Android中Serializable和Parcelable序列化对象详解

作者：Darker 字体：[增加 减小] 类型：转载 时间：2016-02-24 我要评论

这篇文章主要介绍了Android中Serializable和Parcelable序列化对象的相关资料，感兴趣的小伙伴们可以参考一下

本文详细对Android中Serializable和Parcelable序列化对象进行学习，具体内容如下

学习内容：

1.序列化的目的

2.Android中序列化的两种方式

3.Parcelable与Serializable的性能比较

4.Android中如何使用Parcelable进行序列化操作

5.Parcelable的工作原理

6.相关实例

1.序列化的目的

1）.永久的保存对象数据(将对象数据保存在文件当中,或者是磁盘中

2）.通过序列化操作将对象数据在网络上进行传输(由于网络传输是以字节流的方式对数据进行传输的.因此序列化的目的是将对象数据转换成字节流的形式)

3）.将对象数据在进程之间进行传递(Activity之间传递对象数据时,需要在当前的Activity中对对象数据进行序列化操作.在另一个Activity中需要进行反序列化操作讲数据取出)

4）.Java平台允许我们在内存中创建可复用的Java对象，但一般情况下，只有当JVM处于运行时，这些对象才可能存在，即，这些对象的生命周期不会比JVM的生命周期更长（即每个对象都在JVM中）但在现实应用中，就可能要停止JVM运行，但有要保存某些指定的对象，并在将来重新读取被保存的对象。这是Java对象序列化就能够实现该功能。（可选择入数据库、或文件的形式保存）

5）.序列化对象的时候只是针对变量进行序列化,不针对方法进行序列化.

6）.在Intent之间,基本的数据类型直接进行相关传递即可,但是一旦数据类型比较复杂的时候,就需要进行序列化操作了.

2.Android中实现序列化的两种方式

1）.Implements Serializable 接口 **(**声明一下即可**)**

Serializable 的简单实例：

```
public class Person implements Serializable{
 private static final long serialVersionUID = -7060210544600464481L;
 private String name;
 private int age;

 public String getName(){
  return name;
 }

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

 public int getAge(){
  return age;
 }
```

大家感兴趣的内容

1 一看就懂的Android APP开发入门教
2 微信公众平台开发入门教程(图文详
3 Android基础之使用Fragment控制切
4 六款值得推荐的android（安卓）开
5 Android应用开发SharedPreferenc
6 android TextView设置中文字体加
7 Android 动画之TranslateAnimati
8 Android Bitmap详细介绍
9 android PopupWindow 和 Activit
10 android压力测试命令monkey详解

最近更新的内容

Android之AnimationDrawable简单模拟
android中开启actionbar的两种方法
android 获取屏幕尺寸
Android TextView多文本折叠展开效果
Android 区别真机和模拟器的几种方法
Android中layout属性大全
Android自定义实现开关按钮代码
Android NDK中socket的用法以及注意事
Android自定义View实现水面上涨效果
Android开发中关于获取当前Activity的一

常用在线小工具

```
17
18    public void setAge(int age){
19     this.age = age;
20    }
21   }
```

2）.Implements Parcelable 接口(不仅仅需要声明,还需要实现内部的相应方法)

Parcelable的简单实例:

注：写入数据的顺序和读出数据的顺序必须是相同的.

```
1  public class Book implements Parcelable{
2   private String bookName;
3   private String author;
4   private int publishDate;
5
6   public Book(){
7
8   }
9
10  public String getBookName(){
11   return bookName;
12  }
13
14  public void setBookName(String bookName){
15   this.bookName = bookName;
16  }
17
18  public String getAuthor(){
19   return author;
20  }
21
22  public void setAuthor(String author){
23   this.author = author;
24  }
25
26  public int getPublishDate(){
27   return publishDate;
28  }
29
30  public void setPublishDate(int publishDate){
31   this.publishDate = publishDate;
32  }
33
34  @Override
35  public int describeContents(){
36   return 0;
37  }
38
39  @Override
40  public void writeToParcel(Parcel out, int flags){
41   out.writeString(bookName);
42   out.writeString(author);
43   out.writeInt(publishDate);
44  }
45
46  public static final Parcelable.Creator<Book> CREATOR = new Creator<Book>(){
47
48        @Override
49   public Book[] newArray(int size){
50    return new Book[size];
51   }
52
53   @Override
54   public Book createFromParcel(Parcel in){
55    return new Book(in);
56   }
57  };
58
59  public Book(Parcel in){
60   //如果元素数据是list类型的时候需要:  lits = new ArrayList<?> in.readList(list
61   bookName = in.readString();
62   author = in.readString();
63   publishDate = in.readInt();
64  }
65 }
```

我们知道在Java应用程序当中对类进行序列化操作只需要实现Serializable接口就可以,由系统来完成序列化和反序列化操作,但是在Android中序列化操作有另外一种方式来完成,那就是实现Parcelable接口.也是Android中特有的接口来实现类的序列化操作.原因是Parcelable的性能要强于Serializable.因此在绝大多数的情况下,Android还是推荐使用Parcelable来完成对类的序列化操作的.

3.Parcelable与Serializable的性能比较

首先Parcelable的性能要强于Serializable的原因我需要简单的阐述一下

1）.在内存的使用中,前者在性能方面要强于后者

2）. 后者在序列化操作的时候会产生大量的临时变量,(原因是使用了反射机制)从而导致GC的频繁调用,因此在性能上会稍微逊色

3）. Parcelable是以Ibinder作为信息载体的.在内存上的开销比较小,因此在内存之间进行数据传递的时候,Android推荐使用Parcelable,既然是内存方面比价有优势,那么自然就要优先选择.

4）. 在读写数据的时候,Parcelable是在内存中直接进行读写,而Serializable是通过使用IO流的形式将数据读写入在硬盘上.

但是：虽然Parcelable的性能要强于Serializable,但是仍然有特殊的情况需要使用Serializable,而不去使用Parcelable,因为Parcelable无法将数据进行持久化,因此在将数据保存在磁盘的时候,仍然需要使用后者,因为前者无法很好的将数据进行持久化.(原因是在不同的Android版本当中,Parcelable可能会不同,因此数据的持久化方面仍然是使用Serializable)

速度测试：

测试方法：

1）、通过将一个对象放到一个bundle里面然后调用Bundle#writeToParcel(Parcel, int)方法来模拟传递对象给一个activity的过程，然后再把这个对象取出来。

2）、在一个循环里面运行1000 次。

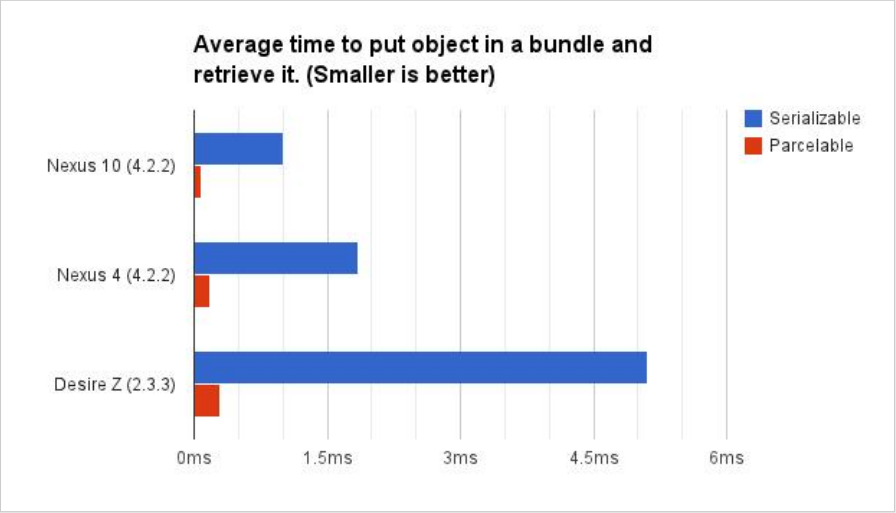
3）、两种方法分别运行10次来减少内存整理，cpu被其他应用占用等情况的干扰。

4）、参与测试的对象就是上面的相关代码

5）、在多种Android软硬件环境上进行测试

1. **LG Nexus 4 – Android 4.2.2**
2. **Samsung Nexus 10 – Android 4.2.2**
3. **HTC Desire Z – Android 2.3.3**

结果如图：



性能差异：

Nexus 10

Serializable: 1.0004ms, Parcelable: 0.0850ms – 提升10.16倍。

Nexus 4

Serializable: 1.8539ms – Parcelable: 0.1824ms – 提升11.80倍。

Desire Z

Serializable: 5.1224ms – Parcelable: 0.2938ms – 提升17.36倍。

由此可以得出: Parcelable 比 Serializable快了10多倍。

从相对的比较我们可以看出,Parcelable的性能要比Serializable要优秀的多,因此在Android中进行序列化操作的时候,我们需要尽可能的选择前者,需要花上大量的时间去实现Parcelable接口中的内部方法.

4.Android中如何使用**Parcelable**进行序列化操作

说了这么多,我们还是来看看Android中如何去使用Parcelable实现类的序列化操作吧.

Implements Parcelable的时候需要实现内部的方法:

1）.writeToParcel 将对象数据序列化成一个Parcel对象(序列化之后成为Parcel对象.以便Parcel容器取出数据)

2）.重写describeContents方法,默认值为0

3）.Public static final Parcelable.Creator<T>CREATOR (将Parcel容器中的数据转换成对象数据) 同时需要实现两个方法:

3.1 CreateFromParcel(从Parcel容器中取出数据并进行转换.)

3.2 newArray(int size)返回对象数据的大小

因此，很明显实现Parcelable并不容易。实现Parcelable接口需要写大量的模板代码，这使得对象代码变得难以阅读和维护。具体的实例就是上面Parcelable的实例代码.就不进行列举了.(有兴趣的可以去看看Android中NetWorkInfo的源代码,是关于网络连接额外信息的一个相关类,内部就实现了序列化操作.大家可以去看看)

5.Parcelable的工作原理

无论是对数据的读还是写都需要使用Parcel作为中间层将数据进行传递.Parcel涉及到的东西就是与C++底层有关了.都是使用JNI.在Java应用层是先创建Parcel(Java)对象,然后再调用相关的读写操作的时候.就拿读写32为Int数据来说吧:

```
static jint android_os_Parcel_readInt(JNIEnv* env, jobject clazz){
 Parcel* parcel = parcelForJavaObject(env, clazz);
 if (parcel != NULL) {
  return parcel->readInt32();
 }
 return 0;
}
```

调用的方法就是这个过程,首先是将Parcel(Java)对象转换成Parcel(C++)对象,然后被封装在Parcel中的相关数据由C++底层来完成数据的序列化操作.

```
status_t Parcel::writeInt32(int32_t val){
 return writeAligned(val);
}
template<class t="">
status_t Parcel::writeAligned(T val) {
 COMPILE_TIME_ASSERT_FUNCTION_SCOPE(PAD_SIZE(sizeof(T)) == sizeof(T));

 if ((mDataPos+sizeof(val)) <= mDataCapacity) {
restart_write:
  *reinterpret_cast<t*>(mData+mDataPos) = val;
  return finishWrite(sizeof(val));
 }

 status_t err = growData(sizeof(val));
 if (err == NO_ERROR) goto restart_write;
 return err;
}
真正的读写过程是由下面的源代码来完成的.
status_t Parcel::continueWrite(size_t desired)
{
 // If shrinking, first adjust for any objects that appear
 // after the new data size.
 size_t objectsSize = mObjectsSize;
 if (desired < mDataSize) {
  if (desired == 0) {
   objectsSize = 0;
  } else {
   while (objectsSize > 0) {
    if (mObjects[objectsSize-1] < desired)
     break;
    objectsSize--;
   }
  }
 }

 if (mOwner) {
  // If the size is going to zero, just release the owner's data.
  if (desired == 0) {
   freeData();
   return NO_ERROR;
  }

  // If there is a different owner, we need to take
  // posession.
  uint8_t* data = (uint8_t*)malloc(desired);
  if (!data) {
   mError = NO_MEMORY;
   return NO_MEMORY;
  }
  size_t* objects = NULL;

  if (objectsSize) {
   objects = (size_t*)malloc(objectsSize*sizeof(size_t));
```

```
    if (!objects) {
     mError = NO_MEMORY;
     return NO_MEMORY;
    }

    // Little hack to only acquire references on objects
    // we will be keeping.
    size_t oldObjectsSize = mObjectsSize;
    mObjectsSize = objectsSize;
    acquireObjects();
    mObjectsSize = oldObjectsSize;
   }

   if (mData) {
    memcpy(data, mData, mDataSize < desired ? mDataSize : desired);
   }
   if (objects && mObjects) {
    memcpy(objects, mObjects, objectsSize*sizeof(size_t));
   }
   //ALOGI("Freeing data ref of %p (pid=%d)\n", this, getpid());
   mOwner(this, mData, mDataSize, mObjects, mObjectsSize, mOwnerCookie);
   mOwner = NULL;

   mData = data;
   mObjects = objects;
   mDataSize = (mDataSize < desired) ? mDataSize : desired;
   ALOGV("continueWrite Setting data size of %p to %d\n", this, mDataSize);
   mDataCapacity = desired;
   mObjectsSize = mObjectsCapacity = objectsSize;
   mNextObjectHint = 0;

  } else if (mData) {
   if (objectsSize < mObjectsSize) {
    // Need to release refs on any objects we are dropping.
    const sp<ProcessState> proc(ProcessState::self());
    for (size_t i=objectsSize; i<mObjectsSize; i++) {
     const flat_binder_object* flat
      = reinterpret_cast<flat_binder_object*>(mData+mObjects[i]);
     if (flat->type == BINDER_TYPE_FD) {
      // will need to rescan because we may have lopped off the only FDs
      mFdsKnown = false;
     }
     release_object(proc, *flat, this);
    }
    size_t* objects =
     (size_t*)realloc(mObjects, objectsSize*sizeof(size_t));
    if (objects) {
     mObjects = objects;
    }
    mObjectsSize = objectsSize;
    mNextObjectHint = 0;
   }

   // We own the data, so we can just do a realloc().
   if (desired > mDataCapacity) {
    uint8_t* data = (uint8_t*)realloc(mData, desired);
    if (data) {
     mData = data;
     mDataCapacity = desired;
    } else if (desired > mDataCapacity) {
     mError = NO_MEMORY;
     return NO_MEMORY;
    }
   } else {
    if (mDataSize > desired) {
     mDataSize = desired;
     ALOGV("continueWrite Setting data size of %p to %d\n", this, mDataSize);
    }
    if (mDataPos > desired) {
     mDataPos = desired;
     ALOGV("continueWrite Setting data pos of %p to %d\n", this, mDataPos);
    }
   }

  } else {
   // This is the first data. Easy!
   uint8_t* data = (uint8_t*)malloc(desired);
   if (!data) {
    mError = NO_MEMORY;
    return NO_MEMORY;
   }

   if(!(mDataCapacity == 0 && mObjects == NULL
     && mObjectsCapacity == 0)) {
    ALOGE("continueWrite: %d/%p/%d/%d", mDataCapacity, mObjects, mObjectsCapa
   }

   mData = data;
   mDataSize = mDataPos = 0;
   ALOGV("continueWrite Setting data size of %p to %d\n", this, mDataSize);
   ALOGV("continueWrite Setting data pos of %p to %d\n", this, mDataPos);
   mDataCapacity = desired;
  }

  return NO_ERROR;
 }
```

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。

6.相关实例

最后上一个例子..

首先是序列化的类Book.class

```
public class Book implements Parcelable{
 private String bookName;
 private String author;
 private int publishDate;

 public Book(){

 }

 public String getBookName(){
  return bookName;
 }

 public void setBookName(String bookName){
  this.bookName = bookName;
 }

 public String getAuthor(){
  return author;
 }

 public void setAuthor(String author){
  this.author = author;
 }

 public int getPublishDate(){
  return publishDate;
 }

 public void setPublishDate(int publishDate){
  this.publishDate = publishDate;
 }

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

 @Override
 public void writeToParcel(Parcel out, int flags){
  out.writeString(bookName);
  out.writeString(author);
  out.writeInt(publishDate);
 }

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

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

  @Override
  public Book createFromParcel(Parcel in){
   return new Book(in);
  }
 };

 public Book(Parcel in){
  //如果元素数据是list类型的时候需要:  lits = new ArrayList<?> in.readList(list
  bookName = in.readString();
  author = in.readString();
  publishDate = in.readInt();
 }
}
```

第一个**Activity,MainActivity**

```
Book book = new Book();
book.setBookname("Darker");
book.setBookauthor("me");
book.setPublishDate(20);
```

```
Bundle bundle = new Bundle();
bundle.putParcelable("book", book);
Intent intent = new Intent(MainActivity.this,AnotherActivity.class);
intent.putExtras(bundle);
```

第二个**Activity,AnotherActivity**

```
Intent intent = getIntent();
Bundle bun = intent.getExtras();
Book book = bun.getParcelable("book");
System.out.println(book);
```

总结：Java应用程序中有Serializable来实现序列化操作,Android中有Parcelable来实现序列化操作,相关的性能也作出了比较,因此在Android中除了对数据持久化的时候需要使用到Serializable来实现序列化操作,其他的时候我们仍然需要使用Parcelable来实现序列化操作,因为在Android中效率并不是最重要的,而是内存,通过比较Parcelable在效率和内存上都要优秀与Serializable,尽管Parcelable实现起来比较复杂,但是如果我们想要成为一名优秀的Android软件工程师,那么我们就需要勤快一些去实现Parcelable,而不是偷懒与实现Serializable.当然实现后者也不是不行,关键在于我们头脑中的那一份思想。

以上就是本文的全部内容，希望对大家的学习有所帮助。

您可能感兴趣的文章:

Android应用中使用XmlSerializer序列化XML数据的教程

解析Android中的Serializable序列化

Android中的Parcelable序列化对象

Android中的序列化浅析

Android xml文件的序列化实现代码

Android序列化XML数据

Android中Intent传递对象的两种方法Serializable,Parcelable

Android中使用Intent在Activity之间传递对象(使用Serializable或者Parcelable)的方法

Android中Parcelable的作用实例解析

很详细的android序列化过程Parcelable

Tags: Android Serializable Parcelable 序列化

相关文章

Handler与Android多线程详解	2013-10-10
Android应用中炫酷的横向和环形进度条的实例分享	2016-04-04
Android用Fragment创建选项卡	2016-10-10
Android BroadcastReceiver广播机制概述	2016-08-08
Android UI效果之绘图篇（四）	2016-02-02
Android Studio使用小技巧：自定义Logcat	2015-05-05
Android之PreferenceActivity应用详解	2012-11-11
Android非XML形式动态生成、调用页面的方法	2015-04-04
Android开发艺术探索学习笔记（七）	2016-01-01
Android提高之ListView实现自适应表格的方法	2014-08-08

最新评论

还没有评论，快来抢沙发吧！

Powered by 畅言