

Activity Results API由浅入深

CoderPig 郭霖 2022-08-19 08:00 发表于江苏



点击上方蓝字即可关注
关注后可查看所有经典文章

/ 今日科技快讯 /

17日腾讯公布了截至2022年6月30日止第二季及上半年未经审核综合业绩。腾讯控股2022年Q2营收1340亿元。财报显示，腾讯第二季度总收入为人民币1340亿元（200亿美元），同比下降3%。按非国际财务报告准则，期内本公司权益持有人应占盈利为人民币281亿元（42亿美元），同比下降17%。

/ 作者简介 /

本篇文章转自coder_pig的博客，文章主要分享了Activity Results API的使用与源码分析，相信会对大家有所帮助！

原文地址：

<https://juejin.cn/post/7094904353667940388>

/ 引言 /

不是什么新玩意了，恰逢最近拆公司项目的BaseFragment时看到介个：

```

@Override
public void onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults) {
    super.onRequestPermissionsResult(requestCode, permissions, grantResults);
    if (requestCode == REQUEST_CODE_PERMISSION) {
        if (verifyPermissions(grantResults)) {
            permissionSuccess(REQUEST_CODE_PERMISSION);
        } else {
            permissionFail(REQUEST_CODE_PERMISSION);
            if (mConfirmListener == null) {
                showTipsDialog();
            } else {
                showTipsDialog(mConfirmListener);
            }
        }
    }
}
}

```

@稀土掘金技术社区

下划线? Deprecated? 点开源码看下啥原因, 有啥替代方案:

```

Deprecated use registerForActivityResult(ActivityResultContract,
    ActivityResultCallback) passing in a androidx.activity.result.contract.
    ActivityResultContracts.RequestMultiplePermissions object for the
    ActivityResultContract and handling the result in the callback.

```

@稀土掘金技术社区

注释说这种写法out了, 可在 ActivityResultContract 中传入一个 RequestMultiplePermissions 对象, 并在回调中处理结果。除此之外, 诸如 startActivityForResult()、onActivityResult() 等都过时了。

搜了一波官方文档《**获取 activity 的结果**》

(<https://developer.android.google.cn/training/basics/intents/result>), 没找到具体原因, 笔者猜测官方旨在帮助开发者: 减少样板代码、解耦。

以前跳转新页面回传数据, 得经历这三步:

- 定义REQUEST_CODE, 同一个页面有多个数据时, 避免重复;
- 调用 startActivityForResult(Intent, REQUEST_CODE)
- 重写 onActivityResult(), 判断requestCode和resultCode, 拿到值后执行后续逻辑;

简易代码示例如下:

```

class FirstActivity : AppCompatActivity() {
    companion object {
        const val REQUEST_CODE = 1
    }
    private lateinit var mContentTv: TextView
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first)
        mContentTv = findViewById(R.id.tv_content)
        mContentTv.setOnClickListener { it: View!
            startActivityForResult(Intent( packageContext: this, SecondActivity::class.java), REQUEST_CODE)
        }
    }

    override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
        super.onActivityResult(requestCode, resultCode, data)
        if (resultCode == Activity.RESULT_OK && requestCode == REQUEST_CODE) {
            mContentTv.text = "页面跳转值为: ${data?.getStringExtra( name: "value")}"
        }
    }
}

```

@稀土掘金技术社区

新页面，一个简单的 setResult， 然后 finish()：

```

class SecondActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second)
        findViewById<TextView>(R.id.tv_content).setOnClickListener { it: View!
            setResult(
                Activity.RESULT_OK,
                Intent().putExtra( name: "value", value: "${System.currentTimeMillis()}")
            )
            finish()
        }
    }
}

```

@稀土掘金技术社区

而使用 **Activity Result API**，你只需定义一个函数，然后 launch() 一下：

```

private val getBackValue = registerForActivityResult(ActivityResultContracts.StartActivityForResult()) { it: ActivityResult?
    if (it?.resultCode == Activity.RESULT_OK) {
        mContentTv.text = "页面跳转值为: ${it?.data?.getStringExtra( name: "value")}"
    }
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_first)
    mContentTv = findViewById(R.id.tv_content)
    mContentTv.setOnClickListener { it: View!
        getBackValue.launch(Intent( packageContext: this, SecondActivity::class.java))
    }
}

```

@稀土掘金技术社区

跳转页面不用动，可以看到：移除了 onActivityResult() 的重写，少写了一个 REQUEST_CODE。看着好像是简洁了一些，对了网上一堆旧文章说要依赖 activity-ktx 和 fragment-ktx，笔者使用 1.4.0 的 activity 和 fragment 包，发现已经内置这些东西了，可以不依赖，技术迭代飞快，建议读者使用时以官方文档为准。

/ 原理浅探 /

```

@NonNull
@Override
public final <I, O> ActivityResultLauncher<I> registerForActivityResult(
    @NonNull ActivityResultContract<I, O> contract,
    @NonNull ActivityResultCallback<O> callback) {
    return registerForActivityResult(contract, mActivityResultRegistry, callback);
}

```

@稀土掘金技术社区

Activity Results API的使用非常简单 (侧面说明封装得好)，它由三个要素组成：**启动器 + 协定 + 结果回调**。

① ActivityResultLauncher → 启动器

registerForActivityResult() 的返回值，用于：**承载启动对象与返回对象**。

```

public abstract class ActivityResultLauncher<I> {

    /** Executes on {@link ActivityResultContract}. ... */
    public void launch(@SuppressWarnings("UnknownNullness") I input) { launch(input, options: null); }

    /** Executes on {@link ActivityResultContract}. ... */
    public abstract void launch(@SuppressWarnings("UnknownNullness") I input,
        @Nullable ActivityOptionsCompat options);

    /** Unregisters this launcher, releasing the underlying result callback, and any references ... */
    @MainThread
    public abstract void unregister();

    /** Get the {@link ActivityResultContract} that was used to create this launcher. ... */
    @NonNull
    public abstract ActivityResultContract<I, ?> getContract();
}

```

@稀土掘金技术社区

② ActivityResultContract → 协定/契约

第一个入参，协定的是：**所需的输入类型** 和 **结果的输出类型**。


```

public abstract class ActivityResultContract<I, O> {

    /** Create an intent that can be used for {@link Activity#startActivityForResult} */
    public abstract @NonNull Intent createIntent(@NonNull Context context,
        @SuppressWarnings("UnknownNullness") I input);

    /** Convert result obtained from {@link Activity#onActivityResult} to O */
    @SuppressWarnings("UnknownNullness")
    public abstract O parseResult(int resultCode, @Nullable Intent intent);

    /** An optional method you can implement that can be used to potentially provide a result in ...
    public @Nullable SynchronousResult<O> getSynchronousResult(
        @NonNull Context context,
        @SuppressWarnings("UnknownNullness") I input) {
        return null;
    }

    /** The wrapper for a result provided in {@link #getSynchronousResult} ... */
    public static final class SynchronousResult<T> {
        private final @SuppressWarnings("UnknownNullness") T mValue;

        Create a new result wrapper
        Params: value – the result value

        public SynchronousResult(@SuppressWarnings("UnknownNullness") T value) { this.mValue = value; }

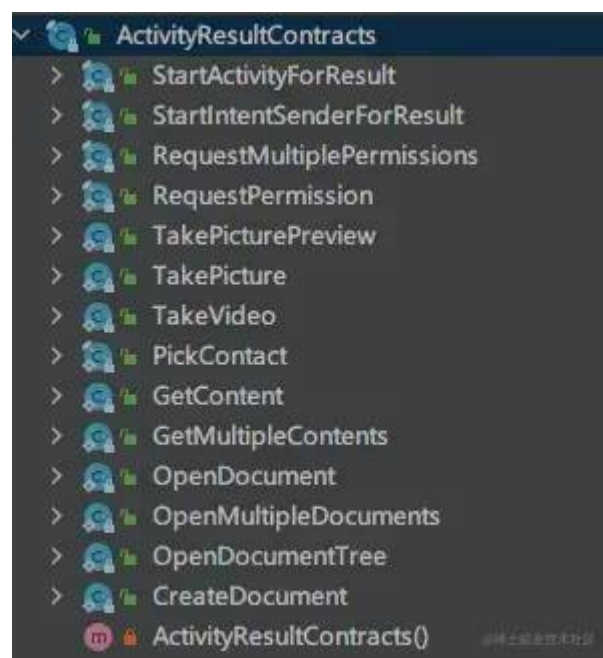
        Returns: the result value

        public @SuppressWarnings("UnknownNullness") T getValue() { return mValue; }
    }
}

```

@稀土掘金技术社区

方法看着有点懵？没关系，找两个个具体实现类看看就知道了。ActivityResultContracts 给我们提供了一些 常用的协定，拿来即用：



罗列下各自的作用（直接看代码实现也能猜到干嘛的~）：

- `StartActivityForResult()` → 通用协定, 不做任何转换, `Intent`作为输入, `ActivityResult`作为输出;
- `StartIntentSenderForResult()` → 内部`Intent`请求;
- `RequestMultiplePermissions()` → 请求一组权限;
- `RequestPermission()` → 请求单个权限;
- `TakePicturePreview()` → 调用`MediaStore.ACTION_IMAGE_CAPTURE`拍照, 返回`Bitmap`图片;
- `TakePicture()` → 调用`MediaStore.ACTION_IMAGE_CAPTURE`拍照, 并将图片保存在给定`Uri`, 返回`true`表示保存成功;
- `TakeVideo()` → 调用`MediaStore.ACTION_VIDEO_CAPTURE`录制, 并将视频保存在给定`Uri`, 返回`true`表示保存成功;
- `PickContact()` → 调用通讯录APP获取联系人;
- `GetContent()` → 提示选择一条内容, 返回一个通过
`ContentResolver#openInputStream(Uri)`访问原生数据的`Uri`地址 (`content://`形式)。默认情况下, 它增加了 `Intent#CATEGORY_OPENABLE`, 返回可以表示流的内容;
- `GetMultipleContents()` → 提示选择多条内容;
- `OpenDocument()` → 提示选择文档, 返回`Uri`;
- `OpenMultipleDocuments()` → 提示用户选择多个文档, 以`List`形式, 返回他们的`Uri`;
- `OpenDocumentTree()` → 提示用户选择目录, 返回`Uri`;
- `CreateDocument()` → 提示用户选择创建新文档的路径, 返回已创建项目的`Uri`。

附: 调用文件选择器, 获取指定类型的文件, 可在`launch()`方法里使用`mimetype`指定调用文件类型, 文件`mimetype`对照表可参见: **media-types**

(<http://www.iana.org/assignments/media-types/media-types.xhtml>)

挑两个协议看看具体代码实现, 先是 `StartActivityForResult`:

```

public static final class StartActivityResult
    extends ActivityResultContract<Intent, ActivityResult> {
    /** Key for the extra containing a (return Android OS bundle) generated from ... */
    public static final String EXTRA_ACTIVITY_OPTIONS_BUNDLE = "androidx.activity.result"
        + ".contract.extra.ACTIVITY_OPTIONS_BUNDLE";

    @NonNull
    @Override
    public Intent createIntent(@NonNull Context context, @NonNull Intent input) {
        return input;
    }

    @NonNull
    @Override
    public ActivityResult parseResult(
        int resultCode, @Nullable Intent intent) {
        return new ActivityResult(resultCode, intent);
    }
}

```

继承ActivityResultContract 泛型指定了输入类型 输出类型

接受Context和输入内容作为参数

构造后返回一个和startActivityResult()配合使用的Intent

这里没做处理，所以直接返回

根据指定resultCode和Intent生成输出内容

这里返回了一个ActivityResult实例

@稀土掘金技术社区

返回类型ActivityResult实现了Parcelable序列化接口，定义了需要用到的两个字段：
mResultCode 和 mData。

```

public final class ActivityResult implements Parcelable {
    private final int mResultCode;
    @Nullable
    private final Intent mData;

    /** Create a new instance ... */
    public ActivityResult(int resultCode, @Nullable Intent data) {
        mResultCode = resultCode;
        mData = data;
    }

    ActivityResult(Parcel in) {
        mResultCode = in.readInt();
        mData = in.readInt() == 0 ? null : Intent.CREATOR.createFromParcel(in);
    }

    Returns: the resultCode
    public int getResultCode() { return mResultCode; }
}

```

@稀土掘金技术社区

接着是 TakePicturePreview:

```

public static class TakePicturePreview extends ActivityResultContract<Void, Bitmap> {

    @CallSuper
    @NonNull
    @Override
    public Intent createIntent(@NonNull Context context, @Nullable Void input) {
        return new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
    }

    @Nullable
    @Override
    public final SynchronousResult<Bitmap> getSynchronousResult(@NonNull Context context,
        @Nullable Void input) {
        return null;
    }

    @Nullable
    @Override
    public final Bitmap parseResult(int resultCode, @Nullable Intent intent) {
        if (intent == null || resultCode != Activity.RESULT_OK) return null;
        return intent.getParcelableExtra(name: "data");
    }
}

```

@稀土掘金技术社区

所以 ActivityResultContract 中的函数意义分别是：

- createIntent(context: Context, input: I): Intent → 创建用于 startActivityForResult 的 intent对象；
- parseResult (resultCode: Int, intent: Intent?): O → 对 onActivityResult 的结果进行转换；
- getSynchronousResult() → 可选，处理一些不需要启动Activity就能知道预期结果的场景，如RequestPermission会用到；

了解函数意义后，如果你觉得内置协定满足不了你，完全可以自定义一波，官方示例如下：

```

class PickRingtone : ActivityResultContract<Int, Uri?>() {
    override fun createIntent(context: Context, ringtoneType: Int) =
        Intent(RingtoneManager.ACTION_RINGTONE_PICKER).apply {
            putExtra(RingtoneManager.EXTRA_RINGTONE_TYPE, ringtoneType)
        }

    override fun parseResult(resultCode: Int, result: Intent?) : Uri? {
        if (resultCode != Activity.RESULT_OK) {
            return null
        }
        return result?.getParcelableExtra(RingtoneManager.EXTRA_RINGTONE_PICKED_URI)
    }
}

```


③ ActivityResultCallback → 结果回调

第二个入参，见名知意：启动Activity并返回当前Activity时的 结果回调。

```
public interface ActivityResultCallback<T> {
    // Called when result is available
    void onActivityResult(@SuppressWarnings("UnknownNullness") T result);
}
```

@稀土掘金技术社区

就定义了一个回调方法，Activity Result API 又是 模版方法模式 封装的思想体现，开发仔按需注入 协定类型 和 结果回调 即可，无需关注底层细节。巴适得很！



杰哥当然不会止步于调别人写好的API，接着再探一波更深层次的原理，弄清楚整条调用链路。

/ 原理再探 /

在Activity、Fragment中可以直接使用 registerForActivityResult()，是因为 ComponentActivity 和 Fragment 都实现了 ActivityResultCaller 接口。

```
public interface ActivityResultCaller {
    /** Register a request to {@link Activity#startActivityForResult start an activity for result}, ...*/
    @NonNull
    <I, O> ActivityResultLauncher<I> registerForActivityResult(
        @NonNull ActivityResultContract<I, O> contract,
        @NonNull ActivityResultCallback<O> callback);

    /** Register a request to {@link Activity#startActivityForResult start an activity for result}, ...*/
    @NonNull
    <I, O> ActivityResultLauncher<I> registerForActivityResult(
        @NonNull ActivityResultContract<I, O> contract,
        @NonNull ActivityResultRegistry registry,
        @NonNull ActivityResultCallback<O> callback);
}
```

@稀土掘金技术社区

① Activity

先跟下 ComponentActivity#registerForActivityResult()：

```
@NonNull
@Override
public final <I, O> ActivityResultLauncher<I> registerForActivityResult(
    @NonNull final ActivityResultContract<I, O> contract,
    @NonNull final ActivityResultRegistry registry,
    @NonNull final ActivityResultCallback<O> callback) {
    return registry.register(
        key: "activity_rq#" + mNextLocalRequestCode.getAndIncrement(), lifecycleOwner: this, contract, callback);
}
```

第一个参数构造了一个key，规则：activity_rq# + 一个自增的AtomicInteger值，怪不得不用另外定义一个REQUEST_CODE，就能进行区分。

继续跟 ActivityResultRegistry#register()：

```

@NonNull
public final <I, O> ActivityResultLauncher<I> register(
    @NonNull final String key,
    @NonNull final LifecycleOwner lifecycleOwner,
    @NonNull final ActivityResultContract<I, O> contract,
    @NonNull final ActivityResultCallback<O> callback) {

    Lifecycle lifecycle = lifecycleOwner.getLifecycle(); 获取生命周期组件的Lifecycle

    if (lifecycle.getCurrentState().isAtLeast(Lifecycle.State.STARTED)) {
        throw new IllegalStateException("LifecycleOwner " + lifecycleOwner + " is "
            + "attempting to register while current state is "
            + lifecycle.getCurrentState() + ". LifecycleOwners must call register before "
            + "they are STARTED."); register() 要在组件处于STARTED状态或之前调用
    }

    registerKey(key); 注册key (为每个key生成一个唯一的整数, 然后存集合)
    LifecycleContainer lifecycleContainer = mKeyToLifecycleContainers.get(key);
    if (lifecycleContainer == null) { 看集合中是否有LifecycleContainer实例
        lifecycleContainer = new LifecycleContainer(lifecycle);
    }

    LifecycleEventObserver observer = new LifecycleEventObserver() {
        @Override 初始化观察者, 重写onStateChanged()
        public void onStateChanged(
            @NonNull LifecycleOwner lifecycleOwner,
            @NonNull Lifecycle.Event event) {
            if (Lifecycle.Event.ON_START.equals(event)) { 协定与回调关联
                mKeyToCallback.put(key, new CallbackAndContract<>(callback, contract));
                if (mParsedPendingResults.containsKey(key)) {
                    /unchecked/
                    final O parsedPendingResult = (O) mParsedPendingResults.get(key);
                    mParsedPendingResults.remove(key);
                    callback.onActivityResult(parsedPendingResult);
                }
                final ActivityResult pendingResult = mPendingResults.getParcelable(key);
                if (pendingResult != null) {
                    mPendingResults.remove(key);
                    callback.onActivityResult(contract.parseResult(
                        pendingResult.getResultCode(),
                        pendingResult.getData())); 调用协定的转换方法
                                                    结果作为参数传给回调
                }
            } else if (Lifecycle.Event.ON_STOP.equals(event)) {
                mKeyToCallback.remove(key);
            } else if (Lifecycle.Event.ON_DESTROY.equals(event)) {
                unregister(key);
                ON_STOP时移除 协定与回调关联
                ON_DESTROY是解绑, 避免内存泄露
            }
        }
    };

    lifecycleContainer.addObserver(observer); 添加观察者
    mKeyToLifecycleContainers.put(key, lifecycleContainer);
}

```

不需要启动
Activity就
知道预期结果
的场景处理,
如申请权限


常规处理

看着好像挺复杂, 其实不然, 核心就是:

添加了一个观察者, 当生命周期组件(传入的第2个参数) 状态切换到 ON_START 时执行回调。

然后下半段返回了一个 `ActivityResultLauncher` 实例：

```
return new ActivityResultLauncher<I>() {  
    @Override  
    public void launch(I input, @Nullable ActivityOptionsCompat options) {  
        Integer innerCode = mKeyToRc.get(key);  
        if (innerCode == null) {  
            throw new IllegalStateException("Attempting to launch an unregistered "  
                + "ActivityResultLauncher with contract " + contract + " and input "  
                + input + ". You must ensure the ActivityResultLauncher is registered "  
                + "before calling launch().");  
        }  
        mLaunchedKeys.add(key);  
        try {  
            onLaunch(innerCode, contract, input, options);  
        } catch (Exception e) {  
            mLaunchedKeys.remove(key);  
            throw e;  
        }  
    }  
};  
  
@Override  
public void unregister() { ActivityResultRegistry.this.unregister(key); }  
  
@NonNull  
@Override  
public ActivityResultContract<I, ?> getContract() { return contract; }  
};
```



跟下 `onLaunch()` 发现是一个抽象方法，具体实现在 `ComponentActivity` 中：


```

public <T, O> void onLaunch(
    final int requestCode,
    @NonNull ActivityResultContract<T, O> contract,
    I input,
    @Nullable ActivityOptionsCompat options) {
    ComponentActivity activity = ComponentActivity.this;

    // Immediate result path
    final ActivityResultContract.SynchronousResult<O> synchronousResult =
        contract.getSynchronousResult(activity, input);
    if (synchronousResult != null) {
        new Handler(Looper.getMainLooper()).post(new Runnable() {
            @Override
            public void run() {
                dispatchResult(requestCode, synchronousResult.getValue());
            }
        });
        return;
    }

    // Start activity path
    Intent intent = contract.createIntent(activity, input);
    Bundle optionsBundle = null;
    // If there are any extras, we should defensively set the classloader
    if (intent.getExtras() != null && intent.getExtras().getClassLoader() == null) {
        intent.setExtrasClassLoader(activity.getClassLoader());
    }
    if (intent.hasExtra(EXTRA_ACTIVITY_OPTIONS_BUNDLE)) {
        optionsBundle = intent.getBundleExtra(EXTRA_ACTIVITY_OPTIONS_BUNDLE);
        intent.removeExtra(EXTRA_ACTIVITY_OPTIONS_BUNDLE);
    } else if (options != null) {
        optionsBundle = options.toBundle();
    }
    if (ACTION_REQUEST_PERMISSIONS.equals(intent.getAction())) {
        // requestPermissions path
        String[] permissions = intent.getStringArrayExtra(EXTRA_PERMISSIONS);

        if (permissions == null) {
            permissions = new String[0];
        }
        ActivityCompat.requestPermissions(activity, permissions, requestCode);
    } else if (ACTION_INTENT_SENDER_REQUEST.equals(intent.getAction())) {
        IntentSenderRequest request =
            intent.getParcelableExtra(EXTRA_INTENT_SENDER_REQUEST);
        try {
            // startIntentSenderForResult path
            ActivityCompat.startIntentSenderForResult(activity, request.getIntentSender(),
                requestCode, request.getFillInIntent(), request.getFlagsMask(),
                request.getFlagsValues(), extraFlags: 0, optionsBundle);
        } catch (final IntentSender.SendIntentException e) {
            new Handler(Looper.getMainLooper()).post(new Runnable() {
                @Override
                public void run() {
                    dispatchResult(requestCode, RESULT_CANCELED,
                        new Intent().setAction(ACTION_INTENT_SENDER_REQUEST)
                            .putExtra(EXTRA_SEND_INTENT_EXCEPTION, e));
                }
            });
        }
    } else {
        // startActivityForResult path
        ActivityCompat.startActivityForResult(activity, intent, requestCode, optionsBundle);
    }
}

```

不需要启动Activity就能
预知结果的场景处理

调用协定的createIntent初始化Intent实例

Bundle初始化

如果是申请权限

请求权限

内部请求

startActivityForResult

©稀土掘金技术社区

到此，基本的调用链条就浮出水面了：

- ① `ComponentActivity` 内部初始化了一个 `ActivityResultRegistry` 实例，并重写了 `onLaunch()`;
- ② 开发者调用 `registerForActivityResult()` 最终调用 `ActivityResultRegistry.register()`，在此添加了一个观察者，当生命周期状态切换到 `ON_START` 时，执行协定 `Contract.parseResult()` 生成输出内容，并把结果作为参数传入回调 `callback.onActivityResult()` 中。
- ③ 注意！②是要生命周期发生改变才会触发的，开发者要调用 `ActivityResultLauncher.launch()` 才会发起跳转，其中回调了 `onLaunch()` 方法，在此调用了协定 `Contract.createIntent()` 返回一个和 `startActivityForResult()` 搭配使用的 `Intent` 实例。
- ④ 跳转目标Activity后返回此页面，生命周期发生改变，然后回调②中的相关代码。

描述起来好像有点拗口，不过你自己照着跟下源码就清楚了，接着跟下Fragment~

② Fragment

同样跟下 `registerForActivityResult()`:

```
@MainThread
@NonNull
@Override
public final <I, O> ActivityResultLauncher<I> registerForActivityResult(
    @NonNull final ActivityResultContract<I, O> contract,
    @NonNull final ActivityResultCallback<O> callback) {
    return prepareCallInternal(contract, new Function<Void, ActivityResultRegistry>() {
        @Override
        public ActivityResultRegistry apply(Void input) {
            if (mHost instanceof ActivityResultRegistryOwner) {
                return ((ActivityResultRegistryOwner) mHost).getActivityResultRegistry();
            }
            return requireActivity().getActivityResultRegistry();
        }
    }, callback);
}
```

注意是函数!

获取宿主Activity的getActivityResultRegistry() 函数

@稀土掘金技术社区

最终调用 `prepareCallInternal()`:

```

@NonNull
private <I, O> ActivityResultLauncher<I> prepareCallInternal(
    @NonNull final ActivityResultContract<I, O> contract,
    @NonNull final Function<Void, ActivityResultRegistry> registryProvider,
    @NonNull final ActivityResultCallback<O> callback) {
    // Throw if attempting to register after the Fragment is CREATED.
    if (mState > CREATED) { Fragment created后才能注册
        throw new IllegalStateException("Fragment " + this + " is attempting to "
            + "registerForActivityResult after being created. Fragments must call "
            + "registerForActivityResult() before they are created (i.e. initialization, "
            + "onAttach(), or onCreate()).");
    }
    // 保证修改ActivityResultLauncher引用时的线程安全
    final AtomicReference<ActivityResultLauncher<I>> ref = new AtomicReference<>();
    // We can't call generateActivityResultKey during initialization of the Fragment
    // since we need to wait for the mWho to be restored from saved instance state
    // so we'll wait until we have all the information needed to register to actually
    // generate the key and register.

    registerOnPreAttachListener() { 生成Fragment专属key
        final String key = generateActivityResultKey();
        ActivityResultRegistry registry = registryProvider.apply( input: null);
        ref.set(registry.register(key, LifecycleOwner Fragment.this, contract, callback));
    }; 获取宿主Activity的ActivityResultRegistry实例，并调用register()方法，并保存

    return new ActivityResultLauncher<I>() { ActivityResultLauncher引用
        @Override
        public void launch(I input, @Nullable ActivityOptionsCompat options) {
            ActivityResultLauncher<I> delegate = ref.get(); 这个引用是Activity中的哦！
            if (delegate == null) {
                throw new IllegalStateException("Operation cannot be started before fragment "
                    + "is in created state");
            }
            delegate.launch(input, options); 所以Fragment的launch其实是交由Activity代理完成~
        }

        @Override
        public void unregister() {
            ActivityResultLauncher<I> delegate = ref.getAndSet( newValue: null);
            if (delegate != null) {
                delegate.unregister();
            }
        }

        @NonNull
        @Override
        public ActivityResultContract<I, ?> getContract() { return contract; }
    };
}

```

@稀土掘金技术社区



思路也很简单，想办法拿到 宿主Activity中的ActivityResultRegistry实例，调它的 register() 拿到返回的 ActivityResultLauncher实例引用。最后返回 新的ActivityResultLauncher 实例，在launch()中调用前面那个Activity的 ActivityResultLauncher实例引用 的launch()方法。TM 调的是Activity的launch()，这一手 委托代理 玩挺6啊。



对了，这有个小细节，生命周期组件传入的是 **Fragment.this**，所以不用担心Fragment销毁没解绑导致的内存泄露问题。

③ 非Activity/Fragment 接收Activity结果

实现一个 LifecycleObserver 用于处理协定的注册和启动器的启动，代码示例如下：



调用处：



④ 亿点小细节：配置改变引起Activity重建的处理

在 ActivityResultRegistry 中还发现了个：

```
onSaveInstanceState(Bundle): void  
onRestoreInstanceState(Bundle): void
```

好家伙，连配置更改导致重建的场景也考虑到了吗？

```

public final void onSaveInstanceState(@NonNull Bundle outState) {
    outState.putIntegerArrayList(KEY_COMPONENT_ACTIVITY_REGISTERED_RCS,
        new ArrayList<>(mKeyToRc.values()));
    outState.putStringArrayList(KEY_COMPONENT_ACTIVITY_REGISTERED_KEYS,
        new ArrayList<>(mKeyToRc.keySet()));
    outState.putStringArrayList(KEY_COMPONENT_ACTIVITY_LAUNCHED_KEYS,
        new ArrayList<>(mLaunchedKeys));
    outState.putBundle(KEY_COMPONENT_ACTIVITY_PENDING_RESULTS,
        (Bundle) mPendingResults.clone());
    outState.putSerializable(KEY_COMPONENT_ACTIVITY_RANDOM_OBJECT, mRandom);
}

```

@稀土掘金技术社区

保存了：key(requestCode)相关的数据、处理结果、Random随机数实例。

```

public final void onRestoreInstanceState(@Nullable Bundle savedInstanceState) {
    if (savedInstanceState == null) {
        return;
    }
    ArrayList<Integer> rcs =
        savedInstanceState.getIntegerArrayList(KEY_COMPONENT_ACTIVITY_REGISTERED_RCS);
    ArrayList<String> keys =
        savedInstanceState.getStringArrayList(KEY_COMPONENT_ACTIVITY_REGISTERED_KEYS);
    if (keys == null || rcs == null) {
        return;
    }
    mLaunchedKeys =
        savedInstanceState.getStringArrayList(KEY_COMPONENT_ACTIVITY_LAUNCHED_KEYS);
    mRandom = (Random) savedInstanceState.getSerializable(KEY_COMPONENT_ACTIVITY_RANDOM_OBJECT);
    mPendingResults.putAll(
        savedInstanceState.getBundle(KEY_COMPONENT_ACTIVITY_PENDING_RESULTS));
    for (int i = 0; i < keys.size(); i++) {
        String key = keys.get(i);
        // Developers may have already registered with this same key by the time we restore
        // state, which caused us to generate a new requestCode that doesn't match what we're
        // about to restore. Clear out the new requestCode to ensure that we use the
        // previously saved requestCode. 恢复状态时，可能已经用了相同的key注册，
        if (mKeyToRc.containsKey(key)) { 会导致生成了一个与恢复内容不匹配的requestCode
            Integer newrequestCode = mKeyToRc.remove(key); 移除它，确保使用之前保存的那个
            // On the chance that developers have already called launch() with this new
            // requestCode, keep the mapping around temporarily to ensure the result is
            // properly delivered to both the new requestCode and the restored requestCode
            if (!mPendingResults.containsKey(key)) {
                mRcToKey.remove(newrequestCode);
            }
            // 如果已经用这个新的requestCode调用了launch()
            // 暂时保留，确保结果正确传递给新的requestCode
            // 和恢复的requestCode
        }
        bindRcKey(rcs.get(i), keys.get(i));
    }
}

```

@稀土掘金技术社区

requestCode和Result得以保留，Activity重建后，再把它们分发给新注册的Callback，避免了数据的丢失。



细节拉满了

⑤ 亿点补充：测试Activity结果调用

默认情况下，`registerForActivityResult()` 会自动使用Activity提供的 `ActivityResultRegistry`，而它还提供了一个重载，支持传入自己的 `ActivityResultRegistry` 实例。能干嘛？拦截结果调用进行测试啊，不会另外启动另一个Activity。代码示例如下：

```
class FirstActivity : AppCompatActivity() {
    private lateinit var mContentView: TextView

    // 自定义ActivityResultRegistry实例，重写onLaunch()
    private val mRegistry = object : ActivityResultRegistry() {
        override fun <I : Any?, O : Any?> onLaunch(
            requestCode: Int,
            contract: ActivityResultContract<I, O>,
            input: I,
            options: ActivityOptionsCompat?
        ) {
            // 不调用startActivityResult(), 而是调用dispatchResult()分发结果
            // 这里模拟了跳转新页面后，设置value后返回
            dispatchResult(
                requestCode,
                contract.parseResult(
                    Activity.RESULT_OK,
                    Intent().putExtra( name: "value", value: "${System.currentTimeMillis()}")
                )
            )
        }
    }

    // 调用registerForActivityResult传入自定义的mRegistry实例
    private val mResult =
        registerForActivityResult(ActivityResultContracts.StartActivityResult(), mRegistry) { it: ActivityResult?
            Log.e( tag: "Test", msg: "收到测试数据: ${it?.data?.getStringExtra( name: "value")}" )
        }

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first)
        mContentView = findViewById(R.id.tv_content)
        mContentView.setOnClickListener { it: View?
            mResult.launch(Intent( packageContext: this, SecondActivity::class.java))
        }
    }
}
```

@稀土掘金技术社区

/ 关于封装 /

Activity Results API 了解得七七八八了，接下来可以放心地用到项目中了，虽然它的API已经很简单易用了。但对于 喜欢偷懒到极致的开发仔 来说还是不够的，可以利用Kotlin相关的语法特性，封装下再少写一些代码。

捋下API使用链条：

- `registerForActivityResult()` → `ActivityResultLauncher`，需在`ON_START`或之前注册，在`OnCreate()`时再初始化会报错，还得传入一个**`ActivityResultContract`** 实例，最后跟一个 `ActivityResultCallback` 回调。
- 调用 `ActivityResultLauncher#launch()` 才触发页面跳转，需要传入一个输入(如`Intent`)实例。

最简单的封装就是写几个 扩展方法，从`ActivityResultLauncher`生成和`launch()`调用处入手：

```
// 扩展
fun ComponentActivity.registerActResult(callback: ActivityResultCallback<ActivityResult>) =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
        callback.onActivityResult(it) }

fun Fragment.registerActResult(callback: ActivityResultCallback<ActivityResult>) =
    registerForActivityResult(ActivityResultContracts.StartActivityForResult()) {
        callback.onActivityResult(it) }

fun Intent.launch(launcher: ActivityResultLauncher<Intent>) {
    launcher.launch(this)
}

// 注册处
private val mLauncher = registerActResult {
    showToast("收到测试数据: ${it.data?.getStringExtra("value")}")
}

// 调用处
Intent(this, SecondActivity::class.java).launch(mLauncher)
```

还可以在优化下，比如改成基于 `ActivityResultCaller` 进行扩展，然后把常用的一些跳转，如权限、打开相机、录像等写成一个扩展函数，用的时候直接调用即可。懒得自己写或者想找参考的可以看看 → `ActivityResult.kt`

如果想代码写得更少更优雅，可以折腾得更复杂些，比如结合生命周期回调，各种简化调用的扩展，甚至弄成DSL调用等，具体可以参考这些：

- **iDeMonnnnnn/DeMon-ARA** (<https://github.com/iDeMonnnnnn/DeMon-ARA>)
- **TxcA/ManageStartActivity** (<https://github.com/TxcA/ManageStartActivity>)

- **Flywith24/Flywith24-ActivityResultRequest**

(<https://github.com/Flywith24/Flywith24-ActivityResultRequest>)

怎么封装看自己，觉得适合就行，笔者就懒得整那么复杂了~



/ 小结 /

借着重构BaseFragment的机缘巧合，过了波Activity Results API的用法，阅读源码了解到背后的实现原理，小试了一下封装。心里有底了，赶紧在重构项目的时候安排上!!!



参考文献

- **官方文档：获取 activity 的结果**

(<https://developer.android.google.cn/training/basics/intents/result>)

- **优雅地封装 Activity Result API，完美地替代**

startActivityResult(<https://juejin.cn/post/6987575150283587592>)

- **Android onActivityResult的替代方法—**

registerForActivityResult(https://blog.csdn.net/weixin_42046829/article/details/116446887)

推荐阅读：

[我的新书，《第一行代码 第3版》已出版！](#)

[Activity Result API详解](#)

[一个Android沉浸式状态栏上的黑科技](#)

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

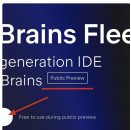
React 性能优化的那些事儿

前端瓶子君



干掉 VScode！JetBrains 官宣推出下一代轻量级 IDE！

沉默王二



React：我爱你，但是你越来越让我失望了

前端之巅

