Android 10 适配

注：关于Android 10 的变更说明

https://developer.android.google.cn/about/versions/10/privacy/changes

一、存储权限

具体参考Android 11 储存适配，那个是Android 10 的加强版

注：在Q以下的系统升级到Q系统，App 会使用兼容模式，此时存储权限跟Q以下没有区别；在Q系统重新下载App 时，此时无法用兼容模式，会使用储存分区模式，可以设置 requestLegacyExternalStorage 为true，用回旧的，在11 系统，此设置无效

二、设备唯一标识符

从 Android Q 开始，应用必须具有 READ_PRIVILEGED_PHONE_STATE 签名权限才能访问设备的不可重置标识符（包含 IMEI 和序列号）

设备唯一标识符需要特别注意，原来的READ_PHONE_STATE权限已经不能获得IMEI和序列号，如果想在Q设备上通过

```
((TelephonyManager) getActivity()
.getSystemService(Context.TELEPHONY_SERVICE)).getDeviceId()
```

获得设备ID，会返回空值(targetSDK<=P)或者报错(targetSDK==Q)。且官方所说的 READ_PRIVILEGED_PHONE_STATE权限只提供给系统app，所以这个方法算是废了。

谷歌官方给予了设备唯一ID最佳做法，但是此方法给出的ID(UUID)可变，我们应该已适配了

三、最小sdk 警告

在 Android Q 中，当用户首次运行以 Android 6.0（API 级别 23）以下的版本为目标平台的任何应用时，Android平台会向用户发出警告。

如果此应用要求用户授予权限，则系统会先向用户提供调整应用权限的机会，然后才会允许此应用首次运行。

谷歌要求运行在Q设备上的应用targetSDK>=23,不然会向用户发出警告。

四、非 SDK 接口限制

为了确保应用稳定性和兼容性，Android 平台开始限制App 可在 Android 9（API 级别 28）中使用一些非 SDK 接口

非SDK接口限制就是某些SDK中的私用方法，如private方法，你通过Java反射等方法获取并调用了。那么这些调用将在target>=P或target>=Q的设备上被限制使用，当你使用了这些方法后，会报错，这个我们应该没用到过

Dalvik instruction referencing a field	NoSuchFieldError thrown
Dalvik instruction referencing a method	NoSuchMethodError thrown
Reflection via Class.getDeclaredField() or Class.getField()	NoSuchFieldException thrown
Reflection via Class.getDeclaredMethod(), Class.getMethod()	NoSuchMethodException thrown
Reflection via Class.getDeclaredFields(), Class.getFields()	Non-SDK members not in results
Reflection via Class.getDeclaredMethods(), Class.getMethods()	Non-SDK members not in results
JNI via env->GetFieldID()	NULL returned, NoSuchFieldError thrown
JNI via env->GetMethodID()	NULL returned, NoSuchMethodError thrown

五、定位权限

1、在后台运行时访问设备位置信息需要权限

Android 10 引入了 ACCESS_BACKGROUND_LOCATION 权限（危险权限）。

该权限允许应用程序在后台访问位置。如果请求此权限，则还必须请求ACCESS_FINE_LOCATION 或 ACCESS_COARSE_LOCATION权限。只请求此权限无效果。

在Android 10的设备上，如果你的应用的 targetSdkVersion < 29，则在请求 ACCESS_FINE_LOCATION 或ACCESS_COARSE_LOCATION权限时，系统会自动同时请求 ACCESS_BACKGROUND_LOCATION。

在请求弹框中，选择“始终允许”表示同意后台获取位置信息，选择“仅在应用使用过程中允许”或"拒绝"选项表示拒绝授权。

如果你的应用的 targetSdkVersion >= 29，则请求ACCESS_FINE_LOCATION 或 ACCESS_COARSE_LOCATION权限表示在前台时拥有访问设备位置信息的权。在请求弹框中，选择“始终允许”表示前后台都可以获取位置信息，选择“仅在应用使用过程中允许”只表示拥有前台的权限。

总结一下就是下图：

表 1. 设备升级至 Android 10 之后位置权限状态发生的变化

目标平台版本	是否授予了 粗略或精确位置信息使用权限?	清单中是否 定义了后台权限?	更新后的默认权限状态
Android 10	是	是	前台和后台访问权
Android 10	是	否	仅前台访问权
Android 10	否	（被系统忽略）	无访问权
Android 9 或更低版本	是	在设备升级时由系统自动添加	前台和后台访问权
Android 9 或更低版本	否	（被系统忽略）	无访问权

注意：即使在系统自动更新应用对设备位置信息的访问权限之后，用户仍然可以选择更改这种访问权限级别。用户可以选择让应用只能在前台访问位置信息，或者完全撤消使用权。在尝试访问设备的位置信息之前，尤其是在前台服务中，您的应用应检查用户是否仍然允许您的应用接收此类位置信息。

其实官方**不推荐你使用申请后台访问权的方式**，因为这样的结果无非就是多请求一个权限，那么这像变更还有什么意义？申请过多的权限，也会造成用户的反感。所以官方推荐使用前台服务来实现，在**前台服务**中获取位置信息。

1）首先在清单中对应的service中添加 android:foregroundServiceType="location"：

```
<service
    android:name="MyNavigationService"
    android:foregroundServiceType="location" ... >
    ...
</service>
```

2）启动前台服务前检查是否具有前台的访问权限：

```
    boolean permissionApproved = ActivityCompat.checkSelfPermission(this,
        Manifest.permission.ACCESS_COARSE_LOCATION) ==
PackageManager.PERMISSION_GRANTED;

    if (permissionApproved) {
       // 启动前台服务
    } else {
       // 请求前台访问位置权限
    }
```

如此一来就可以在Service中获取位置信息。

2、一些电话、蓝牙和WLAN的API需要精确位置权限

下面列举了Android 10中必须具有 ACCESS_FINE_LOCATION 权限才能使用类和方法:

电话

TelephonyManager

- getCellLocation()
- getAllCellInfo()
- requestNetworkScan()
- requestCellInfoUpdate()
- getAvailableNetworks()
- getServiceState()
- TelephonyScanManager
- requestNetworkScan()
- TelephonyScanManager.NetworkScanCallback
- onResults()
- PhoneStateListener
- onCellLocationChanged()
- onCellInfoChanged()
- onServiceStateChanged()

WLAN

- WifiManager
- startScan()
- getScanResults()
- getConnectionInfo()
- getConfiguredNetworks()
- WifiAwareManager
- WifiP2pManager
- WifiRttManager

蓝牙

- BluetoothAdapter
- startDiscovery()
- startLeScan()
- BluetoothAdapter.LeScanCallback
- BluetoothLeScanner
- startScan()

我们可以根据上面提供的具体类和方法，在适配项目中检查是否有使用到并及时处理。

3、ACCESS_MEDIA_LOCATION

Android 10新增权限，上面有提到，不赘述了。

4、PROCESS_OUTGOING_CALLS

Android 10上该权限已废弃。

六、后台启动 Activity 的限制

简单解释就是应用处于后台时，无法启动Activity。

比如点开一个应用会进入启动页或者广告页，一般会有几秒的延时再跳转至首页。如果这期间你退到后台，那么你将无法看到跳转过程。而在之前的版本中，会强制弹出页面至前台。

既然是限制，那么肯定有不受限的情况，主要有以下几点：

- 应用具有可见窗口，例如前台 Activity。
- 应用在前台任务的返回栈中已有的 Activity。
- 应用在 Recents 上现有任务的返回栈中已有的 Activity。Recents 就是我们的任务管理列表。
- 应用收到系统的 PendingIntent 通知。
- 应用收到它应该在其中启动界面的系统广播。示例包括 ACTION_NEW_OUTGOING_CALL 和 SECRET_CODE_ACTION。应用可在广播发送几秒钟后启动 Activity。

用户已向应用授予 SYSTEM_ALERT_WINDOW 权限，或是在应用权限页开启后台弹出页面的开关。

因为此项行为变更适用于在 Android 10 上运行的所有应用，所以这一限制导致最明显的问题就是点击推送信息时，有些应用无法进行正常的跳转（具体的实现问题导致）。所以针对这类问题，可以采取PendingIntent的方式，发送通知时使用setContentIntent方法。

对于全屏 intent，注意设置最高优先级和添加USE_FULL_SCREEN_INTENT权限，这是一个普通权限。比如微信来语音或者视频通话时，弹出的接听页面就是使用这一功能。

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

```
Intent fullScreenIntent = new Intent(this, CallActivity.class);
PendingIntent fullScreenPendingIntent = PendingIntent.getActivity(this, 0,
        fullScreenIntent, PendingIntent.FLAG_UPDATE_CURRENT);
```

```
NotificationCompat.Builder notificationBuilder =
        new NotificationCompat.Builder(this, CHANNEL_ID)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("Incoming call")
    .setContentText("(919) 555-1234")
    .setPriority(NotificationCompat.PRIORITY_HIGH) // <--- 高优先级
    .setCategory(NotificationCompat.CATEGORY_CALL)

    // Use a full-screen intent only for the highest-priority alerts where
you
    // have an associated activity that you would like to launch after the
user
    // interacts with the notification. Also, if your app targets Android
10
    // or higher, you need to request the USE_FULL_SCREEN_INTENT permission
in
    // order for the platform to invoke this notification.
    .setFullScreenIntent(fullScreenPendingIntent, true); // <--- 全屏
intent

Notification incomingCallNotification = notificationBuilder.build();
```

注意：在部分手机上，直接设置setPriority无效（或者说以渠道优先级为准）。所以需要创建通知渠道时将重要性设置为IMPORTANCE_HIGH。

```
NotificationChannel channel = new NotificationChannel(channelId, "xxx",
NotificationManager.IMPORTANCE_HIGH);
```

七、深色主题

Android 10 新增了一个系统级的深色主题（在系统设置中开启）。

适配方法有两种：

1、手动适配（资源替换）

官方文档中提到的继承Theme.AppCompat.DayNight 或者 Theme.MaterialComponents.DayNight的方法，但这只是将我们使用的各种View的默认样式进行了适配，并不太适用于实际项目的适配。因为具体的项目中的View都按照设计的风格进行了重定义。

其实适配的方法很简单，类似屏幕适配、国际化的操作，并不需要继承上面的主题。比如你要修改颜色，就在res 下新建 values-night目录，创建对应的colors.xml文件。将具体要修改的色值定义在里面。图标之类的也是一个思路，创建对应的 drawable-night目录。

只要你之前的代码不是硬编码且代码规范，那么适配起来还是很轻松。

2、自动适配（Force Dark）

Android 10 提供 Force Dark 功能。一如其名，此功能可让开发者快速实现深色主题背景，而无需明确设置 DayNight 主题背景。

如果您的应用采用浅色主题背景，则 Force Dark 会分析应用的每个视图，并在相应视图在屏幕上显示之前，自动应用深色主题背景。有些开发者会混合使用 Force Dark 和本机实现，以缩短实现深色主题背景所需的时间。

应用必须选择启用 Force Dark，方法是在其主题背景中设置 android:forceDarkAllowed="true"。

此属性会在所有系统及 AndroidX 提供的浅色主题背景（例如 Theme.Material.Light）上设置。使用 Force Dark 时，您应确保全面测试应用，并根据需要排除视图。

如果您的应用使用Dark Theme主题（例如Theme.Material），则系统不会应用 Force Dark。同样，如果应用的主题背景继承自 DayNight 主题（例如Theme.AppCompat.DayNight），则系统不会应用 Force Dark，因为会自动切换主题背景。

可以通过 android:forceDarkAllowed 布局属性或 setForceDarkAllowed(boolean) 在特定视图上控制 Force Dark。

上述内容我直接照搬文档的说明。

总结一下，使用Force Dark需要注意几点：

1）如果使用的是 DayNight 或 Dark Theme 主题，则设置forceDarkAllowed 不生效。

2）如果有需要排除适配的部分，可以在对应的View上设置forceDarkAllowed为false。

此方案**整体还是不错的，设置的色值会自动取反。但也因此颜色不受控制，能否达到预期效果是个需要注意的问题。追求快速适配可以采取此方案。**

手动切换主题

使用 AppCompatDelegate.setDefaultNightMode(@NightMode int mode)方法，其中参数mode有以下几种：

- 浅色 - MODE_NIGHT_NO
- 深色 - MODE_NIGHT_YES

- 由省电模式设置 - MODE_NIGHT_AUTO_BATTERY
- 系统默认 - MODE_NIGHT_FOLLOW_SYSTEM

下面的代码是官方Demo中的使用示例：

```
public class ThemeHelper {

    public static final String LIGHT_MODE = "light";
    public static final String DARK_MODE = "dark";
    public static final String DEFAULT_MODE = "default";

    public static void applyTheme(@NonNull String themePref) {
        switch (themePref) {
            case LIGHT_MODE: {

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_NO);
                break;
            }
            case DARK_MODE: {

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_YES);
                break;
            }
            default: {
                if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.Q) {

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_FOLLOW_
SYSTEM);
                } else {

AppCompatDelegate.setDefaultNightMode(AppCompatDelegate.MODE_NIGHT_AUTO_BA
TTERY);
                }
                break;
            }
        }
    }
}
```

通过AppCompatDelegate.getDefaultNightMode()方法，可以获取到当前的模式，这样便于代码中去适配。

监听深色主题是否开启

首先在清单文件中给对应的Activity配置

android:configChanges="uiMode"

```
<activity
    android:name=".MyActivity"
    android:configChanges="uiMode" />
```

这样在onConfigurationChanged方法中就可以获取：

```
@Override
public void onConfigurationChanged(@NonNull Configuration newConfig) {
    super.onConfigurationChanged(newConfig);
    int currentNightMode = newConfig.uiMode &
Configuration.UI_MODE_NIGHT_MASK;
    switch (currentNightMode) {
        case Configuration.UI_MODE_NIGHT_NO:
            // 关闭
            break;
        case Configuration.UI_MODE_NIGHT_YES:
            // 开启
            break;
        default:
            break;
    }
}
```

详细的内容你可以参看官方文档和官方Demo。

https://developer.android.google.cn/guide/topics/ui/look-and-feel/darktheme

https://github.com/android/user-interface-samples/tree/master/DarkTheme

判断深色主题是否开启

其实和上面onConfigurationChanged方法同理：

```
public static boolean isNightMode(Context context) {
    int currentNightMode = context.getResources().getConfiguration().uiMode
&
        Configuration.UI_MODE_NIGHT_MASK;
    return currentNightMode == Configuration.UI_MODE_NIGHT_YES;
}
```

八、限制了对剪贴板数据的访问权限

除非App是默认输入法 (IME) 或是目前处于焦点的应用，否则它无法访问 Android 10 或更高版本平台上的剪贴板数据

详情查看http://note.youdao.com/s/956zlZZ

九、对启用和停用 WLAN 实施了限制

以 Android 10 或更高版本为目标平台的应用无法启用或停用 WLAN。`WifiManager.setWifiEnabled()` 方法始终返回 `false` 。

如果您需要提示用户启用或停用 WLAN，请使用设置面板。

网络操作API有变更，不过我们貌似没有这方面的处理，都是获取网络状态跟标识

十、其他

1、Android10上对折叠屏设备有了更好的支持，对于有折叠屏适配的需求，可以参看为可折叠设备构建应用 和 华为折叠屏应用开发指导。

https://developer.android.google.cn/guide/topics/ui/foldables

https://developer.huawei.com/consumer/cn/doc/90101

2、kotilin 代码适配