
Table of Contents

前言	1.1
Android原生申请权限的用法事例	1.2
RxPermissions申请权限的用法事例	1.3
AndPermission申请权限的用法事例	1.4
EasyPermissions申请权限的用法事例	1.5
PermissionsDispatcher申请权限的用法事例	1.6
框架的对比和选择	1.7
框架的进一步封装	1.8
权限申请时的状态描述	1.8.1

前言

由于**Google在Android-6.0¹**之后对app使用权限作了进一步限制，部分权限需要进行运行时申请，所以无可避免的我们的项目中都会需要增加**运行时权限²**的逻辑处理。

Android系统提供了对应的api给开发者来实现该逻辑，但是如果我们从零开始来实现整套逻辑，势必会耗费我们大量的时间和精力。且目前已经存在很多成熟的运行时权限框架如：[RxPermissions](#)、[AndPermission](#)、[EasyPermissions](#)和[PermissionsDispatcher](#)等。

这里我会以申请 **摄像头权限 (Manifest.permission.CAMERA)** 来对这些列举的框架的用法和优点进行简单说明，且针对最终采取的 框架进行进一步的封装，以更方便和更适用于目前的项目。

¹ 国产ROM部分手机在6.0以下也存在运行时权限申请。但是低于6.0的手机目前已经微乎其微，且这类ROM更是极少一部分，这里暂不讨论。如果确实有需求的请参考[AndPermission](#)和[permissions4m](#)等。

² Runtime权限：运行时权限，是指在app运行过程中，赋予app的权限。这个过程中，会显示明显的权限授予界面，让用户决定是否授予权限。如果app的targetSdkVersion是22（Lollipop MR1）及以下，dangerous权限是安装时权限，否则dangerous权限是运行时权限。如果一个app的targetSdkVersion是23（或者23以上），那么该app所申请的所有dangerous权限都是运行时权限。如果运行在Android 6.0的环境中，该app在运行时必须主动申请这些dangerous权限（调用requestPermissions()），否则该app没有获取到dangerous权限。

Android原生申请摄像头权限

这里使用Android原生api 源码，进行摄像头权限的申请

第一步

在**AndroidManifest**中填写你要申请的权限

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

第二步

在我们需要在打开相机的位置添加申请权限的代码，如：

```
public void requestCamera() {
    // checkSelfPermission 判断是否已经申请了此权限
    if (ContextCompat.checkSelfPermission(this, Manifest.permission.CAMERA)
        != PackageManager.PERMISSION_GRANTED) {
        //申请对应权限
        ActivityCompat.requestPermissions(this, new String[]{Manifest.permission.CAMERA}, 1);
    } else {
        //do something
    }
}
```

第三步

在onRequestPermissionsResult回调方法中去进行处理。代码如下：

```
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == 1) { //code需要是之前申请权限时使用的值
        for (int i = 0; i < permissions.length; i++) {
            if (grantResults[i] == PackageManager.PERMISSION_GRANTED) {
                //申请成功,do something
            } else {
                //申请失败, 判断是否被禁止了弹窗提示
                if (!ActivityCompat.shouldShowRequestPermissionRationale(this,
                    Manifest.permission.CAMERA)) {
                    //弹出设置框, 跳转系统对应的权限设置
                }
            }
        }
    }
}
```

```
        }  
    }  
}  
}
```

RxPermissions申请摄像头权限

这里使用[RxPermissions GITHUB](#), 进行摄像头权限的申请

第一步

在**AndroidManifest**中填写你要申请的权限

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

第二步

在项目中依赖**RxPermissions**, 在项目的 `build.gradle` 中实现依赖

```
implementation 'com.github.tbruyelle:rxpermissions:0.10.2'
```

第三步

在我们需要在打开相机的位置添加申请权限的代码, 如:

```
public void rxRequestCamera() {
    RxPermissions rxPermissions = new RxPermissions(this);
    rxPermissions.request(Manifest.permission.CAMERA).subscribe(new DisposableObserver<Boolean>() {
        @Override
        public void onNext(Boolean aBoolean) {
            if (aBoolean) {
                // do something
            } else {
                // 权限被拒绝
                //申请失败, 判断是否被禁止了弹窗提示
                if (!ActivityCompat.shouldShowRequestPermissionRationale(SettingsActivity.this,
                        Manifest.permission.CAMERA)) {
                    //弹出设置框, 跳转系统对应的权限设置
                }
            }
        }
        @Override
        public void onError(Throwable e) {
            // do something
        }
    });
}
```

```
    @Override
    public void onComplete() {
        }
    });
}
```

AndPermission申请摄像头权限

这里使用[AndPermission GITHUB](#), 进行摄像头权限的申请

第一步

在**AndroidManifest**中填写你要申请的权限

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

第二步

在项目中依赖**AndPermission**, 在项目的 `build.gradle` 中实现依赖

```
implementation 'com.yanzhenjie:permission:2.0.0-rc4'
```

第三步

在我们需要在打开相机的位置添加申请权限的代码, 如:

```
private void requestPermission() {
    AndPermission.with(this)
        .permission(Manifest.permission.CAMERA)
        // 准备方法, 和 okhttp 的拦截器一样, 在请求权限之前先运行改方法, 已经拥有权限
        // 不会触发该方法
        .rationale((context, permissions, executor) -> {
            // 此处可以选择显示提示弹窗
            executor.execute();
        })
        // 用户给权限了
        .onGranted(permissions -> //do something)
        // 用户拒绝权限, 包括不再显示权限弹窗也在此列
        .onDenied(permissions -> {
            // 判断用户是不是不再显示权限弹窗了, 若不再显示的话进入权限设置页
            if (AndPermission.hasAlwaysDeniedPermission(MainActivity.this,
                permissions)) {
                // 打开权限设置页
                AndPermission.permissionSetting(MainActivity.this).execute();
            }
            return;
        })
    })
    .start();
}
```


AndPermission申请摄像头权限

这里使用[EasyPermissions GITHUB](#), 进行摄像头权限的申请

第一步

在AndroidManifest中填写你要申请的权限

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

第二步

在项目中依赖**EasyPermissions**, 在项目的 build.gradle 中实现依赖

```
implementation 'pub.devrel:easypermissions:2.0.0'
```

第三步

在我们需要在打开相机的位置添加申请权限的代码, 如:

```
//参数为请求权限的code
@AfterPermissionGranted(1)
private void methodRequiresPermission() {
    if (EasyPermissions.hasPermissions(this, Manifest.permission.CAMERA)) {
        // Already have permission, do the thing
        // ...
    } else {
        // Do not have permissions, request them now
        //这个方法是用户在拒绝权限之后, 再次申请权限, 才会弹出自定义的dialog, 详情可以查看下
        //源码 shouldShowRequestPermissionRationale()方法
        PermissionRequest request = new PermissionRequest.Builder(SettingActivi
ty.this, 1, Manifest.permission.CAMERA)
            .setRationale(R.string.camera)
            .setPositiveButtonText(R.string.yes)
            .setNegativeButtonText(R.string.no)
            .build();
        EasyPermissions.requestPermissions(request);
    }
}
```

第四步

在onRequestPermissionsResult回调方法中去进行处理。代码如下:

```
//接受系统权限的处理，这里交给EasyPermissions来处理，回调到 EasyPermissions.PermissionCall  
backs接口  
    @Override  
    public void onRequestPermissionsResult(int requestCode, String[] permissions,  
int[] grantResults) {  
        super.onRequestPermissionsResult(requestCode, permissions, grantResults);  
        EasyPermissions.onRequestPermissionsResult(requestCode, permissions, grantsResults,  
this)//注意这个this，内部对实现该方法进行了查询，所以没有this的话，回调结果的方法不  
生效  
    }
```

第五步

实现 EasyPermissions.RationaleCallbacks,重写以下两个方法 (弹出自定义dialog时，用户点击接受和拒绝按钮)

```
@Override  
public void onRationaleDenied(requestCode: Int) {  
    //如果用户点击永远禁止，这个时候就需要跳到系统设置页面去手动打开了  
    if (EasyPermissions.somePermissionPermanentlyDenied(this, perms)) {  
        AppSettingsDialog.Builder(this).build().show()  
    }  
  
}  
  
@Override  
public void onRationaleAccepted(requestCode: Int) {  
    //do something  
}
```

RxPermissions申请摄像头权限

这里使用[PermissionsDispatcher GITHUB](#), 进行摄像头权限的申请

第一步

在**AndroidManifest**中填写你要申请的权限

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

第二步

在项目中依赖**PermissionsDispatcher**, 在项目的 `build.gradle` 中实现依赖

```
implementation "org.permissionsdispatcher:permissionsdispatcher:${latest.version}"
annotationProcessor "org.permissionsdispatcher:permissionsdispatcher-processor:
${latest.version}"
```

第三步

在我们需要在打开相机的Activity实现如下的代码:

```
@RuntimePermissions
public class MainActivity extends AppCompatActivity {

    @NeedsPermission(Manifest.permission.CAMERA)
    void showCamera() {
        getSupportFragmentManager().beginTransaction()
            .replace(R.id.sample_content_fragment, CameraPreviewFragment.newInstance())
            .addToBackStack("camera")
            .commitAllowingStateLoss();
    }

    @OnShowRationale(Manifest.permission.CAMERA)
    void showRationaleForCamera(final PermissionRequest request) {
        new AlertDialog.Builder(this)
            .setMessage(R.string.permission_camera_rationale)
            .setPositiveButton(R.string.button_allow, (dialog, button) -> request.proceed())
            .setNegativeButton(R.string.button_deny, (dialog, button) -> request.cancel())
            .show();
    }
}
```

```

    }

    @OnPermissionDenied(Manifest.permission.CAMERA)
    void showDeniedForCamera() {
        Toast.makeText(this, R.string.permission_camera_denied, Toast.LENGTH_SHORT)
.show();
    }

    @OnNeverAskAgain(Manifest.permission.CAMERA)
    void showNeverAskForCamera() {
        Toast.makeText(this, R.string.permission_camera_neverask, Toast.LENGTH_SHORT)
.show();
    }
}

```

第四步

在我们需要在该Activity实现如下的代码，以代理权限请求的返回状态：

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    findViewById(R.id.button_camera).setOnClickListener(v -> {
        // NOTE: delegate the permission handling to generated method
        MainActivityPermissionsDispatcher.showCameraWithPermissionCheck(this);
    });
}

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    // NOTE: delegate the permission handling to generated method
    MainActivityPermissionsDispatcher.onRequestPermissionsResult(this, requestCode,
    grantResults);
}

```

使用须知：

被权限注解的方法不能用private修饰,详情看下图:

1	@RuntimePermissions	✓	注解在其内部需要使用运行时权限的Activity或Fragment上
2	@NeedsPermission	✓	注解在需要调用运行时权限的方法上，当用户给予权限时会执行该方法
3	@OnShowRationale		注解在用于向用户解释为什么需要调用该权限的方法上，只有当第一次请求权限被用户拒绝，下次请求权限之前会调用
4	@OnPermissionDenied		注解在当用户拒绝了权限请求时需要调用的方法上
5	@OnNeverAskAgain		注解在当用户选中了授权窗口中的 不再询问 复选框后并拒绝了权限请求时需要调用的方法，一般可以向用户解释为何申请此权限，并根据实际需求决定是否再次弹出权限请求对话框

对各个框架的简单说明

通过对 摄像头权限 的申请例子，我们已经了解各个框架的简单用法，现在我们列举一些各个框架明显的优缺点。

原生权限申请

基于Android原生API的运行时权限申请

优点

- 不需要再进行另外的lib引入，减少项目的包大小

缺点

- 针对每个功能的权限申请，都需要设置相应的REQUEST_CODE，且需要在 `onRequestPermissionsResult` 中进行判断
- 只能在Activity和Fragment中使用
- 需要自己判断运行环境的版本
- 需要自己针对部分国产ROM在低于Android 6.0时的权限申请进行判断
- 需要自己对用户禁止权限申请弹框进行判断

RxPermissions权限申请

基于RxPermissions的运行时权限申请

优点

- 开发者不用担心Android运行环境的版本，如果系统是Android 6.0之前的版本，RxPermissions返回的结果是，app请求的每个权限都被允许（granted）
- 不需要为每个功能的权限申请，都需要设置相应的REQUEST_CODE
- 不需要重写 `onRequestPermissionsResult`，且在其中进行授权成功与否判断
- 具备Rx（RxJava）的特性，例如可以使用链式操作，可以执行filter操作，transform操作，等等
- 对用户禁止权限申请弹框做了实现

缺点

- 需要引入依赖
- 初始化需要传入Activity参数

AndPermission权限申请

基于AndPermission的运行时权限申请

优点

- 开发者不用担心Android运行环境的版本，如果系统是**Android 6.0**之前的版本(非特殊ROM)，AndPermission返回的结果是，app请求的每个权限都被允许 (granted)
- 不需要为每个功能的权限申请，都需要设置相应的REQUEST_CODE
- 不需要重写 `onRequestPermissionsResult`，且在其中进行授权成功与否判断
- 在任何地方都可以直接调用
- 针对部分国产ROM在低于**Android 6.0**时的权限申请进行了判断
- 对权限申请时的自定义对话框作了实现
- 对跳转app对应权限设置页面作了实现
- 对用户禁止权限申请弹框做了实现

缺点

- 需要引入依赖
- 做了自定义对话框的实现，所以包体积相对其他会变大，且需要自定义对话框时需要满足对应的规则

EasyPermissions权限申请

基于**EasyPermissions**的运行时权限申请

优点

- 开发者不用担心Android运行环境的版本，如果系统是**Android 6.0**之前的版本(非特殊ROM)，EasyPermissions返回的结果是，app请求的每个权限都被允许 (granted)
- 将权限请求的逻辑从Activity或者Fragment剥离出来做了一个代理操作
- 对用户禁止权限申请弹框做了实现

缺点

- 需要引入依赖
- 每个功能的权限申请，都需要设置相应的REQUEST_CODE
- 需要重写 `onRequestPermissionsResult`
- 做了自定义对话框的实现，所以包体积相对其他会变大，且需要自定义对话框时需要满足对应的规则
- 获取对应的情况需要实现相应的接口

PermissionsDispatcher权限申请

基于**PermissionsDispatcher**的运行时权限申请

优点

- 开发者不用担心Android运行环境的版本，如果系统是**Android 6.0**之前的版本(非特殊ROM)，PermissionsDispatcher返回的结果是，app请求的每个权限都被允许 (granted)
- 对用户禁止权限申请弹框做了实现
- 使用注解对权限申请中的各个情况进行了隔离，让开发者简单明了的处理各种状况

缺点

- 需要引入依赖
- 每个功能的权限申请，都需要设置相应的注解
- 需要重写 `onRequestPermissionsResult` ,里面写上生成类的 `ActivityNamePermissionsDispatcher.onRequestPermissionsResult`，此方法需要编译之后才会产生
- 需要调用权限的地方执行 `ActivityNamePermissionsDispatcher.XXXWithCheck()`，此方法需要编译之后才会产生

总结

通过上述的归纳，不难发现，如果需要实现更多系统的兼容，我们首选应该是**AndPermission**，且其实现了权限提示框的集成和跳转对应权限的设置页面的功能。但是由于目前国内大部分手机都已经属于**Android 6.0**以上，所以对这部分特殊ROM的判断可以暂时不占太大的权重；其次由于RxJava的优越性，基本大部分项目都会基于其开发；最后考虑到权限框架就应该只处理权限相关的问题，而不需要实现对话框的提示，最终选择了 **RxPermissions**。

下面我们对 **RxPermissions** 进行进一步的封装，以让它更好的适用于我们的项目。

基于RxPermissions的权限申请工具类

首先我们需要明确工具类的用途：

- 判断单个权限是否已经被授权
- 申请单个或多个权限且返回申请的状态

其次我们需要确认我们的工具类的使用范围：

- 是否可以在 Activity 或 Fragment 使用
- 是否可以在其他组件中使用

最后根据这些条件我们来进行封装

单个权限是否被授予

一般来说该方法主要用来提示权限未被授予，所以直接返回状态即可。

```
/**
 * 判断单个权限是否已经授予
 * 不需要传递FragmentActivity对象，使得方法可以在任何地方调用
 */
public static boolean rxGranted(String permission) {
    return rxGranted((FragmentActivity) AppManager.getInstance().currentActivity(), permission);
}

/**
 * 判断单个权限是否已经授予
 *
 * @param activity RxPermissions需要用来实例化的FragmentActivity对象
 * @param permission 需要申请的权限，参考{@link android.Manifest.permission#CAMERA}
 */
public static boolean rxGranted(FragmentActivity activity, String permission) {
    RxPermissions rxPermissions = new RxPermissions(activity);
    return rxPermissions.isGranted(permission);
}
```

单个权限的申请

申请权限，我们一般需要知道以下几种情况：

- 权限是否已被授予
- 权限未授予时，是否弹出了系统的权限申请框

这里我们需要一个对象PermissionState来描述这些属性。

```

    /**
     * 运行时权限申请
     *
     * @param activity RxPermissions需要用来实例化的FragmentActivity对象
     * @param permission 需要申请的权限数据, 参考{@link android.Manifest.permission#CAMERA}
     */
    public static Observable<PermissionState> rxPermission(FragmentActivity activity, String permission) {
        PermissionState permissionState = new PermissionState(permission);
        return rxPermission(activity, permissionState);
    }

    /**
     * 运行时权限申请
     * 第一次请求权限时ActivityCompat.shouldShowRequestPermissionRationale=false;
     * 第一次请求权限被禁止, 但未选择【不再提醒】ActivityCompat.shouldShowRequestPermissionRationale=true;
     * 允许某权限后ActivityCompat.shouldShowRequestPermissionRationale=false;
     * 禁止权限, 并选中【禁止后不再询问】ActivityCompat.shouldShowRequestPermissionRationale=false;
     */
    @SuppressLint("CheckResult")
    public static Observable<PermissionState> rxPermission(FragmentActivity activity, final PermissionState permission) {
        RxPermissions rxPermissions = new RxPermissions(activity);
        return rxPermissions.request(permission.getName())
            .flatMap(aBoolean -> {
                permission.setGranted(aBoolean);
                if (!aBoolean) {
                    // 如果禁止权限, 并且未选中【禁止后不再询问】
                    return rxPermissions.shouldShowRequestPermissionRationale(activity, permission.getName());
                }
                return Observable.just(ActivityCompat.shouldShowRequestPermissionRationale(activity, permission.getName()));
            } else {
                return Observable.just(true);
            }
        }).flatMap(it -> {
            permission.setNeedRationale(!it);
            return Observable.just(permission);
        });
    }

```



多个权限的申请

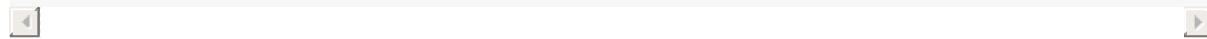
多个权限的申请依赖于单个权限申请实现, 且外部调用时, 直接调用该方法即可。

只返回权限申请的最终结果

```

    /**
     * 多个运行时权限申请，只会返回最终所有权限是否已经授予的状态
     *
     * @param activity RxPermissions需要用来实例化的FragmentActivity对象
     * @param permissions 需要申请的权限数据数组，参考{@link android.Manifest.permission
     #CAMERA}
     */
    public static Observable<Boolean> rxPermissions(FragmentActivity activity, String... permissions) {
        RxPermissions rxPermissions = new RxPermissions(activity);
        return rxPermissions.request(permissions);
    }

```



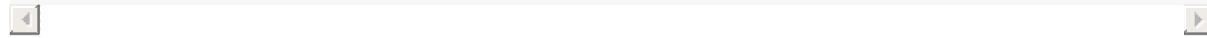
返回每个权限申请的状态对象

```

    /**
     * 多个运行时权限申请，每个权限都会返回对应的PermissionState对象
     */
    public static Observable<PermissionState> rxPermissionList(String... permissions) {
        return rxPermissionList((FragmentActivity) AppManager.getInstance().currentActivity(), permissions);
    }

    /**
     * 多个运行时权限申请，每个权限都会返回对应的PermissionState对象
     *
     * @param activity RxPermissions需要用来实例化的FragmentActivity对象
     * @param permissions 需要申请的权限数据数组，参考{@link android.Manifest.permission
     #CAMERA}
     */
    public static Observable<PermissionState> rxPermissionList(FragmentActivity activity, String... permissions) {
        return Observable.fromArray(permissions)
            .flatMap((Function<String, ObservableSource<PermissionState>>) s ->
            rxPermission(activity, s));
    }

```



描述Android权限的状态

包含Android权限的一些状态和状态的获取以及设置方法。 其中设置方法权限为Default(防止可以在工具类之外设置), 获取方法权限为public。

属性

字段	类型	说明
name	String	权限的名字,使用系统定义的字符串,</br>参考{@link android.Manifest.permission#CAMERA}
explain	String	申请权限失败时, 弹框给用户的说明
isGranted	boolean	权限是否已经获取
needRationale	boolean	是否权限申请失败并且用户关闭了申请权限时的提示

代码如下

```

/**
 * @Description:用来描述对应权限的状态
 * @Author: hekang
 * @CreateDate: 2019/6/26 16:09
 */
public class PermissionState implements Parcelable {

    /**
     * 权限的名字, 使用系统定义的字符串
     * 参考{@link android.Manifest.permission#CAMERA}
     */
    private String name;

    /**
     * 申请权限失败时, 弹框给用户的说明
     */
    private String explain;

    /**
     * 权限是否已经获取
     */
    private boolean isGranted;

    /**
     * 权限申请失败并且用户关闭了申请权限时的提示
     */
    private boolean needRationale;
}

```

```
public PermissionState(String name) {
    this(name, "");
}

public PermissionState(String name, String explain) {
    this.name = name;
    this.explain = explain;
}

public String getName() {
    return name;
}

void setName(String name) {
    this.name = name;
}

public String getExplain() {
    return explain;
}

void setExplain(String explain) {
    this.explain = explain;
}

public boolean isGranted() {
    return isGranted;
}

void setGranted(boolean granted) {
    isGranted = granted;
}

public boolean isNeedRationale() {
    return needRationale;
}

void setNeedRationale(boolean needRationale) {
    this.needRationale = needRationale;
}

@Override
public String toString() {
    return "PermissionState{" +
        "name='" + name + '\'' +
        ", explain='" + explain + '\'' +
        ", isGranted=" + isGranted +
        ", needRationale=" + needRationale +
        '}';
}
```

```
@Override
public int describeContents() {
    return 0;
}

@Override
public void writeToParcel(Parcel dest, int flags) {
    dest.writeString(this.name);
    dest.writeString(this.explain);
    dest.writeByte(this.isGranted ? (byte) 1 : (byte) 0);
    dest.writeByte(this.needRationale ? (byte) 1 : (byte) 0);
}

protected PermissionState(Parcel in) {
    this.name = in.readString();
    this.explain = in.readString();
    this.isGranted = in.readByte() != 0;
    this.needRationale = in.readByte() != 0;
}

public static final Parcelable.Creator<PermissionState> CREATOR = new Parcelable.Creator<PermissionState>() {
    @Override
    public PermissionState createFromParcel(Parcel source) {
        return new PermissionState(source);
    }

    @Override
    public PermissionState[] newArray(int size) {
        return new PermissionState[size];
    }
};
```