Parcelable接口的使用

吃葡萄皮不吐葡萄 关注

0.223 2016.06.14 11:32:29 字数 672 阅读 6,770

为什么要是使用Parcelable接口?

有些时候，我们需要在两个Activity之间，Activity和Service之间等等的情况下传递一些信息，如果是基本数据类型的话，我们可以通过Intent，使用下面方式进行传递

```
//在第一个Activity中向Intent中加入内容
String data = "Hello,SecondActivity.class");
Intent intent = new Intent(FirstActivity.this,SecondActivity.class);
intent.putExtra("alarm",data);
startActivity(intent);

//在第二个Activity中，取出Intent中的内容
Intent intent = getIntent();
String data = intent.getStringExtra("alarm");
```

如果要传递的数据是基本数据类型，这个方法是没有问题的。假如我们要传递的不是基本数据类型，是一个Java中的一个类的对象（比如：Calendar对象），或者是一个我们自定义的对象，那应该怎么办?
有两个解决方案

- 第一个是使用Serializable接口，这个接口是Java SE本身就支持的序列化接口，但是使用这个接口来进行Intent数据的传递有一个缺点。因为这个序列化和反序列化过程中需要大量I/O操作，从而导致开销大效率低。
- 第二个是使用Parcelable接口的使用，也就是本篇文章的主要重点。这种方式是Android中支持的序列化方式，使用起来稍微麻烦点，但是效率更高，所以我们一般使用这个方式来通过Intent传递数据。

Parcelable接口的基本使用方法

在需要进行传递的类中实现Parcelable接口，假如我们要传递一个Person类的对象,我们需要在其中实现Parcelable接口：

```
public class Person implements Parcelable{
      private String name;
      private Int age;

      ...//设置setter() 和 getter()方法

      //下面是实现Parcelable接口的内容
      @Override
    public int describeContents() {
        return 0;                                          //一般返回零就可以了
    }

    @Override
    public void writeToParcel(Parcel dest, int flags) {      //在这个方法中写入这个类的变量
        dest.writeString();                    //对应着 String name;
        dest.writeInt();                        //对应着 Int age;
    }
    //在实现上面的接口方法后，接下来还需要执行反序列化，定义一个变量，并重新定义其中的部分方法
     public static final Parcelable.Creator<Person> CREATOR = new Parcelable.Creator<P

        @Override
        public Person createFromParcel(Parcel source) {                  //在这个方法中反
             //反序列化的属性的顺序必须和之前写入的顺序一致
            Person person = new Person();
            person.name = source.readString();
            person.age = source.readAge();
```

```
            return person;
        }

        @Override
        public Person[] newArray(int size) {
            return new Person[size];                    //一般返回一个数量为size的传递的类
        }
    };
}
```

然后是使用Intent传递的方式：

```
//传递
Person person = new Person("张三",18);
Intent intent = new Intent(FirstActivity.this,SecondActivity.class);
intent.putExtra("person_data",person);
startActivity(intent);

//接受
Person person = (Person) getIntent.getParcelableExtra("person_data);
```

使用Parcelable接口的注意事项

- boolean类型的序列化和反序列化。由于没有直接序列化boolean类型的方式，所以采用如下方式：

```
//序列化
dest.writeByte((Byte)(isChecked ? 1 : 0));

//反序列化
XXX.isChecked = source.readByte != 0;
```

- 当类中某一个属性也是一个自定义类时，有两种方案解决，一种就是将该类也实现Parcelalbe接口，另外一种见下一条注意事项。

```
Public class Person implements Parcelable {
       ...    //一些属性
      School school;                    //School 是一个已经实现Parcleable接口的自定义类
      ...
      //序列化该School属性
      dest.writeParcelable(school,0);

      //反序列化School属性,由于school是另一个可序列化对象，所以它的反序列化过程需要传递当前线程的上
      XXX.school = source.readParcelable(Thread.currentThread().getContextClassLoader(
}
```

- 如果传递的对象的某个属性所属的类是系统自带的同时没有也不可能实现Parcelable接口的类（比如：Calendar），或者是一个第三方类，但是不方便实现Parcelable接口，那么可以在将该对象拆为若干个必要的属性进行传递，反序列化后再组装。

```
public class Person implements Parcelable{
      ... //一些属性
      Calendar calendar;
      ...
      //序列化Calendar对象，Calendar对象中最重要的就是Time和TimeZone两个属性，所以可以将其拆为这
      dest.writeLong(calendar.getTimeInMillis());
      dest.writeInt(calendar.getTimeZone().getID());

      //进行反序列化
      person.calendar.setTimeInMillis(source.readLong());
      person.calendar.getTimeZone().setID(source.readString());
}
```

- 如果有一个属性是Enum数组

```
public class Person implements Parcelable {
      ...//其他属性
      private Day[] days = {Day.SUNDAY,Day.MONDAY,Day.TUESDAY,Day.WEDNSDAY,Day.THURSDA
```

```
4
5       //进行序列化，将Enum数组转换成Int类型的List,利用每个Enum对象都有自己的需要order()
6       List<Integer> dayInts = new ArrayList<Integer>();
7        for (Day tempDay : days){
8            dayInts.add(tempDay.ordinal());
9        }
10      dest.writeList(dayInts);
11
12      //进行反序列化，将上面的过程反过来
13      List<Integer> dayInts = new ArrayList<Integer>();
14      source.readList(dayInts,null);
15      List<Day> dayEnums = new ArrayList<Day>();
16      for(int temp : dayInts){
17            dayEnums.add(Day.values()[temp]);
18      }
19      XXX.days = daysEnums.toArray(new Day[dayEnum.size()]);
20  }
```

- 如果打算将对象序列化到存储设备或者通过网络传送，建议使用Serializable接口，因为使用Parcelable接口会过于复杂。