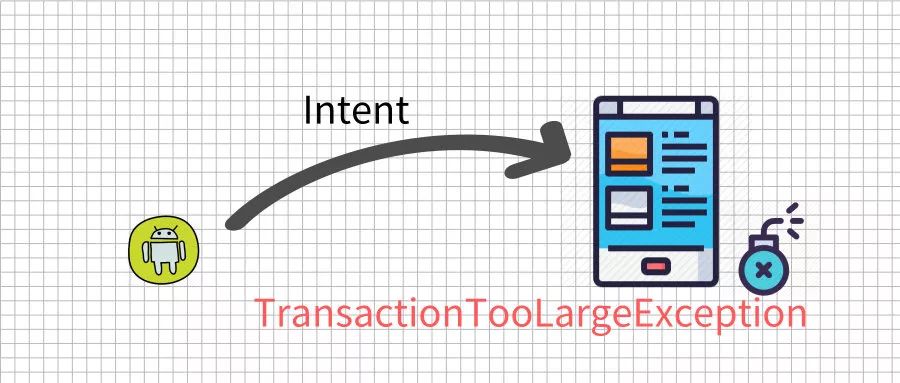
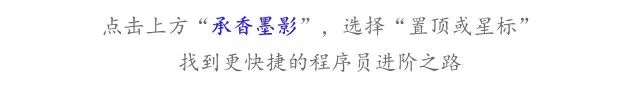
面试常客：Intent 能传递多大 Size 的数据？

团队号 [承香墨影的独家号](https://toutiao.io/subjects/16900) 作者 @承香墨影[原文链接](https://mp.weixin.qq.com/s/v57mDRnSEZ6hl7-bc3ugwA)



**一、序**

作为 Android 开发，日常 Coding 时，最频繁的操作应该就是操作 App 内的一系列 Activity。而在 Activity 间传递数据，就需要借助 Intent。

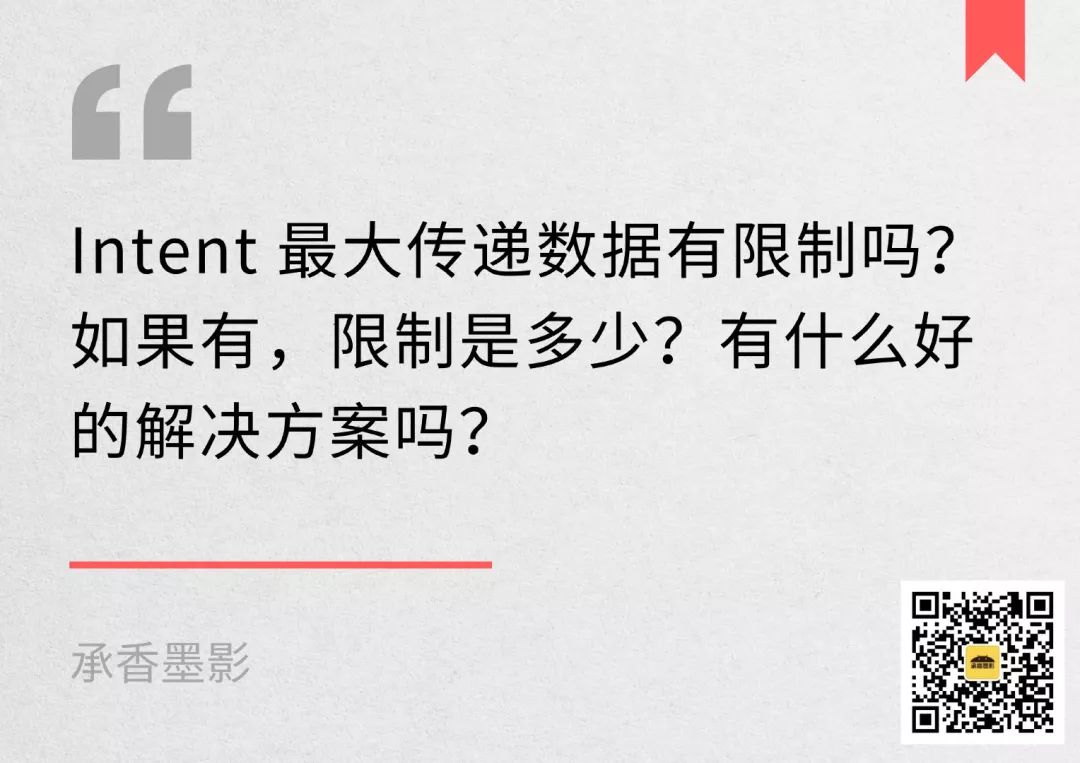
不少资料中写到，Intent 在 Activity 间传递基础类型数据或者可序列化的对象数据。但是 Intent 对数据大小是有限制的，当超过这个限制后，就会触发 TransactionTooLargeException 异常。

那么今天就来聊聊 Intent 传递大数据时，为什么会抛异常，以及如何解决它。

**二、为什么会出现异常？**

**2.1 异常原因**

Intent 传递大数据，会出现 TransactionTooLargeException 的场景，这本身也是一道面试题，经常在面试中被问到。



其实这个问题，如果遇到了，查查文档就知道了。

在 TransactionTooLargeException(https://developer.android.com/reference/android/os/TransactionTooLargeException.html) 的文档中，其实已经将触发原因详细说明了。

简单来说，Intent 传输数据的机制中，用到了 Binder。Intent 中的数据，会作为 Parcel 被存储在 Binder 的事务缓冲区(Binder transaction buffer)中的对象进行传输。

而这个 **Binder 事务缓冲区具有一个有限的固定大小，当前为 1MB**。你可别以为传递 1MB 以下的数据就安全了，这里的 1MB 空间并不是当前操作独享的，而是由当前进程所共享。也就是说 Intent 在 Activity 间传输数据，本身也不适合传递太大的数据。

**2.2 Bundle 的锅？**

这里再补充一些细节，Intent 使用 Bundle 存储数据，到底是值传递(深拷贝)还是引用传递？

Intent 传输的数据，都存放在一个 Bundle 类型的对象 mExtras 中，Bundle 要求所有存储的数据，都是可被序列化的。

在 Android 中，序列化数据需要实现 Serializable 或者 Parcelable。对于基础数据类型的包装类，本身就是实现了 Serializable，而我们自定义的对象，按需实现这两个序列化接口的其中一个即可。

**那是不是只要通过 Bundle 传递数据，就会面临序列化的问题？**

并不是，Activity 之间传递数据，首先要考虑跨进程的问题，而 Android 中又是通过 Binder 机制来解决跨进程通信的问题。涉及到跨进程，对于复杂数据就要涉及到序列化和反序列化的过程，这就注定是一次值传递（深拷贝）的过程。

这个问题用反证法也可以解释，如果是引用传递，那传递过去的只是对象的引用，指向了对象的存储地址，就只相当于一个 Int 的大小，也就根本不会出现 TransactionTooLargeException 异常。

**传输数据序列化和 Bundle 没有关系，只与 Binder 的跨进程通信有关。**

为什么要强调这个呢？

在 Android 中，使用 Bundle 传输数据，并非 Intent 独有的。例如使用弹窗时，DialogFragment 中也可以通过setArguments(Bundle)传递一个 Bundle 对象给对话框。

Fragment 本身是不涉及跨进程的，这里虽然使用了 Bundle 传输数据，但是并没有通过 Binder，也就是不存在序列化和反序列化。和 Fragment 数据传递相关的 Bundle，其实传递的是原对象的引用。

有兴趣可以做个试验，弹出 Dialog 时传递一个对象，Dialog 中修改数据后，在 Activity 中检查数据是否被修改了。

**三、如何解决这个异常？**

**3.1 解决思路**

知道异常的原因，就好解决了。

既然原因在于 Binder 传输限制了数据的大小，那我们不走 Binder 通信就好了。

可以从数据源上来考虑。

例如 Bitmap，本身就已经实现了 Parcelable 是可以支持序列化的。用 Intent 传输，稍微大一点的图一定会出现 TransactionTooLargeException。当然真是业务场景，肯定不存在传递 Bitmap 的情况。

那就先看看这个图片的数据源。Drawable？本地文件？线上图片？无论数据源在哪里，我们只需要传递一个 drawable\_id、路径、URL，就可以还原这张图片，无需将这个 Bitmap 对象传递过去。大数据总有数据源，从数据源还原数据，对我们而言只是调用一个方法而已。

此前阿里发布的《**Android 开发者手册**》中，就提到了这个问题的解决建议。



阿里给出的方案，是通过 EventBus 来传递数据。

**3.2 EventBus 的 粘性事件**

很多商业项目其实都用到了 EventBus，这里就简单介绍如何使用 EventBus 的粘性事件来完成数据在 Activity 间的传递。

EventBus 是一个 Android 端优化的 Publish/subscribe 消息总线，简化了应用程序内各个组件间、组件与后台线程间的通信。

在 Activity 中使用 EventBus，需要根据 Activity 的生命周期，成对调用register()和unregister()方法。普通的事件，只会发生在register()之后，在注册前发生的事件，统统都收不到。

这里利用的 EventBus 的粘性事件(Sticky Event)来实现，EventBus 内部维护了一个 Map 对象stickyEvents，用于缓存粘性事件。

粘性事件使用postSticky()方法发送，它会将事件缓存到stickyEvents这个 Map 对象中，以待下次注册时，将这个事件取出，抛给注册的组件。以此来达到一个粘性的滞后事件发送和接收。

接下来我们看看 EventBus 粘性事件的使用细节。

*1.* **注解的区别**

粘性事件的注解和普通事件的注解略有区别，需要添加 threadMode 和 sticky 参数。

@Subscribe(threadMode = ThreadMode.MAIN,sticky = **true**)

**public** **void** **onStickyEvent**(MyStickyEvent **event**){

    //...

}

注意，这两个额外的参数是必须的。

*2.* **调用方法的区别**

在发送消息的时候，需要使用postSticky()来替换掉post()方法。

需要注意的是，粘性事件是使用 Map 结构缓存的，并且是使用事件对象类型当 Key 进行缓存，所以对于同类型的数据，它只会缓存最后发送的数据。

*3.* **注意清理事件**

前面也提到，粘性事件是存储在一个 Map 对象中的，它是不会主动清理其中存储的对象的，需要开发者手动清理。

EventBus 提供了两类方法removeStickyEvent()和removeAllStickyEvents()方法，分别用来清理固定数据以及全部数据。

我们需要在合适的时机，手动的调用这两类方法，清理粘性事件。如果不对粘性事件进行清理，每次register()的时候，都会收到粘性事件。

*4.* **EventBus 粘性事件的问题**

粘性事件本身是脱离了 Android Intent 数据传递的这一套机制的，要知道 Activity 会在一些特殊情况下被销毁重建，在此情况下，通过 Intent 传递的数据，是可以继续从 Intent 中获取恢复到上一次页面传递的数据。

而通过 EventBus 的粘性事件，则可能在销毁重建时，造成数据丢失。

如果想要使用 EventBus 的粘性事件，来在页面间传递大数据，还是有不少细节，需要根据业务来调整的。

**四、小结时刻**

今天我们聊到了在 Activity 间，通过 Intent 传递大数据会触发 TransactionTooLargeException 异常的原因，以及如何解决它，最后再简单总结一下。

1. Intent 无法传递大数据是因为其内部使用了 Binder 通信机制，Binder 事务缓冲区限制了传递数据的大小。
2. Binder 事务缓冲区的大小限定在 1MB，但是这个尺寸是共享的，也就是并不是传递 1MB 以下的数据就绝对安全，要视当前的环境而定。
3. 不要挑战 Intent 传递数据大小的极限，对于大数据，例如长字符串、Bitmap 等，不要考虑 Intent 传递数据的方案。
4. 解决大数据传递问题，可以从数据源出发，根据数据的标识，还原数据，或者先持久化再还原。也可以使用 EventBus 的粘性事件来解决。

reference：https://developer.android.com/reference/android/os/TransactionTooLargeException.html

https://www.wanandroid.com/blogimgs/a2609aed-1000-4039-93c3-7541aaa2013b.pdf