCODE_BACK
- 传统导航模式和 Swipe-Up 导航模式下的返回按钮：简称为**Back KeyButton**
- 全局返回手势：简称为**Back Gesture**

Back KeyButton



Back Gesture

后续将按照如下几个方面去阐述：

1. 新返回导航的具体影响
2. 如何确定是否受影响
3. 适配方案的选择
4. 适配方案的详述
5. SDK API 适配方案的深入探讨
6. 新返回导航支持与否的深入比较和原理分析
7. 注意和残留事项

/ 新返回的影响 /

简单来说会产生如下影响：

返回手势的可预见型 UI 的增强：展示返回触发前上层画面

原有 API 废弃：

- KEYCODE_BACK：详述见小章节
- Activity/Dialog:onBackPressed()

引入全新的 SDK 返回相关 API：

- Manifest 中 enableOnBackInvokedCallback 属性
- Activity/Dialog/Window：getOnBackInvokedDispatcher()
- OnBackInvokedDispatcher
- OnBackInvokedCallback

备注：**无关TargetSDKVersion** ，运行在 13 上只要支持新返回导航均会受收到如上的影响。

KEYCODE_BACK 非推荐

准确含义是 13 上一旦开启新返回导航支持，无论是 Back Gesture 的触发还是 Back KeyButton 的点击，App 均无法监听到 KEYCODE_BACK 事件。即相关的如下 API 将无法被回调：

Activity：

- dispatchKeyEvent()
- onKeyDown()
- onKeyUp()
- onBackPressed()

Dialog：API 同上

/ 如何确定受影响 /

除了上述提到的具体变更以外，所有 KEYCODE_BACK 的相关逻辑都得测试一下是否存在问题，比如容易忽略的 View、Dialog$Builder。

简单来说，检查下现有代码是否用到了如下 API：

1. Activity/Dialog#onBackPressed()
2. Activity：dispatchKeyEvent()、onKeyDown()、onKeyUp()，监听 KEYCODE_BACK
3. Activity：使用 AndroidX 的 OnBackPressedDispatcher、OnBackPressedCallback API
4. Dialog：dispatchKeyEvent()、onKeyDown()、onKeyUp()、setOnKeyListener()，监听 KEYCODE_BACK
5. AlertDialog$Builder：setOnKeyListener()，监听 KEYCODE_BACK
6. View：dispatchKeyEvent()、onKeyDown()、onKeyUp()、setOnKeyListener()，监听 KEYCODE_BACK

/ 适配方案 /

大多数 App 都会选择自定义返回导航，可选的方式包括 SDK 的原生 API 和 AndroidX 的 Callback API。依据这些情况的不同、App 适配的意愿不同，适配的方案也不一样。

没有自定义返回导航的场景

加入新返回导航的支持即可，具体见《4.1 加入新返回导航的支持》章节。

自定义返回导航的场景

需要按照现有 API 是否接入了 AndroidX 的 OnBackPressedDispatcher 进行分情况适配。

/ 方案详述 /

1. 加入新返回导航的支持

Manifest 中针对新返回导航特性引入的属性 enableOnBackInvokedCallback 默认是 false，即默认不支持该特性，支持的话需要声明为 true。

```
<application
   ...
   android:enableOnBackInvokedCallback="true"
   ... >
...
</application>
```

实测发现：即便声明成了 false，但如果代码中残存了 13 的新 API（比如 OnBackInvokedCallback）的使用，仍会导致新返回导航发生作用。也就是说，不支持的话，就不要使用任何新的返回相关 API。

2. 关闭新返回导航的支持

正如上面所述，按照如下即可关闭对新返回导航的支持：

enableOnBackInvokedCallback 声明为 false（不声明亦可）
不要使用 OnBackInvokedCallback 等返回相关 API

3. 升级已有的 AndroidX 返回 API

对于已使用 AndroidX 返回 API 的 App 只需开启新返回导航的支持，其他的适配工作交由 AndroidX 框架来完成。

Supporting the predictive back gesture requires updating your app, using the OnBackPressedCallback ***AppCompat 1.6.0-alpha03*** *(AndroidX) or higher API.*

笔者按照官方说明将 AppCompat 包升级到了 1.6.0-alpha03。

```
dependencies {
  implementation 'androidx.appcompat:appcompat:1.6.0-alpha03'
}
```

使用其提供的 OnBackPressedCallback API 监听 Activity 的 Back 操作如下：

```
class BackKeyTestActivityAppCompat : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    ...
    onBackPressedDispatcher.addCallback(object : OnBackPressedCallback(true) {
      override fun handleOnBackPressed() {
        Log.d("BackGesture", "Activity#handleOnBackPressed()")
      }
    })
  }

  override fun dispatchKeyEvent(event: KeyEvent): Boolean {
    Log.d("BackGesture", "Activity#dispatchKeyEvent() event:$event")
    return super.dispatchKeyEvent(event)
  }

  override fun onKeyDown(keyCode: Int, event: KeyEvent): Boolean {
    Log.d("BackGesture", "Activity#onKeyDown() event:$event")
    return super.onKeyDown(keyCode, event)
  }

  override fun onKeyUp(keyCode: Int, event: KeyEvent): Boolean {
    Log.d("BackGesture", "Activity#onKeyUp() event:$event")
    return super.onKeyUp(keyCode, event)
  }
```

```
    override fun onBackPressed() {
        Log.d("BackGesture", "onBackPressed()")
        super.onBackPressed()
    }
}
```

可是实测发现：

- 即便在 13 上开启了新返回导航，无论是 Back Gesture 还是 Back KeyButton，Callback 和 KeyEvent 回调均未执行，Activity 将直接结束
- 但同样的代码运行在 12 上的话，Back Gesture 和 Back KeyButton 下 Callback 和 KeyEvent 均能被回调

12-Back Gesture 的执行日志：

```
05-31 10:35:28.732 11267 11267 D BackGesture: Activity#dispatchKeyEvent() event:KeyEvent { actior
05-31 10:35:28.733 11267 11267 D BackGesture: Activity#onKeyDown() event:KeyEvent { action=ACTION
05-31 10:35:28.733 11267 11267 D BackGesture: Activity#dispatchKeyEvent() event:KeyEvent { actior
05-31 10:35:28.733 11267 11267 D BackGesture: Activity#onKeyUp() event:KeyEvent { action=ACTION_U
05-31 10:35:28.733 11267 11267 D BackGesture: onBackPressed()
05-31 10:35:28.734 11267 11267 D BackGesture: Activity#handleOnBackPressed()
```

12-Back KeyButton 的执行日志：

```
05-31 10:37:21.724 11267 11267 D BackGesture: Activity#dispatchKeyEvent() event:KeyEvent { actior
05-31 10:37:21.724 11267 11267 D BackGesture: Activity#onKeyDown() event:KeyEvent { action=ACTION
05-31 10:37:21.846 11267 11267 D BackGesture: Activity#dispatchKeyEvent() event:KeyEvent { actior
05-31 10:37:21.846 11267 11267 D BackGesture: Activity#onKeyUp() event:KeyEvent { action=ACTION_U
05-31 10:37:21.846 11267 11267 D BackGesture: onBackPressed()
05-31 10:37:21.846 11267 11267 D BackGesture: Activity#handleOnBackPressed()
```

调试了一下，发现 AppCompat 框架里使用 13 的新 SDK API 前的版本判断有问题：

```
public class ComponentActivity {
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        ...
        if (Build.VERSION.SDK_INT >= 33) {
```

```
            mOnBackPressedDispatcher.setOnBackInvokedDispatcher(getOnBackInvokedDispatcher());
        }
        ...
    }
}

public final class OnBackPressedDispatcher {
    Cancellable addCancellableCallback(@NonNull OnBackPressedCallback onBackPressedCallback) {
        ...
        if (Build.VERSION.SDK_INT >= 33) {
            updateBackInvokedCallbackState();
            onBackPressedCallback.setIsEnabledConsumer(mEnabledConsumer);
        }
        return cancellable;
    }
}
```

Beta 版的 SDK_INT 常量仍然是 12L 的 32，到正式发布才会改为 33，所以版本判断应当使用 BuildCompat 的如下 API：

```
// BuildCompat.java
public static boolean isAtLeastT() {
    return VERSION.SDK_INT >= 33
            || (VERSION.SDK_INT >= 32
            && isAtLeastPreReleaseCodename("Tiramisu", VERSION.CODENAME));
}
```

官方文档提示说的是使用 1.6.0-alpha03 及以上，那么 03 应该是首次引入上述适配的版本，可能还没做好。查了下 AppCompat 包是否出现最新版本，果然有个 **1.6.0-alpha04**。

更新了后确实好了，即 13 上开启支持的话，无论是 Back Gesture 还是 Back KeyButton，**能像预期的那样都只会输出 androidX 版本的 Callback，Back 相关 KeyEvent 回调将不再执行。**

```
05-31 10:55:10.773  5041  5041 D BackGesture: Activity#handleOnBackPressed()
```

但仍有一点未达预期：

按理说 13 上关闭支持的话，无论是 Back Gesture 还是 Back KeyButton，运行结果应该和 12 保持一致，即收到 Back 相关 KeyEvent 回调以及 OnBackPressedCallback

可实测发现：只有 Back KeyButton 点击是上述结果，Back Gesture 的话只收到了 Callback、没有 KeyEvent 回调，这里有点奇怪

4. 迁移非推荐 SDK 返回 API 到 AndroidX API

适配步骤：

迁移已有的系统返回处理逻辑到 AndroidX 的 OnBackPressedDispatcher API，他需要指定 OnBackPressedCallback 实现，详细的可参考**如何提供自定义返回导航** (https://developer.android.google.cn/guide/navigation/navigation-custom-back?hl=zh-cn)

对于 Activity：

```
class MyActivity : AppCompatActivity() {
  override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    val callback = onBackPressedDispatcher.addCallback(this) {
      // Handle the back button event
    }
  }
  ...
}
```

对于 Fragment：

```
public class FormEntryFragment extends Fragment {
  @Override
  public void onAttach(@NonNull Context context) {
    super.onAttach(context);
    OnBackPressedCallback callback = new OnBackPressedCallback(
      true // default to enabled
    ) {
      @Override
      public void handleOnBackPressed() {
        showAreYouSureDialog();
      }
    };
```

```
        requireActivity().getOnBackPressedDispatcher().addCallback(
          this, // LifecycleOwner
          callback);
    }
}
```

禁用原有的系统返回手势回调，比如 onBackPressed()、KEYCODE_BACK

解释：getOnBackPressedDispatcher 早在 13 之前就已经支持，既然换了就没必要保留 SDK API 逻辑。

最后记得加入新返回导航的支持。

5. 迁移非推荐 SDK 返回 API 到新 SDK 返回 API

适配步骤：

运行在 13 及之后的版本上使用全新的 SDK API 即 OnBackInvokedCallback，12及之前的版本仍可使用旧的返回 API

在 Activity、Dialog、Window 等 Window 级别的组件里需要监听返回手势的逻辑处注册实现了 onBackInvoked 方法的 OnBackInvokedCallback。这将阻止当前的 Activity 被结束，这样的话当用户触发了系统返回操作的话你的 Callback 将有机会执行你预期的返回动作

为了确保正确支持系统“后退导航”的未来增强功能，你的 App 必须注销 OnBackInvokedCallback。否则，用户在使用系统后退导航时可能会看到不良行为，例如，在视图之间“卡住”并强制他们退出应用。

*To ensure that future enhancements to the system Back navigation are properly supported, your app **MUST** unregister the OnBackInvokedCallback. Otherwise, users may see undesirable behavior when using a system Back navigation—for example, "getting stuck" between views and forcing them to force quit your app.*

```
@Override
void onCreate() {
 if (BuildCompat.isAtLeastT()) {
  getOnBackInvokedDispatcher().registerOnBackInvokedCallback(
    OnBackInvokedDispatcher.PRIORITY_DEFAULT,
    () -> {
      // ...
    }
  );
 }
}
```

比如 WebView 需要拦截返回手势以回退网页，当已经返回到主画面的时候应当注销该 Callback 让系统来处理 finish。

同样的，加入新返回导航的支持。

备注：onBackPressed() 逻辑保留也没有关系，并不会发生冲突，而且为了兼容 13 之前的系统功能本就应该保留。

registerOnBackInvokedCallback() 说明

registerOnBackInvokedCallback() 调用的时候需要提供如下两个参数：
priority：按照注册的逆序进行，但如果是高优先级的先回调。可选范围：int 型，亦可选如下预设常量：

- PRIORITY_DEFAULT：值为 0，普通回调
- PRIORITY_OVERLAY：值为 1000000，优先回调

但不可以是负值、否则会发生 IllegalArgumentException 异常
java.lang.IllegalArgumentException: Application registered OnBackInvokedCallback cannot have negative priority. Priority: -1
callback：OnBackInvokedCallback 实例，会在 Back Gesture 触发、Back KeyButton 按压的时候被回调

实际结果：只有最后一个 register 的 Callback 得到调用，但如果列表里存在 PRIORITY_OVERLAY 等更高优先级的 Callback 的话则优先。与如下描述不符：

When back is triggered, callbacks on the in-focus window are invoked in reverse order in which they are added within the same priority. Between different priorities, callbacks with higher priority are invoked first.

/ 深入探讨 /

案例

和 KEYCODE_BACK 相关的有很多 API 可以处理、场景也很繁杂，简单举例如下：

1. 覆写 Activity#onKeyDown() 处理 KEYCODE_BACK 的 DOWN：

```
class Activity {
  override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean {
    if ( ... ) return false

    when (keyCode) {
      KeyEvent.KEYCODE_BACK -> { methodA() }
      KeyEvent.KEYCODE_MENU -> { ... }
      else -> {}
    }

    return if ( ... ) {
      true
    } else super.onKeyDown(keyCode, event)
  }
}
```

2. 覆写 Activity#onKeyUp() 处理 KEYCODE_BACK 的 UP

3. 覆写 Activity#dispatchKeyEvent() 将 KeyEvent 传递到 Fragment 处理

4. 覆写 Activity#onBackPressed() 处理返回回调

5. 调用 Dialog#setOnKeyListener() 处理 KEYCODE_BACK

6. 调用 AlertDialog.Builder#setOnKeyListener() 处理 KEYCODE_BACK

7. 覆写 Dialog#dispatchKeyEvent() 处理 KEYCODE_BACK

8. 覆写 EditText#onKeyPreIme() 处理 KEYCODE_BACK

9. 甚至还有覆写 View 的 dispatchKeyEvent() 等函数处理 KEYCODE_BACK

适配

适配的目的在于确保如下：

- 12 及以前的设备上 Back Gesture、Back KeyButton 以及其他 Key 抵达的时候，onKeyUp() 等回调能正常收到
- 13 上开启新返回导航支持的话：Back Gesture 和 Back KeyButton 能在对应的 Callback 里回调，并和之前的 Back 动作保持一致。同时，其他 Key 仍能在 onKeyUp() 等原有函数里监听到

以上述的案例 1 的代码为例，如下是如何改造以保证能在 12 和 13 上运行一样的 Key 相关动作：

```
class Activity {
  private var onBackInvokedCallback: OnBackInvokedCallback? = null

  override fun onCreate(savedInstanceState: Bundle?) {
    ...

    if (BuildCompat.isAtLeastT()) {
      onBackInvokedCallback = OnBackInvokedCallback {
        onBackEvent()
      }.also {
        onBackInvokedDispatcher.registerOnBackInvokedCallback(
          OnBackInvokedDispatcher.PRIORITY_DEFAULT,
          it
        )
```

```
        }
      }
    }

    override fun onDestroy() {
      super.onDestroy()

      if (BuildCompat.isAtLeastT()) {
        onBackInvokedCallback?.let {
          onBackInvokedDispatcher.unregisterOnBackInvokedCallback(it)
        }
      }
    }

    private fun onBackEvent() {
      // if ( ... ) return false
      if ( ... ) return

      // when (keyCode) {
      //   KeyEvent.KEYCODE_BACK -> { methodA() }
      //   KeyEvent.KEYCODE_MENU -> { ... }
      //   else -> {}
      // }
      methodA()

      // return if ( ... ) {
      //   true
      // } else super.onKeyDown(keyCode, event)
    }

    // 为兼容旧版仍需完全保留
    override fun onKeyDown(keyCode: Int, event: KeyEvent?): Boolean {
      ...
    }
  }
```

如上适配的关键点在于：除了在 Manifest 中将 enableOnBackInvokedCallback 属性打开和注册 OnBackInvokedCallback() 以外，重点在于如何实现 onBackInvoked() 来达到旧版的同等返回逻辑：

- 删除掉 Back KeyButton 以外的逻辑，因为 Callback 只针对 Back 事件，没有可能收到其他 KEY 事件
- 删除掉 KEYCODE_BACK 的检查，因为 Callback 只针对 Back 事件、没有必要检查

- 按照原有的 dispatchtKeyEvent()、onKeyDown()、onKeyUp() 的逻辑决定 return true、false 以及 super 的改写办法
- 兼容旧版本保留所有的 KeyEvent 的处理逻辑

此外，需要留意如下一些细节：

新的 Callback 如何区分 dispatchKeyEvent()、onKeyDown()、onKeyUp() 的时机?

无法区分，开启新返回导航之后只有一个 OnBackInvokedCallback 回调时机，其在 Back Gesture Trigger 或 Back KeyButton Up 时触发。
原本时序：dispatchKeyEvent(DOWN) -> onKeyDown() -> dispatchKeyEvent(UP) -> onKeyUp()

新的 Callback 如何针对 KEYCODE_BACK 的 DOWN 和 UP 作区分?

无法区分，开启新返回导航之后只有最终的 Callback，没有 DOWN 和 UP 之分。

新的 Callback 针对 dispatchKeyEvent() 等处理的 return true、false、super 如何区分?

- false：本意是不处理，对应于现在的 Callback 可以是什么也不做或直接 return
- true：本意是处理，对应于现在的 Callback 可以是处理外加 return
- super：本意是交由父类处理，对应于现在的 Callback 可以是 return 或者直接删除，这取决于原来的 super 调用位置，也可以考虑在某条件满足的时候提前注销 Callback 这种思路

Back 以外，比如 Menu KeyEvent 的监听是否受影响?

不受影响。之前的 Menu Key 等监听在 13 上仍可以监听到、正常运行，可以保留。

如何兼容 13 以前的版本呢?

新老处理共存，判断运行版本：13 上开启的话执行新逻辑，13 以前继续沿用旧逻辑。

集成到 Base 中统一处理

Activity、Fragment 以及 Dialog 众多的情况下，可在 Base 类里加入统一的注册和销毁 Callback 的复用代码。

为了不干预不需要处理的子类，默认不进行注册。需要的子类覆写 isNeedInterceptBackEvent() 返回 true 并实现自己的 Callback 逻辑即可。

如下的 BaseActivity 事例代码：

```
open class BaseActivity: AppCompatActivity() {
    private var onBackInvokedCallback: OnBackInvokedCallback? = null

    /**
     * Inner class for handle back callback totally.
     */
    internal class OnBackInvokedCallbackInner constructor(baseActivity: BaseActivity) :
        OnBackInvokedCallback {
        private val activity: WeakReference<BaseActivity>

        override fun onBackInvoked() {
            activity.get()?.apply {
                onBackEvent()
            }
        }

        init {
            activity = WeakReference(baseActivity)
        }
    }

    /**
     * Override this method and return true if child wanna handle back event.
     */
    open fun isNeedInterceptBackEvent(): Boolean = false

    /**
     * Default back operation is invoking onBackPressed().
     * Child activity could override and implement its own operation.
     */
    open fun onBackEvent() {
        onBackPressed()
    }
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        ...
        if (isNeedInterceptBackEvent() && BuildCompat.isAtLeastT()) {
            onBackInvokedCallback = OnBackInvokedCallbackInner(this).also {
                onBackInvokedDispatcher.registerOnBackInvokedCallback(
                    OnBackInvokedDispatcher.PRIORITY_DEFAULT,
                    it
                )
            }
        }
    }

    override fun onDestroy() {
        ...
        if (BuildCompat.isAtLeastT()) {
            onBackInvokedCallback?.let {
                onBackInvokedDispatcher.unregisterOnBackInvokedCallback(it)
            }
        }
    }
}
```

需要的子类进行覆写。

```
class BackKeyHandleActivity : BaseActivity() {
    ...
    override fun isNeedInterceptBackEvent(): Boolean = true

    override fun onBackEvent() { ... }

    // 兼容 13 之前的逻辑
    override fun onBackPressed() { ... }
}
```

/ 比较和原理分析 /

针对采用新 SDK 返回 API 方案分别在 13 上开启和关闭新返回导航的支持，观察 KeyEvent 相关的 Log 输出，并尝试分析一些原理方面的差异。

1. 开启支持

- **Back Gesture**

开启新的返回手势支持的话，只能收到 OnBackInvokedCallback 回调，确实无法像以前一样灵活、精细地处理 KEYCODE_BACK 了。

如下的系统日志可以瞥见 Callback 处理的一些细节。

```
05-26 10:26:27.929  787  787 D NoBackGesture: Start gesture: MotionEvent { action=ACTION_DOWN ...
05-26 10:26:27.929  787  787 D NoBackGesture: Prediction [1653531987929,47,633,-1,0.000000,1]
05-26 10:26:27.930  787  787 D NoBackGesture: reset mTriggerBack=false
05-26 10:26:27.931  787  852 D ShellBackPreview: initAnimation mMotionStarted=false
05-26 10:26:27.932  787  787 D NoBackGesture: Gesture [1653531987932,alw=TRUE,TRUE,TRUE,FALSE,di
05-26 10:26:27.933  599 2725 D CoreBackPreview: Focused window found using getFocusedWindowToken
05-26 10:26:27.933  599 2725 D CoreBackPreview: startBackNavigation currentTask=Task{1d3c440 #50
05-26 10:26:27.934  787  852 D ShellBackPreview: Received backNavigationInfo:BackNavigationInfo{
05-26 10:26:27.963  787  787 D OnBackInvokedDispatcher: ViewRootImpl.registerBackCallbackOnWindo
05-26 10:26:27.968  787  787 V OnBackInvokedDispatcher: Proxy setActual android.window.WindowOnB
05-26 10:26:27.968  787  787 V OnBackInvokedDispatcher: Proxy transferring 0 callbacks to android.
05-26 10:26:28.271 3978 3978 D BackGesture: onBackInvoked()
```

通过 adb shell dumpsys input 命令确实也没有看到 InputFlinger 发送 KEYCODE_BACK 的记录。

```
MotionEvent(deviceId=8, eventTime=2965229468000, source=TOUCHSCREEN | STYLUS, displayId=0, act
MotionEvent(deviceId=8, eventTime=2965457324000, source=TOUCHSCREEN | STYLUS, displayId=0, act
...
MotionEvent(deviceId=8, eventTime=2965524225000, source=TOUCHSCREEN | STYLUS, displayId=0, act
```

- **Back KeyButton**

Back KeyButton 场景也是一样，开启新返回导航支持的话，只能收到 OnBackInvokedCallback 回调。

```
05-26 10:59:05.854 4497 4497 D OnBackInvokedDispatcher: ViewRootImpl.registerBackCallbackOnWin
05-26 10:59:05.904 4497 4497 V OnBackInvokedDispatcher: Proxy setActual android.window.WindowOn
05-26 10:59:05.904 4497 4497 V OnBackInvokedDispatcher: Proxy transferring 0 callbacks to androi
05-26 10:59:05.977 7700 7700 D BackGesture: onBackInvoked()
```

```
05-26 10:59:06.495  7700  7700 V OnBackInvokedDispatcher: Proxy unregister android.app.Activity$$
05-26 10:59:06.495  7700  7700 V OnBackInvokedDispatcher: Proxy unregister com.example.tiramisu_d
05-26 10:59:27.696  4497  4497 D OnBackInvokedDispatcher: ViewRootImpl.registerBackCallbackOnWin
05-26 10:59:27.707  4497  4497 V OnBackInvokedDispatcher: Proxy setActual android.window.WindowOn
05-26 10:59:27.707  4497  4497 V OnBackInvokedDispatcher: Proxy transferring 0 callbacks to androi
```

但 dump input 却出现了 Back 的 KeyEvent 记录，这是为什么呢?

此处留个悬念，后面会揭开谜底。

```
MotionEvent(deviceId=8, eventTime=2276120343000, source=TOUCHSCREEN | STYLUS, displayId=0, ac
KeyEvent(deviceId=-1, eventTime=2276124000000, source=KEYBOARD, displayId=0, action=DOWN, fla
MotionEvent(deviceId=8, eventTime=2276205324000, source=TOUCHSCREEN | STYLUS, displayId=0, ac
KeyEvent(deviceId=-1, eventTime=2276266000000, source=KEYBOARD, displayId=0, action=UP, flags
```

2. 关闭支持

- **Back Gesture**

当关闭支持后 Back Gesture 场景下能和旧版本一样收到 KEYCODE_BACK 了。

```
05-26 11:09:28.235  6784  6784 D BackGesture: dispatchKeyEvent() event:KeyEvent { action=ACTION_D(
05-26 11:09:28.236  6784  6784 D BackGesture: dispatchKeyEvent() event:KeyEvent { action=ACTION_UI
05-26 11:09:28.240  6784  6784 D BackGesture: onBackPressed()
```

dump input 也可以证实该 KeyEvent 的真实存在，而且可以看到 **Back Gesture 的 UP 之后连续注入了 KEYCODE_BACK 的 DOWN 和 UP 的细节。**

```
MotionEvent(deviceId=8, eventTime=585598303000, source=0x00005002, displayId=0, action=DOWN .
MotionEvent(deviceId=8, eventTime=585812734000, source=0x00005002, displayId=0, action=MOVE .
...
MotionEvent(deviceId=8, eventTime=585858936000, source=0x00005002, displayId=0, action=UP ...
KeyEvent(deviceId=-1, eventTime=585859000000, source=0x00000101, displayId=0, action=DOWN, fl
KeyEvent(deviceId=-1, eventTime=585860000000, source=0x00000101, displayId=0, action=UP, flag
```

- **Back KeyButton**

自不必说，Back KeyButton 的按下当然也可以收到 KEYCODE_BACK。

```
05-26 10:48:21.580  5817  5817 D BackGesture: dispatchKeyEvent() event:KeyEvent { action=ACTION_D(
05-26 10:48:21.635  5817  5817 D BackGesture: dispatchKeyEvent() event:KeyEvent { action=ACTION_UI
05-26 10:48:21.635  5817  5817 D BackGesture: onBackPressed()
```

但与 Gesture 不同，dump input 的结果可以看到：**在 Back KeyButton 上按下时注入了 KEYCODE_BACK 的 DOWN，抬起注入了 UP。**

```
MotionEvent(deviceId=8, eventTime=352268883000, source=0x00005002, displayId=0, action=DOWN .
KeyEvent(deviceId=-1, eventTime=352289000000, source=0x00000101, displayId=0, action=DOWN ...
MotionEvent(deviceId=8, eventTime=352378721000, source=0x00005002, displayId=0, action=UP ...
KeyEvent(deviceId=-1, eventTime=352386000000, source=0x00000101, displayId=0, action=UP...)
```

3. Back 相关时序的变化总结

对比点	关闭支持-Back Gesture	关闭支持-Back KeyButton	开启支持-Back Gesture	开启支持-Back KeyButton
ViewRootImpl#processKeyEvent()	YES	YES	**NO**	YES
dispatchKeyEvent(DOWN)	YES **trigger 时连续发送 DOWN 和 UP**	YES **按下时发送 DOWN**	NO	NO
onKeyDown()	YES	YES	NO	NO
dispatchKeyEvent(UP)	YES	YES **抬起时发送 UP**	NO	NO
onKeyUp()	YES	YES	NO	NO
onBackPressed()	YES	YES	NO	NO
OnBackInvokedCallback	NO	NO	YES	YES

4. 开启支持的原理分析

	13 开启支持	12
Back Gesture	Callback + KEYCODE_BACK 无法监听	KEYCODE_BACK 可以监听
Back KeyButton	Callback + KEYCODE_BACK 无法监听 **但 KEYCODE_BACK 实际存在**	KEYCODE_BACK 可以监听

13 上开启支持之后，如果是点击 Back KeyButton，从 dump 来看仍然发出了 KEYCODE_BACK，猜测与你大体是这样：

- Back Gesture 触发的时候，如果发现支持了 新返回导航，那就不再注入 KEYCODE_BACK，而是通过 Binder 告知 App 进程直接处理 Callback 的回调
- Back KeyButton 仍然像以前一样注入 KEYCODE_BACK，但 ViewRootImpl 接收到该事件的时候，发现支持了 新返回导航，则没有向 View 树分发，而是取出 Callback 直接回调

这里不禁产生一个疑问：

Back Gesture 和 Back KeyButton 缘何没有采用同一个处理方式?

经过思考，觉得不免又如下几点可能：

Back Gesture 和 Back KeyButton 功能定位有区别：前者是返回手势，需要展示返回图标和背面视图的动画，它的处理在 EdgeBackGesture 里；后者是虚拟按键，在 NavigationBar 的 KeyButtonView 中处理

- 13 之前没有引入可预测型动画的时候两者功能雷同，所以 Back Gesture 采用了和 Back KeyButton 一样的逻辑
- 13 引入了和 Back KeyButton 完全不同的返回预测动画，需要实现一套自己的回调路径，不需要再依赖原来的 KeyEvent 路径

另外从是否属于按键的角度上来讲

- Back Gesture 不是虚拟按键、也不是实体按键，没有必要发送 KeyEvent
- Back KeyButton 是虚拟按键，需要遵从 Key 的 Map 规范，是需要发送对应 KeyEvent 出来的。而且即便后面会被 App 拦截，但对于前期的系统 PhoneWindowManager、InputFilter 可能也需要处理

需要说明的是，当关闭新返回导航支持后，为了兼容旧的 API，Back Gesture 仍像以前一样发送 KEYCODE_BACK。当然这肯定是暂时的，后续系统肯定会强制使用该特性，到时候这个 Back Gesture 就再也不用发送 KEYCODE_BACK 了。

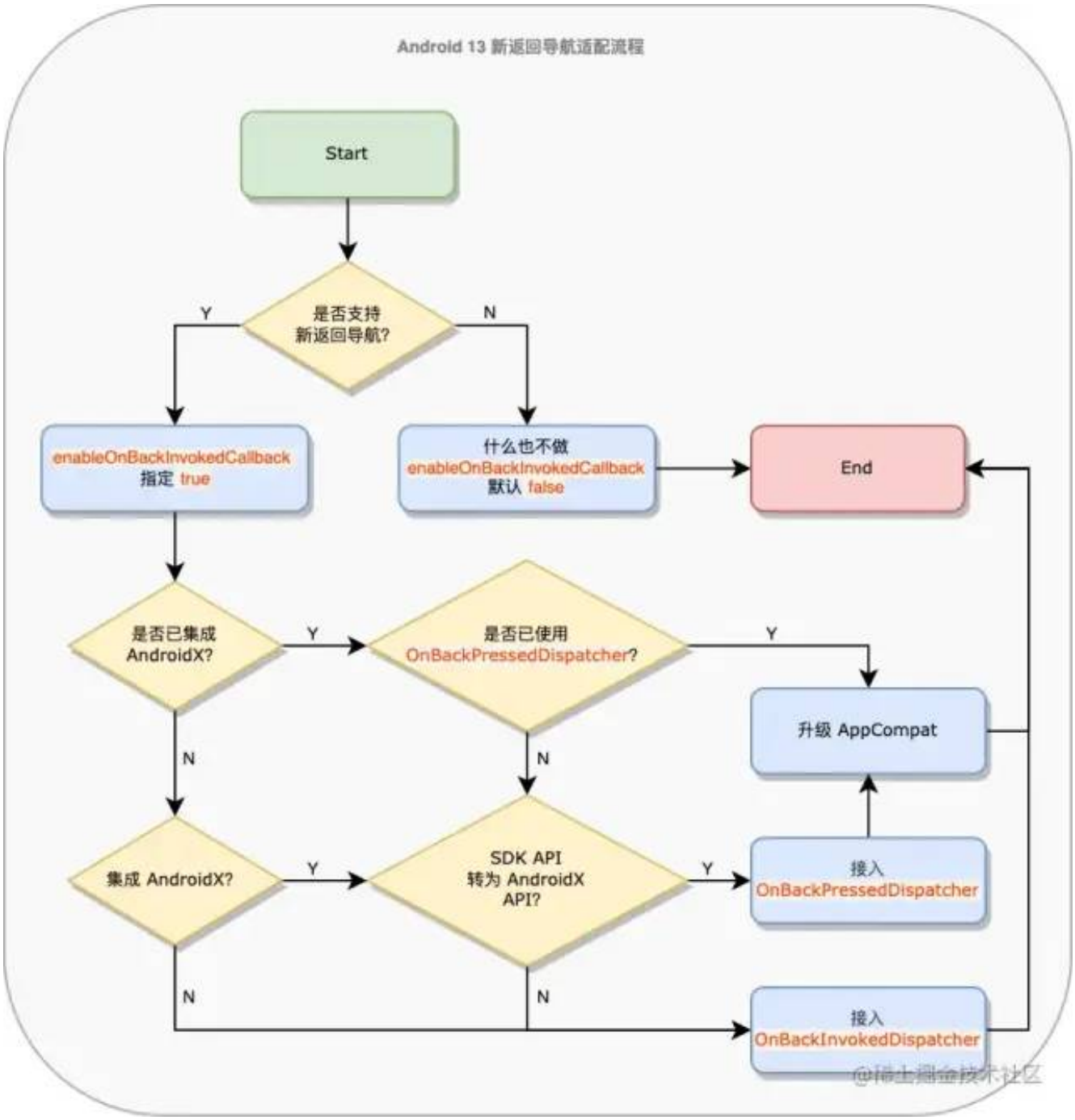
/ 注意和残留事项 /

1. 本次变更跟 TargetSDKVersion 无关，运行在 13 上的 App 都需要思考是否收到影响、如何适配
2. 直到 Android 13 最终版可预见型返回手势的动画才能生效：Settings > System > Developer options > Predictive back animations
3. 新 SDK 返回 API OnBackInvokedDispatcher 中注册的 OnBackInvokedCallback 回调不是按照文档描述的逆序，而是只回调最后一个高优先级的 Callback
4. Manifest 文件里 enableOnBackInvokedCallback 属性关闭的话，不要残留注册 OnBackInvokedCallback 的逻辑，不然新返回导航可能仍然有效
5. Dialog 场景使用新版 SDK 返回 API 没有效果，原因未知
6. View 监听 KEYCODE_BACK 的逻辑是否受影响，暂未实验
7. 对于新 SDK 返回 API 的注册和销毁的时机可以选择：onCreate() + onDestroy()，onCreate() + onStop()、onResume() + onPause() 的组合亦可，但要注意是否会发生画面展示前的 Back Gesture 或 Back KeyButton 无法被监听以及画面进入后台了但 Callback 未被注销等问题。当然注册和注销的时机可依据需要的条件灵活选择，没有绝对的要求
8. 使用 AndroidX API 方案要注意升级 AppCompat 到 1.6.0-alpha04，不然不生效
9. 另外，采用 AndroidX API 方案但关闭了支持的话，Back Gesture 没有像 Back KeyButton 一样，只能收到 OnBackPressedCallback，没有 KeyEvent 回调，原因未知

10. 对于某些场景下不希望 Callback 而希望系统处理的话，对于 SDK API 而言可以使用 unregister 方法注销该 Callback；对于 AndroidX API 而言可以将 Callback 状态置为 disabled

/ 总结 /

制作了一张 Android 13 新返回导航适配流程图供大家快速查阅。



做个简单总结：

如果决定支持新返回导航即声明 enableOnBackInvokedCallback 为 true，之后需依据 App 集成了 SDK API 还是 AndroidX API 决定适配的方案。

- SDK 方案的话需要引入新的 OnBackInvokedDispatcher 相关API，并留意 Activity、Dialog、Window、View 上现有的 Back 逻辑是否会收到影响，以及如何改造。当然需要判断运行版本，并为了兼容 13 之前的设备保留现有的 Back 逻辑
- AndroidX 方案的话使用专属的OnBackPressedDispatcher API，AppCompat 库升级之后会自行完成内部的 SDK API 迁移

另外还需要留意上述章节提及的注意事项和残留事项。

当然如果没有余力适配，决定舍弃可预测型返回手势、OnBackInvokedDispatcher 新 API 以及 KEYCODE_BACK 等一系列变更，可以选择什么也不做。

但早在 13 之前，官方已推荐使用 AndroidX 的 OnBackPressedDispatcher 来取代 onBackPressed，13 花这么大精力完全废弃 onBackPressed 并向 AOSP 新增了 OnBackInvokedDispatcher 等系列 API。

从这个趋势来看，估计到 Android 14 这个新返回导航就会成为强制要求，开发者们当尽早适配才是！

推荐阅读：

我的新书，《第一行代码 第3版》已出版！

PermissionX 1.7发布，全面支持Android 13运行时权限

Android自定义View，九宫格解锁

欢迎关注我的公众号

学习技术或投稿

▼

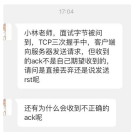
长按上图，识别图中二维码即可关注

阅读原文

喜欢此内容的人还喜欢

字节一面：TCP 三次握手，问的好细！

小林coding



入职一年，升职一次，涨薪两次

拓跋阿秀

他俩都曾是技术大牛，创业这些年来有怎样的苦与乐？

头哥侃码