链滴

几种 Java 序列化方式的实现

作者：aqjun
原文链接：https://ld246.com/article/1488641672451
来源网站：链滴

0、前言

本文主要对几种常见Java序列化方式进行实现。包括Java原生以流的方法进行的序列化、Json序列化FastJson序列化、Protobuff序列化。

1、Java原生序列化

Java原生序列化方法即通过Java原生流(InputStream和OutputStream之间的转化)的方式进行转化需要注意的是JavaBean实体类必须实现Serializable接口，否则无法序列化。Java原生序列化代码示如下所示：

```
package serialize;

import java.io.BufferedInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.List;
/**
 *
 * @author liqqc
 *
 */
public class JavaSerialize {
    public static void main(String[] args) throws ClassNotFoundException, IOException {
        new JavaSerialize().start();
    }

    public void start() throws IOException, ClassNotFoundException {
        User u = new User();
        List friends = new ArrayList<>();
        u.setUserName("张三");
        u.setPassWord("123456");
        u.setUserInfo("张三是一个很牛逼的人");
        u.setFriends(friends);

        User f1 = new User();
        f1.setUserName("李四");
        f1.setPassWord("123456");
        f1.setUserInfo("李四是一个很牛逼的人");

        User f2 = new User();
        f2.setUserName("王五");
        f2.setPassWord("123456");
        f2.setUserInfo("王五是一个很牛逼的人");

        friends.add(f1);
        friends.add(f2);
```

```
        Long t1 = System.currentTimeMillis();
        ByteArrayOutputStream out = new ByteArrayOutputStream();
        ObjectOutputStream obj = new ObjectOutputStream(out);
        for(int i = 0; i<10; i++) {
            obj.writeObject(u);
        }
        System.out.println("java serialize: " +(System.currentTimeMillis() - t1) + "ms; 总大小：" +
ut.toByteArray().length );

        Long t2 = System.currentTimeMillis();
        ObjectInputStream ois = new ObjectInputStream(new BufferedInputStream(new java.io.
yteArrayInputStream(out.toByteArray())));
        User user = (User) ois.readObject();
        System.out.println("java deserialize: " + (System.currentTimeMillis() - t2) + "ms; User: " +
user);
    }

}
```

运行结果：

```
java serialize: 8ms; 总大小：420
java deserialize: 1ms; User: User [userId=null, userName=张三, passWord=123456, userInfo=
三是一个很牛逼的人, friends=[User [userId=null, userName=李四, passWord=123456, userInfo
李四是一个很牛逼的人, friends=null], User [userId=null, userName=王五, passWord=123456, us
rInfo=王五是一个很牛逼的人, friends=null]]]
```

2、Json序列化

Json序列化一般会使用jackson包，通过ObjectMapper类来进行一些操作，比如将对象转化为byte组或者将json串转化为对象。现在的大多数公司都将json作为服务器端返回的数据格式。比如调用一服务器接口，通常的请求为xxx.json?a=xxx&b=xxx的形式。Json序列化示例代码如下所示：

```
package serialize;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.fasterxml.jackson.databind.ObjectMapper;
/**
 *
 * @author liqqc
 *
 */
public class JsonSerialize {
    public static void main(String[] args) throws IOException {
        new JsonSerialize().start();
    }

    public void start() throws IOException {
```

```
        User u = new User();
        List friends = new ArrayList<>();
        u.setUserName("张三");
        u.setPassWord("123456");
        u.setUserInfo("张三是一个很牛逼的人");
        u.setFriends(friends);

        User f1 = new User();
        f1.setUserName("李四");
        f1.setPassWord("123456");
        f1.setUserInfo("李四是一个很牛逼的人");

        User f2 = new User();
        f2.setUserName("王五");
        f2.setPassWord("123456");
        f2.setUserInfo("王五是一个很牛逼的人");

        friends.add(f1);
        friends.add(f2);

        ObjectMapper mapper = new ObjectMapper();
        Long t1 = System.currentTimeMillis();
        byte[] writeValueAsBytes = null;
        for (int i = 0; i < 10; i++) {
            writeValueAsBytes = mapper.writeValueAsBytes(u);
        }
        System.out.println("json serialize: " + (System.currentTimeMillis() - t1) + "ms; 总大小：" +
writeValueAsBytes.length);
        Long t2 = System.currentTimeMillis();
        User user = mapper.readValue(writeValueAsBytes, User.class);
        System.out.println("json deserialize: " + (System.currentTimeMillis() - t2) + "ms; User: " +
user);

    }
}
```

运行结果：

```
json serialize: 55ms; 总大小：341
json deserialize: 35ms; User: User [userId=null, userName=张三, passWord=123456, userInfo
张三是一个很牛逼的人, friends=[User [userId=null, userName=李四, passWord=123456, userInf
=李四是一个很牛逼的人, friends=null], User [userId=null, userName=王五, passWord=123456,
serInfo=王五是一个很牛逼的人, friends=null]]]
```

3、FastJson序列化

fastjson 是由阿里巴巴开发的一个性能很好的Java 语言实现的 Json解析器和生成器。特点：速度快，试表明fastjson具有极快的性能，超越任其他的Java json parser。功能强大，完全支持java bean、合、Map、日期、Enum，支持范型和自省。无依赖，能够直接运行在Java SE 5.0以上版本

支持Android。使用时候需引入FastJson第三方jar包。FastJson序列化代码示例如下所示：

```java
package serialize;

import java.util.ArrayList;
import java.util.List;

import com.alibaba.fastjson.JSON;
/**
 *
 * @author liqqc
 *
 */
public class FastJsonSerialize {

    public static void main(String[] args) {
        new FastJsonSerialize().start();
    }

    public void start(){
        User u = new User();
        List friends = new ArrayList<>();
        u.setUserName("张三");
        u.setPassWord("123456");
        u.setUserInfo("张三是一个很牛逼的人");
        u.setFriends(friends);

        User f1 = new User();
        f1.setUserName("李四");
        f1.setPassWord("123456");
        f1.setUserInfo("李四是一个很牛逼的人");

        User f2 = new User();
        f2.setUserName("王五");
        f2.setPassWord("123456");
        f2.setUserInfo("王五是一个很牛逼的人");

        friends.add(f1);
        friends.add(f2);

        //序列化
        Long t1 = System.currentTimeMillis();
        String text = null;
        for(int i = 0; i<10; i++) {
            text = JSON.toJSONString(u);
        }
        System.out.println("fastJson serialize: " +(System.currentTimeMillis() - t1) + "ms; 总大小:
 + text.getBytes().length);
        //反序列化
        Long t2 = System.currentTimeMillis();
        User user = JSON.parseObject(text, User.class);
        System.out.println("fastJson serialize: " + (System.currentTimeMillis() -t2) + "ms; User: " +
user);
    }
}
```

运行结果：

fastJson serialize: 284ms; 总大小：269
fastJson serialize: 26ms; User: User [userId=null, userName=张三, passWord=123456, userInf
=张三是一个很牛逼的人, friends=[User [userId=null, userName=李四, passWord=123456, userI
fo=李四是一个很牛逼的人, friends=null], User [userId=null, userName=王五, passWord=123456
 userInfo=王五是一个很牛逼的人, friends=null]]]

4、ProtoBuff序列化

ProtocolBuffer是一种轻便高效的结构化数据存储格式，可以用于结构化数据序列化。适合做数据存或 RPC 数据交换格式。可用于通讯协议、数据存储等领域的语言无关、平台无关、可扩展的序列化构数据格式。

优点：跨语言；序列化后数据占用空间比JSON小，JSON有一定的格式，在数据量上还有可以压缩的间。

缺点：它以二进制的方式存储，无法直接读取编辑，除非你有 .proto 定义，否则无法直接读出 Protouffer的任何内容。

其与thrift的对比：两者语法类似，都支持版本向后兼容和向前兼容，thrift侧重点是构建跨语言的可缩的服务，支持的语言多，同时提供了全套RPC解决方案，可以很方便的直接构建服务，不需要做太其他的工作。 Protobuffer主要是一种序列化机制，在数据序列化上进行性能比较，Protobuffer相对好。

ProtoBuff序列化对象可以很大程度上将其压缩，可以大大减少数据传输大小，提高系统性能。对于量数据的缓存，也可以提高缓存中数据存储量。原始的ProtoBuff需要自己写.proto文件，通过编译将其转换为java文件，显得比较繁琐。百度研发的jprotobuf框架将Google原始的protobuf进行了封，对其进行简化，仅提供序列化和反序列化方法。其实用上也比较简洁，通过对JavaBean中的字段行注解就行，不需要撰写.proto文件和实用编译器将其生成.java文件，百度的jprotobuf都替我们做这些事情了。

一个带有jprotobuf注解的JavaBean如下所示，如果你想深入学习可以参照https://github.com/google/protobuf。

```
package serialize;

import java.io.Serializable;
import java.util.List;
import com.baidu.bjf.remoting.protobuf.FieldType;
import com.baidu.bjf.remoting.protobuf.annotation.Protobuf;

public class User implements Serializable {
    private static final long serialVersionUID = -7890663945232864573L;

    @Protobuf(fieldType = FieldType.INT32, required = false, order = 1)
    private Integer userId;

    @Protobuf(fieldType = FieldType.STRING, required = false, order = 2)
    private String userName;

    @Protobuf(fieldType = FieldType.STRING, required = false, order = 3)
```

```
    private String passWord;

    @Protobuf(fieldType = FieldType.STRING, required = false, order = 4)
    private String userInfo;

    @Protobuf(fieldType = FieldType.OBJECT, required = false, order = 5)
    private List friends;

    public Integer getUserId() {
        return userId;
    }

    public void setUserId(Integer userId) {
        this.userId = userId;
    }

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
    }

    public String getPassWord() {
        return passWord;
    }

    public void setPassWord(String passWord) {
        this.passWord = passWord;
    }

    public String getUserInfo() {
        return userInfo;
    }

    public void setUserInfo(String userInfo) {
        this.userInfo = userInfo;
    }

    public List getFriends() {
        return friends;
    }

    public void setFriends(List friends) {
        this.friends = friends;
    }

    @Override
    public String toString() {
        return "User [userId=" + userId + ", userName=" + userName + ", passWord=" + passW
rd + ", userInfo=" + userInfo
                + ", friends=" + friends + "]";
    }
```

```
}
```

jprotobuf序列化代码示例如下所示：

```
package serialize;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import com.baidu.bjf.remoting.protobuf.Codec;
import com.baidu.bjf.remoting.protobuf.ProtobufProxy;
/**
 *
 * @author liqqc
 *
 */
public class ProtoBuffSerialize {

    public static void main(String[] args) throws IOException {
        new ProtoBuffSerialize().start();
    }

    public void start() throws IOException {
        Codec studentClassCodec = ProtobufProxy.create(User.class, false);

        User u2 = new User();
        List friends = new ArrayList<>();
        u2.setUserName("张三");
        u2.setPassWord("123456");
        u2.setUserInfo("张三是一个很牛逼的人");
        u2.setFriends(friends);

        User f1 = new User();
        f1.setUserName("李四");
        f1.setPassWord("123456");
        f1.setUserInfo("李四是一个很牛逼的人");

        User f2 = new User();
        f2.setUserName("王五");
        f2.setPassWord("123456");
        f2.setUserInfo("王五是一个很牛逼的人");
        friends.add(f1);
        friends.add(f2);

        Long stime_jpb_encode = System.currentTimeMillis();
        byte[] bytes = null;
        for(int i = 0; i<10; i++) {
            bytes = studentClassCodec.encode(u2);
        }
        System.out.println("jprotobuf序列化耗时：" + (System.currentTimeMillis() - stime_jpb_en
ode) + "ms; 总大小：" + bytes.length);
```

```
        Long stime_jpb_decode = System.currentTimeMillis();
        User user = studentClassCodec.decode(bytes);
        Long etime_jpb_decode = System.currentTimeMillis();
        System.out.println("jprotobuf反序列化耗时："+ (etime_jpb_decode-stime_jpb_decode) +
ms; User: " + user);
    }

}
```

运行结果：

```
jprotobuf序列化耗时：9ms; 总大小：148
jprotobuf反序列化耗时：0ms; User: User [userId=null, userName=张三, passWord=123456, use
Info=张三是一个很牛逼的人, friends=[User [userId=null, userName=李四, passWord=123456, u
erInfo=李四是一个很牛逼的人, friends=null], User [userId=null, userName=王五, passWord=12
456, userInfo=王五是一个很牛逼的人, friends=null]]]
```

5、总结

我们通过Main方法来进行对比测试，（但是通过测试发现少量数据无法准确显示每种序列化方式的劣，故这里无法给出比较好的答案，仅供参考）。示例代码如下所示：

```
package serialize;

import java.io.IOException;

/**
 * @author liqqc
 */
public class Main {

    public static void main(String[] args) throws IOException, ClassNotFoundException {

        ProtoBuffSerialize protoBuffSerialize = new ProtoBuffSerialize();
        protoBuffSerialize.start();

        System.err.println();
        System.err.println();

        JavaSerialize javaSerialize = new JavaSerialize();
        javaSerialize.start();
        System.err.println();

        JsonSerialize jsonSerialize = new JsonSerialize();
        jsonSerialize.start();
        System.err.println();

        FastJsonSerialize fastJsonSerialize = new FastJsonSerialize();
        fastJsonSerialize.start();
    }
}
```

运行结果：

jprotobuf序列化耗时：7ms; 总大小：148
jprotobuf反序列化耗时：0ms

java serialize: 6ms; 总大小：420
java deserialize: 1ms

json serialize: 37ms; 总大小：341
json deserialize: 27ms

fastJson serialize: 173ms; 总大小：269
fastJson serialize: 35ms

上面的测试仅供参考，并不能代表通过大量数据进行测试的结果。可以发现：序列化后对象的所占大上：protobuff序列化所占总大小是最少的；其次是fastJson序列化；最后是json序列化和java原生序化。对于序列化耗时，上面的测试不准。

还是去看看专业测试分析吧，具体情况可以进去看看https://github.com/eishay/jvm-serializers/wiki

本文仅仅简单介绍了下几种序列化方式的实现，并未经过大量测试对其进行对比分析，待后续有时间精力在进行补充。