TODO

笔记:http://m.blog.csdn.net/article/details?id=50726311

http://blog.csdn.net/u010342865/article/details/52460387

# IPC机制

IPC是 Inter-Process Communication的缩写，含义为进程间通信或者跨进程通信，是指两个进程之间进行数据交换的过程。进程间通信的方式有：Bundle、文件共享、AIDL、Messenger、ContentProvider和Socket等，最简单的情况下，一个进程中只可以有一个线程，即主线程，也叫UI线程。IPC不是Android中所独有的，任何一个操作系统都需要相应的IPC机制，比如Windows上可以通过剪贴板等来进行进程间通信。Android是一种基于Linux内核的移动操作系统，它的进程间通信方式并不能完全继承自Linux，相反，它有自己的进程间通信方式。在Android中最有特色的进程间通信方式就是Binder了。

why we need 多进程，可能有很多，比如有些模块由于特殊原因需要运行在单独的进程中，或者为了加 大一个应用可使用的内存所以需要通过多进程来获取多份内存空间。Android对单个应用所使用的最大内存做了16M限制。另一种情况是当前应用需要向其他应用获取数据，由于是两个应用，所以必须采用跨进程的方式来获取所需的数据，甚至我们通过系统提供的ContentPra}ider去查询数据的时候，建议DJI的midware这么搞？

--------------------------------------------------------------------------------

开启多进程模式

在Android中使用多进程只有一种方法：给四大组件(Activity、Service、Receiver、ContentProvider)在AndroidManifest中指定android:process属性，除此之外没有其他办法(除去通过JNI在native层去fork):

● android:process=”:remote”，私有进程，这种方式的标记的进程名为包名:remote，“:“的含义是指要在当前的进程名前附加包名，其次，进程名以“：”开头的进程属于当前应用的私有进程，其他应用的组件不可以和它跑在同一个进程中。

● android:process=”com.example.wpp.remote”，全局进程，这种声明方式是完整的命名方式，不会附件包名信息，属于全局进程，其他应用通过ShareUID方式可以和它跑在同一个进程中

Android系统会为每个应用分配一个唯一的UID，具有相同UID的应用才能共享数据，而两个应用通过ShareUID跑在同一个进程需要这两个应用有相同的ShareUID并且签名相同才可以，在这种情况下，它们可以互相 访问对方的私有数据，比如data目录、组件信息等，不管它们是否泡在同一个进程中。当然如果它们跑在同一个进程中，那么除了能共享data目录、组件信息，还可以共享内存数据

使用多进程会造成如下几方面的问题：

1. 静态成员和单例模式完全失效 （每个进程都分配一个独立的虚拟机，有着不同的虚拟机空间）

2. 线程同步机制完全失效。（不管是锁对象还是锁全局类都无法保证线程同步，因为不同进程锁的对象也不同了）

3. SharedPreferences的可靠性下降

4. Application会多次创建。

Q:多进程如果共享数据？

--------------------------------------------------------------------------------

3 IPC基础概念介绍

基本概念：Serializable接口、Parcelable接口以及Binder

Serializable接口

Parcelable接口

用来保存到本地的类特别好，取代了shareprefer

SerialiaabIe是Java中的序列化, 其使用起来简单但是开销很大，序列化到存储设备中或者将对象序列化后通过网络传输

Parcelable是Android中的序列化方式，使用起来稍微麻烦点，但是它的效率很高，主要用在内存序列化

Parcelable接口是Android提供的新的序列化方式，Parcelable也是一个接口，下面的示例是一个典型的用法。

public class User implements Parcelable{

public int userId;

public String userName;

public boolean isMale;

public Book book;

public User(int userId, String userName, boolean isMale){

this.userId = userId;

this.userName = userName;

this.isMale = isMale;

}

public int describeContents(){

return 0;

}

public void writeToParcel(Parcel out, int flags){

out.writeInt(userId);

out.wirteString(userName);

out.wirteInt(isMale ? 1 : 0);

out.writeParcelable(book, 0);

}

public static final Parcelable.Creator<User> CREATOR = new Parcelable.Creator<User>(){

public User createFromParcel(Parcel in){

return new User(in);

}

public User[] new Array(int size){

return new User[size];

}

};

private User(Parcel in){

userId = in.readInt();

userName = in.readString();

isMale = in.readInt()==1;

book = in.readParcelable(Thread.currentThread().getContextClassLoader());

}

}

Parcelable方法说明



方法功能标记位createFromParcel(Parcel in)从序列化后的对象中创建原对象new Array(int size)创建指定长度的原始对象数组User(Parcel in)从序列化后的对象中创建原始对象writeToParcel(Parcel out, int flag)将当前对象写入序列化结构中，其中flag标识有两种值：0或者1.为1时标识当前对象需要作为返回值返回，不能立即释放资源，几乎所有情况都为0PARCELABLE\_WRITE\_RETURN\_VALUEdescribeContents返回当前对象的内容描述，如果还有文件描述符，返回1，否则返回0，几乎所有情况都返回0

系统已经为我们提供了许多实现了Parcelable接口的类，它们都是可以直接序列化的，比如Intent、Bundle、Bitmap等，同时List和Map也可以序列化，前提是它们里面每个元素都是可序列化的。

Binder

选择AIDL的使用场合

官方文档特别提醒我们何时使用AIDL是必要的：只有你允许客户端从不同的应用程序为了进程间的通信而去访问你的service，以及想在你的service处理多线程。

2.4.2使用文件共享

对数据同步要求不高的进程之间进行遁信，基本走java的路线了

oos = new ObjectOutputStream( new FileOutputStream(file));

oos.writeObject(user);

注:SharePreferences是一个轻量级的存储方案，但是系统对这个的读写存在一定的缓存策略，在多进程高并发的情况下，丢失数据的概率很高，因此进程之间不推荐用这种方式通信(之前和zhongping就是一次失败的情况)

2.4.3使用Messenger

服务端

messenger = new Messenger(new MyHandler());

客户端

messenger = new Messenger(iBinder);

m.replyTo = mGetReplyMessenger;

2.4.6 使用Socket

# Parcel

d Parcel

一、 Android源码

（一）代码

内部用C++

package android.os;import android.text.TextUtils;import android.util.ArrayMap;import android.util.Log;import android.util.Size;import android.util.SizeF;import android.util.SparseArray;import android.util.SparseBooleanArray;import java.io.ByteArrayInputStream;import java.io.ByteArrayOutputStream;import java.io.FileDescriptor;import java.io.FileNotFoundException;import java.io.IOException;import java.io.ObjectInputStream;import java.io.ObjectOutputStream;import java.io.ObjectStreamClass;import java.io.Serializable;import java.lang.reflect.Field;import java.lang.reflect.Modifier;import java.util.ArrayList;import java.util.Arrays;import java.util.HashMap;import java.util.List;import java.util.Map;import java.util.Set;/\*\* \* Container for a message (data and object references) that can \* be sent through an IBinder. A Parcel can contain both flattened data \* that will be unflattened on the other side of the IPC (using the various \* methods here for writing specific types, or the general \* {@link Parcelable} interface), and references to live {@link IBinder} \* objects that will result in the other side receiving a proxy IBinder \* connected with the original IBinder in the Parcel. \* <p/> \* <p class="note">Parcel is <strong>not</strong> a general-purpose \* serialization mechanism. This class (and the corresponding \* {@link Parcelable} API for placing arbitrary objects into a Parcel) is \* designed as a high-performance IPC transport. As such, it is not \* appropriate to place any Parcel data in to persistent storage: changes \* in the underlying implementation of any of the data in the Parcel can \* render older data unreadable.</p> \* <p/> \* <p>The bulk of the Parcel API revolves around reading and writing data \* of various types. There are six major classes of such functions available.</p> \* <p/> \* <h3>Primitives</h3> \* <p/> \* <p>The most basic data functions are for writing and reading primitive \* data types: {@link #writeByte}, {@link #readByte}, {@link #writeDouble}, \* {@link #readDouble}, {@link #writeFloat}, {@link #readFloat}, {@link #writeInt}, \* {@link #readInt}, {@link #writeLong}, {@link #readLong}, \* {@link #writeString}, {@link #readString}. Most other \* data operations are built on top of these. The given data is written and \* read using the endianess of the host CPU.</p> \* <p/> \* <h3>Primitive Arrays</h3> \* <p/> \* <p>There are a variety of methods for reading and writing raw arrays \* of primitive objects, which generally result in writing a 4-byte length \* followed by the primitive data items. The methods for reading can either \* read the data into an existing array, or create and return a new array. \* These available types are:</p> \* <p/> \* <ul> \* <li> {@link #writeBooleanArray(boolean[])}, \* {@link #readBooleanArray(boolean[])}, {@link #createBooleanArray()} \* <li> {@link #writeByteArray(byte[])}, \* {@link #writeByteArray(byte[], int, int)}, {@link #readByteArray(byte[])}, \* {@link #createByteArray()} \* <li> {@link #writeCharArray(char[])}, {@link #readCharArray(char[])}, \* {@link #createCharArray()} \* <li> {@link #writeDoubleArray(double[])}, {@link #readDoubleArray(double[])}, \* {@link #createDoubleArray()} \* <li> {@link #writeFloatArray(float[])}, {@link #readFloatArray(float[])}, \* {@link #createFloatArray()} \* <li> {@link #writeIntArray(int[])}, {@link #readIntArray(int[])}, \* {@link #createIntArray()} \* <li> {@link #writeLongArray(long[])}, {@link #readLongArray(long[])}, \* {@link #createLongArray()} \* <li> {@link #writeStringArray(String[])}, {@link #readStringArray(String[])}, \* {@link #createStringArray()}. \* <li> {@link #writeSparseBooleanArray(SparseBooleanArray)}, \* {@link #readSparseBooleanArray()}. \* </ul> \* <p/> \* <h3>Parcelables</h3> \* <p/> \* <p>The {@link Parcelable} protocol provides an extremely efficient (but \* low-level) protocol for objects to write and read themselves from Parcels. \* You can use the direct methods {@link #writeParcelable(Parcelable, int)} \* and {@link #readParcelable(ClassLoader)} or \* {@link #writeParcelableArray} and \* {@link #readParcelableArray(ClassLoader)} to write or read. These \* methods write both the class type and its data to the Parcel, allowing \* that class to be reconstructed from the appropriate class loader when \* later reading.</p> \* <p/> \* <p>There are also some methods that provide a more efficient way to work \* with Parcelables: {@link #writeTypedObject}, {@link #writeTypedArray}, \* {@link #writeTypedList}, {@link #readTypedObject}, \* {@link #createTypedArray} and {@link #createTypedArrayList}. These methods \* do not write the class information of the original object: instead, the \* caller of the read function must know what type to expect and pass in the \* appropriate {@link Parcelable.Creator Parcelable.Creator} instead to \* properly construct the new object and read its data. (To more efficient \* write and read a single Parceable object that is not null, you can directly \* call {@link Parcelable#writeToParcel Parcelable.writeToParcel} and \* {@link Parcelable.Creator#createFromParcel Parcelable.Creator.createFromParcel} \* yourself.)</p> \* <p/> \* <h3>Bundles</h3> \* <p/> \* <p>A special type-safe container, called {@link Bundle}, is available \* for key/value maps of heterogeneous values. This has many optimizations \* for improved performance when reading and writing data, and its type-safe \* API avoids difficult to debug type errors when finally marshalling the \* data contents into a Parcel. The methods to use are \* {@link #writeBundle(Bundle)}, {@link #readBundle()}, and \* {@link #readBundle(ClassLoader)}. \* <p/> \* <h3>Active Objects</h3> \* <p/> \* <p>An unusual feature of Parcel is the ability to read and write active \* objects. For these objects the actual contents of the object is not \* written, rather a special token referencing the object is written. When \* reading the object back from the Parcel, you do not get a new instance of \* the object, but rather a handle that operates on the exact same object that \* was originally written. There are two forms of active objects available.</p> \* <p/> \* <p>{@link Binder} objects are a core facility of Android's general cross-process \* communication system. The {@link IBinder} interface describes an abstract \* protocol with a Binder object. Any such interface can be written in to \* a Parcel, and upon reading you will receive either the original object \* implementing that interface or a special proxy implementation \* that communicates calls back to the original object. The methods to use are \* {@link #writeStrongBinder(IBinder)}, \* {@link #writeStrongInterface(IInterface)}, {@link #readStrongBinder()}, \* {@link #writeBinderArray(IBinder[])}, {@link #readBinderArray(IBinder[])}, \* {@link #createBinderArray()}, \* {@link #writeBinderList(List)}, {@link #readBinderList(List)}, \* {@link #createBinderArrayList()}.</p> \* <p/> \* <p>FileDescriptor objects, representing raw Linux file descriptor identifiers, \* can be written and {@link ParcelFileDescriptor} objects returned to operate \* on the original file descriptor. The returned file descriptor is a dup \* of the original file descriptor: the object and fd is different, but \* operating on the same underlying file stream, with the same position, etc. \* The methods to use are {@link #writeFileDescriptor(FileDescriptor)}, \* {@link #readFileDescriptor()}. \* <p/> \* <h3>Untyped Containers</h3> \* <p/> \* <p>A final class of methods are for writing and reading standard Java \* containers of arbitrary types. These all revolve around the \* {@link #writeValue(Object)} and {@link #readValue(ClassLoader)} methods \* which define the types of objects allowed. The container methods are \* {@link #writeArray(Object[])}, {@link #readArray(ClassLoader)}, \* {@link #writeList(List)}, {@link #readList(List, ClassLoader)}, \* {@link #readArrayList(ClassLoader)}, \* {@link #writeMap(Map)}, {@link #readMap(Map, ClassLoader)}, \* {@link #writeSparseArray(SparseArray)}, \* {@link #readSparseArray(ClassLoader)}. \*/public final class Parcel { private static final boolean DEBUG\_RECYCLE = false; private static final boolean DEBUG\_ARRAY\_MAP = false; private static final String TAG = "Parcel"; @SuppressWarnings({"UnusedDeclaration"}) private long mNativePtr; // used by native code /\*\* \* Flag indicating if {@link #mNativePtr} was allocated by this object, \* indicating that we're responsible for its lifecycle. \*/ private boolean mOwnsNativeParcelObject; private RuntimeException mStack; private static final int POOL\_SIZE = 6; private static final Parcel[] sOwnedPool = new Parcel[POOL\_SIZE];//类变量 private static final Parcel[] sHolderPool = new Parcel[POOL\_SIZE]; private static final int VAL\_NULL = -1; private static final int VAL\_STRING = 0; private static final int VAL\_INTEGER = 1; private static final int VAL\_MAP = 2; private static final int VAL\_BUNDLE = 3; private static final int VAL\_PARCELABLE = 4; private static final int VAL\_SHORT = 5; private static final int VAL\_LONG = 6; private static final int VAL\_FLOAT = 7; private static final int VAL\_DOUBLE = 8; private static final int VAL\_BOOLEAN = 9; private static final int VAL\_CHARSEQUENCE = 10; private static final int VAL\_LIST = 11; private static final int VAL\_SPARSEARRAY = 12; private static final int VAL\_BYTEARRAY = 13; private static final int VAL\_STRINGARRAY = 14; private static final int VAL\_IBINDER = 15; private static final int VAL\_PARCELABLEARRAY = 16; private static final int VAL\_OBJECTARRAY = 17; private static final int VAL\_INTARRAY = 18; private static final int VAL\_LONGARRAY = 19; private static final int VAL\_BYTE = 20; private static final int VAL\_SERIALIZABLE = 21; private static final int VAL\_SPARSEBOOLEANARRAY = 22; private static final int VAL\_BOOLEANARRAY = 23; private static final int VAL\_CHARSEQUENCEARRAY = 24; private static final int VAL\_PERSISTABLEBUNDLE = 25; private static final int VAL\_SIZE = 26; private static final int VAL\_SIZEF = 27; // The initial int32 in a Binder call's reply Parcel header: private static final int EX\_SECURITY = -1; private static final int EX\_BAD\_PARCELABLE = -2; private static final int EX\_ILLEGAL\_ARGUMENT = -3; private static final int EX\_NULL\_POINTER = -4; private static final int EX\_ILLEGAL\_STATE = -5; private static final int EX\_NETWORK\_MAIN\_THREAD = -6; private static final int EX\_UNSUPPORTED\_OPERATION = -7; private static final int EX\_HAS\_REPLY\_HEADER = -128; // special; see below private static native int nativeDataSize(long nativePtr); private static native int nativeDataAvail(long nativePtr); private static native int nativeDataPosition(long nativePtr); private static native int nativeDataCapacity(long nativePtr); private static native void nativeSetDataSize(long nativePtr, int size); private static native void nativeSetDataPosition(long nativePtr, int pos); private static native void nativeSetDataCapacity(long nativePtr, int size); private static native boolean nativePushAllowFds(long nativePtr, boolean allowFds); private static native void nativeRestoreAllowFds(long nativePtr, boolean lastValue); private static native void nativeWriteByteArray(long nativePtr, byte[] b, int offset, int len); private static native void nativeWriteBlob(long nativePtr, byte[] b, int offset, int len); private static native void nativeWriteInt(long nativePtr, int val); private static native void nativeWriteLong(long nativePtr, long val); private static native void nativeWriteFloat(long nativePtr, float val); private static native void nativeWriteDouble(long nativePtr, double val); private static native void nativeWriteString(long nativePtr, String val); private static native void nativeWriteStrongBinder(long nativePtr, IBinder val); private static native void nativeWriteFileDescriptor(long nativePtr, FileDescriptor val); private static native byte[] nativeCreateByteArray(long nativePtr); private static native byte[] nativeReadBlob(long nativePtr); private static native int nativeReadInt(long nativePtr); private static native long nativeReadLong(long nativePtr); private static native float nativeReadFloat(long nativePtr); private static native double nativeReadDouble(long nativePtr); private static native String nativeReadString(long nativePtr); private static native IBinder nativeReadStrongBinder(long nativePtr); private static native FileDescriptor nativeReadFileDescriptor(long nativePtr); private static native long nativeCreate(); private static native void nativeFreeBuffer(long nativePtr); private static native void nativeDestroy(long nativePtr); private static native byte[] nativeMarshall(long nativePtr); private static native void nativeUnmarshall( long nativePtr, byte[] data, int offset, int length); private static native void nativeAppendFrom( long thisNativePtr, long otherNativePtr, int offset, int length); private static native boolean nativeHasFileDescriptors(long nativePtr); private static native void nativeWriteInterfaceToken(long nativePtr, String interfaceName); private static native void nativeEnforceInterface(long nativePtr, String interfaceName); private static native long nativeGetBlobAshmemSize(long nativePtr); public final static Parcelable.Creator<String> STRING\_CREATOR = new Parcelable.Creator<String>() { public String createFromParcel(Parcel source) { return source.readString(); } public String[] newArray(int size) { return new String[size]; } }; /\*\* \* Retrieve a new Parcel object from the pool.sOwnedPool默认有6个池子，选出一个非空的包裹内存出来 \*/ public static Parcel obtain() { final Parcel[] pool = sOwnedPool; synchronized (pool) { Parcel p; for (int i = 0; i < POOL\_SIZE; i++) { p = pool[i]; if (p != null) { pool[i] = null; if (DEBUG\_RECYCLE) { p.mStack = new RuntimeException(); } return p; } } } return new Parcel(0); } /\*\* \* Put a Parcel object back into the pool. You must not touch \* the object after this call.回收内存，把这个Parcel回收gei类的缓存 \*/ public final void recycle() { if (DEBUG\_RECYCLE) mStack = null; freeBuffer(); final Parcel[] pool; if (mOwnsNativeParcelObject) { pool = sOwnedPool; } else { mNativePtr = 0; pool = sHolderPool; } synchronized (pool) { for (int i = 0; i < POOL\_SIZE; i++) { if (pool[i] == null) {//刚才用obtain分配的 pool[i] = this;//为空了 return; } } } } /\*\* \* @hide \*/ public static native long getGlobalAllocSize(); /\*\* \* @hide \*/ public static native long getGlobalAllocCount(); /\*\* \* Returns the total amount of data contained in the parcel. \*/ public final int dataSize() { return nativeDataSize(mNativePtr); } /\*\* \* Returns the amount of data remaining to be read from the \* parcel. That is, {@link #dataSize}-{@link #dataPosition}. \*/ public final int dataAvail() { return nativeDataAvail(mNativePtr); } /\*\* \* Returns the current position in the parcel data. Never \* more than {@link #dataSize}. \*/ public final int dataPosition() { return nativeDataPosition(mNativePtr); } /\*\* \* Returns the total amount of space in the parcel. This is always \* >= {@link #dataSize}. The difference between it and dataSize() is the \* amount of room left until the parcel needs to re-allocate its \* data buffer. \*/ public final int dataCapacity() { return nativeDataCapacity(mNativePtr); } /\*\* \* Change the amount of data in the parcel. Can be either smaller or \* larger than the current size. If larger than the current capacity, \* more memory will be allocated. \* \* @param size The new number of bytes in the Parcel. \*/ public final void setDataSize(int size) { nativeSetDataSize(mNativePtr, size); } /\*\* \* Move the current read/write position in the parcel. \* \* @param pos New offset in the parcel; must be between 0 and \* {@link #dataSize}. \*/ public final void setDataPosition(int pos) { nativeSetDataPosition(mNativePtr, pos); } /\*\* \* Change the capacity (current available space) of the parcel. \* \* @param size The new capacity of the parcel, in bytes. Can not be \* less than {@link #dataSize} -- that is, you can not drop existing data \* with this method. \*/ public final void setDataCapacity(int size) { nativeSetDataCapacity(mNativePtr, size); } /\*\* \* @hide \*/ public final boolean pushAllowFds(boolean allowFds) { return nativePushAllowFds(mNativePtr, allowFds); } /\*\* \* @hide \*/ public final void restoreAllowFds(boolean lastValue) { nativeRestoreAllowFds(mNativePtr, lastValue); } /\*\* \* Returns the raw bytes of the parcel. \* <p/> \* <p class="note">The data you retrieve here <strong>must not</strong> \* be placed in any kind of persistent storage (on local disk, across \* a network, etc). For that, you should use standard serialization \* or another kind of general serialization mechanism. The Parcel \* marshalled representation is highly optimized for local IPC, and as \* such does not attempt to maintain compatibility with data created \* in different versions of the platform. \*/ public final byte[] marshall() { return nativeMarshall(mNativePtr); } /\*\* \* Set the bytes in data to be the raw bytes of this Parcel. \*/ public final void unmarshall(byte[] data, int offset, int length) { nativeUnmarshall(mNativePtr, data, offset, length); } public final void appendFrom(Parcel parcel, int offset, int length) { nativeAppendFrom(mNativePtr, parcel.mNativePtr, offset, length); } /\*\* \* Report whether the parcel contains any marshalled file descriptors. \*/ public final boolean hasFileDescriptors() { return nativeHasFileDescriptors(mNativePtr); } /\*\* \* Store or read an IBinder interface token in the parcel at the current \* {@link #dataPosition}. This is used to validate that the marshalled \* transaction is intended for the target interface. \*/ public final void writeInterfaceToken(String interfaceName) { nativeWriteInterfaceToken(mNativePtr, interfaceName); } public final void enforceInterface(String interfaceName) { nativeEnforceInterface(mNativePtr, interfaceName); } /\*\* \* Write a byte array into the parcel at the current {@link #dataPosition}, \* growing {@link #dataCapacity} if needed. \* \* @param b Bytes to place into the parcel. \*/ public final void writeByteArray(byte[] b) { writeByteArray(b, 0, (b != null) ? b.length : 0); } /\*\* \* Write a byte array into the parcel at the current {@link #dataPosition}, \* growing {@link #dataCapacity} if needed. \* \* @param b Bytes to place into the parcel. \* @param offset Index of first byte to be written. \* @param len Number of bytes to write. \*/ public final void writeByteArray(byte[] b, int offset, int len) { if (b == null) { writeInt(-1); return; } Arrays.checkOffsetAndCount(b.length, offset, len); nativeWriteByteArray(mNativePtr, b, offset, len); } /\*\* \* Write a blob of data into the parcel at the current {@link #dataPosition}, \* growing {@link #dataCapacity} if needed. \* \* @param b Bytes to place into the parcel. \* {@hide} \* {@SystemApi} \*/ public final void writeBlob(byte[] b) { writeBlob(b, 0, (b != null) ? b.length : 0); } /\*\* \* Write a blob of data into the parcel at the current {@link #dataPosition}, \* growing {@link #dataCapacity} if needed. \* \* @param b Bytes to place into the parcel. \* @param offset Index of first byte to be written. \* @param len Number of bytes to write. \* {@hide} \* {@SystemApi} \*/ public final void writeBlob(byte[] b, int offset, int len) { if (b == null) { writeInt(-1); return; } Arrays.checkOffsetAndCount(b.length, offset, len); nativeWriteBlob(mNativePtr, b, offset, len); } /\*\* \* Write an integer value into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeInt(int val) { nativeWriteInt(mNativePtr, val); } /\*\* \* Write a long integer value into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeLong(long val) { nativeWriteLong(mNativePtr, val); } /\*\* \* Write a floating point value into the parcel at the current \* dataPosition(), growing dataCapacity() if needed. \*/ public final void writeFloat(float val) { nativeWriteFloat(mNativePtr, val); } /\*\* \* Write a double precision floating point value into the parcel at the \* current dataPosition(), growing dataCapacity() if needed. \*/ public final void writeDouble(double val) { nativeWriteDouble(mNativePtr, val); } /\*\* \* Write a string value into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeString(String val) { nativeWriteString(mNativePtr, val); } /\*\* \* Write a CharSequence value into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \* \* @hide \*/ public final void writeCharSequence(CharSequence val) { TextUtils.writeToParcel(val, this, 0); } /\*\* \* Write an object into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeStrongBinder(IBinder val) { nativeWriteStrongBinder(mNativePtr, val); } /\*\* \* Write an object into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeStrongInterface(IInterface val) { writeStrongBinder(val == null ? null : val.asBinder()); } /\*\* \* Write a FileDescriptor into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \* <p/> \* <p class="caution">The file descriptor will not be closed, which may \* result in file descriptor leaks when objects are returned from Binder \* calls. Use {@link ParcelFileDescriptor#writeToParcel} instead, which \* accepts contextual flags and will close the original file descriptor \* if {@link Parcelable#PARCELABLE\_WRITE\_RETURN\_VALUE} is set.</p> \*/ public final void writeFileDescriptor(FileDescriptor val) { nativeWriteFileDescriptor(mNativePtr, val); }可见一个byte也是占32位 /\*\* \* Write a byte value into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeByte(byte val) { writeInt(val); } /\*\* \* Please use {@link #writeBundle} instead. Flattens a Map into the parcel \* at the current dataPosition(), \* growing dataCapacity() if needed. The Map keys must be String objects. \* The Map values are written using {@link #writeValue} and must follow \* the specification there. \* <p/> \* <p>It is strongly recommended to use {@link #writeBundle} instead of \* this method, since the Bundle class provides a type-safe API that \* allows you to avoid mysterious type errors at the point of marshalling. \*/ public final void writeMap(Map val) { writeMapInternal((Map<String, Object>) val); } /\*\* \* Flatten a Map into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. The Map keys must be String objects. \*/ /\* package \*/ void writeMapInternal(Map<String, Object> val) { if (val == null) { writeInt(-1); return; } Set<Map.Entry<String, Object>> entries = val.entrySet(); writeInt(entries.size()); for (Map.Entry<String, Object> e : entries) { writeValue(e.getKey()); writeValue(e.getValue()); } } /\*\* \* Flatten an ArrayMap into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. The Map keys must be String objects. \*/ /\* package \*/ void writeArrayMapInternal(ArrayMap<String, Object> val) { if (val == null) { writeInt(-1); return; } final int N = val.size(); writeInt(N); if (DEBUG\_ARRAY\_MAP) { RuntimeException here = new RuntimeException("here"); here.fillInStackTrace(); Log.d(TAG, "Writing " + N + " ArrayMap entries", here); } int startPos; for (int i = 0; i < N; i++) { if (DEBUG\_ARRAY\_MAP) startPos = dataPosition(); writeString(val.keyAt(i)); writeValue(val.valueAt(i)); if (DEBUG\_ARRAY\_MAP) Log.d(TAG, " Write #" + i + " " + (dataPosition() - startPos) + " bytes: key=0x" + Integer.toHexString(val.keyAt(i) != null ? val.keyAt(i).hashCode() : 0) + " " + val.keyAt(i)); } } /\*\* \* @hide For testing only. \*/ public void writeArrayMap(ArrayMap<String, Object> val) { writeArrayMapInternal(val); } /\*\* \* Flatten a Bundle into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeBundle(Bundle val) { if (val == null) { writeInt(-1); return; } val.writeToParcel(this, 0); } /\*\* \* Flatten a PersistableBundle into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writePersistableBundle(PersistableBundle val) { if (val == null) { writeInt(-1); return; } val.writeToParcel(this, 0); } /\*\* \* Flatten a Size into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeSize(Size val) { writeInt(val.getWidth()); writeInt(val.getHeight()); } /\*\* \* Flatten a SizeF into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. \*/ public final void writeSizeF(SizeF val) { writeFloat(val.getWidth()); writeFloat(val.getHeight()); } /\*\* \* Flatten a List into the parcel at the current dataPosition(), growing \* dataCapacity() if needed. The List values are written using \* {@link #writeValue} and must follow the specification there. \*/ public final void writeList(List val) { if (val == null) { writeInt(-1); return; } int N = val.size(); int i = 0; writeInt(N); while (i < N) { writeValue(val.get(i)); i++; } } /\*\* \* Flatten an Object array into the parcel at the current dataPosition(), \* growing dataCapacity() if needed. The array values are written using \* {@link #writeValue} and must follow the specification there. \*/ public final void writeArray(Object[] val) { if (val == null) { writeInt(-1); return; } int N = val.length; int i = 0; writeInt(N); while (i < N) { writeValue(val[i]); i++; } } /\*\* \* Flatten a generic SparseArray into the parcel at the current \* dataPosition(), growing dataCapacity() if needed. The SparseArray \* values are written using {@link #writeValue} and must follow the \* specification there. \*/ public final void writeSparseArray(SparseArray<Object> val) { if (val == null) { writeInt(-1); return; } int N = val.size(); writeInt(N); int i = 0; while (i < N) { writeInt(val.keyAt(i)); writeValue(val.valueAt(i)); i++; } } public final void writeSparseBooleanArray(SparseBooleanArray val) { if (val == null) { writeInt(-1); return; } int N = val.size(); writeInt(N); int i = 0; while (i < N) { writeInt(val.keyAt(i)); writeByte((byte) (val.valueAt(i) ? 1 : 0)); i++; } } public final void writeBooleanArray(boolean[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeInt(val[i] ? 1 : 0); } } else { writeInt(-1); } } public final boolean[] createBooleanArray() { int N = readInt(); // >>2 as a fast divide-by-4 works in the create\*Array() functions // because dataAvail() will never return a negative number. 4 is // the size of a stored boolean in the stream. if (N >= 0 && N <= (dataAvail() >> 2)) { boolean[] val = new boolean[N]; for (int i = 0; i < N; i++) { val[i] = readInt() != 0; } return val; } else { return null; } } public final void readBooleanArray(boolean[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = readInt() != 0; } } else { throw new RuntimeException("bad array lengths"); } } public final void writeCharArray(char[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeInt((int) val[i]); } } else { writeInt(-1); } } public final char[] createCharArray() { int N = readInt(); if (N >= 0 && N <= (dataAvail() >> 2)) { char[] val = new char[N]; for (int i = 0; i < N; i++) { val[i] = (char) readInt(); } return val; } else { return null; } } public final void readCharArray(char[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = (char) readInt(); } } else { throw new RuntimeException("bad array lengths"); } } public final void writeIntArray(int[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeInt(val[i]); } } else { writeInt(-1); } } public final int[] createIntArray() { int N = readInt(); if (N >= 0 && N <= (dataAvail() >> 2)) { int[] val = new int[N]; for (int i = 0; i < N; i++) { val[i] = readInt(); } return val; } else { return null; } } public final void readIntArray(int[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = readInt(); } } else { throw new RuntimeException("bad array lengths"); } } public final void writeLongArray(long[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeLong(val[i]); } } else { writeInt(-1); } } public final long[] createLongArray() { int N = readInt(); // >>3 because stored longs are 64 bits if (N >= 0 && N <= (dataAvail() >> 3)) { long[] val = new long[N]; for (int i = 0; i < N; i++) { val[i] = readLong(); } return val; } else { return null; } } public final void readLongArray(long[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = readLong(); } } else { throw new RuntimeException("bad array lengths"); } } public final void writeFloatArray(float[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeFloat(val[i]); } } else { writeInt(-1); } } public final float[] createFloatArray() { int N = readInt(); // >>2 because stored floats are 4 bytes if (N >= 0 && N <= (dataAvail() >> 2)) { float[] val = new float[N]; for (int i = 0; i < N; i++) { val[i] = readFloat(); } return val; } else { return null; } } public final void readFloatArray(float[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = readFloat(); } } else { throw new RuntimeException("bad array lengths"); } } public final void writeDoubleArray(double[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeDouble(val[i]); } } else { writeInt(-1); } } public final double[] createDoubleArray() { int N = readInt(); // >>3 because stored doubles are 8 bytes if (N >= 0 && N <= (dataAvail() >> 3)) { double[] val = new double[N]; for (int i = 0; i < N; i++) { val[i] = readDouble(); } return val; } else { return null; } } public final void readDoubleArray(double[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = readDouble(); } } else { throw new RuntimeException("bad array lengths"); } } public final void writeStringArray(String[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeString(val[i]); } } else { writeInt(-1); } } public final String[] createStringArray() { int N = readInt(); if (N >= 0) { String[] val = new String[N]; for (int i = 0; i < N; i++) { val[i] = readString(); } return val; } else { return null; } } public final void readStringArray(String[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = readString(); } } else { throw new RuntimeException("bad array lengths"); } } public final void writeBinderArray(IBinder[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeStrongBinder(val[i]); } } else { writeInt(-1); } } /\*\* \* @hide \*/ public final void writeCharSequenceArray(CharSequence[] val) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { writeCharSequence(val[i]); } } else { writeInt(-1); } } /\*\* \* @hide \*/ public final void writeCharSequenceList(ArrayList<CharSequence> val) { if (val != null) { int N = val.size(); writeInt(N); for (int i = 0; i < N; i++) { writeCharSequence(val.get(i)); } } else { writeInt(-1); } } public final IBinder[] createBinderArray() { int N = readInt(); if (N >= 0) { IBinder[] val = new IBinder[N]; for (int i = 0; i < N; i++) { val[i] = readStrongBinder(); } return val; } else { return null; } } public final void readBinderArray(IBinder[] val) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { val[i] = readStrongBinder(); } } else { throw new RuntimeException("bad array lengths"); } } /\*\* \* Flatten a List containing a particular object type into the parcel, at \* the current dataPosition() and growing dataCapacity() if needed. The \* type of the objects in the list must be one that implements Parcelable. \* Unlike the generic writeList() method, however, only the raw data of the \* objects is written and not their type, so you must use the corresponding \* readTypedList() to unmarshall them. \* \* @param val The list of objects to be written. \* @see #createTypedArrayList \* @see #readTypedList \* @see Parcelable \*/ public final <T extends Parcelable> void writeTypedList(List<T> val) { if (val == null) { writeInt(-1); return; } int N = val.size(); int i = 0; writeInt(N); while (i < N) { T item = val.get(i); if (item != null) { writeInt(1); item.writeToParcel(this, 0); } else { writeInt(0); } i++; } } /\*\* \* Flatten a List containing String objects into the parcel, at \* the current dataPosition() and growing dataCapacity() if needed. They \* can later be retrieved with {@link #createStringArrayList} or \* {@link #readStringList}. \* \* @param val The list of strings to be written. \* @see #createStringArrayList \* @see #readStringList \*/ public final void writeStringList(List<String> val) { if (val == null) { writeInt(-1); return; } int N = val.size(); int i = 0; writeInt(N); while (i < N) { writeString(val.get(i)); i++; } } /\*\* \* Flatten a List containing IBinder objects into the parcel, at \* the current dataPosition() and growing dataCapacity() if needed. They \* can later be retrieved with {@link #createBinderArrayList} or \* {@link #readBinderList}. \* \* @param val The list of strings to be written. \* @see #createBinderArrayList \* @see #readBinderList \*/ public final void writeBinderList(List<IBinder> val) { if (val == null) { writeInt(-1); return; } int N = val.size(); int i = 0; writeInt(N); while (i < N) { writeStrongBinder(val.get(i)); i++; } } /\*\* \* Flatten a heterogeneous array containing a particular object type into \* the parcel, at \* the current dataPosition() and growing dataCapacity() if needed. The \* type of the objects in the array must be one that implements Parcelable. \* Unlike the {@link #writeParcelableArray} method, however, only the \* raw data of the objects is written and not their type, so you must use \* {@link #readTypedArray} with the correct corresponding \* {@link Parcelable.Creator} implementation to unmarshall them. \* \* @param val The array of objects to be written. \* @param parcelableFlags Contextual flags as per \* {@link Parcelable#writeToParcel(Parcel, int) Parcelable.writeToParcel()}. \* @see #readTypedArray \* @see #writeParcelableArray \* @see Parcelable.Creator \*/ public final <T extends Parcelable> void writeTypedArray(T[] val, int parcelableFlags) { if (val != null) { int N = val.length; writeInt(N); for (int i = 0; i < N; i++) { T item = val[i]; if (item != null) { writeInt(1); item.writeToParcel(this, parcelableFlags); } else { writeInt(0); } } } else { writeInt(-1); } } /\*\* \* Flatten the Parcelable object into the parcel. \* \* @param val The Parcelable object to be written.y \* @param parcelableFlags Contextual flags as per \* {@link Parcelable#writeToParcel(Parcel, int) Parcelable.writeToParcel()}. \* @see #readTypedObject \*/ public final <T extends Parcelable> void writeTypedObject(T val, int parcelableFlags) { if (val != null) { writeInt(1); val.writeToParcel(this, parcelableFlags); } else { writeInt(0); } } /\*\* \* Flatten a generic object in to a parcel. The given Object value may \* currently be one of the following types: \* <p/> \* <ul> \* <li> null \* <li> String \* <li> Byte \* <li> Short \* <li> Integer \* <li> Long \* <li> Float \* <li> Double \* <li> Boolean \* <li> String[] \* <li> boolean[] \* <li> byte[] \* <li> int[] \* <li> long[] \* <li> Object[] (supporting objects of the same type defined here). \* <li> {@link Bundle} \* <li> Map (as supported by {@link #writeMap}). \* <li> Any object that implements the {@link Parcelable} protocol. \* <li> Parcelable[] \* <li> CharSequence (as supported by {@link TextUtils#writeToParcel}). \* <li> List (as supported by {@link #writeList}). \* <li> {@link SparseArray} (as supported by {@link #writeSparseArray(SparseArray)}). \* <li> {@link IBinder} \* <li> Any object that implements Serializable (but see \* {@link #writeSerializable} for caveats). Note that all of the \* previous types have relatively efficient implementations for \* writing to a Parcel; having to rely on the generic serialization \* approach is much less efficient and should be avoided whenever \* possible. \* </ul> \* <p/> \* <p class="caution">{@link Parcelable} objects are written with \* {@link Parcelable#writeToParcel} using contextual flags of 0. When \* serializing objects containing {@link ParcelFileDescriptor}s, \* this may result in file descriptor leaks when they are returned from \* Binder calls (where {@link Parcelable#PARCELABLE\_WRITE\_RETURN\_VALUE} \* should be used).</p>先写类型，再写数据 \*/ public final void writeValue(Object v) { if (v == null) { writeInt(VAL\_NULL); } else if (v instanceof String) { writeInt(VAL\_STRING); writeString((String) v); } else if (v instanceof Integer) { writeInt(VAL\_INTEGER); writeInt((Integer) v); } else if (v instanceof Map) { writeInt(VAL\_MAP); writeMap((Map) v); } else if (v instanceof Bundle) { // Must be before Parcelable writeInt(VAL\_BUNDLE); writeBundle((Bundle) v); } else if (v instanceof Parcelable) { writeInt(VAL\_PARCELABLE); writeParcelable((Parcelable) v, 0); } else if (v instanceof Short) { writeInt(VAL\_SHORT); writeInt(((Short) v).intValue()); } else if (v instanceof Long) { writeInt(VAL\_LONG); writeLong((Long) v); } else if (v instanceof Float) { writeInt(VAL\_FLOAT); writeFloat((Float) v); } else if (v instanceof Double) { writeInt(VAL\_DOUBLE); writeDouble((Double) v); } else if (v instanceof Boolean) { writeInt(VAL\_BOOLEAN); writeInt((Boolean) v ? 1 : 0); } else if (v instanceof CharSequence) { // Must be after String writeInt(VAL\_CHARSEQUENCE); writeCharSequence((CharSequence) v); } else if (v instanceof List) { writeInt(VAL\_LIST); writeList((List) v); } else if (v instanceof SparseArray) { writeInt(VAL\_SPARSEARRAY); writeSparseArray((SparseArray) v); } else if (v instanceof boolean[]) { writeInt(VAL\_BOOLEANARRAY); writeBooleanArray((boolean[]) v); } else if (v instanceof byte[]) { writeInt(VAL\_BYTEARRAY); writeByteArray((byte[]) v); } else if (v instanceof String[]) { writeInt(VAL\_STRINGARRAY); writeStringArray((String[]) v); } else if (v instanceof CharSequence[]) { // Must be after String[] and before Object[] writeInt(VAL\_CHARSEQUENCEARRAY); writeCharSequenceArray((CharSequence[]) v); } else if (v instanceof IBinder) { writeInt(VAL\_IBINDER); writeStrongBinder((IBinder) v); } else if (v instanceof Parcelable[]) { writeInt(VAL\_PARCELABLEARRAY); writeParcelableArray((Parcelable[]) v, 0); } else if (v instanceof int[]) { writeInt(VAL\_INTARRAY); writeIntArray((int[]) v); } else if (v instanceof long[]) { writeInt(VAL\_LONGARRAY); writeLongArray((long[]) v); } else if (v instanceof Byte) { writeInt(VAL\_BYTE); writeInt((Byte) v); } else if (v instanceof PersistableBundle) { writeInt(VAL\_PERSISTABLEBUNDLE); writePersistableBundle((PersistableBundle) v); } else if (v instanceof Size) { writeInt(VAL\_SIZE); writeSize((Size) v); } else if (v instanceof SizeF) { writeInt(VAL\_SIZEF); writeSizeF((SizeF) v); } else { Class<?> clazz = v.getClass(); if (clazz.isArray() && clazz.getComponentType() == Object.class) { // Only pure Object[] are written here, Other arrays of non-primitive types are // handled by serialization as this does not record the component type. writeInt(VAL\_OBJECTARRAY); writeArray((Object[]) v); } else if (v instanceof Serializable) { // Must be last writeInt(VAL\_SERIALIZABLE); writeSerializable((Serializable) v); } else { throw new RuntimeException("Parcel: unable to marshal value " + v); } } } /\*\* \* Flatten the name of the class of the Parcelable and its contents \* into the parcel. \* \* @param p The Parcelable object to be written. \* @param parcelableFlags Contextual flags as per \* {@link Parcelable#writeToParcel(Parcel, int) Parcelable.writeToParcel()}. \*/ public final void writeParcelable(Parcelable p, int parcelableFlags) { if (p == null) { writeString(null); return; } writeParcelableCreator(p); p.writeToParcel(this, parcelableFlags); } /\*\* \* @hide \*/ public final void writeParcelableCreator(Parcelable p) { String name = p.getClass().getName(); writeString(name); } /\*\* \* Write a generic serializable object in to a Parcel. It is strongly \* recommended that this method be avoided, since the serialization \* overhead is extremely large, and this approach will be much slower than \* using the other approaches to writing data in to a Parcel. \*/ public final void writeSerializable(Serializable s) { if (s == null) { writeString(null); return; } String name = s.getClass().getName(); writeString(name); ByteArrayOutputStream baos = new ByteArrayOutputStream(); try { ObjectOutputStream oos = new ObjectOutputStream(baos); oos.writeObject(s); oos.close(); writeByteArray(baos.toByteArray()); } catch (IOException ioe) { throw new RuntimeException("Parcelable encountered " + "IOException writing serializable object (name = " + name + ")", ioe); } } /\*\* \* Special function for writing an exception result at the header of \* a parcel, to be used when returning an exception from a transaction. \* Note that this currently only supports a few exception types; any other \* exception will be re-thrown by this function as a RuntimeException \* (to be caught by the system's last-resort exception handling when \* dispatching a transaction). \* <p/> \* <p>The supported exception types are: \* <ul> \* <li>{@link BadParcelableException} \* <li>{@link IllegalArgumentException} \* <li>{@link IllegalStateException} \* <li>{@link NullPointerException} \* <li>{@link SecurityException} \* <li>{@link NetworkOnMainThreadException} \* </ul> \* \* @param e The Exception to be written. \* @see #writeNoException \* @see #readException \*/ public final void writeException(Exception e) { int code = 0; if (e instanceof SecurityException) { code = EX\_SECURITY; } else if (e instanceof BadParcelableException) { code = EX\_BAD\_PARCELABLE; } else if (e instanceof IllegalArgumentException) { code = EX\_ILLEGAL\_ARGUMENT; } else if (e instanceof NullPointerException) { code = EX\_NULL\_POINTER; } else if (e instanceof IllegalStateException) { code = EX\_ILLEGAL\_STATE; } else if (e instanceof NetworkOnMainThreadException) { code = EX\_NETWORK\_MAIN\_THREAD; } else if (e instanceof UnsupportedOperationException) { code = EX\_UNSUPPORTED\_OPERATION; } writeInt(code); StrictMode.clearGatheredViolations(); if (code == 0) { if (e instanceof RuntimeException) { throw (RuntimeException) e; } throw new RuntimeException(e); } writeString(e.getMessage()); } /\*\* \* Special function for writing information at the front of the Parcel \* indicating that no exception occurred. \* \* @see #writeException \* @see #readException \*/ public final void writeNoException() { // Despite the name of this function ("write no exception"), // it should instead be thought of as "write the RPC response // header", but because this function name is written out by // the AIDL compiler, we're not going to rename it. // // The response header, in the non-exception case (see also // writeException above, also called by the AIDL compiler), is // either a 0 (the default case), or EX\_HAS\_REPLY\_HEADER if // StrictMode has gathered up violations that have occurred // during a Binder call, in which case we write out the number // of violations and their details, serialized, before the // actual RPC respons data. The receiving end of this is // readException(), below. if (StrictMode.hasGatheredViolations()) { writeInt(EX\_HAS\_REPLY\_HEADER); final int sizePosition = dataPosition(); writeInt(0); // total size of fat header, to be filled in later StrictMode.writeGatheredViolationsToParcel(this); final int payloadPosition = dataPosition(); setDataPosition(sizePosition); writeInt(payloadPosition - sizePosition); // header size setDataPosition(payloadPosition); } else { writeInt(0); } } /\*\* \* Special function for reading an exception result from the header of \* a parcel, to be used after receiving the result of a transaction. This \* will throw the exception for you if it had been written to the Parcel, \* otherwise return and let you read the normal result data from the Parcel. \* \* @see #writeException \* @see #writeNoException \*/ public final void readException() { int code = readExceptionCode(); if (code != 0) { String msg = readString(); readException(code, msg); } } /\*\* \* Parses the header of a Binder call's response Parcel and \* returns the exception code. Deals with lite or fat headers. \* In the common successful case, this header is generally zero. \* In less common cases, it's a small negative number and will be \* followed by an error string. \* <p/> \* This exists purely for android.database.DatabaseUtils and \* insulating it from having to handle fat headers as returned by \* e.g. StrictMode-induced RPC responses. \* \* @hide \*/ public final int readExceptionCode() { int code = readInt(); if (code == EX\_HAS\_REPLY\_HEADER) { int headerSize = readInt(); if (headerSize == 0) { Log.e(TAG, "Unexpected zero-sized Parcel reply header."); } else { // Currently the only thing in the header is StrictMode stacks, // but discussions around event/RPC tracing suggest we might // put that here too. If so, switch on sub-header tags here. // But for now, just parse out the StrictMode stuff. StrictMode.readAndHandleBinderCallViolations(this); } // And fat response headers are currently only used when // there are no exceptions, so return no error: return 0; } return code; } /\*\* \* Throw an exception with the given message. Not intended for use \* outside the Parcel class. \* \* @param code Used to determine which exception class to throw. \* @param msg The exception message. \*/ public final void readException(int code, String msg) { switch (code) { case EX\_SECURITY: throw new SecurityException(msg); case EX\_BAD\_PARCELABLE: throw new BadParcelableException(msg); case EX\_ILLEGAL\_ARGUMENT: throw new IllegalArgumentException(msg); case EX\_NULL\_POINTER: throw new NullPointerException(msg); case EX\_ILLEGAL\_STATE: throw new IllegalStateException(msg); case EX\_NETWORK\_MAIN\_THREAD: throw new NetworkOnMainThreadException(); case EX\_UNSUPPORTED\_OPERATION: throw new UnsupportedOperationException(msg); } throw new RuntimeException("Unknown exception code: " + code + " msg " + msg); } /\*\* \* Read an integer value from the parcel at the current dataPosition(). \*/ public final int readInt() { return nativeReadInt(mNativePtr); } /\*\* \* Read a long integer value from the parcel at the current dataPosition(). \*/ public final long readLong() { return nativeReadLong(mNativePtr); } /\*\* \* Read a floating point value from the parcel at the current \* dataPosition(). \*/ public final float readFloat() { return nativeReadFloat(mNativePtr); } /\*\* \* Read a double precision floating point value from the parcel at the \* current dataPosition(). \*/ public final double readDouble() { return nativeReadDouble(mNativePtr); } /\*\* \* Read a string value from the parcel at the current dataPosition(). \*/ public final String readString() { return nativeReadString(mNativePtr); } /\*\* \* Read a CharSequence value from the parcel at the current dataPosition(). \* \* @hide \*/ public final CharSequence readCharSequence() { return TextUtils.CHAR\_SEQUENCE\_CREATOR.createFromParcel(this); } /\*\* \* Read an object from the parcel at the current dataPosition(). \*/ public final IBinder readStrongBinder() { return nativeReadStrongBinder(mNativePtr); } /\*\* \* Read a FileDescriptor from the parcel at the current dataPosition(). \*/ public final ParcelFileDescriptor readFileDescriptor() { FileDescriptor fd = nativeReadFileDescriptor(mNativePtr); return fd != null ? new ParcelFileDescriptor(fd) : null; } /\*\* \* {@hide} \*/ public final FileDescriptor readRawFileDescriptor() { return nativeReadFileDescriptor(mNativePtr); } /\*package\*/ static native FileDescriptor openFileDescriptor(String file, int mode) throws FileNotFoundException; /\*package\*/ static native FileDescriptor dupFileDescriptor(FileDescriptor orig) throws IOException; /\*package\*/ static native void closeFileDescriptor(FileDescriptor desc) throws IOException; /\*package\*/ static native void clearFileDescriptor(FileDescriptor desc); /\*\* \* Read a byte value from the parcel at the current dataPosition().取低位 \*/ public final byte readByte() { return (byte) (readInt() & 0xff); } /\*\* \* Please use {@link #readBundle(ClassLoader)} instead (whose data must have \* been written with {@link #writeBundle}. Read into an existing Map object \* from the parcel at the current dataPosition(). \*/ public final void readMap(Map outVal, ClassLoader loader) { int N = readInt(); readMapInternal(outVal, N, loader); } /\*\* \* Read into an existing List object from the parcel at the current \* dataPosition(), using the given class loader to load any enclosed \* Parcelables. If it is null, the default class loader is used. \*/ public final void readList(List outVal, ClassLoader loader) { int N = readInt(); readListInternal(outVal, N, loader); } /\*\* \* Please use {@link #readBundle(ClassLoader)} instead (whose data must have \* been written with {@link #writeBundle}. Read and return a new HashMap \* object from the parcel at the current dataPosition(), using the given \* class loader to load any enclosed Parcelables. Returns null if \* the previously written map object was null. \*/ public final HashMap readHashMap(ClassLoader loader) { int N = readInt(); if (N < 0) { return null; } HashMap m = new HashMap(N); readMapInternal(m, N, loader); return m; } /\*\* \* Read and return a new Bundle object from the parcel at the current \* dataPosition(). Returns null if the previously written Bundle object was \* null. \*/ public final Bundle readBundle() { return readBundle(null); } /\*\* \* Read and return a new Bundle object from the parcel at the current \* dataPosition(), using the given class loader to initialize the class \* loader of the Bundle for later retrieval of Parcelable objects. \* Returns null if the previously written Bundle object was null. \*/ public final Bundle readBundle(ClassLoader loader) { int length = readInt(); if (length < 0) { if (Bundle.DEBUG) Log.d(TAG, "null bundle: length=" + length); return null; } final Bundle bundle = new Bundle(this, length); if (loader != null) { bundle.setClassLoader(loader); } return bundle; } /\*\* \* Read and return a new Bundle object from the parcel at the current \* dataPosition(). Returns null if the previously written Bundle object was \* null. \*/ public final PersistableBundle readPersistableBundle() { return readPersistableBundle(null); } /\*\* \* Read and return a new Bundle object from the parcel at the current \* dataPosition(), using the given class loader to initialize the class \* loader of the Bundle for later retrieval of Parcelable objects. \* Returns null if the previously written Bundle object was null. \*/ public final PersistableBundle readPersistableBundle(ClassLoader loader) { int length = readInt(); if (length < 0) { if (Bundle.DEBUG) Log.d(TAG, "null bundle: length=" + length); return null; } final PersistableBundle bundle = new PersistableBundle(this, length); if (loader != null) { bundle.setClassLoader(loader); } return bundle; } /\*\* \* Read a Size from the parcel at the current dataPosition(). \*/ public final Size readSize() { final int width = readInt(); final int height = readInt(); return new Size(width, height); } /\*\* \* Read a SizeF from the parcel at the current dataPosition(). \*/ public final SizeF readSizeF() { final float width = readFloat(); final float height = readFloat(); return new SizeF(width, height); } /\*\* \* Read and return a byte[] object from the parcel. \*/ public final byte[] createByteArray() { return nativeCreateByteArray(mNativePtr); } /\*\* \* Read a byte[] object from the parcel and copy it into the \* given byte array. \*/ public final void readByteArray(byte[] val) { // TODO: make this a native method to avoid the extra copy. byte[] ba = createByteArray(); if (ba.length == val.length) { System.arraycopy(ba, 0, val, 0, ba.length); } else { throw new RuntimeException("bad array lengths"); } } /\*\* \* Read a blob of data from the parcel and return it as a byte array. \* {@hide} \* {@SystemApi} \*/ public final byte[] readBlob() { return nativeReadBlob(mNativePtr); } /\*\* \* Read and return a String[] object from the parcel. \* {@hide} \*/ public final String[] readStringArray() { String[] array = null; int length = readInt(); if (length >= 0) { array = new String[length]; for (int i = 0; i < length; i++) { array[i] = readString(); } } return array; } /\*\* \* Read and return a CharSequence[] object from the parcel. \* {@hide} \*/ public final CharSequence[] readCharSequenceArray() { CharSequence[] array = null; int length = readInt(); if (length >= 0) { array = new CharSequence[length]; for (int i = 0; i < length; i++) { array[i] = readCharSequence(); } } return array; } /\*\* \* Read and return an ArrayList&lt;CharSequence&gt; object from the parcel. \* {@hide} \*/ public final ArrayList<CharSequence> readCharSequenceList() { ArrayList<CharSequence> array = null; int length = readInt(); if (length >= 0) { array = new ArrayList<CharSequence>(length); for (int i = 0; i < length; i++) { array.add(readCharSequence()); } } return array; } /\*\* \* Read and return a new ArrayList object from the parcel at the current \* dataPosition(). Returns null if the previously written list object was \* null. The given class loader will be used to load any enclosed \* Parcelables. \*/ public final ArrayList readArrayList(ClassLoader loader) { int N = readInt(); if (N < 0) { return null; } ArrayList l = new ArrayList(N); readListInternal(l, N, loader); return l; } /\*\* \* Read and return a new Object array from the parcel at the current \* dataPosition(). Returns null if the previously written array was \* null. The given class loader will be used to load any enclosed \* Parcelables. \*/ public final Object[] readArray(ClassLoader loader) { int N = readInt(); if (N < 0) { return null; } Object[] l = new Object[N]; readArrayInternal(l, N, loader); return l; } /\*\* \* Read and return a new SparseArray object from the parcel at the current \* dataPosition(). Returns null if the previously written list object was \* null. The given class loader will be used to load any enclosed \* Parcelables. \*/ public final SparseArray readSparseArray(ClassLoader loader) { int N = readInt(); if (N < 0) { return null; } SparseArray sa = new SparseArray(N); readSparseArrayInternal(sa, N, loader); return sa; } /\*\* \* Read and return a new SparseBooleanArray object from the parcel at the current \* dataPosition(). Returns null if the previously written list object was \* null. \*/ public final SparseBooleanArray readSparseBooleanArray() { int N = readInt(); if (N < 0) { return null; } SparseBooleanArray sa = new SparseBooleanArray(N); readSparseBooleanArrayInternal(sa, N); return sa; } /\*\* \* Read and return a new ArrayList containing a particular object type from \* the parcel that was written with {@link #writeTypedList} at the \* current dataPosition(). Returns null if the \* previously written list object was null. The list <em>must</em> have \* previously been written via {@link #writeTypedList} with the same object \* type. \* \* @return A newly created ArrayList containing objects with the same data \* as those that were previously written. \* @see #writeTypedList \*/ public final <T> ArrayList<T> createTypedArrayList(Parcelable.Creator<T> c) { int N = readInt(); if (N < 0) { return null; } ArrayList<T> l = new ArrayList<T>(N); while (N > 0) { if (readInt() != 0) { l.add(c.createFromParcel(this)); } else { l.add(null); } N--; } return l; } /\*\* \* Read into the given List items containing a particular object type \* that were written with {@link #writeTypedList} at the \* current dataPosition(). The list <em>must</em> have \* previously been written via {@link #writeTypedList} with the same object \* type. \* \* @return A newly created ArrayList containing objects with the same data \* as those that were previously written. \* @see #writeTypedList \*/ public final <T> void readTypedList(List<T> list, Parcelable.Creator<T> c) { int M = list.size(); int N = readInt(); int i = 0; for (; i < M && i < N; i++) { if (readInt() != 0) { list.set(i, c.createFromParcel(this)); } else { list.set(i, null); } } for (; i < N; i++) { if (readInt() != 0) { list.add(c.createFromParcel(this)); } else { list.add(null); } } for (; i < M; i++) { list.remove(N); } } /\*\* \* Read and return a new ArrayList containing String objects from \* the parcel that was written with {@link #writeStringList} at the \* current dataPosition(). Returns null if the \* previously written list object was null. \* \* @return A newly created ArrayList containing strings with the same data \* as those that were previously written. \* @see #writeStringList \*/ public final ArrayList<String> createStringArrayList() { int N = readInt(); if (N < 0) { return null; } ArrayList<String> l = new ArrayList<String>(N); while (N > 0) { l.add(readString()); N--; } return l; } /\*\* \* Read and return a new ArrayList containing IBinder objects from \* the parcel that was written with {@link #writeBinderList} at the \* current dataPosition(). Returns null if the \* previously written list object was null. \* \* @return A newly created ArrayList containing strings with the same data \* as those that were previously written. \* @see #writeBinderList \*/ public final ArrayList<IBinder> createBinderArrayList() { int N = readInt(); if (N < 0) { return null; } ArrayList<IBinder> l = new ArrayList<IBinder>(N); while (N > 0) { l.add(readStrongBinder()); N--; } return l; } /\*\* \* Read into the given List items String objects that were written with \* {@link #writeStringList} at the current dataPosition(). \* \* @return A newly created ArrayList containing strings with the same data \* as those that were previously written. \* @see #writeStringList \*/ public final void readStringList(List<String> list) { int M = list.size(); int N = readInt(); int i = 0; for (; i < M && i < N; i++) { list.set(i, readString()); } for (; i < N; i++) { list.add(readString()); } for (; i < M; i++) { list.remove(N); } } /\*\* \* Read into the given List items IBinder objects that were written with \* {@link #writeBinderList} at the current dataPosition(). \* \* @return A newly created ArrayList containing strings with the same data \* as those that were previously written. \* @see #writeBinderList \*/ public final void readBinderList(List<IBinder> list) { int M = list.size(); int N = readInt(); int i = 0; for (; i < M && i < N; i++) { list.set(i, readStrongBinder()); } for (; i < N; i++) { list.add(readStrongBinder()); } for (; i < M; i++) { list.remove(N); } } /\*\* \* Read and return a new array containing a particular object type from \* the parcel at the current dataPosition(). Returns null if the \* previously written array was null. The array <em>must</em> have \* previously been written via {@link #writeTypedArray} with the same \* object type. \* \* @return A newly created array containing objects with the same data \* as those that were previously written. \* @see #writeTypedArray \*/ public final <T> T[] createTypedArray(Parcelable.Creator<T> c) { int N = readInt(); if (N < 0) { return null; } T[] l = c.newArray(N); for (int i = 0; i < N; i++) { if (readInt() != 0) { l[i] = c.createFromParcel(this); } } return l; } public final <T> void readTypedArray(T[] val, Parcelable.Creator<T> c) { int N = readInt(); if (N == val.length) { for (int i = 0; i < N; i++) { if (readInt() != 0) { val[i] = c.createFromParcel(this); } else { val[i] = null; } } } else { throw new RuntimeException("bad array lengths"); } } /\*\* \* @hide \* @deprecated \*/ @Deprecated public final <T> T[] readTypedArray(Parcelable.Creator<T> c) { return createTypedArray(c); } /\*\* \* Read and return a typed Parcelable object from a parcel. \* Returns null if the previous written object was null. \* The object <em>must</em> have previous been written via \* {@link #writeTypedObject} with the same object type. \* \* @return A newly created object of the type that was previously \* written. \* @see #writeTypedObject \*/ public final <T> T readTypedObject(Parcelable.Creator<T> c) { if (readInt() != 0) { return c.createFromParcel(this); } else { return null; } } /\*\* \* Write a heterogeneous array of Parcelable objects into the Parcel. \* Each object in the array is written along with its class name, so \* that the correct class can later be instantiated. As a result, this \* has significantly more overhead than {@link #writeTypedArray}, but will \* correctly handle an array containing more than one type of object. \* \* @param value The array of objects to be written. \* @param parcelableFlags Contextual flags as per \* {@link Parcelable#writeToParcel(Parcel, int) Parcelable.writeToParcel()}. \* @see #writeTypedArray \*/ public final <T extends Parcelable> void writeParcelableArray(T[] value, int parcelableFlags) { if (value != null) { int N = value.length; writeInt(N); for (int i = 0; i < N; i++) { writeParcelable(value[i], parcelableFlags); } } else { writeInt(-1); } } /\*\* \* Read a typed object from a parcel. The given class loader will be \* used to load any enclosed Parcelables. If it is null, the default class \* loader will be used. \*/ public final Object readValue(ClassLoader loader) { int type = readInt(); switch (type) { case VAL\_NULL: return null; case VAL\_STRING: return readString(); case VAL\_INTEGER: return readInt(); case VAL\_MAP: return readHashMap(loader); case VAL\_PARCELABLE: return readParcelable(loader); case VAL\_SHORT: return (short) readInt(); case VAL\_LONG: return readLong(); case VAL\_FLOAT: return readFloat(); case VAL\_DOUBLE: return readDouble(); case VAL\_BOOLEAN: return readInt() == 1; case VAL\_CHARSEQUENCE: return readCharSequence(); case VAL\_LIST: return readArrayList(loader); case VAL\_BOOLEANARRAY: return createBooleanArray(); case VAL\_BYTEARRAY: return createByteArray(); case VAL\_STRINGARRAY: return readStringArray(); case VAL\_CHARSEQUENCEARRAY: return readCharSequenceArray(); case VAL\_IBINDER: return readStrongBinder(); case VAL\_OBJECTARRAY: return readArray(loader); case VAL\_INTARRAY: return createIntArray(); case VAL\_LONGARRAY: return createLongArray(); case VAL\_BYTE: return readByte(); case VAL\_SERIALIZABLE: return readSerializable(loader); case VAL\_PARCELABLEARRAY: return readParcelableArray(loader); case VAL\_SPARSEARRAY: return readSparseArray(loader); case VAL\_SPARSEBOOLEANARRAY: return readSparseBooleanArray(); case VAL\_BUNDLE: return readBundle(loader); // loading will be deferred case VAL\_PERSISTABLEBUNDLE: return readPersistableBundle(loader); case VAL\_SIZE: return readSize(); case VAL\_SIZEF: return readSizeF(); default: int off = dataPosition() - 4; throw new RuntimeException( "Parcel " + this + ": Unmarshalling unknown type code " + type + " at offset " + off); } } /\*\* \* Read and return a new Parcelable from the parcel. The given class loader \* will be used to load any enclosed Parcelables. If it is null, the default \* class loader will be used. \* \* @param loader A ClassLoader from which to instantiate the Parcelable \* object, or null for the default class loader. \* @return Returns the newly created Parcelable, or null if a null \* object has been written. \* @throws BadParcelableException Throws BadParcelableException if there \* was an error trying to instantiate the Parcelable. \*/ @SuppressWarnings("unchecked") public final <T extends Parcelable> T readParcelable(ClassLoader loader) { Parcelable.Creator<?> creator = readParcelableCreator(loader); if (creator == null) { return null; } if (creator instanceof Parcelable.ClassLoaderCreator<?>) { Parcelable.ClassLoaderCreator<?> classLoaderCreator = (Parcelable.ClassLoaderCreator<?>) creator; return (T) classLoaderCreator.createFromParcel(this, loader); } return (T) creator.createFromParcel(this); } /\*\* \* @hide \*/ @SuppressWarnings("unchecked") public final <T extends Parcelable> T readCreator(Parcelable.Creator<?> creator, ClassLoader loader) { if (creator instanceof Parcelable.ClassLoaderCreator<?>) { Parcelable.ClassLoaderCreator<?> classLoaderCreator = (Parcelable.ClassLoaderCreator<?>) creator; return (T) classLoaderCreator.createFromParcel(this, loader); } return (T) creator.createFromParcel(this); } /\*\* \* @hide \*/ public final Parcelable.Creator<?> readParcelableCreator(ClassLoader loader) { String name = readString(); if (name == null) { return null; } Parcelable.Creator<?> creator; synchronized (mCreators) { HashMap<String, Parcelable.Creator<?>> map = mCreators.get(loader); if (map == null) { map = new HashMap<>(); mCreators.put(loader, map); } creator = map.get(name); if (creator == null) { try { // If loader == null, explicitly emulate Class.forName(String) "caller // classloader" behavior. ClassLoader parcelableClassLoader = (loader == null ? getClass().getClassLoader() : loader); // Avoid initializing the Parcelable class until we know it implements // Parcelable and has the necessary CREATOR field. http://b/1171613. Class<?> parcelableClass = Class.forName(name, false /\* initialize \*/, parcelableClassLoader); if (!Parcelable.class.isAssignableFrom(parcelableClass)) { throw new BadParcelableException("Parcelable protocol requires that the " + "class implements Parcelable"); } Field f = parcelableClass.getField("CREATOR"); if ((f.getModifiers() & Modifier.STATIC) == 0) { throw new BadParcelableException("Parcelable protocol requires " + "the CREATOR object to be static on class " + name); } Class<?> creatorType = f.getType(); if (!Parcelable.Creator.class.isAssignableFrom(creatorType)) { // Fail before calling Field.get(), not after, to avoid initializing // parcelableClass unnecessarily. throw new BadParcelableException("Parcelable protocol requires a " + "Parcelable.Creator object called " + "CREATOR on class " + name); } creator = (Parcelable.Creator<?>) f.get(null); } catch (IllegalAccessException e) { Log.e(TAG, "Illegal access when unmarshalling: " + name, e); throw new BadParcelableException( "IllegalAccessException when unmarshalling: " + name); } catch (ClassNotFoundException e) { Log.e(TAG, "Class not found when unmarshalling: " + name, e); throw new BadParcelableException( "ClassNotFoundException when unmarshalling: " + name); } catch (NoSuchFieldException e) { throw new BadParcelableException("Parcelable protocol requires a " + "Parcelable.Creator object called " + "CREATOR on class " + name); } if (creator == null) { throw new BadParcelableException("Parcelable protocol requires a " + "non-null Parcelable.Creator object called " + "CREATOR on class " + name); } map.put(name, creator); } } return creator; } /\*\* \* Read and return a new Parcelable array from the parcel. \* The given class loader will be used to load any enclosed \* Parcelables. \* \* @return the Parcelable array, or null if the array is null \*/ public final Parcelable[] readParcelableArray(ClassLoader loader) { int N = readInt(); if (N < 0) { return null; } Parcelable[] p = new Parcelable[N]; for (int i = 0; i < N; i++) { p[i] = readParcelable(loader); } return p; } /\*\* \* Read and return a new Serializable object from the parcel. \* \* @return the Serializable object, or null if the Serializable name \* wasn't found in the parcel. \*/ public final Serializable readSerializable() { return readSerializable(null); } private final Serializable readSerializable(final ClassLoader loader) { String name = readString(); if (name == null) { // For some reason we were unable to read the name of the Serializable (either there // is nothing left in the Parcel to read, or the next value wasn't a String), so // return null, which indicates that the name wasn't found in the parcel. return null; } byte[] serializedData = createByteArray(); ByteArrayInputStream bais = new ByteArrayInputStream(serializedData); try { ObjectInputStream ois = new ObjectInputStream(bais) { @Override protected Class<?> resolveClass(ObjectStreamClass osClass) throws IOException, ClassNotFoundException { // try the custom classloader if provided if (loader != null) { Class<?> c = Class.forName(osClass.getName(), false, loader); if (c != null) { return c; } } return super.resolveClass(osClass); } }; return (Serializable) ois.readObject(); } catch (IOException ioe) { throw new RuntimeException("Parcelable encountered " + "IOException reading a Serializable object (name = " + name + ")", ioe); } catch (ClassNotFoundException cnfe) { throw new RuntimeException("Parcelable encountered " + "ClassNotFoundException reading a Serializable object (name = " + name + ")", cnfe); } } // Cache of previously looked up CREATOR.createFromParcel() methods for // particular classes. Keys are the names of the classes, values are // Method objects. private static final HashMap<ClassLoader, HashMap<String, Parcelable.Creator<?>>> mCreators = new HashMap<>(); /\*\* \* @hide for internal use only. \*/ static protected final Parcel obtain(int obj) { throw new UnsupportedOperationException(); } /\*\* \* @hide \*/ static protected final Parcel obtain(long obj) { final Parcel[] pool = sHolderPool; synchronized (pool) { Parcel p; for (int i = 0; i < POOL\_SIZE; i++) { p = pool[i]; if (p != null) { pool[i] = null; if (DEBUG\_RECYCLE) { p.mStack = new RuntimeException(); } p.init(obj); return p; } } } return new Parcel(obj); } //私有构造函数 private Parcel(long nativePtr) { if (DEBUG\_RECYCLE) { mStack = new RuntimeException(); } //Log.i(TAG, "Initializing obj=0x" + Integer.toHexString(obj), mStack); init(nativePtr);//初始化本地指针？ } private void init(long nativePtr) { if (nativePtr != 0) { mNativePtr = nativePtr; mOwnsNativeParcelObject = false; } else {//自己创建包裹 mNativePtr = nativeCreate(); mOwnsNativeParcelObject = true; } } private void freeBuffer() { if (mOwnsNativeParcelObject) { nativeFreeBuffer(mNativePtr); } } private void destroy() { if (mNativePtr != 0) { if (mOwnsNativeParcelObject) { nativeDestroy(mNativePtr); } mNativePtr = 0; } } @Override protected void finalize() throws Throwable { if (DEBUG\_RECYCLE) { if (mStack != null) { Log.w(TAG, "Client did not call Parcel.recycle()", mStack); } } destroy(); } /\* package \*/ void readMapInternal(Map outVal, int N, ClassLoader loader) { while (N > 0) { Object key = readValue(loader); Object value = readValue(loader); outVal.put(key, value); N--; } } /\* package \*/ void readArrayMapInternal(ArrayMap outVal, int N, ClassLoader loader) { if (DEBUG\_ARRAY\_MAP) { RuntimeException here = new RuntimeException("here"); here.fillInStackTrace(); Log.d(TAG, "Reading " + N + " ArrayMap entries", here); } int startPos; while (N > 0) { if (DEBUG\_ARRAY\_MAP) startPos = dataPosition(); String key = readString(); Object value = readValue(loader); if (DEBUG\_ARRAY\_MAP) Log.d(TAG, " Read #" + (N - 1) + " " + (dataPosition() - startPos) + " bytes: key=0x" + Integer.toHexString((key != null ? key.hashCode() : 0)) + " " + key); outVal.append(key, value); N--; } outVal.validate(); } /\* package \*/ void readArrayMapSafelyInternal(ArrayMap outVal, int N, ClassLoader loader) { if (DEBUG\_ARRAY\_MAP) { RuntimeException here = new RuntimeException("here"); here.fillInStackTrace(); Log.d(TAG, "Reading safely " + N + " ArrayMap entries", here); } while (N > 0) { String key = readString(); if (DEBUG\_ARRAY\_MAP) Log.d(TAG, " Read safe #" + (N - 1) + ": key=0x" + (key != null ? key.hashCode() : 0) + " " + key); Object value = readValue(loader); outVal.put(key, value); N--; } } /\*\* \* @hide For testing only. \*/ public void readArrayMap(ArrayMap outVal, ClassLoader loader) { final int N = readInt(); if (N < 0) { return; } readArrayMapInternal(outVal, N, loader); } private void readListInternal(List outVal, int N, ClassLoader loader) { while (N > 0) { Object value = readValue(loader); //Log.d(TAG, "Unmarshalling value=" + value); outVal.add(value); N--; } } private void readArrayInternal(Object[] outVal, int N, ClassLoader loader) { for (int i = 0; i < N; i++) { Object value = readValue(loader); //Log.d(TAG, "Unmarshalling value=" + value); outVal[i] = value; } } private void readSparseArrayInternal(SparseArray outVal, int N, ClassLoader loader) { while (N > 0) { int key = readInt(); Object value = readValue(loader); //Log.i(TAG, "Unmarshalling key=" + key + " value=" + value); outVal.append(key, value); N--; } } private void readSparseBooleanArrayInternal(SparseBooleanArray outVal, int N) { while (N > 0) { int key = readInt(); boolean value = this.readByte() == 1; //Log.i(TAG, "Unmarshalling key=" + key + " value=" + value); outVal.append(key, value); N--; } } /\*\* \* @hide For testing \*/ public long getBlobAshmemSize() { return nativeGetBlobAshmemSize(mNativePtr); }}

二、测试代码

（一）深入解释

一、Android中的Parcel是什么

转自： http://blog.csdn.net/nkmnkm/article/details/6451699

Parcel，翻译过来是“打包”的意思。打包干什么呢？是为了序列化。

如果要在进程之间传递一个整数，很简单，直接传就是行了；如果要传一个字符串，就稍微复杂了点：需先分配一块可以容纳字符串的 内存，然后将字符串复制到内存中，再传递（新手可能问：为啥不直接把字符串的引用传过去呢？学过C/C++的地球人都知道：进程有自己的内存地址空间，一 个进程中的1000地址可能在另一个进程中是100000，java对象的引用跟本上还是内存地址）；再如果要传递一个类的实例呢？也是先为类分配内存， 然后复制一份再传递可以吗？我认为不可以，我至少可以找到一个理由：类中成员除了属性还有方法，即使属性能完整传过去，但还有方法呢？方法是独立于类对象 存在的，所以到另一个进程中再引用同一个方法就要出错了，还是因为独立地址空间的原因。

Android开发中，很经常在各activity之间传递数据，而跟据Android的设计架构，即使同一个程序中的Activity都不一定运行在同 一个进程中，所以处理数据传递时你不能老假设两个activity都运行于同一进程，那么只能按进程间传递数据来处理，使之具有最广泛的适应性。

　 　那么到底如何在进程之间传递类对象呢？简单来说可以这样做：在进程Ａ中把类中的非默认值的属性和类的唯一标志打成包（这就叫序列化），把这个包传递到进 程Ｂ，进程Ｂ接收到包后，跟据类的唯一标志把类创建出来，然后把传来的属性更新到类对象中，这样进程Ａ和进程Ｂ中就包含了两个完全一样的类对象。

二、 探索Android中的Parcel机制（上）

转自：http://blog.csdn.net/caowenbin/article/details/6532217 (作者：曹文斌)

一．先从Serialize说起

我们都知道JAVA中的Serialize机制，译成串行化、序列化……，其作用是能将数据对象存入字节流当中，在需要时重新生成对象。主要应用是利用外部存储设备保存对象状态，以及通过网络传输对象等。

二．Android中的新的序列化机制

在Android系统中，定位为针对内存受限的设备，因此对性能要求更高，另外系统中采用了新的IPC（进程间通信）机制，必然 要求使用性能更出色的对象传输方式。在这样的环境下，Parcel被设计出来，其定位就是轻量级的高效的对象序列化和反序列化机制。

三．Parcel类的背后

在Framework中有parcel类，源码路径是：

Frameworks/base/core/java/android/os/Parcel.java

典型的源码片断如下：

[html] view plain copy print ?

/\*\*

\* Write an integer value into the parcel at the current dataPosition(),

\* growing dataCapacity() if needed.

\*/

public final native void writeInt(int val);

/\*\*

\* Write a long integer value into the parcel at the current dataPosition(),

\* growing dataCapacity() if needed.

\*/

public final native void writeLong(long val);

从中我们看到，从这个源程序文件中我们看不到真正的功能是如何实现的，必须透过JNI往下走了。于是，Frameworks/base/core/jni/android\_util\_Binder.cpp中找到了线索

[html] view plain copy print ?

static void android\_os\_Parcel\_writeInt(JNIEnv\* env, jobject clazz, jint val)

{

Parcel\* parcel = parcelForJavaObject(env, clazz);

if (parcel != NULL) {

const status\_t err = parcel->writeInt32(val);

if (err != NO\_ERROR) {

jniThrowException(env, "java/lang/OutOfMemoryError", NULL);

}

}

}

static void android\_os\_Parcel\_writeLong(JNIEnv\* env, jobject clazz, jlong val)

{

Parcel\* parcel = parcelForJavaObject(env, clazz);

if (parcel != NULL) {

const status\_t err = parcel->writeInt64(val);

if (err != NO\_ERROR) {

jniThrowException(env, "java/lang/OutOfMemoryError", NULL);

}

}

}

## Parcel指针

从这里我们可以得到的信息是函数的实现依赖于Parcel指针，因此还需要找到Parcel的类定义，注意，这里的类已经是用C++语言实现的了。

找到Frameworks/base/include/binder/parcel.h和Frameworks/base/libs/binder/parcel.cpp。终于找到了最终的实现代码了。

有兴趣的朋友可以自己读一下，不难理解，这里把基本的思路总结一下：

1. 整个读写全是在内存中进行，主要是通过malloc()、realloc()、memcpy()等内存操作进行，所以效率比JAVA序列化中使用外部存储器会高很多；

2. 读写时是4字节对齐的，可以看到#define PAD\_SIZE(s) (((s)+3)&~3)这句宏定义就是在做这件事情；

3. 如果预分配的空间不够时newSize = ((mDataSize+len)\*3)/2;会一次多分配50%；

4. 对于普通数据，使用的是mData内存地址，对于IBinder类型的数据以及FileDescriptor使用的是 mObjects内存地址。后者是通过flatten\_binder()和unflatten\_binder()实现的，目的是反序列化时读出的对象就是 原对象而不用重新new一个新对象。

三、探索Android中的Parcel机制（下）

上一篇中我们透过源码看到了Parcel背后的机制，本质上把它当成一个Serialize就可以了，只是它是在内存中完成的序列化和反序列化，利用的是连续的内存空间，因此会更加高效。

我们接下来要说的是Parcel类如何应用。就应用程序而言，最常见使用Parcel类的场景就是在Activity间传递数据。没错，在Activity间使用Intent传递数据的时候，可以通过Parcelable机制传递复杂的对象。

在下面的程序中，MyColor用于保存一个颜色值，MainActivity在用户点击屏幕时将MyColor对象设成红色， 传递到SubActivity中，此时SubActivity的TextView显示为红色的背景；当点击SubActivity时，将颜色值改为绿色， 返回MainActivity，期望的是MainActivity的TextView显示绿色背景。

来看一下MyColor类的实现代码：

[html] view plain copy print ?

package com.wenbin.test;

import android.graphics.Color;

import android.os.Parcel;

import android.os.Parcelable;

/\*\*

\* @author 曹文斌

\* http://blog.csdn.net/caowenbin

\*

\*/

public class MyColor implements Parcelable {

private int color=Color.BLACK;

MyColor(){

color=Color.BLACK;

}

MyColor(Parcel in){

color=in.readInt();

}

public int getColor(){

return color;

}

public void setColor(int color){

this.color=color;

}

@Override

public int describeContents() {

return 0;

}

@Override

public void writeToParcel(Parcel dest, int flags) {

dest.writeInt(color);

}

public static final Parcelable.Creator<MyColor> CREATOR

= new Parcelable.Creator<MyColor>() {

public MyColor createFromParcel(Parcel in) {

return new MyColor(in);

}

public MyColor[] newArray(int size) {

return new MyColor[size];

}

};

}

该类实现了Parcelable接口，提供了默认的构造函数，同时也提供了可从Parcel对象开始的构造函数，另外还实现了一个static的构造器用于构造对象和数组。代码很简单，不一一解释了。

再看MainActivity的代码：

[html] view plain copy print ?

package com.wenbin.test;

import android.app.Activity;

import android.content.Intent;

import android.graphics.Color;

import android.os.Bundle;

import android.view.MotionEvent;

/\*\*

\* @author 曹文斌

\* http://blog.csdn.net/caowenbin

\*

\*/

public class MainActivity extends Activity {

private final int SUB\_ACTIVITY=0;

private MyColor color=new MyColor();

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

}

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode==SUB\_ACTIVITY){

if (resultCode==RESULT\_OK){

if (data.hasExtra("MyColor")){

color=data.getParcelableExtra("MyColor"); //Notice

findViewById(R.id.text).setBackgroundColor(color.getColor());

}

}

}

}

@Override

public boolean onTouchEvent(MotionEvent event){

if (event.getAction()==MotionEvent.ACTION\_UP){

Intent intent=new Intent();

intent.setClass(this, SubActivity.class);

color.setColor(Color.RED);

intent.putExtra("MyColor", color);

startActivityForResult(intent,SUB\_ACTIVITY);

}

return super.onTouchEvent(event);

}

}

下面是SubActivity的代码：

[html] view plain copy print ?

package com.wenbin.test;

import android.app.Activity;

import android.content.Intent;

import android.graphics.Color;

import android.os.Bundle;

import android.view.MotionEvent;

import android.widget.TextView;

/\*\*

\* @author 曹文斌

\* http://blog.csdn.net/caowenbin

\*

\*/

public class SubActivity extends Activity {

private MyColor color;

@Override

public void onCreate(Bundle savedInstanceState) {

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

((TextView)findViewById(R.id.text)).setText("SubActivity");

Intent intent=getIntent();

if (intent!=null){

if (intent.hasExtra("MyColor")){

color=intent.getParcelableExtra("MyColor");

findViewById(R.id.text).setBackgroundColor(color.getColor());

}

}

}

@Override

public boolean onTouchEvent(MotionEvent event){

if (event.getAction()==MotionEvent.ACTION\_UP){

Intent intent=new Intent();

if (color!=null){

color.setColor(Color.GREEN);

intent.putExtra("MyColor", color);

}

setResult(RESULT\_OK,intent);

finish();

}

return super.onTouchEvent(event);

}

}

下面是main.xml的代码：

[html] view plain copy print ?

<?xml version="1.0" encoding="utf-8"?>

<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"

android:orientation="vertical"

android:layout\_width="fill\_parent"

android:layout\_height="fill\_parent"

>

<TextView

android:layout\_width="fill\_parent"

android:layout\_height="wrap\_content"

android:text="@string/hello"

android:id="@+id/text"

/>

</LinearLayout>

注意的是在MainActivity的onActivityResult()中，有一句 color=data.getParcelableExtra("MyColor")，这说明的是反序列化后是一个新的MyColor对象，因此要想使用 这个对象，我们做了这个赋值语句。

记得在上一篇《探索Android中的Parcel机制（上）》 中提到，如果数据本身是IBinder类型，那么反序列化的结果就是原对象，而不是新建的对象，很显然，如果是这样的话，在反序列化后在 MainActivity中就不再需要color=data.getParcelableExtra("MyColor")这句了。因此，换一种 MyColor的实现方法，令其中的int color成员变量使用IBinder类型的成员变量来表示。

新建一个BinderData类继承自Binder，代码如下：

[html] view plain copy print ?

package com.wenbin.test;

import android.os.Binder;

/\*\*

\* @author 曹文斌

\* http://blog.csdn.net/caowenbin

\*

\*/

public class BinderData extends Binder {

public int color;

}

修改MyColor的代码如下：

[html] view plain copy print ?

package com.wenbin.test;

import android.graphics.Color;

import android.os.Parcel;

import android.os.Parcelable;

/\*\*

\* @author 曹文斌

\* http://blog.csdn.net/caowenbin

\*

\*/

public class MyColor implements Parcelable {

private BinderData data=new BinderData();

MyColor(){

data.color=Color.BLACK;

}

MyColor(Parcel in){

data=(BinderData) in.readValue(BinderData.class.getClassLoader());

}

public int getColor(){

return data.color;

}

public void setColor(int color){

data.color=color;

}

@Override

public int describeContents() {

return 0;

}

@Override

public void writeToParcel(Parcel dest, int flags) {

dest.writeValue(data);

}

public static final Parcelable.Creator<MyColor> CREATOR

= new Parcelable.Creator<MyColor>() {

public MyColor createFromParcel(Parcel in) {

return new MyColor(in);

}

public MyColor[] newArray(int size) {

return new MyColor[size];

}

};

}

去掉MainActivity的onActivityResult()中的color=data.getParcelableExtra("MyColor")一句，变成：

[html] view plain copy print ?

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data) {

super.onActivityResult(requestCode, resultCode, data);

if (requestCode==SUB\_ACTIVITY){

if (resultCode==RESULT\_OK){

if (data.hasExtra("MyColor")){

findViewById(R.id.text).setBackgroundColor(color.getColor());

}

}

}

}

再次运行程序，结果符合预期。

以上就是Parcel在应用程序中的使用方法，与Serialize还是挺相似的，详细的资料当然还是要参考Android SDK的开发文档了。

三、Android Parcel理解

android 中Parcel 的使用，他是一个存储基本数据类型和引用数据类型的容器，在andorid 中通过IBinder来绑定数据在进程间传递数据。

Parcel parcel = Parcel.obtain();// 获取一个Parcel 对象

下面就可以对其进行方法进行操作了，createXXX(),wirteXXX(),readXXX(),

其中 dataPosition(),返回当前Parcel 当前对象存储数据的偏移量，而setDataPosition(),设置当前Parcel 对象的偏移量，方便读取parcel 中的数据，可问题就出在我读取出来的数据要么是空（null）,要么永远是第一个偏移量处的值，存储和读取数据的。Parcel采用什么机制实现的，是以 什么形式的存储的，然后我才能任意对其操作，读取目标数据。

基本数据类型的取值范围，

boolean 1bit

short 16bit

int 32bit

long 64bit

float 32bit

double 64bit

char 16bit

byte 8bit

由此我 可以猜想，Parcel 32bit 作为基本单位存储写入的变量,4byte\*8=32bit,在内存中的引用地址变量是采用16进制进行编码，且作为偏移量，即偏移量是4的倍 数，0,4,8,12,16,20,24,28,32,36,40,44,48......4\*N,

f(x) = 4\*y｛y>=0&y是自然数｝

我想绝对不会出现向偏移量是3，6，9这样的数据。。。

由此我们可以推断出，无论他存储的是基本数据类型或引用数据类型的变量，都是以32bit基本单位作为偏移量，

parcel.writeInt(1);

parcel.writeInt(2);

parcel.writeInt(3);

parcel.writeInt(4);

parcel.writeInt(5);

parcel.writeInt(6);

parcel.writeInt(7);

parcel.writeInt(81011111);

parcel.writeFloat(1f);

parcel.writeFloat(1000000000000000000000000000000000000f);

parcel.writeXXX(), 每写一次数据，在32bit的空间里能够存储要放入的变量，怎只占一个偏移量，也就之一动4个位置，而当存储的数据如 parcel.writeFloat(1000000000000000000000000000000000000f);他就自动往后移动，

parcel.writeString("a");

parcel.writeString("b");

parcel.writeString("d");

parcel.writeString("c");

和

parcel.writeString("abcd"); 的区别。有此可见，他的内存的分配原来是这样的。

那我怎样才能把我存进去的书据依次的去出来呢？setDataPosition(),设置parcel 的偏移量，在readXXX(),读取数据

int size = parcel.dataSize();

int i = 0;

while (i <= size ) {

parcel.setDataPosition(i);

int curr\_int = parcel.readInt();

i+=4;

int j = 0;

j++;

}

由此可 见parcel 写入数据是按照32bit 为基本的容器，依次存储写入的数据，基本和引用（其实引用的也是有多个基本数据类型组合而成OBJECTS－属性｜方法），读取的时候我们就可以按照这种 规律根据目标数据的偏移量的位置（curr\_position），以及偏移量的大小(size)，,取出已经存进去的数据了

int i ＝ curr\_position；

while (i <= size ) {

parcel.setDataPosition(i);

int curr\_int = parcel.readXXXt();

i+=4;

int j = 0;

j++;

}

这样就ok 了

他的createXXX（）方法现在没用，用了在说吧！

总结一句话，java 中 基本数据类型的取值范围，引用类型的数据，相当于c中的指针，以及各进制之间的相互转换和灵活的引用，以及定制自己想要的任意进制数据类型。

四、 Android开发：什么是Parcel(2)

转自：http://blog.csdn.net/nkmnkm/article/details/6453391

上回书解释了IBinder,这回详细解释一下Parcel，以下是对android sdk 文档的翻议：

Parcel是一个容器，它主要用于存储序列化数据，然后可以通过Binder在进程间传递这些数据（要了解为什么要序列化，请参考：http://blog.csdn.net/nkmnkm/archive/2011/05/28/6451699.aspx）。Parcel可以包含原始数据类型（用各种对应的方法写入，比如writeInt(),writeFloat()等），可以包含Parcelable对象，它还包含了一个活动的IBinder对象的引用，这个引用导致另一端接收到一个指向这个IBinder的代理IBinder。

注：Parcel不是一般目的的序列化机制。这个类被设计用于高性能的IPC传输。因此不适合把Parcel写入永久化存储中，因为Parcel中的数据类型的实现的改变会导致旧版的数据不可读。

Parcel的一坨一坨的API用于解决不同类型数据的读写。这些函数们主要有六种类型。

１原始类

这类方法们主要读写原始数据类型。它们是：writeByte(byte), readByte(), writeDouble(double), readDouble(), writeFloat(float), readFloat(), writeInt(int), readInt(), writeLong(long), readLong(), writeString(String), readString(). 大多数其它数据的操作都是基于这些方法。

２原始数组类

这类方法用于读写原始数据组成的数组。在向数组写数据时先写入数组的长度再写入数据。读数组的方法可以将数据读到已存在的数组中，也可以创建并返回一个新数组。它们是：

writeBooleanArray(boolean[]), readBooleanArray(boolean[]), createBooleanArray()

writeByteArray(byte[]), writeByteArray(byte[], int, int), readByteArray(byte[]), createByteArray()

writeCharArray(char[]), readCharArray(char[]), createCharArray()

writeDoubleArray(double[]), readDoubleArray(double[]), createDoubleArray()

writeFloatArray(float[]), readFloatArray(float[]), createFloatArray()

writeIntArray(int[]), readIntArray(int[]), createIntArray()

writeLongArray(long[]), readLongArray(long[]), createLongArray()

writeStringArray(String[]), readStringArray(String[]), createStringArray().

writeSparseBooleanArray(SparseBooleanArray), readSparseBooleanArray().

3 Parcelable类

Parcelable为对象从Parcel中读写自己提供了极其高效的协议。你可以使用直接的方法 writeParcelable(Parcelable, int) 和 readParcelable(ClassLoader) 或 writeParcelableArray(T[], int) and readParcelableArray(ClassLoader) 进行读写。这些方法们把类的信息和数据都写入Parcel，以使将来能使用合适的类装载器重新构造类的实例。

还有一些方法提供了更高效的操作Parcelable们的途径，它们是：writeTypedArray(T[], int), writeTypedList(List), readTypedArray(T[], Parcelable.Creator) and readTypedList(List, Parcelable.Creator)。这些方法不会写入类的信息，取而代之的是：读取时必须能知道数据属于哪个类并传入正确的 Parcelable.Creator来创建对象而不是直接构造新对象。（更加高效的读写单个Parcelable对象的方法是：直接调用 Parcelable.writeToParcel()和Parcelable.Creator.createFromParcel()）

4 Bundles类

Bundles是一种类型安全的Map型容器，可用于存储任何不同类型的数据。它具有很多对讀写数据的性能优化，并且它的类型安全机制避免了当把它的数据 封送到Parcel中时由于类型错误引起的BUG的调试的麻烦，可以使用的方法为： writeBundle(Bundle), readBundle(), and readBundle(ClassLoader)。

5 活动对象类

Parcel的一个非同寻常的特性是读写活对象的能力。对于活动对象，它们的内容实际上并没有写入，而是仅写入了一个令牌来引用这个对象。当从Parcel中读取这个对象时，你不会获取一个新的对象实例，而是直接得到那个写入的对象。有两种活动对象可操作：

Binder对象。它是 Android跨进程通讯的基础。这种对象可被写入Parcel，并在读取时你将得到原始的对象或一个代理对象（可以想象：在进程内时得到原始的对象，在 进程间时得到代理对象）。可以使用的方法们是： writeStrongBinder(IBinder), writeStrongInterface(IInterface), readStrongBinder(), writeBinderArray(IBinder[]), readBinderArray(IBinder[]), createBinderArray(), writeBinderList(List), readBinderList(List), createBinderArrayList()。

FileDescriptor对象。 它代表了原始的Linux文件描述符，它可以被写入Parcel并在读取时返回一个ParcelFileDescriptor对象用于操作原始的文件描述 符。ParcelFileDescriptor是原始描述符的一个复制：对象和fd不同，但是都操作于同一文件流，使用同一个文件位置指针，等等。可以使 用的方法是：writeFileDescriptor(FileDescriptor), readFileDescriptor()。

6无类型容器类

一类final方法，用于读写标准的java容器类。这些方法们是：writeArray(Object[]), readArray(ClassLoader), writeList(List), readList(List, ClassLoader), readArrayList(ClassLoader), writeMap(Map), readMap(Map, ClassLoader), writeSparseArray(SparseArray), readSparseArray(ClassLoader)。

三、参考

1.android parcelable 详解

http://jingyan.baidu.com/article/f96699bbae42cd894e3c1b15.html

2. 探索Android中的Parcel

<http://my.oschina.net/zhuzihasablog/blog/102316>

# 10 android进程间使用例子

<http://note.youdao.com/noteshare?id=4e5cd8395d29d2e8fc0dc4e026e5812a&sub=D711E7F7F24446A7867324E746876F08>

参考：http://blog.csdn.net/victory08/article/details/8696252

由于android系统中应用程序之间不能共享内存。因此，在不同应用程序之间交互数据（跨进程通讯）就稍微麻烦一些。在android SDK中提供了4种用于跨进程通讯的方式。这4种方式正好对应于android系统中4种应用程序组件：Activity、Content Provider、Broadcast和Service。其中Activity可以跨进程调用其他应用程序的Activity；Content Provider可以跨进程访问其他应用程序中的数据（以Cursor对象形式返回），当然，也可以对其他应用程序的数据进行增、删、改操 作；Broadcast可以向android系统中所有应用程序发送广播，而需要跨进程通讯的应用程序可以监听这些广播；Service和Content Provider类似，也可以访问其他应用程序中的数据，但不同的是，Content Provider返回的是Cursor对象，而Service返回的是Java对象，这种可以跨进程通讯的服务叫AIDL服务。

完整示例请参阅本文提供的源代码。

方式一：访问其他应用程序的Activity

Activity既可以在进程内（同一个应用程序）访问，也可以跨进程访问。如果想在同一个应用程序中访问Activity，需要指定Context对象和Activity的Class对象，代码如下：

[java] view plaincopy

Intent intent = new Intent(this, Test.class);

startActivity(intent);

Activity的跨进程访问与进程内访问略有不同。虽然它们都需要Intent对象，但跨进程访问并不需要指定Context对象和Activity的Class对象，而需要指定的是要访问的Activity所对应的Action（一个字符串）。有些Activity还需要指定一个Uri（通过Intent构造方法的第2个参数指定）。

在android系统中有很多应用程序提供了可以跨进程访问的Activity，例如，下面的代码可以直接调用拨打电话的Activity。

[java] view plaincopy

Intent callIntent = new Intent(Intent.ACTION\_CALL, Uri.parse("tel:12345678");

startActivity(callIntent);

执行上面的代码后，系统会自动拨号，界面如图1所示。

在调用拨号程序的代码中使用了一个Intent.ACTION\_CALL常量，该常量的定义如下：

[java] view plaincopy

public static final String ACTION\_CALL = "android.intent.action.CALL";

这个常量是一个字符串常量，也是我们在这节要介绍的跨进程调用Activity的关键。如果在应用程序中要共享某个Activity，需要为这个Activity指定一个字符串ID，也就是Action。也可以将这个Action看做这个Activity的key。在其他的应用程序中只要通过这个Action就可以找到与Action对应的Activity，并通过startActivity方法来启动这个Activity。

下面先来看一下如何将应用程序的Activity共享出来，读者可按如下几步来共享Activity：

1. 在AndroidManifest.xml文件中指定Action。指定Action要使用<action>标签，并在该标签的android:name属性中指定Action

2. 在AndroidManifest.xml文件中指定访问协议。在指定Uri（Intent类的第2个参数）时需要访问协议。访问协议需要使 用<data>标签的android:scheme属性来指定。如果该属性的值是“abc”，那么Uri就应该是“abc://Uri的主体 部分”，也就是说，访问协议是Uri的开头部分。

3. 通过getIntent().getData().getHost()方法获得协议后的Uri的主体部分。这个Host只是个称谓，并不一定是主机名。读者可以将其看成是任意的字符串。

4. 从Bundle对象中获得其他应用程序传递过来的数据。

5. 这一步当然是获得数据后做进一步的处理了。至于如何处理这些数据，就得根据具体的需求决定了。

下面来根据这些步骤共享一个Activity。首先建立一个android工程（ActionActivity），工程的主Activity是Main。在 本例中我们会共享这个Main类。首先打开AndroidManifest.xml文件，添加一个<activity>标签，并重新定义了Main的相应属性。AndroidManifest.xml文件的内容如下：

[java] view plaincopy

<!-- 重新配置Main -->

<activity android:name=".Main" android:label="@string/app\_name">

<intent-filter>

<action android:name="net.blogjava.mobile.MYACTION" />

<data android:scheme="info" />

<category android:name="android.intent.category.DEFAULT" />

</intent-filter>

</activity>

在配置AndroidManifest.xml时要注意，不能在同一个<activity>中配置多个动作，否则会覆盖MAIN动作以使该程序无法正常启动（虽然其他应用程序调用Main是正常的）。

从上面的代码可以看出，<action>标签的android:name属性值是net.blogjava.mobile.MYACTION，这就是Main自定义的动作。<data>标签指定了Url的协议。如果指定 了<data>标签的android:scheme属性值（info），则在调用Main时需要使用如下的URL:

[java] view plaincopy

info://任意字符串

一般<category>标签的android:name属性值可以设成android.intent.category.DEFAULT。

下面来看看如何在Main类的onCreate方法中获得其他应用程序传递过来的数据。

package net.blogjava.mobile.actionactivity;

... ...

public class Main extends Activity implements OnClickListener

{

private EditText editText;

@Override

public void onClick(View view)

{

// 单击按钮，会显示文本框中的内容（以Toast信息框形式显示）

Toast.makeText(this , editText.getText().toString(), Toast.LENGTH\_LONG)

.show();

}

@Override

public void onCreate(Bundle savedInstanceState)

{

super .onCreate(savedInstanceState);

setContentView(R.layout.main);

Button button = (Button) findViewById(R.id.button);

button.setOnClickListener(this );

editText = (EditText) findViewById(R.id.edittext);

// 获得其他应用程序传递过来的数据

if (getIntent().getData() != null )

{

// 获得Host，也就是info://后面的内容

String host = getIntent().getData().getHost();

Bundle bundle = getIntent().getExtras();

// 其他的应用程序会传递过来一个value值，在该应用程序中需要获得这个值

String value = bundle.getString("value" );

// 将Host和Value组合在一下显示在EditText组件中

editText.setText(host + ":" + value);

// 调用了按钮的单击事件，显示Toast信息提示框

onClick(button);

}

}

}

[java] view plaincopy

package net.blogjava.mobile.actionactivity;

... ...

public class Main extends Activity implements OnClickListener

{

private EditText editText;

@Override

public void onClick(View view)

{

// 单击按钮，会显示文本框中的内容（以Toast信息框形式显示）

Toast.makeText(this, editText.getText().toString(), Toast.LENGTH\_LONG)

.show();

}

@Override

public void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(R.layout.main);

Button button = (Button) findViewById(R.id.button);

button.setOnClickListener(this);

editText = (EditText) findViewById(R.id.edittext);

// 获得其他应用程序传递过来的数据

if (getIntent().getData() != null)

{

// 获得Host，也就是info://后面的内容

String host = getIntent().getData().getHost();

Bundle bundle = getIntent().getExtras();

// 其他的应用程序会传递过来一个value值，在该应用程序中需要获得这个值

String value = bundle.getString("value");

// 将Host和Value组合在一下显示在EditText组件中

editText.setText(host + ":" + value);

// 调用了按钮的单击事件，显示Toast信息提示框

onClick(button);

}

}

}

从上面的程序可以看出，首先通过getIntent().getData()来判断其他的应用程序是否传递了Uri（getData方法返回了一个Uri 对象）。如果运行该程序，Uri为null，因此，不会执行if语句里面的代码。当其他的应用程序传递了Uri对象后，系统会执行if语句里面的代码。当 运行ActionActivity后，在文本框中输入“Running”，单击“显示文本框的内容”按钮，会显示如图2所示的Toast提示信息框。

下面来看一下其他的应用程序是如何调用ActionActivity中的Main。新建一个android工程（InvokeActivity），并添加一个按钮，按钮的单击事件方法代码如下：

public void onClick(View view)

{

// 需要使用Intent类的第2个参数指定Uri

Intent intent = new Intent("net.blogjava.mobile.MYACTION" , Uri

.parse("info://调用其他应用程序的Activity" ));

// 设置value属性值

intent.putExtra("value" , "调用成功" );

// 调用ActionActivity中的Main

startActivity(intent);

}

[java] view plaincopy

public void onClick(View view)

{

// 需要使用Intent类的第2个参数指定Uri

Intent intent = new Intent("net.blogjava.mobile.MYACTION", Uri

.parse("info://调用其他应用程序的Activity"));

// 设置value属性值

intent.putExtra("value", "调用成功");

// 调用ActionActivity中的Main

startActivity(intent);

}

在运行InvokeActivity之前，先要运行ActionActivity以便在android模拟器中安装该程序。然后单击InvokeActivity中的按钮，就会显示如图3所示的效果。

当然，也可以使用startActivityForResult方法来启动其他应用程序的Activity，以便获得Activity的返回值。例如，可以将ActionActivity中Main类的onClick代码修改为下面的形式。

public void onClick(View view)

{

Toast.makeText(this , editText.getText().toString(), Toast.LENGTH\_LONG).show();

Intent intent = new Intent();

// 设置要返回的属性值

intent.putExtra("result" , editText.getText().toString());

// 设置返回码和Intent对象

setResult(2 , intent);

// 关闭Activity

finish();

}

[java] view plaincopy

public void onClick(View view)

{

Toast.makeText(this, editText.getText().toString(), Toast.LENGTH\_LONG).show();

Intent intent = new Intent();

// 设置要返回的属性值

intent.putExtra("result", editText.getText().toString());

// 设置返回码和Intent对象

setResult(2, intent);

// 关闭Activity

finish();

}

然后在InvokeActivity中使用下面的代码来调用Main。

intent = new Intent("net.blogjava.mobile.MYACTION" , Uri

.parse("info://调用其他应用程序的Activity" ));

// 传递数据

intent.putExtra("value" , "调用成功" );

startActivityForResult(intent, 1 ); // 1为请求码

[java] view plaincopy

intent = new Intent("net.blogjava.mobile.MYACTION", Uri

.parse("info://调用其他应用程序的Activity"));

// 传递数据

intent.putExtra("value", "调用成功");

startActivityForResult(intent, 1); // 1为请求码

要想接收Activity返回的值，需要覆盖onActivityResult事件方法，代码如下：

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data)

{

Toast.makeText(this , "返回值：" + data.getExtras().getString("result" ),

Toast.LENGTH\_LONG).show();

}

[java] view plaincopy

@Override

protected void onActivityResult(int requestCode, int resultCode, Intent data)

{

Toast.makeText(this, "返回值：" + data.getExtras().getString("result"),

Toast.LENGTH\_LONG).show();

}

当单击InvokeActivity中的相应按钮后，并且Main关闭后，会显示如图4所示的Toast信息提示框。

从本节的介绍可以看出，跨进程访问Activity（访问其他应用程序中的Activity）主要是通过一个Action来完成的，如果要传递数据，还需 要指定一个Uri。当然，传递数据也可以通过Intent来完成。传递数据的过程可以是双向的。如果要想从调用的Activity中返回数据，就需要使用startActivityForResult方法来启动Activity了。

## 方式二：Content Provider

Android应用程序可以使用文件或SqlLite数据库来存储数据。Content Provider提供了一种在多个应用程序之间数据共享的方式（跨进程共享数据）。应用程序可以利用Content Provider完成下面的工作

1. 查询数据

2. 修改数据

3. 添加数据

4. 删除数据

虽然Content Provider也可以在同一个应用程序中被访问，但这么做并没有什么意义。Content Provider存在的目的向其他应用程序共享数据和允许其他应用程序对数据进行增、删、改操作。

Android系统本身提供了很多Content Provider，例如，音频、视频、联系人信息等等。我们可以通过这些Content Provider获得相关信息的列表。这些列表数据将以Cursor对象返回。因此，从Content Provider返回的数据是二维表的形式。

对于访问Content Provider的程序，需要使用ContentResolver对象。该对象需要使用getContentResolver方法获得，

[java] view plaincopy

ContentResolver cr = getContentResolver();

与Activity一样，Content Provider也需要与一个URI对应。每一个Content Provider可以控制多个数据集，在这种情况下，每一个数据集会对应一个单独的URI。所有的URI必须以“content://”开头。

为了程序更容易维护，也为了简化程序代码，一般将URI定义成一个常量。例如，下面的常量表示系统的联系人电话号码。

[java] view plaincopy

android.provider.Contacts.Phones.CONTENT\_URI

下面来看一下编写Content Provider的具体步骤。

1. 编写一个继承于android.content.ContentProvider的子类。该类是ContentProvider的核心类。在该类中会实现query、insert、update及delete方法。实际上调用ContentResolver类的这4个方法就是调用ContentProvider类中与之要对应的方法。在本文中只介绍query。至于insert、update、delete和query的用法类 似。也是通过Uri传递参数，然后在这些方法中接收这些参数，并做进一步地处理。

2. 在AndroidManifest.xml文件中配置ContentProvider。要想唯一确定一个ContentProvider，需要指定这个ContentProvider的URI，除此之外，还需要指定URI所对应的ContentProvider类。这有些象Servlet的定义，除了要 指定Servlet对应的Web地址，还要指定这个地址所对应的Servlet类。

现在来看一下Uri的具体格式，先看一下如图5所示的URI。

下面对图5所示的URI的4个部分做一下解释。

A：Content Provider URI的固定前缀，也就是说，所有的URI必须以content://开头。

B：URI中最重要的部分。该部分是Content Provider的唯一标识。对于第三方应用程序来说，该部分最后使用完整的类名（包名+类名），以确保URI的唯一性。该部分需要在AndroidManifest.xml文件中<provider>标签中定义，代码如下：

<provider name=".TransportationProvider" authorities="com.example.transportationprovider"

. . . >

[java] view plaincopy

<provider name=".TransportationProvider" authorities="com.example.transportationprovider"

. . . >

C：这部分是URI的路径（path）。表示URI中各种被请求的数据。这部分是可选的， 如果Content Provider仅仅提供一种请求的数据，那么这部分可以省略。如果Content Provider要提供多种请求数据。就需要添加多个路径，甚至是子路径。例如，“land/bus”、“land/train”、“sea/ship” 就指定了3种可能提供的数据。

D：这部分也是可选的。如果要传递一个值给Content Provider，可以通过这部分传递。当然，如果不需要传值，这部分也可以省略，省略后的URI如下所示：

content://com.example.transportationprovider/trains

[java] view plaincopy

content://com.example.transportationprovider/trains

本例利用了《基于android SDK1.5的英文电子词典的实现》一文中实现的电子词典程序。通过ContentProvider，将电子词典的查词功能共享成Cursor对象。这样 其他的应用程序就可以通过ContentProvider来查词英文单词了。关于英文词典的具体实现细节，读者可以通过如下的地址查看《基于android SDK1.5的英文电子词典的实现》一文。

http://www. androidsdn.com/article/show/111

[java] view plaincopy

http://www.ophonesdn.com/article/show/111

在电子词典程序中需要一个DictionaryContentProvider类，该类是ContentProvider的子类。在该类中实现了query方法，并根据不同的URI来返回不同的结果。让我们先看一下DictionaryContentProvider类，然后再对这些代码做一些解 释。

... ...

public class DictionaryContentProvider extends ContentProvider

{

private static UriMatcher uriMatcher;

private static final String AUTHORITY = "net.blogjava.mobile.dictionarycontentprovider" ;

private static final int SINGLE\_WORD = 1 ;

private static final int PREFIX\_WORDS = 2 ;

public static final String DATABASE\_PATH = android.os.Environment

.getExternalStorageDirectory().getAbsolutePath()

+ "/dictionary" ;

public static final String DATABASE\_FILENAME = "dictionary.db" ;

private SQLiteDatabase database;

static

{

// 添加访问ContentProvider的Uri

uriMatcher = new UriMatcher(UriMatcher.NO\_MATCH);

uriMatcher.addURI(AUTHORITY, "single" , SINGLE\_WORD);

uriMatcher.addURI(AUTHORITY, "prefix/\*" , PREFIX\_WORDS);

}

// 该方法在Activity的onCreate方法之前调用

@Override

public boolean onCreate()

{

database = openDatabase();

return true ;

}

// 在本例中只实现了query方法，其他的方法（insert、update和delete）与query方法的实现

// 类似

@Override

public Cursor query(Uri uri, String[] projection, String selection,

String[] selectionArgs, String sortOrder)

{

Cursor cursor = null ;

switch (uriMatcher.match(uri))

{

case SINGLE\_WORD:

// 查找指定的单词

cursor = database.query("t\_words" , projection, selection,

selectionArgs, null , null , sortOrder);

break ;

case PREFIX\_WORDS:

String word = uri.getPathSegments().get(1 );

// 查找以指定字符串开头的单词集合

cursor = database

.rawQuery(

"select english as \_id, chinese from t\_words where english like ?" ,

new String[]

{ word + "%" });

break ;

default :

throw new IllegalArgumentException("<" + uri + ">格式不正确." );

}

return cursor;

}

... ...

}

[java] view plaincopy

... ...

public class DictionaryContentProvider extends ContentProvider

{

private static UriMatcher uriMatcher;

private static final String AUTHORITY = "net.blogjava.mobile.dictionarycontentprovider";

private static final int SINGLE\_WORD = 1;

private static final int PREFIX\_WORDS = 2;

public static final String DATABASE\_PATH = android.os.Environment

.getExternalStorageDirectory().getAbsolutePath()

+ "/dictionary";

public static final String DATABASE\_FILENAME = "dictionary.db";

private SQLiteDatabase database;

static

{

// 添加访问ContentProvider的Uri

uriMatcher = new UriMatcher(UriMatcher.NO\_MATCH);

uriMatcher.addURI(AUTHORITY, "single", SINGLE\_WORD);

uriMatcher.addURI(AUTHORITY, "prefix/\*", PREFIX\_WORDS);

}

// 该方法在Activity的onCreate方法之前调用

@Override

public boolean onCreate()

{

database = openDatabase();

return true;

}

// 在本例中只实现了query方法，其他的方法（insert、update和delete）与query方法的实现

// 类似

@Override

public Cursor query(Uri uri, String[] projection, String selection,

String[] selectionArgs, String sortOrder)

{

Cursor cursor = null;

switch (uriMatcher.match(uri))

{

case SINGLE\_WORD:

// 查找指定的单词

cursor = database.query("t\_words", projection, selection,

selectionArgs, null, null, sortOrder);

break;

case PREFIX\_WORDS:

String word = uri.getPathSegments().get(1);

// 查找以指定字符串开头的单词集合

cursor = database

.rawQuery(

"select english as \_id, chinese from t\_words where english like ?",

new String[]

{ word + "%" });

break;

default:

throw new IllegalArgumentException("<" + uri + ">格式不正确.");

}

return cursor;

}

... ...

}

关于DictionaryContentProvider类的代码需要做如下的解释。

1. 在DictionaryContentProvider类的开头定义的AUTHORITY是访问ContentProvider的URI的前半部分。

2. 访问ContentProvider的URI的后半部分由uriMatcher.addURI(...)方法指定。该方法的第1个参数就是AUTHORITY（Uri的前半部分），第2个参数是Uri的后半部分，第3个参数是与第2个参数值对应的代码。当其他的应用程序通过Uri访问ContentProvider时。系统解析Uri后，将addURI方法的第2个参数值转换成与之对应的代码（第3个参数值）。

3. addURI的第2个参数值可以使用通配符。例如，prefix/\*中的\*表示所有字符。prefix/abc、prefix/xxx都会匹配成功。

4. 访问ContentProvider的URI是addURI的第1个和第2个参数值的组件，例如，按着DictionaryContentProvider中设置的两个URI，可以分别匹配下面的两个URI。

content://net.blogjava.mobile.dictionarycontentprovider/single

content://net.blogjava.mobile.dictionarycontentprovider/prefix/wo

[java] view plaincopy

content://net.blogjava.mobile.dictionarycontentprovider/single

content://net.blogjava.mobile.dictionarycontentprovider/prefix/wo

要注意的是，访问ContentProvider的URI必须以“content://”开头。

5. 在query方法中建议使用SQLiteDatabase对象的query方法查询。因为query方法的参数正好和DictionaryContentProvider类中的query方法的参数对应，这样使用起来比较方便。

6. 由于安装了ContentProvider的应用程序会先调用ContentProvider的onCreate方法（该方法会在Activity的onCreate方法之前调用），因此，只需要将打开或复制数据库的方法（openDatabase）放在DictionaryContentProvider类中，并在onCreate方法中调用即可。

7. 在DictionaryContentProvider类中只实现了query方法。在该方法中判断了其他应用程序发送的是哪一个Uri。并进行相应的处理。这两个Uri一个是查询指定单词的，另外一个是查询以某个字符串开头的所有单词的（用于显示单词列表）。

下面在AndroidManifest.xml文件中配置DictionaryContentProvider类。

<provider android:name="DictionaryContentProvider"

android:authorities="net.blogjava.mobile.dictionarycontentprovider" />

[java] view plaincopy

<provider android:name="DictionaryContentProvider"

android:authorities="net.blogjava.mobile.dictionarycontentprovider" />

OK，现在来看看应用程序如何调用ContentProvider。调用ContentProvider的关键是使用getContentResolver方法来获得一个ContentResolver对象，并通过ContentResolver对象的query方法来 访问ContentProvider。

首先来定义两个访问ContentProvider的常量。

public final String DICTIONARY\_SINGLE\_WORD\_URI = "content://net.blogjava.mobile.dictionarycontentprovider/single" ;

public final String DICTIONARY\_PREFIX\_WORD\_URI = "content://net.blogjava.mobile.dictionarycontentprovider/prefix" ;

[java] view plaincopy

public final String DICTIONARY\_SINGLE\_WORD\_URI = "content://net.blogjava.mobile.dictionarycontentprovider/single";

public final String DICTIONARY\_PREFIX\_WORD\_URI = "content://net.blogjava.mobile.dictionarycontentprovider/prefix";

然后在查询按钮的单击事件中编写如下的代码来查询单词。

public void onClick(View view)

{

Uri uri = Uri.parse(DICTIONARY\_SINGLE\_WORD\_URI);

// 通过ContentProvider查询单词，并返回Cursor对象，然后的操作就和直接从数据中获得

// Cursor对象后的操作是一样的了

Cursor cursor = getContentResolver().query(uri, null , "english=?" ,

new String[]{ actvWord.getText().toString() }, null );

String result = "未找到该单词." ;

if (cursor.getCount() > 0 )

{

cursor.moveToFirst();

result = cursor.getString(cursor.getColumnIndex("chinese" ));

}

new AlertDialog.Builder(this ).setTitle("查询结果" ).setMessage(result)

.setPositiveButton("关闭" , null ).show();

}

[java] view plaincopy

public void onClick(View view)

{

Uri uri = Uri.parse(DICTIONARY\_SINGLE\_WORD\_URI);

// 通过ContentProvider查询单词，并返回Cursor对象，然后的操作就和直接从数据中获得

// Cursor对象后的操作是一样的了

Cursor cursor = getContentResolver().query(uri, null, "english=?",

new String[]{ actvWord.getText().toString() }, null);

String result = "未找到该单词.";

if (cursor.getCount() > 0)

{

cursor.moveToFirst();

result = cursor.getString(cursor.getColumnIndex("chinese"));

}

new AlertDialog.Builder(this).setTitle("查询结果").setMessage(result)

.setPositiveButton("关闭", null).show();

}

下面是显示单词列表的代码。

public void afterTextChanged(Editable s)

{

if ("" .equals(s.toString()))

return ;

Uri uri = Uri.parse(DICTIONARY\_PREFIX\_WORD\_URI + "/" + s.toString());

// 从ContentProvider中获得以某个字符串开头的所有单词的Cursor对象

Cursor cursor = getContentResolver().query(uri, null , null , null , null );

DictionaryAdapter dictionaryAdapter = new DictionaryAdapter(this ,

cursor, true );

actvWord.setAdapter(dictionaryAdapter);

}

[java] view plaincopy

public void afterTextChanged(Editable s)

{

if ("".equals(s.toString()))

return;

Uri uri = Uri.parse(DICTIONARY\_PREFIX\_WORD\_URI + "/" + s.toString());

// 从ContentProvider中获得以某个字符串开头的所有单词的Cursor对象

Cursor cursor = getContentResolver().query(uri, null, null, null, null);

DictionaryAdapter dictionaryAdapter = new DictionaryAdapter(this,

cursor, true);

actvWord.setAdapter(dictionaryAdapter);

}

现在来运行本例，会看到如图6所示的界面。当查询单词时会显示如图7所示的单词列表，查询出结果后，会显示如图8所示的界面。

## 方式三：广播（Broadcast）

广播是一种被动跨进程通讯的方式。当某个程序向系统发送广播时，其他的应用程序只能被动地接收广播数据。这就象电台进行广播一样，听众只能被动地收听，而不能主动与电台进行沟通。

在应用程序中发送广播比较简单。只需要调用sendBroadcast方法即可。该方法需要一个Intent对象。通过Intent对象可以发送需要广播的数据。

先建一个android工程：sendbroadcast。在XML布局文件中放两个组件：EditText和Button，当单击按钮后，会弹出显示EditText组件中文本的对话框，关闭对话框后， 会使用sendBroadcast方法发送消息，并将EditText组件的文本通过Intent对象发送出去。完整的代码如下：

package net.blogjava.mobile.sendbroadcast;

... ...

public class Main extends Activity implements OnClickListener

{

private EditText editText;

@Override

public void onClick(View view)

{

new AlertDialog.Builder(this ).setMessage(editText.getText().toString())

.setPositiveButton("确定" , null ).show();

// 通过Intent类的构造方法指定广播的ID

Intent intent = new Intent("net.blogjava.mobile.MYBROADCAST" );

// 将要广播的数据添加到Intent对象中

intent.putExtra("text" , editText.getText().toString());

// 发送广播

sendBroadcast(intent);

}

... ...

}

[java] view plaincopy

package net.blogjava.mobile.sendbroadcast;

... ...

public class Main extends Activity implements OnClickListener

{

private EditText editText;

@Override

public void onClick(View view)

{

new AlertDialog.Builder(this).setMessage(editText.getText().toString())

.setPositiveButton("确定", null).show();

// 通过Intent类的构造方法指定广播的ID

Intent intent = new Intent("net.blogjava.mobile.MYBROADCAST");

// 将要广播的数据添加到Intent对象中

intent.putExtra("text", editText.getText().toString());

// 发送广播

sendBroadcast(intent);

}

... ...

}

发送广播并不需要在AndroidManifest.xml文件中注册，但接收广播必须在AndroidManifest.xml文件中注册receiver。下面来编写一个接收广播的应用程序。首先建立一个android工程：receiver。然后编写一个MyReceiver类，该类是BroadcastReceiver的子类，代码如下：

package net.blogjava.mobile.receiver;

... ...

public class MyReceiver extends BroadcastReceiver

{

// 当sendbroadcast发送广播时，系统会调用onReceive方法来接收广播

@Override

public void onReceive(Context context, Intent intent)

{

// 判断是否为sendbroadcast发送的广播

if ("net.blogjava.mobile.MYBROADCAST" .equals(intent.getAction()))

{

Bundle bundle = intent.getExtras();

if (bundle != null )

{

String text = bundle.getString("text" );

Toast.makeText(context, "成功接收广播：" + text, Toast.LENGTH\_LONG).show();

}

}

}

}

[java] view plaincopy

package net.blogjava.mobile.receiver;

... ...

public class MyReceiver extends BroadcastReceiver

{

// 当sendbroadcast发送广播时，系统会调用onReceive方法来接收广播

@Override

public void onReceive(Context context, Intent intent)

{

// 判断是否为sendbroadcast发送的广播

if ("net.blogjava.mobile.MYBROADCAST".equals(intent.getAction()))

{

Bundle bundle = intent.getExtras();

if (bundle != null)

{

String text = bundle.getString("text");

Toast.makeText(context, "成功接收广播：" + text, Toast.LENGTH\_LONG).show();

}

}

}

}

当应用程序发送广播时，系统会调用onReceive方法来接收广播，并通过intent.getAction()方法返回广播的ID，也就是在发送广播时Intent构造方法指定的字符串。然后就可以从Bundle对象中获得相应的数据了。

最后还需要在AndroidManifest.xml文件中注册receiver，代码如下：

<!-- 注册receiver 

<receiver android:name="MyReceiver" >

<intent-filter>

<action android:name="net.blogjava.mobile.MYBROADCAST" />

</intent-filter>

</receiver>

[java] view plaincopy

<!-- 注册receiver 

<receiver android:name="MyReceiver">

<intent-filter>

<action android:name="net.blogjava.mobile.MYBROADCAST" />

</intent-filter>

</receiver>

在注册MyReceiver类时需要使用<receiver>标签，android:name属性指定MyReceiver类，<action>标签的android:name指定了广播的ID。

首先运行receiver程序，然后就可以关闭receiver程序了。接收广播并不依赖于程序的状态。就算程序关闭了，仍然可以接收广播。然后再启动sendbroadcast程序。并在文本框中输入“android”，然后单击按钮，会弹出一个显示文本框内容的对话框，如图9所示。当关闭对话框后，会 显示一个Toast信息提示框，这个信息框是由receiver程序弹出的。如图10所示。

方式四：AIDL服务

服务（Service）是android系统中非常重要的组件。Service可以脱离应用程序运行。也就是说，应用程序只起到一个启动Service的作用。一但Service被启动，就算应用程序关闭，Service仍然会在后台运行。

android系统中的Service主要有两个作用：后台运行和跨进程通讯。后台运行就不用说了，当Service启动后，就可以在Service对象中 运行相应的业务代码，而这一切用户并不会察觉。而跨进程通讯是这一节的主题。如果想让应用程序可以跨进程通讯，就要使用我们这节讲的AIDL服 务，AIDL的全称是Android Interface Definition Language，也就是说，AIDL实际上是一种接口定义语言。通过这种语言定义接口后，Eclipse插件（ODT）会自动生成相应的Java代码接 口代码。下面来看一下编写一个AIDL服务的基本步骤。

1. 在Eclipse工程的package目录中建立一个扩展名为aidl的文件。package目录就是Java类所在的目录。该文件的语法类似于Java代码。aidl文件中定义的是AIDL服务的接口。这个接口需要在调用AIDL服务的程序中访问。

2. 如果aidl文件的内容是正确的，Eclipse插件会自动生成一个Java接口文件（\*.java）。

3. 建立一个服务类（Service的子类）。

4. 实现由aidl文件生成的Java接口。

5. 在AndroidManifest.xml文件中配置AIDL服务，尤其要注意的是，<action>标签的android:name属性值就是客户端要引用该服务的ID，也就是Intent类构造方法的参数值。

现在我们来编写一个AIDL服务，首先建立一个android工程：aidlservice。在aidlservice工程中有一个Main类，在Main类所有的目录建立一个IMyService.aidl文件，内容如下：

[java] view plaincopy

package net.blogjava.mobile.aidlservice;

interface IMyService

{

String getValue(); // 为AIDL服务的接口方法，调用AIDL服务的程序需要调用该方法

}

在保存IMyService.aidl文件后，ODT会在gen目录下产生一个IMyService.java文件，读者可以不必管这个文件中的内容，也 不需要修改该文件的内容。这个文件是由ODT自动维护的，只要修改了IMyService.aidl文件的内容，IMyService.java文件的内 容就会随之改变。

然后建立一个MyService类，该类是Service的子类，代码如下：

package net.blogjava.mobile.aidlservice;

public class MyService extends Service

{

// IMyService.Stub类是根据IMyService.aidl文件生成的类，该类中包含了接口方法（getValue）

public class MyServiceImpl extends IMyService.Stub

{

@Override

public String getValue() throws RemoteException

{

return "从AIDL服务获得的值.";

}

}

@Override

public IBinder onBind(Intent intent)

{

// 该方法必须返回MyServiceImpl类的对象实例

return new MyServiceImpl();

}

}

最后需要在AndroidManifest.xml文件中配置MyService类，代码如下：

[java] view plaincopy

<!-- 注册服务-->

<service android:name=".MyService">

<intent-filter>

<!-- 指定调用AIDL服务的ID -->

<action android:name="net.blogjava.mobile.aidlservice.IMyService" />

</intent-filter>

</service>

下面来看看如何调用这个AIDL服务。首先建立一个android工程：aidlclient。然后将aidlservice工程中自动生成的IMyService.java文件复制到aidlclient工程中。在调用AIDL服务之前需要先使用bindService方法绑定AIDL服务。bindService方法需要一个ServiceConnection对象。ServiceConnection有一个onServiceConnected方法，当成功绑定AIDL服务且，该方法被调用。并通过service参数返回AIDL服务对象。下面是调用AIDL服务的完成代码。

[java] view plaincopy

package net.blogjava.mobile.aidlclient;

public class Main extends Activity implements OnClickListener

{

private IMyService myService = null;

// 创建ServiceConnection对象

private ServiceConnection serviceConnection = new ServiceConnection()

{

@Override

public void onServiceConnected(ComponentName name, IBinder service)

{

// 获得AIDL服务对象

myService = IMyService.Stub.asInterface(service);

try

{

// 调用AIDL服务对象中的getValue方法，并以对话框中显示该方法的返回值

new AlertDialog.Builder(Main.this).setMessage(

myService.getValue()).setPositiveButton("确定", null)

.show();

}

catch (Exception e)

{

}

}

@Override

public void onServiceDisconnected(ComponentName name)

{

}

};

@Override

public void onClick(View view)

{

// 绑定AIDL服务

bindService(new Intent("net.blogjava.mobile.aidlservice.IMyService"),

serviceConnection, Context.BIND\_AUTO\_CREATE);

}

... ...

}

在编写AIDL服务和客户端时要注意如下两点：

1. AIDL服务中的onBind方法必须返回AIDL接口对象（MyServiceImpl对象）。该对象也是onServiceConnected事件方法的第2个参数值。

2. bindService方法的第1个参数是Intent对象，该对象构造方法的参数需要指定AIDL服务的ID，也就是在AndroidManifest.xml文件中<service>标签的<action>子标签的android:name属性 的值。

现在先运行aidlservice程序，以便安装AIDL服务，然后运行aidlclient程序，并单击按钮，会显示如图11所示的对话框。对话框中的信息就是AIDL服务接口中getValue方法的返回值。

总结

本文介绍了4种跨进程通讯的方式：Activity、ContentProvider、Broadcast和AIDL Service。其中Activity可以跨进程调用其他应用程序的Activity；ContentProvider可以访问其他应用程序返回的Cursor对象；Broadcast采用的是被动接收的方法，也就是说，客户端只能接收广播数据，而不能向发送广播的程序发送信息。AIDL Service可以将程序中的某个接口公开，这样在其他的应用程序中就可以象访问本地对象一样访问AIDL服务对象了。这4种跨进程通讯的方式可以应用在 不同的场合，例如，在需要显示可视化的界面时可以用Activity，需要返回记录集时可以用ContentProvider。至于在应用程序中具体要用 到哪一种或几种方式进行跨进程通讯，读者可以根据实际情况进行选择。