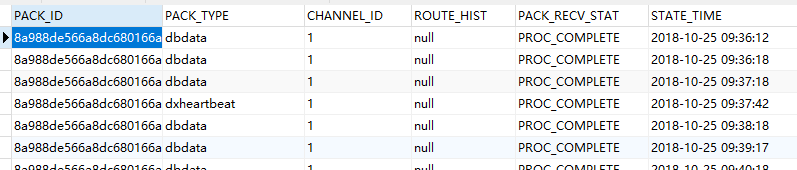
# Java序列化方法比较

## 序列化数据

16124条mysql数据库记录：



把数据放入包装对象中序列化。

## 序列化方法：

### Java序列化

java对象序列化不仅保留一个对象的数据，而且递归保存[对象引用](https://www.baidu.com/s?wd=%E5%AF%B9%E8%B1%A1%E5%BC%95%E7%94%A8&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "_blank)的每个对象的数据。可以将整个对象层次写入字节流中，可以保存在文件中或在网络连接上传递。利用对象序列化可以进行对象的"深复制"，即复制对象本身及引用的对象本身。序列化一个对象可能得到整个对象序列。

优点：支持的序列化多

缺点：时间比较久和大小比较大

### FastJson序列化

FastJson是啊里巴巴的的开源库，用于对JSON格式的数据进行解析和打包。

FastJson特点如下：

（1）能够支持将java bean序列化成JSON字符串，也能够将JSON字符串反序列化成Java bean。

（2）顾名思义，FastJson操作JSON的速度是非常快的。

（3）无其他包的依赖。

（4）使用比较方便。

（5）完全支持Java Bean、集合、Map、日期、Enum，支持范型，支持自省；无依赖，能够直接运行在Java SE5.0以上版本；支持Android；开源 (Apache 2.0)

### gsonSerialize序列化

GSON是Google提供的用来在Java对象和JSON数据之间进行映射的Java类库。可以将一个Json字符转成一个Java对象，或者将一个Java转化为Json字符串。

特点：a、快速、高效

    b、代码量少、简洁

    c、面向对象

    d、数据传递和解析方便

### KryoSerialize序列化

Kryo是一个快速高效的Java序列化框架，旨在提供快速、高效和易用的API。无论文件、数据库或网络数据Kryo都可以随时完成序列化。Kryo还可以执行自动深拷贝（克隆）、浅拷贝（克隆）。这是对象到对象的直接拷贝，非对象->字节->对象的拷贝。

优点：

速度快，序列化后体积小

缺点：

跨语言支持较复杂

### ProtoStuffSerialize序列化

Protocol Buffer是谷歌出品的一种数据交换格式，独立于语言和平台，类似于json。Google提供了多种语言的实现：java、c++、go和python。对象序列化城Protocol Buffer之后可读性差，但是相比xml，json，它占用小，速度快。适合做数据存储或 RPC 数据交换格式。

相对我们常用的json来说，Protocol Buffer门槛更高，因为需要编写.proto文件，再把它编译成目标语言，这样使用起来就很麻烦。但是现在有了protostuff之后，就不需要依赖.proto文件了，他可以直接对POJO进行序列化和反序列化，使用起来非常简单

protobuf是谷歌推出的与语言无关、平台无关的通信协议，一个对象经过protobuf序列化后将变成二进制格式的数据，所以他可读性差，但换来的是占用空间小，速度快。使用protobuf要先使用特定的语法编写一个.proto文件，该文件与语言无关，然后使用特殊的编译器对该文件进行编译，生成与语言相关的文件，如java，那么将生成java的类，该类不仅有我们自己定义的属性，还提供了序列化，反序列化等其他方法。直接把该类copy到项目中，就可以使用了。不过缺点是，假如我们是数据的发送方，那么接受方也要有一个通过相同的.proto编译出来的“类”（假设对方使用java语言），才可以顺利地进行反编译。这样一来，假如我们对proto 2.6版本的编辑器对.proto文件进行编译，而对方使用的是2.3版本的编译器进行编译，那么编译出来的类是不一样的，且两个版本互不兼容。所以两方的版本要保持一致。这么一来，假如一方升级，但没及时通知另一方，那么可能导致对方无法反序列化！这个缺点也是不小的。

优点:

1：序列化后体积相比Json和XML很小，适合网络传输，空间效率是JSON的2-5倍，时间效率要高

2：支持跨平台多语言

3：消息格式升级和兼容性还不错

4：序列化反序列化速度很快，快于Json的处理速度

缺点：

1：消息结构可读性不高，序列化后的字节序列为二进制序列不能简单的分析有效性；

2：目前使用不广泛，只支持java,C++和Python；

### fstSerialise序列化

优点：

fst是完全兼容JDK序列化协议的系列化框架，序列化速度大概是JDK的4-10倍，大小是JDK大小的1/3左右。而且兼容 JDK 原生的序列化。要求 JDK 1.7 支持。

序列化速度更快，对象变得更小。指的是序列化简单对象，如果序列化复杂对象和java1大小差不多

简易替代者。类序列化时不需要getters、setters、Constructors和Interfaces；扩展了Outputstream，实现了ObjectInput/ObjectOutput接口。 使用时对原来的代码改动很小。

完全支持JDK序列化的主要功能，例如Externalizable中的writeObject、readObject、readReplace、validation、putField、getField方法，可自定义钩子等等。JDK序列化的对象可以直接用FST处理。

序列化的对象中保留的连接关系图与JDK默认序列化结果保持一致。

通过注解、自定义serializer支持自定义优化。

支持序列化版本（仅2.x支持）

JSON序列化/反序列化支持（可选）

条件解码（可根据要求跳过对象/流中部分内容的解码）

## 时间和大小比较

说明：

1、avro和protoBuff序列化需要模版，利用插件生成javabean，比较复杂，无法和其他方法在一起比较，暂时抛弃。

2、在这里的序列化时间指的是：从1.6w个数据封装后的类进行序列化后，生成的字节或者字符串写入BufferedOutputStream，FileOutputStream，进行存储；

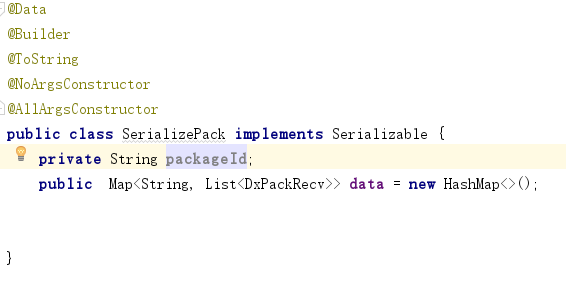
读入是相反的操作或者有的采用序列化方式自带的读入文件输入流。

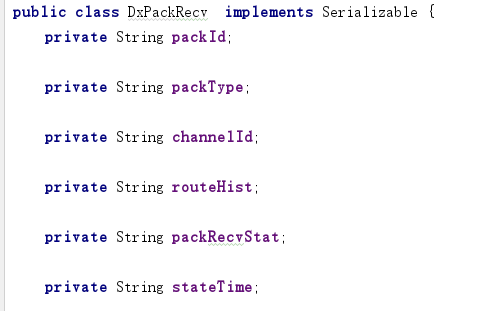
serializeTime=序列化+存储到文件

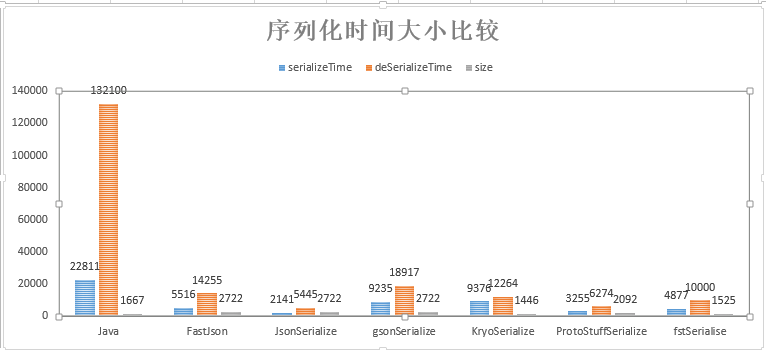
deSerializeTime=序列化+存储到文件+读出文件+反序列成Object

3、序列化次数是执行100次

4、序列化对象：







## 总结

只针对对数据进行封装后的性能：（若单纯序列化简单对象，性能会出现不一样的情况）

从大小看：kroy<fst<java

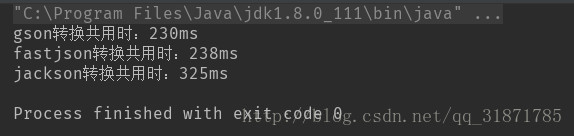
从序列化时间：json<protoStuff<fastJson

从反序列化时间：json<ProtoStuff<fstJson

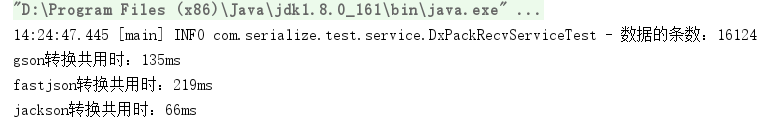
比起来原生的json序列化和protoStuff更优秀一点。

## 关于json、fastJson、jackson比较：

直接序列化对象：



序列化封装的对象（对象里有map，map里有16k数据）：



说明序列化简单的对象是最好的分别是gson、fastJson、jackson。

序列化复杂对象最好的分别是jackson、gson、fastson。

#### 总结

把Java对象JSON序列化，Jackson速度最快，在测试中比Gson快接近50%，FastJSON和Gson速度接近。   
把JSON反序列化成Java对象，FastJSON、Jackson速度接近，Gson速度稍慢，不过差距很小。

## 关于java fst kroy比较

序列化简单对象：

序列化 ， 反序列化 对比测试：

原生序列化方案[序列化10000次]耗时：458ms size:=1160000

fst序列化方案[序列化10000次]耗时：184ms size:=550000

kryo序列化方案[序列化10000次]耗时：462ms size:=390000

序列化复杂对象：fst和java大小差不多，素服fst快很多，kroy和fst差不多。

#### 总结

依据业务场景进行选择，简单对象是fst快很多，大小也很小。