# 序列化

# 序列化介绍

## 1.1、什么是序列化和反序列化

**序列化：**将对象转化为二进制序列的过程。

**反序列化：**将二进制序列转化为对象的过程。

## 1.2、为什么需要使用序列化

一般情况下序列化可以划分为下面几种情况：

1. 永久性保存对象，保存字节序列在本地存储；
2. 对象在网络中传递；
3. 对象在IPC（Inter-Process Communication，进程间通信）中传递；

## 1.3、怎么实现序列化

在Android中实现序列化方式有两种：

1. 实现Serializable接口（java可用）
2. 实现Parceable接口（Android可用）

## 1.4、使用选择

1）如果对象需要将二进制数据存储在磁盘上，那么Parceable就不合适了，毕竟它的载体是内存，这时候就需要选择Serializable。

2）其他的情况在Android中建议使用Parceable，频繁的使用Serializable容易频繁触发GC。

例如，使用Intent在组件中传递数据的时候，建议使用Parceable，传递一些复杂对象。

# Serializable

## 2.1、介绍

Serializable接口实际上就是作为一个标记接口，只要直接实现Serializable接口的对象就可以进行序列化和反序列化了。具体进行序列化靠的是ObjectOutputStream和ObjectInputStream来完成的，中间会牵扯到一些反射的东西，我也不太会就暂时跳过了。

**注意事项：**

1、serializableUID用来标识当前序列化的对象类版本，最好自己定义以下，如果没有定义的话，后面会自动生成一个UID；

private static final long *serialVersionUID* = 1L;

2、被transient描述的域和类的静态变量是不会被序列化的，序列化是针对类实例。

3、如果一个实现了Serializable类的对象继承自另外一个类，那么这个类要么需要继承自Serializable，要么需要提供一个无参构造器。

4、反序列化产生的对象并不是通过构造器创建的，那么很多依赖于构造器保证的约束条件在对象反序列化时都无法保证。

5、需要进行序列化的对象所有的域都必须实现Serializable接口，不然会直接报错NotSerializableException。当然，有两个例外：域为空或者域被transient描述是不会报错的。

## 2.2、例子

一个实现接口的对象：

public class Book implements Serializable {  
 private String BookId;  
 private String BookName;  
  
 public Book(String bookId, String bookName) {  
 BookId = bookId;  
 BookName = bookName;  
 }  
  
 public String getBookId() {  
 return BookId;  
 }  
  
 public void setBookId(String bookId) {  
 BookId = bookId;  
 }  
  
 public String getBookName() {  
 return BookName;  
 }  
  
 public void setBookName(String bookName) {  
 BookName = bookName;  
 }  
}

# Parcelable

## 3.1、什么是Parcel

简单来说，Parcel提供了一套机制，可以将序列化之后的数据写入到一个共享内存中，其他进程通过Parcel可以从这块共享内存中读出字节流，并反序列化成对象,下图是这个过程的模型。

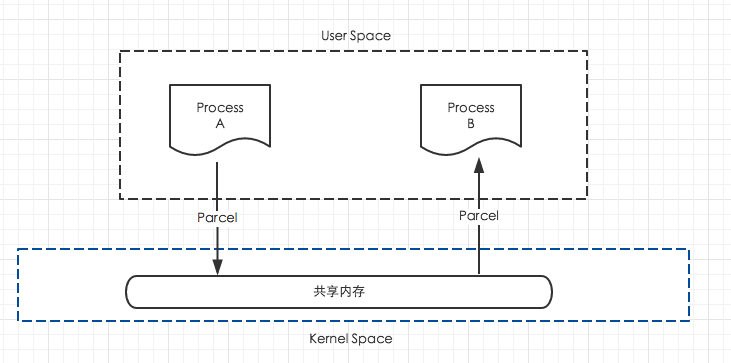


图3-1 parcel

高深的东西在这里不存在的，看得懂就ok了。基本概念嘛。

## 3.2、介绍

Parcelable接口实际上是google提供的一个Android接口。相对Serializable就复杂多了。

先看例子代码：

public class Dog implements Parcelable {  
 private String name;  
  
 public Dog(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public int describeContents() {  
 return 0;// 内容接口描述，默认返回0即可。  
 }  
 //序列化使用  
 @Override  
 public void writeToParcel(Parcel dest, int flags) {  
 dest.writeString(this.name);  
 }  
 //在Parcelable.Creator中 反序列化中使用  
 protected Dog(Parcel in) {  
 this.name = in.readString();  
 }  
  
 public static final Parcelable.Creator<Dog> *CREATOR* = new Parcelable.Creator<Dog>() {  
 @Override  
 public Dog createFromParcel(Parcel source) {  
 return new Dog(source);// 在构造函数里面完成了 读取 的工作  
 }  
 //供反序列化本类数组时调用的  
 @Override  
 public Dog[] newArray(int size) {  
 return new Dog[size];  
 }  
 };  
}

是不是比那个多了好多方法，复杂了不少。

1、实现Parcelable接口。

2、writeToParcel()方法，将需要持久化的字段保存到一个Parcel中。

3、重写describeContents方法，内容接口描述，一般情况下直接返回0就好了。

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

可是，这个东西怎么复杂写起来这么麻烦，有没有简便方法呢？

当然是有的啊。

使用插件，自动化实现Parcelable接口，如下图。

我是已经装好了，所有右边显示的是卸载。

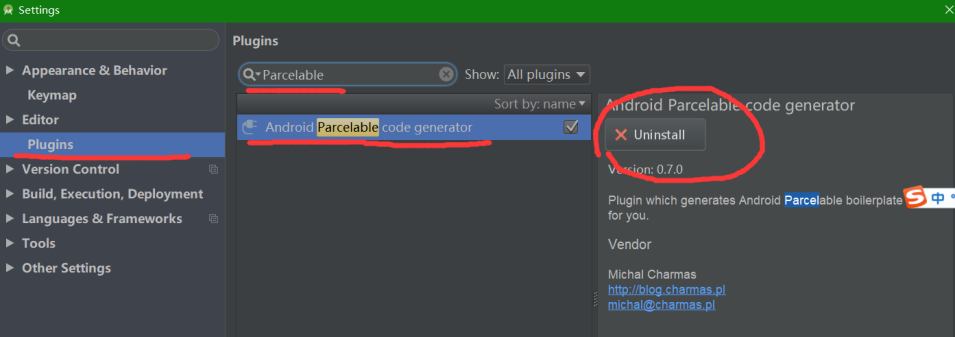


图3-2 插件安装

**如何使用呢？非常简单！！！！安装后重启as哦**

在创建的javabean的class中右键：

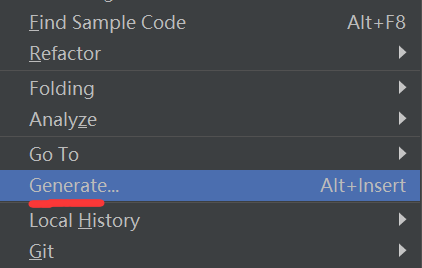


图3-3 右键

选中后，选择

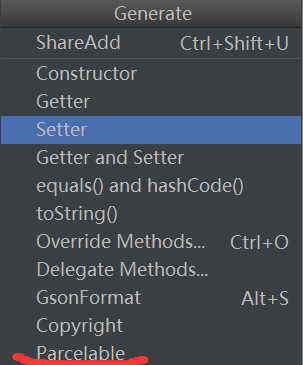


图3-4 Parcelable

就这样自动创建完成了，简单粗暴吧

# 例子

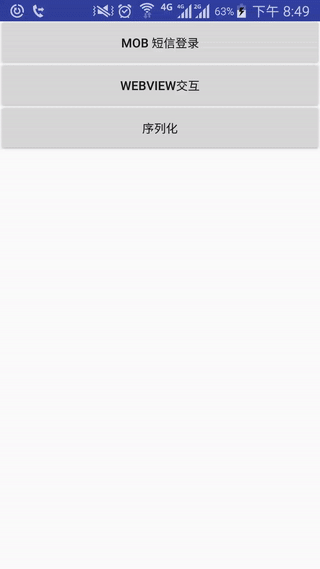


图4-1 gif