6.1日福袋事故原因分享

> **6.1日凌晨，接到客诉：购买福袋抽不出奖品，经定位发现 直接原因是没有建表，之后盲目的处理过程当中认为mq没发生重试，对账过程当中又没有核对数据明细， 错误的补发了金币，针对此次事故，分享出来一些犯过的错误，与大家共勉，引以为戒！！！**

主要问题清单：

- 核账
- fastjson 序列化需注意问题
- 日志规范
- 代码严谨性

按时间顺序还原问题经过，内容如下：

代码严谨性

问题代码1-自动建表job

job方法片段：

```
@Override
public ReturnT<String> execute(String... params) throws SQLException {
    logger.info("CreateTableJobHandler execute params->{}", (Object[]) params);
    Integer year = null;
    Integer month = null;
    try {
        year = Integer.parseInt(params[0]);
        month = Integer.parseInt(params[1]);
        tableTemplate.operate(year, month);
        logger.info("CreateTableJobHandler.execute success");
    } catch (Exception e) {
        logger.error("CreateTableJobHandler params error:{}", (Object[]) params);
    }
    return ReturnT.SUCCESS;
}
```

tableTemplate.operate 方法片段

```
public void operate(Integer year, Integer month) throws SQLException {
    //月表
    for (String monthValue : monthList) {
            Calendar calendar = DateUtils.getPreMonthCalendarByYearAndMonth(year,
month);
            String monthString = DateUtils.getFormatedDateString(calendar.getTime(),
DateUtils.FORMATE_YYYYMM);
            String sql = MessageFormat.format(monthValue,monthString);
            Integer result = (Integer) sqlService.executeSql(sql, getConnection());
            if (result.equals(-1)) {
                logger.error("sql error:{}", sql);
            }
        }
        //天表
        for (String dayValue : dayList) {
            List<Date> datesByYearAndMonth = DateUtils.getPreDatesByYearAndMonth(year,
month);
            for (Date date : datesByYearAndMonth) {
                String dayString = DateUtils.getFormatedDateString(date,
DateUtils.FORMATE_YYYYMMDD);
                String sql = MessageFormat.format(dayValue,dayString);
                Integer result = (Integer) sqlService.executeSql(sql,
getConnection());
                if (result.equals(-1)) {
                    logger.error("sql error:{}", sql);
                }
            }
        }

    }
```

job设计功能及缺陷

1. 功能:

- 每月定期四次执行，如果入参为空，则建立下个月的表，年末顺延下一年度
- 如果入参不为空，则建立指定指定参数下一个月的表，年月正确性校验

2. 缺陷:

- 如果年月是一个已经过期的，未处理，

  case，输入2021,4 则当月分执行可生成5月份表，到5月份的时候，6月份的表不会建立

类似问题思考：方法入参校验，调用链路上方法分不清边界，参数完整性校验，异常抛出问题，都很值得思考

解决方案：建立有效的建表审查复查机制，用技术的角度解决认为疏忽的问题。目前已经建立完成

核账

当初问题发生后，运营的建议是补发金币，代码如下，先贴出来，我们后分析

```java
@Override
    public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs,
ConsumeConcurrentlyContext context) {


        try {
            GiftSendMessage message = JSON.parseObject(msgs.get(0).getBody(),
GiftSendMessage.class);
            GiftSendFilterConfig filter = this.config.getGiftSendFilterConfig();

            if (!filter.getEnabled()) {
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }

            if
(!filter.getGiftIdBySend().containsKey(String.valueOf(message.getGiftId()))) {
                return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
            }

            UserInfo userInfo = new UserInfo();
            userInfo.setMemberId(String.valueOf(message.getFromId()));
            userInfo.setScId(message.getScId());
            userInfo.setAnchorScid(message.getAnchorScid());
            userInfo.setAnchorId(message.getAnchorId());
            userInfo.setGiftSendToId(message.getToId());

            if (message.getLiveType() != null) {
                userInfo.setLiveType(String.valueOf(message.getLiveType()));
            }

            if (!Strings.isNullOrEmpty(message.getAppId())) {
                userInfo.setAppId(Integer.valueOf(message.getAppId()));
            }

            userInfo.setSdkId(message.getSdkId());
            userInfo.setPkgName(message.getPkgName());
            userInfo.setAppVersion(message.getAppVersion());
            userInfo.setDeviceId(message.getDeviceId());
            userInfo.setDeviceType(message.getDeviceType());

            if (!Strings.isNullOrEmpty(message.getVideoSource())) {
                userInfo.setVideoSource(Integer.valueOf(message.getVideoSource()));
            }

            userInfo.setPkId(message.getPkId());
            userInfo.setMsgSerialId(message.getMsgSerialId());

            DrawByGiftSendConfig draw =
filter.getGiftIdBySend().get(String.valueOf(message.getGiftId()));

            GiftSendRequest request = new GiftSendRequest();
            request.setActivityId(draw.getActivityId());
            request.setBoxType(draw.getBoxType());
            request.setGiftOrderId(message.getGiftOrderId());
            request.setBatch(message.getAmount());
            request.setSendTime(message.getSendTime());
            request.setUserInfo(userInfo);
            request.setPaymentVersion(PAYMENT_VERSION);
```

```
                this.userDrawService.giftSend(request);
                LOG.info("DRAW BY GIFT SEND WITH SANTA CHECKOUT, MQ-MESSAGE={}, REQUEST=
{}", JSON.toJSONString(message),
                        JSON.toJSONString(request));
            } catch (Exception e) {
                LOG.error("GIFT-SEND-QUEUE CONSUME FATAL ERROR ON PARSING, DROPPED, MSG=
{}, EXCEPTION={}, EXCEPTION-MESSAGE={}", JSON.toJSONString(msgs),
e.getClass().getSimpleName(), e.getMessage(), e);
            }
            return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
        }
```

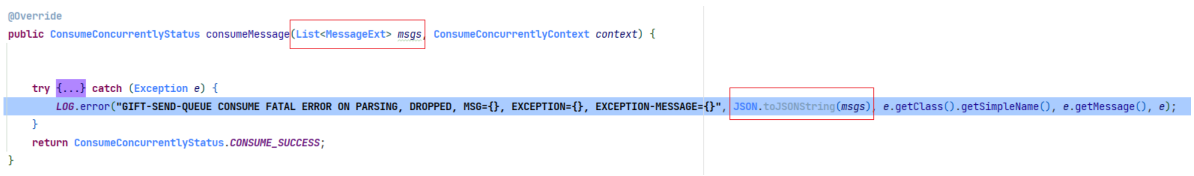
MQ消费者中有try catch块，核账的时候未详细的核对数据。其实mq已经发生重试，重试发生在有表的时候，执行逻辑正常，保证了数据完整性。

思考：分析过程不代表事实，所有故障，一般都发生在 ‘自以为的不科学‘，事实却从不说谎。

解决方案：建立行之有效的核账机制，用管理和制度的维度，杜绝不规范操作， 制度正在建立中

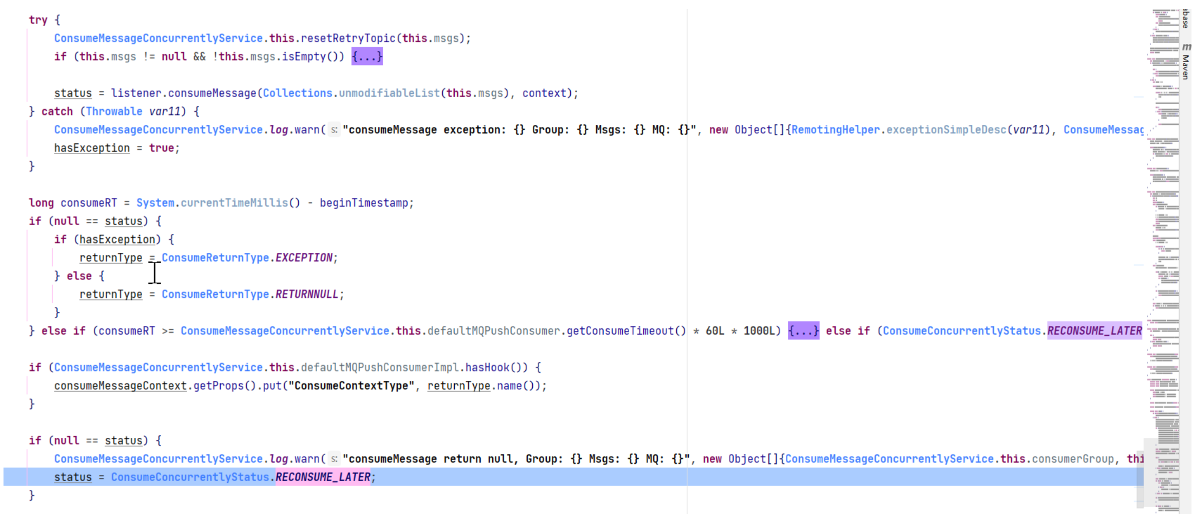
问题分析

上面的代码，问题出现在



```
@Override
public ConsumeConcurrentlyStatus consumeMessage(List<MessageExt> msgs, ConsumeConcurrentlyContext context) {


    try {...} catch (Exception e) {
        LOG.error("GIFT-SEND-QUEUE CONSUME FATAL ERROR ON PARSING, DROPPED, MSG={}, EXCEPTION={}, EXCEPTION-MESSAGE={}", JSON.toJSONString(msgs), e.getClass().getSimpleName(), e.getMessage(), e);
    }
    return ConsumeConcurrentlyStatus.CONSUME_SUCCESS;
}
```

图中红色方框当中，用的序列化方式为 fastjson，此行代码会抛出异常，导致消费失败，进入重试队列，且没有任何业务日志输出。MQ源码如下：



```
try {
    ConsumeMessageConcurrentlyService.this.resetRetryTopic(this.msgs);
    if (this.msgs != null && !this.msgs.isEmpty()) {...}

    status = listener.consumeMessage(Collections.unmodifiableList(this.msgs), context);
} catch (Throwable var11) {
    ConsumeMessageConcurrentlyService.log.warn( s: "consumeMessage exception: {} Group: {} Msgs: {} MQ: {}", new Object[]{RemotingHelper.exceptionSimpleDesc(var11), ConsumeMessag
    hasException = true;
}

long consumeRT = System.currentTimeMillis() - beginTimestamp;
if (null == status) {
    if (hasException) {
        returnType = ConsumeReturnType.EXCEPTION;
    } else {
        returnType = ConsumeReturnType.RETURNNULL;
    }
} else if (consumeRT >= ConsumeMessageConcurrentlyService.this.defaultMQPushConsumer.getConsumeTimeout() * 60L * 1000L) {...} else if (ConsumeConcurrentlyStatus.RECONSUME_LATER

if (ConsumeMessageConcurrentlyService.this.defaultMQPushConsumerImpl.hasHook()) {
    consumeMessageContext.getProps().put("ConsumeContextType", returnType.name());
}

if (null == status) {
    ConsumeMessageConcurrentlyService.log.warn( s: "consumeMessage return null, Group: {} Msgs: {} MQ: {}", new Object[]{ConsumeMessageConcurrentlyService.this.consumerGroup, thi
    status = ConsumeConcurrentlyStatus.RECONSUME_LATER;
}
```

如果异常，返回 ConsumeConcurrentlyStatus.RECONSUME_LATER;

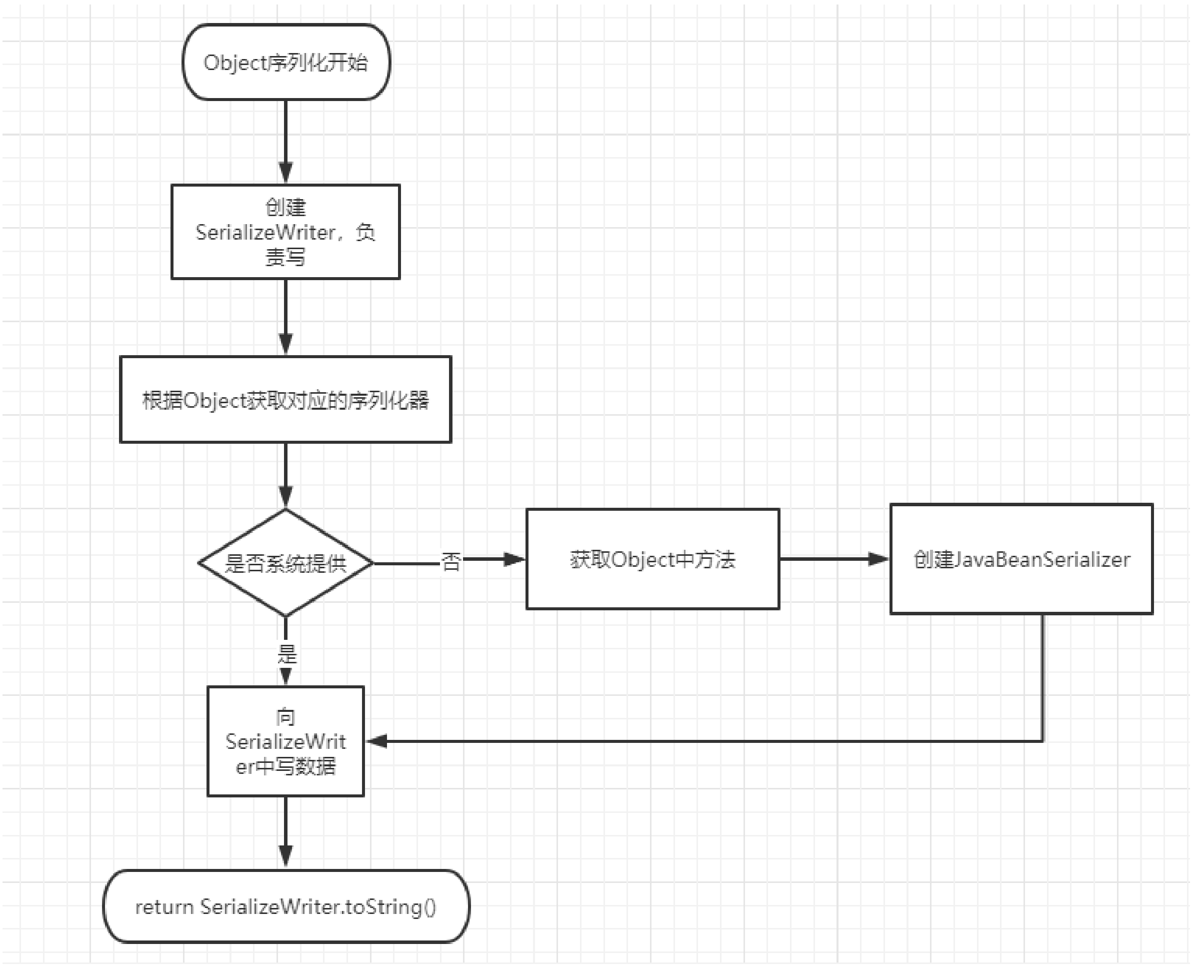
结论：无论是kafka，还是RocketMq，消费者方法参数中的MessageExt对象不能被 fastjson默认的方式序列化

原因：

环境：福袋项目采用1.2.31 (最新版本1.2.73)

接下来，我们分析下fastjson序列化的完整过程

fastjson反序列化的方式默认为采用 get方法、is方法作为序列化属性 字段的,序列化流程如下：


Object序列化开始
创建SerializeWriter，负责写
根据Object获取对应的序列化器
是否系统提供
否
是
获取Object中方法
创建JavaBeanSerializer
向SerializeWriter中写数据
return SerializeWriter.toString()

其中：在获取对象序列化的时候，MessageExt中有返回 ByteBuffer的get方法，代码如下：

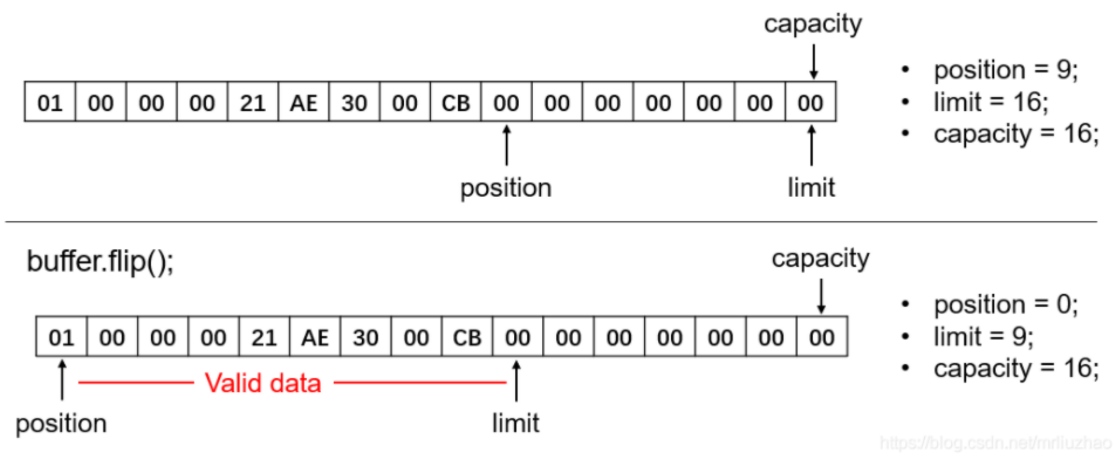
```
    public ByteBuffer getStoreHostBytes() {
        return socketAddress2ByteBuffer(this.storeHost);
    }

    //socketAddress2ByteBuffer
    public static ByteBuffer socketAddress2ByteBuffer(SocketAddress socketAddress) {
        ByteBuffer byteBuffer = ByteBuffer.allocate(8);
        return socketAddress2ByteBuffer(socketAddress, byteBuffer);
    }
    //socketAddress2ByteBuffer
    public static ByteBuffer socketAddress2ByteBuffer(SocketAddress socketAddress,
ByteBuffer byteBuffer) {
        InetSocketAddress inetSocketAddress = (InetSocketAddress)socketAddress;
        byteBuffer.put(inetSocketAddress.getAddress().getAddress(), 0, 4);
        byteBuffer.putInt(inetSocketAddress.getPort());
        byteBuffer.flip();
        return byteBuffer;
    }
```

Mq消息在接收到消息时，构造了返回了ByteBuffer对象的方法，该方法是nio中设计用于保存数据到缓冲区的目的。

主要的属性如下：

- position: 其实是指从buffer读取或写入buffer的下一个元素位置。比如，已经写入buffer 3个元素那那么position就是指向第4个位置，即position设置为3（数组从0开始计）。
- limit：还有多少数据需要从buffer中取出，或还有多少空间可以放入。postition总是<=limit。
- capacity: 表示buffer本身底层数组的容量。limit绝不能>capacity。

  数据结构如下：



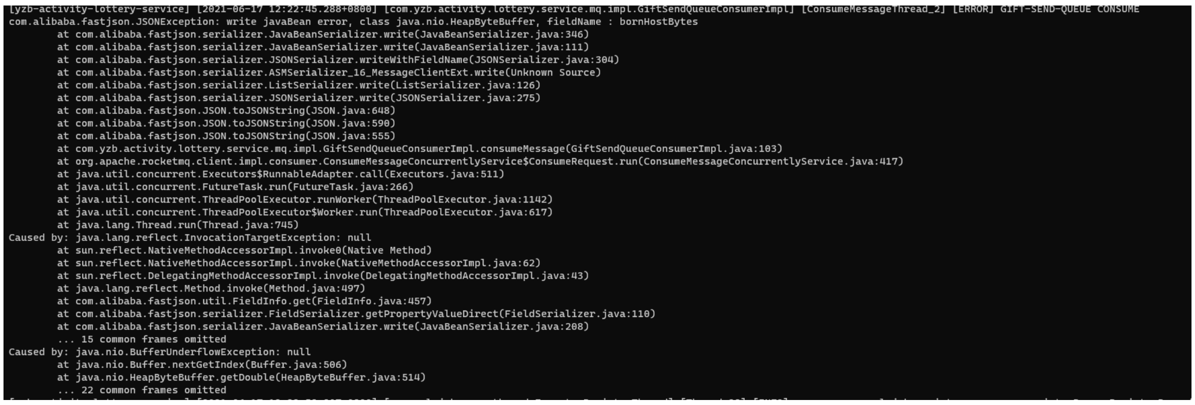
☐ get()方法，一字节一字节读

☐ getChar()、getShort()、getInt()、getFloat()、getLong()、getDouble()读取相应字节数的数据

至此：问题显而易见，fastjson在1.2.31及之前，没有提供ByteBuffer 序列化器，所以用了默认的javabean序列化器，而默认的javabean序列化器，又通过get方法反序列化，当遇见ByteBuffer时，ByteBuffer中会先遇到如下方法，getLong(),

```
    public long getLong() {
        return Bits.getLong(this, ix(nextGetIndex(8)), bigEndian);
    }
    //nextGetIndex

    final int nextGetIndex(int nb) {                    // package-private
        if (limit - position < nb)
            throw new BufferUnderflowException();
        int p = position;
        position += nb;
        return p;
    }
```

每次读取position偏移8个字节，而MessageExt中，构建的ByteBuffer存储的时4个字节，所以会报错，完整的堆栈如下：



```
[yzb-activity-lottery-service] [2021-06-17 12:22:45.288+0800] [com.yzb.activity.lottery.service.mq.impl.GiftSendQueueConsumerImpl] [ConsumeMessageThread_2] [ERROR] GIFT-SEND-QUEUE CONSUME
com.alibaba.fastjson.JSONException: write javaBean error, class java.nio.HeapByteBuffer, fieldName : bornHostBytes
        at com.alibaba.fastjson.serializer.JavaBeanSerializer.write(JavaBeanSerializer.java:346)
        at com.alibaba.fastjson.serializer.JavaBeanSerializer.write(JavaBeanSerializer.java:111)
        at com.alibaba.fastjson.serializer.JSONSerializer.writeWithFieldName(JSONSerializer.java:304)
        at com.alibaba.fastjson.serializer.ASMSerializer_16_MessageClientExt.write(Unknown Source)
        at com.alibaba.fastjson.serializer.ListSerializer.write(ListSerializer.java:126)
        at com.alibaba.fastjson.serializer.JSONSerializer.write(JSONSerializer.java:275)
        at com.alibaba.fastjson.JSON.toJSONString(JSON.java:648)
        at com.alibaba.fastjson.JSON.toJSONString(JSON.java:590)
        at com.alibaba.fastjson.JSON.toJSONString(JSON.java:555)
        at com.yzb.activity.lottery.service.mq.impl.GiftSendQueueConsumerImpl.consumeMessage(GiftSendQueueConsumerImpl.java:103)
        at org.apache.rocketmq.client.impl.consumer.ConsumeMessageConcurrentlyService$ConsumeRequest.run(ConsumeMessageConcurrentlyService.java:417)
        at java.util.concurrent.Executors$RunnableAdapter.call(Executors.java:511)
        at java.util.concurrent.FutureTask.run(FutureTask.java:266)
        at java.util.concurrent.ThreadPoolExecutor.runWorker(ThreadPoolExecutor.java:1142)
        at java.util.concurrent.ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617)
        at java.lang.Thread.run(Thread.java:745)
Caused by: java.lang.reflect.InvocationTargetException: null
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
        at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:62)
        at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
        at java.lang.reflect.Method.invoke(Method.java:497)
        at com.alibaba.fastjson.util.FieldInfo.get(FieldInfo.java:457)
        at com.alibaba.fastjson.serializer.FieldSerializer.getPropertyValueDirect(FieldSerializer.java:110)
        at com.alibaba.fastjson.serializer.JavaBeanSerializer.write(JavaBeanSerializer.java:208)
        ... 15 common frames omitted
Caused by: java.nio.BufferUnderflowException: null
        at java.nio.Buffer.nextGetIndex(Buffer.java:506)
        at java.nio.HeapByteBuffer.getDouble(HeapByteBuffer.java:514)
        ... 22 common frames omitted
```

以下内容是fastjson序列化的过程，仅供参考：

1-JSON.toJSONString方法

```
    public static String toJSONString(Object object, int defaultFeatures,
SerializerFeature... features) {
        //写数据的类，存储序列化过程的数据，最后通过 out.toString()转化为json字符串
        SerializeWriter out = new SerializeWriter((Writer)null, defaultFeatures,
features);

        String var5;
        try {
            //Json序列化解析对象的类，解析过程中向out写入数据
            JSONSerializer serializer = new JSONSerializer(out);
            //解析传入的对象，保存在out中
            serializer.write(object);
            //将解析的结果转成String输出
            var5 = out.toString();
        } finally {
            out.close();
        }

        return var5;
    }
```

2-JSONSerializer#write(java.lang.Object)方法

```
    public final void write(Object object) {
        if (object == null) {
            this.out.writeNull();
        } else {
            Class<?> clazz = object.getClass();
            //获取序列化器
            ObjectSerializer writer = this.getObjectWriter(clazz);

            try {
                writer.write(this, object, (Object)null, (Type)null, 0);
            } catch (IOException var5) {
                throw new JSONException(var5.getMessage(), var5);
            }
        }
    }
```

3-SerializeConfig#getObjectWriter(java.lang.Class<?>, boolean)方法

获取对应的序列化器

```
private ObjectSerializer getObjectWriter(Class<?> clazz, boolean create) {
        ObjectSerializer writer = (ObjectSerializer)this.serializers.get(clazz);
        ClassLoader classLoader;
        Iterator var5;
        Object o;
        AutowiredObjectSerializer autowired;
        Iterator var8;
        Type forType;
        if (writer == null) {
            try {
                classLoader = Thread.currentThread().getContextClassLoader();
                var5 = ServiceLoader.load(AutowiredObjectSerializer.class,
classLoader).iterator();

                label254:
                while(true) {
                    do {
                        if (!var5.hasNext()) {
                            break label254;
                        }

                        o = var5.next();
                    } while(!(o instanceof AutowiredObjectSerializer));

                    autowired = (AutowiredObjectSerializer)o;
                    var8 = autowired.getAutowiredFor().iterator();

                    while(var8.hasNext()) {
                        forType = (Type)var8.next();
                        this.put((Type)forType, (ObjectSerializer)autowired);
                    }
                }
            } catch (ClassCastException var17) {
            }
            writer = (ObjectSerializer)this.serializers.get(clazz);
        }

        if (writer == null) {
            classLoader = JSON.class.getClassLoader();
            if (classLoader != Thread.currentThread().getContextClassLoader()) {
                try {
                    var5 = ServiceLoader.load(AutowiredObjectSerializer.class, 
classLoader).iterator();

                    label235:
                    while(true) {
                        do {
                            if (!var5.hasNext()) {
                                break label235;
                            }

                            o = var5.next();
                        } while(!(o instanceof AutowiredObjectSerializer));

                        autowired = (AutowiredObjectSerializer)o;
                        var8 = autowired.getAutowiredFor().iterator();

                        while(var8.hasNext()) {
                            forType = (Type)var8.next();
                            this.put((Type)forType, (ObjectSerializer)autowired);
                        }
                    }
                } catch (ClassCastException var16) {
                }

                writer = (ObjectSerializer)this.serializers.get(clazz);
            }
        }

        if (writer == null) {
            if (Map.class.isAssignableFrom(clazz)) {
                this.put((Type)clazz, (ObjectSerializer)MapSerializer.instance);
            } else if (List.class.isAssignableFrom(clazz)) {
                this.put((Type)clazz, (ObjectSerializer)ListSerializer.instance);
            } else if (Collection.class.isAssignableFrom(clazz)) {
                this.put((Type)clazz, (ObjectSerializer)CollectionCodec.instance);
            } else if (Date.class.isAssignableFrom(clazz)) {
                this.put((Type)clazz, (ObjectSerializer)DateCodec.instance);
            } else if (JSONAware.class.isAssignableFrom(clazz)) {
                this.put((Type)clazz, (ObjectSerializer)JSONAwareSerializer.instance);
            } else if (JSONSerializable.class.isAssignableFrom(clazz)) {
                this.put((Type)clazz, 
(ObjectSerializer)JSONSerializableSerializer.instance);
            } else if (JSONStreamAware.class.isAssignableFrom(clazz)) {
                this.put((Type)clazz, (ObjectSerializer)MiscCodec.instance);
            } else if (clazz.isEnum() || clazz.getSuperclass() != null && 
clazz.getSuperclass().isEnum()) {
                JSONType jsonType = (JSONType)clazz.getAnnotation(JSONType.class);
                if (jsonType != null && jsonType.serializeEnumAsJavaBean()) {
                    this.put((Type)clazz, 
(ObjectSerializer)this.createJavaBeanSerializer(clazz));
```

```
                } else {
                    this.put((Type)clazz, (ObjectSerializer)EnumSerializer.instance);
                }
            } else if (clazz.isArray()) {
                Class<?> componentType = clazz.getComponentType();
                ObjectSerializer compObjectSerializer =
this.getObjectWriter(componentType);
                this.put((Type)clazz, (ObjectSerializer)(new
ArraySerializer(componentType, compObjectSerializer)));
            } else if (Throwable.class.isAssignableFrom(clazz)) {
                SerializeBeanInfo beanInfo = TypeUtils.buildBeanInfo(clazz, (Map)null,
this.propertyNamingStrategy);
                beanInfo.features |= SerializerFeature.WriteClassName.mask;
                this.put((Type)clazz, (ObjectSerializer)(new
JavaBeanSerializer(beanInfo)));
            } else if (!TimeZone.class.isAssignableFrom(clazz) &&
!Entry.class.isAssignableFrom(clazz)) {
                if (Appendable.class.isAssignableFrom(clazz)) {
                    this.put((Type)clazz,
(ObjectSerializer)AppendableSerializer.instance);
                } else if (Charset.class.isAssignableFrom(clazz)) {
                    this.put((Type)clazz,
(ObjectSerializer)ToStringSerializer.instance);
                } else if (Enumeration.class.isAssignableFrom(clazz)) {
                    this.put((Type)clazz,
(ObjectSerializer)EnumerationSerializer.instance);
                } else if (!Calendar.class.isAssignableFrom(clazz) &&
!XMLGregorianCalendar.class.isAssignableFrom(clazz)) {
                    if (Clob.class.isAssignableFrom(clazz)) {
                        this.put((Type)clazz,
(ObjectSerializer)ClobSeriliazer.instance);
                    } else if (TypeUtils.isPath(clazz)) {
                        this.put((Type)clazz,
(ObjectSerializer)ToStringSerializer.instance);
                    } else if (Iterator.class.isAssignableFrom(clazz)) {
                        this.put((Type)clazz, (ObjectSerializer)MiscCodec.instance);
                    } else {
                        String className = clazz.getName();
                        if (className.startsWith("java.awt.") &&
AwtCodec.support(clazz)) {
                            if (!awtError) {
                                try {
                                    this.put((Type)Class.forName("java.awt.Color"),
(ObjectSerializer)AwtCodec.instance);
                                    this.put((Type)Class.forName("java.awt.Font"),
(ObjectSerializer)AwtCodec.instance);
                                    this.put((Type)Class.forName("java.awt.Point"),
(ObjectSerializer)AwtCodec.instance);

 this.put((Type)Class.forName("java.awt.Rectangle"),
(ObjectSerializer)AwtCodec.instance);
                                } catch (Throwable var10) {
                                    awtError = true;
                                }
                            }

                            return AwtCodec.instance;
                        }
                        if (!jdk8Error && (className.startsWith("java.time.") ||
className.startsWith("java.util.Optional") ||
className.equals("java.util.concurrent.atomic.LongAdder") ||
className.equals("java.util.concurrent.atomic.DoubleAdder"))) {
                            try {

 this.put((Type)Class.forName("java.time.LocalDateTime"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.LocalDate"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.LocalTime"),
(ObjectSerializer)Jdk8DateCodec.instance);

 this.put((Type)Class.forName("java.time.ZonedDateTime"),
(ObjectSerializer)Jdk8DateCodec.instance);

 this.put((Type)Class.forName("java.time.OffsetDateTime"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.OffsetTime"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.ZoneOffset"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.ZoneRegion"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.Period"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.Duration"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.time.Instant"),
(ObjectSerializer)Jdk8DateCodec.instance);
                                this.put((Type)Class.forName("java.util.Optional"),
(ObjectSerializer)OptionalCodec.instance);

 this.put((Type)Class.forName("java.util.OptionalDouble"),
(ObjectSerializer)OptionalCodec.instance);
                                this.put((Type)Class.forName("java.util.OptionalInt"),
(ObjectSerializer)OptionalCodec.instance);

 this.put((Type)Class.forName("java.util.OptionalLong"),
(ObjectSerializer)OptionalCodec.instance);

 this.put((Type)Class.forName("java.util.concurrent.atomic.LongAdder"),
(ObjectSerializer)AdderSerializer.instance);

 this.put((Type)Class.forName("java.util.concurrent.atomic.DoubleAdder"),
(ObjectSerializer)AdderSerializer.instance);
                                writer =
(ObjectSerializer)this.serializers.get(clazz);
                                if (writer != null) {
                                    return writer;
                                }
                            } catch (Throwable var15) {
                                jdk8Error = true;
                            }
                        }

                        if (!oracleJdbcError && className.startsWith("oracle.sql.")) {
                            try {
                                this.put((Type)Class.forName("oracle.sql.DATE"),
(ObjectSerializer)DateCodec.instance);
                                this.put((Type)Class.forName("oracle.sql.TIMESTAMP"),
(ObjectSerializer)DateCodec.instance);
                                writer =
(ObjectSerializer)this.serializers.get(clazz);
                                if (writer != null) {
                                    return writer;
                                }
                            } catch (Throwable var14) {
                                oracleJdbcError = true;
                            }
                        }

                        if (!springfoxError &&
className.equals("springfox.documentation.spring.web.json.Json")) {
                            try {

 this.put((Type)Class.forName("springfox.documentation.spring.web.json.Json"),
(ObjectSerializer)SwaggerJsonSerializer.instance);
                                writer =
(ObjectSerializer)this.serializers.get(clazz);
                                if (writer != null) {
                                    return writer;
                                }
                            } catch (ClassNotFoundException var13) {
                                springfoxError = true;
                            }
                        }

                        if (!guavaError &&
className.startsWith("com.google.common.collect.")) {
                            try {

 this.put((Type)Class.forName("com.google.common.collect.HashMultimap"),
(ObjectSerializer)GuavaCodec.instance);

 this.put((Type)Class.forName("com.google.common.collect.LinkedListMultimap"),
(ObjectSerializer)GuavaCodec.instance);

 this.put((Type)Class.forName("com.google.common.collect.ArrayListMultimap"),
(ObjectSerializer)GuavaCodec.instance);

 this.put((Type)Class.forName("com.google.common.collect.TreeMultimap"),
(ObjectSerializer)GuavaCodec.instance);
                                writer =
(ObjectSerializer)this.serializers.get(clazz);
                                if (writer != null) {
                                    return writer;
                                }
                            } catch (ClassNotFoundException var12) {
                                guavaError = true;
                            }
                        }

                        if (className.equals("net.sf.json.JSONNull")) {
                            try {
```

```
                                this.put((Type)Class.forName("net.sf.json.JSONNull"),
(ObjectSerializer)MiscCodec.instance);
                            } catch (ClassNotFoundException var11) {
                            }

                            writer = (ObjectSerializer)this.serializers.get(clazz);
                            if (writer != null) {
                                return writer;
                            }
                        }

                        if (TypeUtils.isProxy(clazz)) {
                            Class<?> superClazz = clazz.getSuperclass();
                            ObjectSerializer superWriter =
this.getObjectWriter(superClazz);
                            this.put((Type)clazz, (ObjectSerializer)superWriter);
                            return superWriter;
                        }

                        if (create) {
                            //默认返回 JavaBeanSerializer
                            this.put((Type)clazz,
(ObjectSerializer)this.createJavaBeanSerializer(clazz));
                        }
                    }
                } else {
                    this.put((Type)clazz, (ObjectSerializer)CalendarCodec.instance);
                }
            } else {
                this.put((Type)clazz, (ObjectSerializer)MiscCodec.instance);
            }

            writer = (ObjectSerializer)this.serializers.get(clazz);
        }

        return writer;
    }
```

4-SerializeConfig#createJavaBeanSerializer(java.lang.Class<?>)方法

```
    private final ObjectSerializer createJavaBeanSerializer(Class<?> clazz) {
        //构造SerializeBeanInfo 对象，里面存储序列化的字段等信息
        SerializeBeanInfo beanInfo = TypeUtils.buildBeanInfo(clazz, (Map)null,
this.propertyNamingStrategy, this.fieldBase);
        return (ObjectSerializer)(beanInfo.fields.length == 0 &&
Iterable.class.isAssignableFrom(clazz) ? MiscCodec.instance :
this.createJavaBeanSerializer(beanInfo));
    }
```

5-TypeUtils#buildBeanInfo(java.lang.Class<?>, java.util.Map<java.lang.String,java.lang.String>, com.alibaba.fastjson.PropertyNamingStrategy, boolean)方法

```
public static SerializeBeanInfo buildBeanInfo(Class<?> beanType, Map<String, String>
aliasMap, PropertyNamingStrategy propertyNamingStrategy, boolean fieldBased) {
        JSONType jsonType = (JSONType)beanType.getAnnotation(JSONType.class);
        Map<String, Field> fieldCacheMap = new HashMap();
        ParserConfig.parserAllFieldToCache(beanType, fieldCacheMap);
        //fieldBased==false，执行computeGetters(beanType, jsonType, aliasMap,
fieldCacheMap, false, propertyNamingStrategy) 逻辑
        List<FieldInfo> fieldInfoList = fieldBased ?
computeGettersWithFieldBase(beanType, aliasMap, false, propertyNamingStrategy) :
computeGetters(beanType, jsonType, aliasMap, fieldCacheMap, false,
propertyNamingStrategy);
        FieldInfo[] fields = new FieldInfo[fieldInfoList.size()];
        fieldInfoList.toArray(fields);
        String[] orders = null;
        String typeName = null;
        int features;
        if (jsonType != null) {
            orders = jsonType.orders();
            typeName = jsonType.typeName();
            if (typeName.length() == 0) {
                typeName = null;
            }

            features = SerializerFeature.of(jsonType.serialzeFeatures());
        } else {
            features = 0;
        }

        Object sortedFieldList;
        if (orders != null && orders.length != 0) {
            sortedFieldList = fieldBased ? computeGettersWithFieldBase(beanType,
aliasMap, true, propertyNamingStrategy) : computeGetters(beanType, jsonType, aliasMap,
fieldCacheMap, true, propertyNamingStrategy);
        } else {
            sortedFieldList = new ArrayList(fieldInfoList);
            Collections.sort((List)sortedFieldList);
        }

        FieldInfo[] sortedFields = new FieldInfo[((List)sortedFieldList).size()];
        ((List)sortedFieldList).toArray(sortedFields);
        if (Arrays.equals(sortedFields, fields)) {
            sortedFields = fields;
        }

        return new SerializeBeanInfo(beanType, jsonType, typeName, features, fields,
sortedFields);
    }
```

6-TypeUtils#computeGetters(java.lang.Class<?>, com.alibaba.fastjson.annotation.JSONType, java.util.Map<java.lang.String,java.lang.String>, java.util.Map<java.lang.String,java.lang.reflect.Field>, boolean, com.alibaba.fastjson.PropertyNamingStrategy)方法

```
public static List<FieldInfo> computeGetters(Class<?> clazz, JSONType jsonType,
Map<String, String> aliasMap, Map<String, Field> fieldCacheMap, boolean sorted,
PropertyNamingStrategy propertyNamingStrategy) {
        Map<String, FieldInfo> fieldInfoMap = new LinkedHashMap();
        //反射获取所有方法
        Method[] var7 = clazz.getMethods();
        int var8 = var7.length;

        for(int var9 = 0; var9 < var8; ++var9) {
            Method method = var7[var9];
            String methodName = method.getName();
            int ordinal = 0;
            int serialzeFeatures = 0;
            int parserFeatures = 0;
            String label = null;
            //静态方法等过滤
            if (!Modifier.isStatic(method.getModifiers()) &&
!method.getReturnType().equals(Void.TYPE) && method.getParameterTypes().length == 0 &&
method.getReturnType() != ClassLoader.class &&
(!method.getName().equals("getMetaClass") ||
!method.getReturnType().getName().equals("groovy.lang.MetaClass"))) {
                JSONField annotation =
(JSONField)method.getAnnotation(JSONField.class);
                if (annotation == null) {
                    annotation = getSuperMethodAnnotation(clazz, method);
                }

                if (annotation != null) {
                    if (!annotation.serialize()) {
                        continue;
                    }

                    ordinal = annotation.ordinal();
                    serialzeFeatures =
SerializerFeature.of(annotation.serialzeFeatures());
                    parserFeatures = Feature.of(annotation.parseFeatures());
                    if (annotation.name().length() != 0) {
                        String propertyName = annotation.name();
                        if (aliasMap != null) {
                            propertyName = (String)aliasMap.get(propertyName);
                            if (propertyName == null) {
                                continue;
                            }
                        }

                        FieldInfo fieldInfo = new FieldInfo(propertyName, method,
(Field)null, clazz, (Type)null, ordinal, serialzeFeatures, parserFeatures, annotation,
(JSONField)null, label);
                        fieldInfoMap.put(propertyName, fieldInfo);
                        continue;
                    }

                    if (annotation.label().length() != 0) {
                        label = annotation.label();
                    }
                }

                char c2;
                String propertyName;
                //get方法
                if (methodName.startsWith("get")) {
                    if (methodName.length() < 4 || methodName.equals("getClass") ||
methodName.equals("getDeclaringClass") && clazz.isEnum()) {
                        continue;
                    }

                    c2 = methodName.charAt(3);
                    if (!Character.isUpperCase(c2) && c2 <= 512) {
                        if (c2 == '_') {
                            propertyName = methodName.substring(4);
                        } else if (c2 == 'f') {
                            propertyName = methodName.substring(3);
                        } else {
                            if (methodName.length() < 5 ||
!Character.isUpperCase(methodName.charAt(4))) {
                                continue;
                            }

                            propertyName = decapitalize(methodName.substring(3));
                        }
                    } else {
                        if (compatibleWithJavaBean) {
                            propertyName = decapitalize(methodName.substring(3));
                        } else {
                            propertyName = Character.toLowerCase(methodName.charAt(3))
+ methodName.substring(4);
                        }

                        propertyName =
getPropertyNameByCompatibleFieldName(fieldCacheMap, methodName, propertyName, 3);
                    }

                    boolean ignore = isJSONTypeIgnore(clazz, propertyName);
                    if (ignore) {
                        continue;
                    }

                    Field field = ParserConfig.getFieldFromCache(propertyName,
fieldCacheMap);
                    if (field == null && propertyName.length() > 1) {
                        char ch = propertyName.charAt(1);
                        if (ch >= 'A' && ch <= 'Z') {
                            String javaBeanCompatiblePropertyName =
decapitalize(methodName.substring(3));
                            field =
ParserConfig.getFieldFromCache(javaBeanCompatiblePropertyName, fieldCacheMap);
                        }
                    }

                    JSONField fieldAnnotation = null;
                    if (field != null) {
                        fieldAnnotation =
(JSONField)field.getAnnotation(JSONField.class);
                        if (fieldAnnotation != null) {
                            if (!fieldAnnotation.serialize()) {
                                continue;
                            }

                            ordinal = fieldAnnotation.ordinal();
                            serialzeFeatures =
SerializerFeature.of(fieldAnnotation.serialzeFeatures());
                            parserFeatures =
Feature.of(fieldAnnotation.parseFeatures());
                            if (fieldAnnotation.name().length() != 0) {
                                propertyName = fieldAnnotation.name();
                                if (aliasMap != null) {
                                    propertyName = (String)aliasMap.get(propertyName);
                                    if (propertyName == null) {
                                        continue;
                                    }
                                }
                            }

                            if (fieldAnnotation.label().length() != 0) {
                                label = fieldAnnotation.label();
                            }
                        }
                    }

                    if (aliasMap != null) {
                        propertyName = (String)aliasMap.get(propertyName);
                        if (propertyName == null) {
                            continue;
                        }
                    }

                    if (propertyNamingStrategy != null) {
                        propertyName = propertyNamingStrategy.translate(propertyName);
                    }

                    FieldInfo fieldInfo = new FieldInfo(propertyName, method, field,
clazz, (Type)null, ordinal, serialzeFeatures, parserFeatures, annotation,
fieldAnnotation, label);
                    fieldInfoMap.put(propertyName, fieldInfo);
                }
                //返回boolean的is开头的方法
                if (methodName.startsWith("is") && methodName.length() >= 3 &&
(method.getReturnType() == Boolean.TYPE || method.getReturnType() == Boolean.class)) {
                    c2 = methodName.charAt(2);
                    if (Character.isUpperCase(c2)) {
                        if (compatibleWithJavaBean) {
                            propertyName = decapitalize(methodName.substring(2));
                        } else {
                            propertyName = Character.toLowerCase(methodName.charAt(2))
+ methodName.substring(3);
                        }

                        propertyName =
getPropertyNameByCompatibleFieldName(fieldCacheMap, methodName, propertyName, 2);
                    } else if (c2 == '_') {
                        propertyName = methodName.substring(3);
                    } else {
                        if (c2 != 'f') {
                            continue;
                        }

                        propertyName = methodName.substring(2);
                    }

                    Field field = ParserConfig.getFieldFromCache(propertyName,
fieldCacheMap);
                    if (field == null) {
                        field = ParserConfig.getFieldFromCache(methodName,
fieldCacheMap);
                    }

                    JSONField fieldAnnotation = null;
                    if (field != null) {
                        fieldAnnotation =
(JSONField)field.getAnnotation(JSONField.class);
                        if (fieldAnnotation != null) {
                            if (!fieldAnnotation.serialize()) {
                                continue;
                            }

                            ordinal = fieldAnnotation.ordinal();
                            serialzeFeatures =
SerializerFeature.of(fieldAnnotation.serialzeFeatures());
                            parserFeatures =
Feature.of(fieldAnnotation.parseFeatures());
                            if (fieldAnnotation.name().length() != 0) {
                                propertyName = fieldAnnotation.name();
                                if (aliasMap != null) {
                                    propertyName = (String)aliasMap.get(propertyName);
                                    if (propertyName == null) {
                                        continue;
                                    }
                                }
                            }

                            if (fieldAnnotation.label().length() != 0) {
                                label = fieldAnnotation.label();
                            }
                        }
                    }

                    if (aliasMap != null) {
                        propertyName = (String)aliasMap.get(propertyName);
                        if (propertyName == null) {
                            continue;
                        }
                    }

                    if (propertyNamingStrategy != null) {
                        propertyName = propertyNamingStrategy.translate(propertyName);
                    }

                    if (!fieldInfoMap.containsKey(propertyName)) {
                        FieldInfo fieldInfo = new FieldInfo(propertyName, method,
field, clazz, (Type)null, ordinal, serialzeFeatures, parserFeatures, annotation,
fieldAnnotation, label);
                        fieldInfoMap.put(propertyName, fieldInfo);
                    }
                }
            }
```

```
        }

        Field[] fields = clazz.getFields();
        computeFields(clazz, aliasMap, propertyNamingStrategy, fieldInfoMap, fields);
        return getFieldInfos(clazz, sorted, fieldInfoMap);
    }
```

上面方法可证明，fastjson序列化是依赖的java方法

getXxx()

boolean isXxx()