Java序列化：

Java序列化会把要序列化的对象类的元数据和业务数据全部序列化为字节流，而且是把整个继承关系上的东西全部序列化了。**它序列化出来的字节流是对那个对象结构到内容的完全描述**，包含所有的信息，因此**效率较低而且字节流比较大**。但是由于确实是序列化了所有内容，所以可以说**什么都可以传输，因此也更可用和可靠**。

hession序列化：

它的实现机制是着重于数据，附带简单的类型信息的方法。就像Integer a = 1，hessian会序列化成I 1这样的流，I表示int or Integer，1就是数据内容。而对于复杂对象，通过Java的反射机制，**hessian把对象所有的属性当成一个Map来序列化**，产生类似M className propertyName1 I 1 propertyName S stringValue（大概如此，确切的忘了）这样的流，**包含了基本的类型描述和数据内容。**而在序列化过程中，如果一个对象之前出现过，hessian会直接插入一个R index这样的块来表示一个引用位置，从而省去再次序列化和反序列化的时间。这样做的代价就是hessian需要对不同的类型进行不同的处理（因此hessian直接偷懒不支持short），而且遇到某些特殊对象还要做特殊的处理（比如StackTraceElement）。**而且同时因为并没有深入到实现内部去进行序列化，所以在某些场合会发生一定的不一致，比如通过Collections.synchronizedMap得到的map。**

问题 序列化日期异常

```
// esb 调用     2020-05-01 00:00:00     dubbo 调用 Fri May 01 00:00:00 CST 2020
hessian
JavaSerializer#getFieldSerializer
```

```
.....
  else if (java.util.Date.class.equals(type)
        || java.sql.Date.class.equals(type)
        || java.sql.Timestamp.class.equals(type)
        || java.sql.Time.class.equals(type)) {
    return DateFieldSerializer.SER;
```

```
} else
    return FieldSerializer.SER;

static class DateFieldSerializer extends FieldSerializer {
    static final FieldSerializer SER = new DateFieldSerializer();
    @Override
    void serialize(AbstractHessianOutput out, Object obj, Field field)
            throws IOException {
        java.util.Date value = null;
        try {
            value = (java.util.Date) field.get(obj);
        } catch (IllegalAccessException e) {
            log.log(Level.FINE, e.toString(), e);
        }
        if (value == null)
            out.writeNull();
        else
            out.writeUTCDate(value.getTime());
    }
}
```