|  |
| --- |
| 1、序列化与反序列化  内存中的对象 <---> 二进制数据  2、在 Java 中使用对象流来完成序列化和反序列化。  ObjectOutputStream:通过 writeObject()方法做序列化操作  ObjectInputStream:通过 readObject() 方法做反序列化操作  3、为什么要序列化？  （持久化、网络传递、进程间传递）  1）永久性保存对象，保存对象的字节序列到本地文件中；  2）通过序列化对象在网络中传递对象；  3）通过序列化在进程间传递对象。  4）服务器钝化。  4、实现序列化的方法  （Parcelable接口复杂，但是效率高。）  Android中实现序列化有两个选择：  1）实现Serializable接口（是JavaSE本身就支持的），使用简单。  Serializable是由Java提供的序列化接口，它是一个空接口。  这种序列化是通过反射机制从而削弱了性能，这种机制也创建了大量的临时对象从而引起GC频繁回收调用资源。  2）实现Parcelable接口，是Android特有功能，实现稍微复杂，但是像你高。  运用真实的序列化处理代替反射，大量的引入代码但是速度会远快于Serializable。所以优先选择Parcelable。  5、选择序列化方法的原则  1）在使用内存的时候，Parcelable比Serializable性能高，所以推荐使用Parcelable。  2）Serializable在序列化的时候会产生大量的临时变量，从而引起频繁的GC。  3）Parcelable不能使用在要将数据存储在磁盘上的情况，尽管Serializable效率低点，但此时还是建议使用Serializable 。 |

# 标题：Android中Parcelable接口用法

# 1、Parcelable接口

Interface for classes whose instances can be written to and restored from a Parcel。 Classes implementing the Parcelable interface must also have a static field called CREATOR， which is an object implementing the Parcelable.Creator interface。

# 2、为什么要序列化？

（持久化、网络传递、进程间传递）

实现Parcelable就是为了进行序列化，那么，为什么要序列化？

1）永久性保存对象，保存对象的字节序列到本地文件中；

2）通过序列化对象在网络中传递对象；

3）通过序列化在进程间传递对象。

# 3、实现序列化的方法

（Parcelable接口复杂，但是效率高。）

Android中实现序列化有两个选择：一是实现Serializable接口（是JavaSE本身就支持的），一是实现Parcelable接口（是Android特有功能，效率比实现Serializable接口高效，可用于Intent数据传递，也可以用于进程间通信（IPC））。实现Serializable接口非常简单，声明一下就可以了，而实现Parcelable接口稍微复杂一些，但效率更高，推荐用这种方法提高性能。

注：Android中Intent传递对象有两种方法：一是Bundle.putSerializable(Key，Object)，另一种是Bundle.putParcelable(Key，Object)。当然这些Object是有一定的条件的，前者是实现了Serializable接口，而后者是实现了Parcelable接口。

# 4、选择序列化方法的原则

1）在使用内存的时候，Parcelable比Serializable性能高，所以推荐使用Parcelable。

2）Serializable在序列化的时候会产生大量的临时变量，从而引起频繁的GC。

3）Parcelable不能使用在要将数据存储在磁盘上的情况，因为Parcelable不能很好的保证数据的持续性在外界有变化的情况下。尽管Serializable效率低点，但此时还是建议使用Serializable 。

# 5、应用场景

（Intent意图传递复制对象。）

需要在多个部件(Activity或Service)之间通过Intent传递一些数据，简单类型（如：数字、字符串）的可以直接放入Intent。复杂类型必须实现Parcelable接口。

# 6、Parcelable接口定义

public interface Parcelable

{

//内容描述接口，基本不用管

public int describeContents();

//写入接口函数，打包

public void writeToParcel(Parcel dest, int flags);

//读取接口，目的是要从Parcel中构造一个实现了Parcelable的类的实例处理。因为实现类在这里还是不可知的，所以需要用到模板的方式，继承类名通过模板参数传入

//为了能够实现模板参数的传入，这里定义Creator嵌入接口,内含两个接口函数分别返回单个和多个继承类实例

public interface Creator<T>

{

public T createFromParcel(Parcel source);

public T[] newArray(int size);

}

}

# 7、实现Parcelable步骤

（ private static final long serialVersionUID

内容接口描述、写入接口函数、实例化静态内部对象）

1）implements Parcelable。

2）重写writeToParcel方法，将你的对象序列化为一个Parcel对象，即：将类的数据写入外部提供的Parcel中，打包需要传递的数据到Parcel容器保存，以便从 Parcel容器获取数据。

3）重写describeContents方法，内容接口描述，默认返回0就可以。

4）实例化静态内部对象CREATOR实现接口Parcelable.Creator

public static final Parcelable.Creator<T> CREATOR

注：其中public static final一个都不能少，内部对象CREATOR的名称也不能改变，必须全部大写。需重写本接口中的两个方法：createFromParcel(Parcel in) 实现从Parcel容器中读取传递数据值，封装成Parcelable对象返回逻辑层，newArray(int size) 创建一个类型为T，长度为size的数组，仅一句话即可（return new T[size]），供外部类反序列化本类数组使用。

简而言之：通过writeToParcel将你的对象映射成Parcel对象，再通过createFromParcel将Parcel对象映射成你的对象。也可以将Parcel看成是一个流，通过writeToParcel把对象写到流里面，在通过createFromParcel从流里读取对象，只不过这个过程需要你来实现，因此写的顺序和读的顺序必须一致。

代码如下：

public class MyParcelable implements Parcelable

{

private int mData;

public int describeContents()

{

return 0;

}

public void writeToParcel(Parcel out, int flags)

{

out.writeInt(mData);

}

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

{

public MyParcelable createFromParcel(Parcel in)

{

return new MyParcelable(in);

}

public MyParcelable[] newArray(int size)

{

return new MyParcelable[size];

}

};

private MyParcelable(Parcel in)

{

mData = in.readInt();

}

}

# 8、Serializable实现与Parcelabel实现的区别

1）Serializable的实现，只需要implements Serializable 即可。这只是给对象打了一个标记，系统会自动将其序列化。

2）Parcelabel的实现，不仅需要implements Parcelabel，还需要在类中添加一个静态成员变量CREATOR，这个变量需要实现 Parcelable.Creator 接口。

两者代码比较：

1）创建Person类，实现Serializable

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;

}

public void setAge(int age)

{

this.age = age;

}

}

2）创建Book类，实现Parcelable

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)

{

bookName = in.readString();

author = in.readString();

publishDate = in.readInt();

}

}

# 总结Serializable 和 Parcelable

Serializable和Parcelable是两个序列化接口，如果使用Bundle在Intent之间传递对象需要先进行序列化。

•序列化的目的

1、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)

2、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)

3、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）

4、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。

Serializable

Serializable是由Java提供的序列化接口，它是一个空接口。

这种序列化是通过反射机制从而削弱了性能，这种机制也创建了大量的临时对象从而引起GC频繁回收调用资源。

（想要使用反射机制，就必须要先获取到该类的字节码文件对象(.class)，通过字节码文件对象，就能够通过该类中的方法获取到我们想要的所有信息(方法，属性，类名，父类名，实现的所有接口等等)，每一个类对应着一个字节码文件也就对应着一个Class类型的对象，也就是字节码文件对象。）

Parcelable

Parcelable是由Android提供的序列化接口，google做了大量的优化。运用真实的序列化处理代替反射，大量的引入代码但是速度会远快于Serializable。所以优先选择Parcelable。