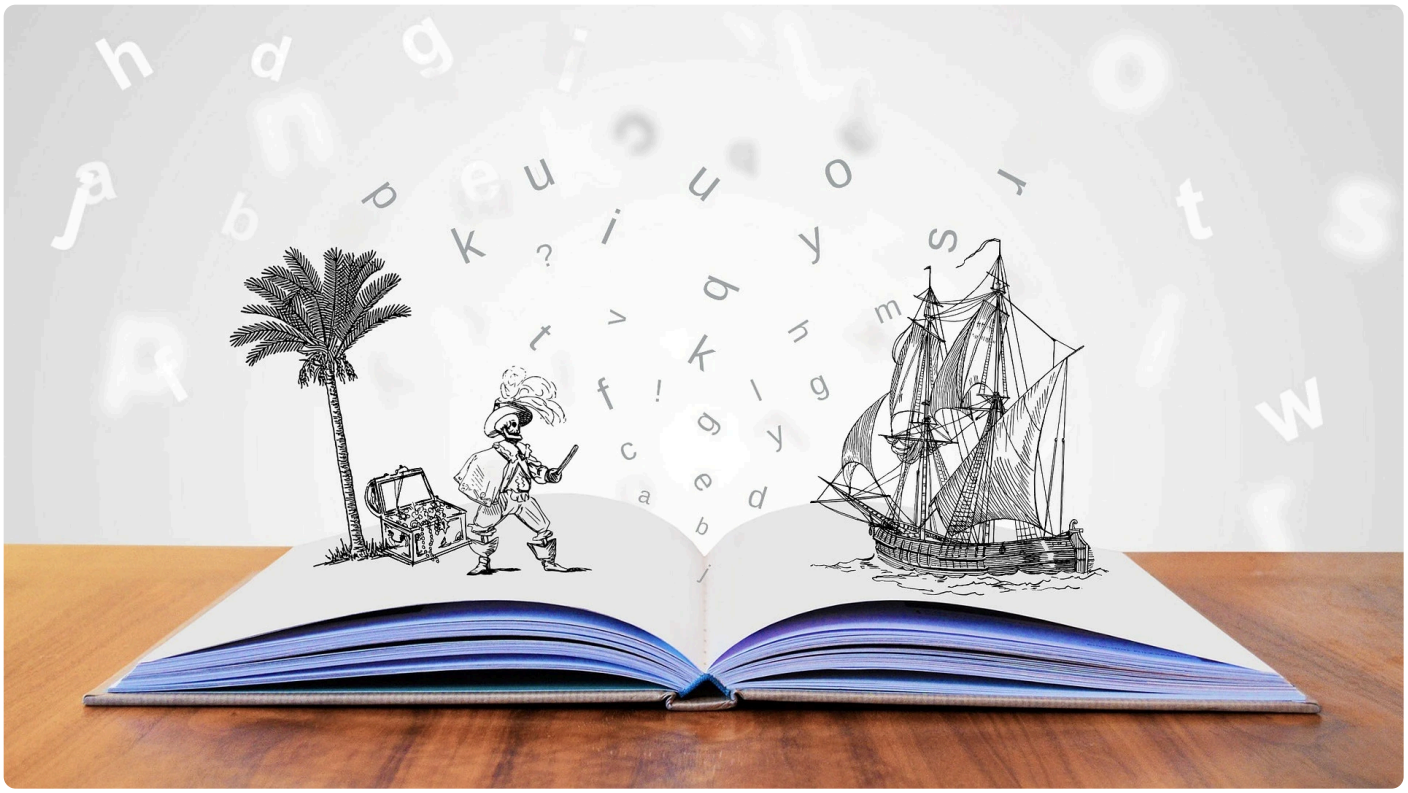
16 | 揭开神秘的“位移主题”面纱

胡夕 · Kafka核心技术与实战



你好，我是胡夕。今天我要和你分享的内容是：Kafka 中神秘的内部主题（Internal Topic）__consumer_offsets。

__consumer_offsets 在 Kafka 源码中有个更为正式的名字，叫**位移主题**，即 Offsets Topic。为了方便今天的讨论，我将统一使用位移主题来指代 __consumer_offsets。需要注意的是，它有两个下划线哦。

好了，我们开始今天的内容吧。首先，我们有必要探究一下位移主题被引入的背景及原因，即位移主题的前世今生。

在上一期中，我说过老版本 Consumer 的位移管理是依托于 Apache ZooKeeper 的，它会自动或手动地将位移数据提交到 ZooKeeper 中保存。当 Consumer 重启后，它能自动从 ZooKeeper 中读取位移数据，从而在上次消费截止的地方继续消费。这种设计使得 Kafka Broker 不需要保存位移数据，减少了 Broker 端需要持有的状态空间，因而有利于实现高伸缩性。

但是，ZooKeeper 其实并不适用于这种高频的写操作，因此，Kafka 社区自 0.8.2.x 版本开始，就在酝酿修改这种设计，并最终在新版本 Consumer 中正式推出了全新的位移管理机制，自然也包括这个新的位移主题。

新版本 Consumer 的位移管理机制其实也很简单，就是**将 Consumer 的位移数据作为一条条普通的 Kafka 消息，提交到 __consumer_offsets 中。可以这么说，__consumer_offsets 的主要作用是保存 Kafka 消费者的位移信息。**它要求这个提交过程不仅要实现高持久性，还要支持高频的写操作。显然，Kafka 的主题设计天然就满足这两个条件，因此，使用 Kafka 主题来保存位移这件事情，实际上就是一个水到渠成的想法了。

这里我想再次强调一下，和你创建的其他主题一样，位移主题就是普通的 Kafka 主题。你可以手动地创建它、修改它，甚至是删除它。只不过，它同时也是一个内部主题，大部分情况下，你其实并不需要“搭理”它，也不用花心思去管理它，把它丢给 Kafka 就完事了。

虽说位移主题是一个普通的 Kafka 主题，但**它的消息格式却是 Kafka 自己定义的**，用户不能修改，也就是说你不能随意地向这个主题写消息，因为一旦你写入的消息不满足 Kafka 规定的格式，那么 Kafka 内部无法成功解析，就会造成 Broker 的崩溃。事实上，Kafka Consumer 有 API 帮你提交位移，也就是向位移主题写消息。你千万不要自己写个 Producer 随意向该主题发送消息。

你可能会好奇，这个主题存的到底是什么格式的消息呢？所谓的消息格式，你可以简单地理解为是一个 KV 对。Key 和 Value 分别表示消息的键值和消息体，在 Kafka 中它们就是字节数组而已。想象一下，如果让你来设计这个主题，你觉得消息格式应该长什么样子呢？我先不说社区的设计方案，我们自己先来设计一下。

首先从 Key 说起。一个 Kafka 集群中的 Consumer 数量会有很多，既然这个主题保存的是 Consumer 的位移数据，那么消息格式中必须要有字段来标识这个位移数据是哪个 Consumer 的。这种数据放在哪个字段比较合适呢？显然放在 Key 中比较合适。

现在我们知道该主题消息的 Key 中应该保存标识 Consumer 的字段，那么，当前 Kafka 中什么字段能够标识 Consumer 呢？还记得之前我们说 Consumer Group 时提到的 Group ID 吗？没错，就是这个字段，它能够标识唯一的 Consumer Group。

说到这里，我再多说几句。除了 Consumer Group，Kafka 还支持独立 Consumer，也称 Standalone Consumer。它的运行机制与 Consumer Group 完全不同，但是位移管理的机制却是相同的。因此，即使是 Standalone Consumer，也有自己的 Group ID 来标识它自己，所以也适用于这套消息格式。

Okay，我们现在知道 Key 中保存了 Group ID，但是只保存 Group ID 就可以了吗？别忘了，Consumer 提交位移是在分区层面上进行的，即它提交的是某个或某些分区的位移，那么很显然，Key 中还应该保存 Consumer 要提交位移的分区。

好了，我们来总结一下我们的结论。**位移主题的 Key 中应该保存 3 部分内容：<Group ID，主题名，分区号 >**。如果你认同这样的结论，那么恭喜你，社区就是这么设计的！

接下来，我们再来看看消息体的设计。也许你会觉得消息体应该很简单，保存一个位移值就可以了。实际上，社区的方案要复杂得多，比如消息体还保存了位移提交的一些其他元数据，诸如时间戳和用户自定义的数据等。保存这些元数据是为了帮助 Kafka 执行各种各样后续的操作，比如删除过期位移消息等。但总体来说，我们还是可以简单地认为消息体就是保存了位移值。

当然了，位移主题的消息格式可不是只有这一种。事实上，它有 3 种消息格式。除了刚刚我们说的这种格式，还有 2 种格式：

1. 用于保存 Consumer Group 信息的消息。
2. 用于删除 Group 过期位移甚至是删除 Group 的消息。

第 1 种格式非常神秘，以至于你几乎无法在搜索引擎中搜到它的身影。不过，你只需要记住它是用来注册 Consumer Group 的就可以了。

第 2 种格式相对更加有名一些。它有个专属的名字：tombstone 消息，即墓碑消息，也称 delete mark。下次你在 Google 或百度中见到这些词，不用感到惊讶，它们指的是一个东西。这些消息只出现在源码中而不暴露给你。它的主要特点是它的消息体是 null，即空消息体。

那么，何时会写入这类消息呢？一旦某个 Consumer Group 下的所有 Consumer 实例都停止了，而且它们的位移数据都已被删除时，Kafka 会向位移主题的对应分区写入 tombstone 消息，表明要彻底删除这个 Group 的信息。

好了，消息格式就说这么多，下面我们来说说位移主题是怎么被创建的。通常来说，**当 Kafka 集群中的第一个 Consumer 程序启动时，Kafka 会自动创建位移主题**。我们说过，位移主题就是普通的 Kafka 主题，那么它自然也有对应的分区数。但如果是 Kafka 自动创建的，分区数是怎么设置的呢？这就要看 Broker 端参数 offsets.topic.num.partitions 的取值了。它的默认值是 50，因此 Kafka 会自动创建一个 50 分区的位移主题。如果你曾经惊讶于 Kafka 日志路径下冒出很多 __consumer_offsets-xxx 这样的目录，那么现在应该明白了吧，这就是 Kafka 自动帮你创建的位移主题啊。

你可能会问，除了分区数，副本数或备份因子是怎么控制的呢？答案也很简单，这就是 Broker 端另一个参数 offsets.topic.replication.factor 要做的事情了。它的默认值是 3。

总结一下，**如果位移主题是 Kafka 自动创建的，那么该主题的分区数是 50，副本数是 3**。

当然，你也可以选择手动创建位移主题，具体方法就是，在 Kafka 集群尚未启动任何 Consumer 之前，使用 Kafka API 创建它。手动创建的好处在于，你可以创建满足你实际场景需要的位移主题。比如很多人说 50 个分区对我来讲太多了，我不想要这么多分区，那么你可以自己创建它，不用理会 offsets.topic.num.partitions 的值。

不过我给你的建议是，还是让 Kafka 自动创建比较好。目前 Kafka 源码中有一些地方硬编码了 50 分区数，因此如果你自行创建了一个不同于默认分区数的位移主题，可能会碰到各种各样奇怪的问题。这是社区的一个 Bug，目前代码已经修复了，但依然在审核中。

创建位移主题当然是为了用的，那么什么地方会用到位移主题呢？我们前面一直在说 Kafka Consumer 提交位移时会写入该主题，那 Consumer 是怎么提交位移的呢？目前 Kafka Consumer 提交位移的方式有两种：**自动提交位移和手动提交位移。**

Consumer 端有个参数叫 enable.auto.commit，如果值是 true，则 Consumer 在后台默默地为你定期提交位移，提交间隔由一个专属的参数 auto.commit.interval.ms 来控制。自动提交

位移有一个显著的优点，就是省事，你不用操心位移提交的事情，就能保证消息消费不会丢失。但这一点同时也是缺点。因为它太省事了，以至于丧失了很大的灵活性和可控性，你完全没法把控 Consumer 端的位移管理。

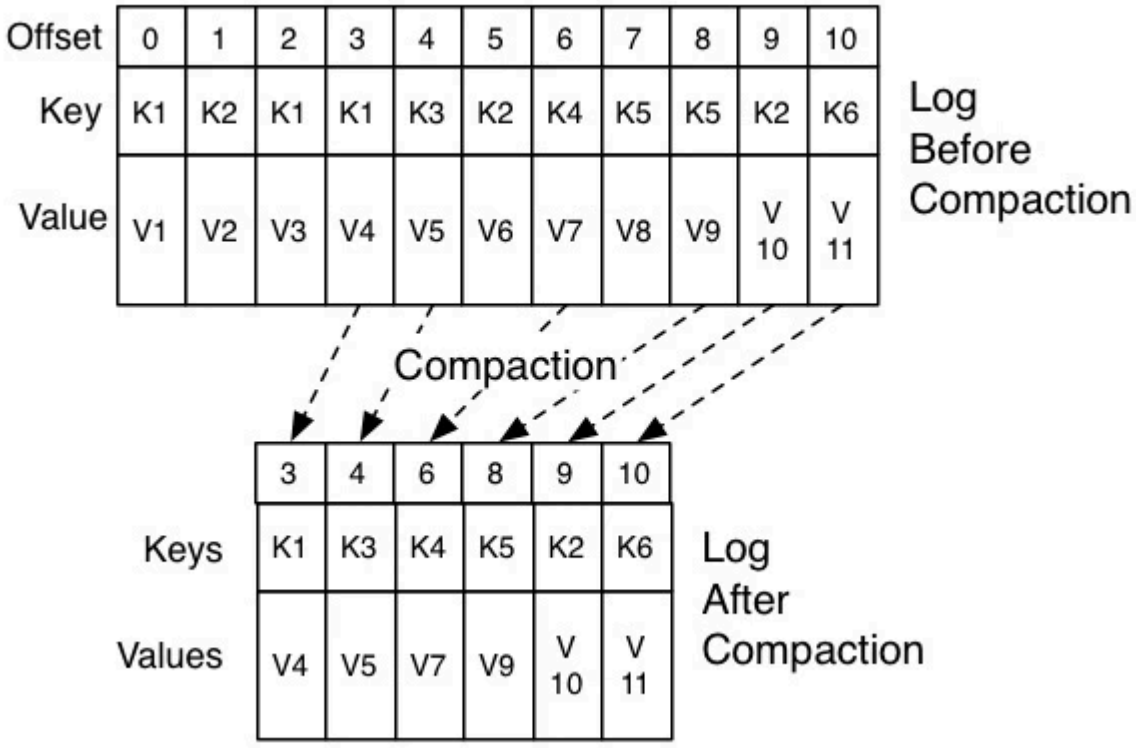
事实上，很多与 Kafka 集成的大数据框架都是禁用自动提交位移的，如 Spark、Flink 等。这就引出了另一种位移提交方式：**手动提交位移**，即设置 enable.auto.commit = false。一旦设置了 false，作为 Consumer 应用开发的你就要承担起位移提交的责任。Kafka Consumer API 为你提供了位移提交的方法，如 consumer.commitSync 等。当调用这些方法时，Kafka 会向位移主题写入相应的消息。

如果你选择的是自动提交位移，那么就可能存在一个问题：只要 Consumer 一直启动着，它就会无限期地向位移主题写入消息。

我们来举个极端一点的例子。假设 Consumer 当前消费到了某个主题的最新一条消息，位移是 100，之后该主题没有任何新消息产生，故 Consumer 无消息可消费了，所以位移永远保持在 100。由于是自动提交位移，位移主题中会不停地写入位移 =100 的消息。显然 Kafka 只需要保留这类消息中的最新一条就可以了，之前的消息都是可以删除的。这就要求 Kafka 必须要有针对位移主题消息特点的消息删除策略，否则这种消息会越来越多，最终撑爆整个磁盘。

Kafka 是怎么删除位移主题中的过期消息的呢？答案就是 Compaction。国内很多文献都将其翻译成压缩，我个人是有一点保留意见的。在英语中，压缩的专有术语是 Compression，它的原理和 Compaction 很不相同，我更倾向于翻译成压实，或干脆采用 JVM 垃圾回收中的术语：整理。

不管怎么翻译，Kafka 使用 **Compact 策略**来删除位移主题中的过期消息，避免该主题无限期膨胀。那么应该如何定义 Compact 策略中的过期呢？对于同一个 Key 的两条消息 M1 和 M2，如果 M1 的发送时间早于 M2，那么 M1 就是过期消息。Compact 的过程就是扫描日志的所有消息，剔除那些过期的消息，然后把剩下的消息整理在一起。我在这里贴一张来自官网的图片，来说明 Compact 过程。



图中位移为 0、2 和 3 的消息的 Key 都是 K1。Compact 之后，分区只需要保存位移为 3 的消息，因为它是最新发送的。

Kafka 提供了专门的后台线程定期地巡检待 Compact 的主题，看看是否存在满足条件的可删除数据。这个后台线程叫 Log Cleaner。很多实际生产环境中都出现过位移主题无限膨胀占用过多磁盘空间的问题，如果你的环境中也有这个问题，我建议你去检查一下 Log Cleaner 线程的状态，通常都是这个线程挂掉了导致的。

小结

总结一下，今天我跟你分享了 Kafka 神秘的位移主题 __consumer_offsets，包括引入它的契机与原因、它的作用、消息格式、写入的时机以及管理策略等，这对我们了解 Kafka 特别是 Kafka Consumer 的位移管理是大有帮助的。实际上，将很多元数据以消息的方式存入 Kafka 内部主题的做法越来越流行。除了 Consumer 位移管理，Kafka 事务也是利用了这个方法，当然那是另外的一个内部主题了。

社区的想法很简单：既然 Kafka 天然实现了高持久性和高吞吐量，那么任何有这两个需求的子服务自然也就不必求助于外部系统，用 Kafka 自己实现就好了。

Kakfa的位移主题__consumer_offsets

- 新版本Consumer的位移管理机制，就是将Consumer的位移数据作为一条条普通的Kafka消息，提交到_ _consumer_offsets中。可以说，_ _consumer_offsets的主要作用是保存Kafka消费者的位移信息。

- 虽说位移主题是一个普通的Kafka主题，但它的消息格式却是Kafka自己定义的，用户不能修改。

- 当Kafka集群中的第一个Consumer程序启动时，Kafka会自动创建位移主题。

- Kafka使用Compact策略来删除位移主题中的过期消息，避免该主题无限期膨胀。

开放讨论

今天我们说了位移主题的很多好处，请思考一下，与 ZooKeeper 方案相比，它可能的劣势是什么？

欢迎写下你的思考和答案，我们一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。

AI智能总结

Kafka位移主题__consumer_offsets在Kafka中扮演着重要的角色，用于保存消费者的位移信息。该主题通过将位移数据作为普通的Kafka消息提交到__consumer_offsets中，实现了高持久性和支持高频的写操作。位移主题的消息格式包括保存Consumer Group信息的消息、用于删除Group过期位移的消息以及用于删除Group信息的tombstone消息。位移主题通常在Kafka集群中第一个Consumer程序启动时自动创建，分区数和副本数由相应的Broker端参数控制。Kafka Consumer提交位移的方式有自动提交和手动提交两种，其中手动提交位移需要开发者承担位移提交的责任。此外，Kafka使用Compact策略来删除位移主题中的过期消息，避免该主题无限期膨胀。对于同一个Key的两条消息，如果早于后一条消息，则被视为过期消息。Kafka提供了专门的后台线程Log Cleaner来定期巡检待Compact的主题，以删除满足条件的可删除数据。这些技术特点使得位移主题在Kafka中具有重要意义，对于理解Kafka的位移管理机制和消费者位移信息的保存具有重要意义。文章总结了Kafka位移主题__consumer_offsets的重要性和作用，以及其消息格式、提交方式和管理策略。通过将元数据以消息的方式存入Kafka内部主题，Kafka实现了高持久性和高吞吐量，满足了子服务的需求，避免了对外部系统的依赖。与ZooKeeper方案相比，位移主题可能存在的劣势需要进一步思考和讨论。

全部留言 (126)

最新　精选

此方彼方Francis
2020-02-14

之前遇到过的一个问题跟大家分享一下，原因描述不正确的地方还请大佬指正：
log cleaner线程挂掉还有可能导致消费端出现：Marking Coordinator Dead！

原因大概如下：
log cleaner线程挂掉之后会导致磁盘上位移主题的文件越来越多（当然，大部分是过期数据，只是依旧存在），broker内存中会维护offsetMap，从名字上看这个map就是维护消费进度的，而这个map和位移主题的文件有关联，文件越来越多会导致offsetMap越来越大，甚至导致offsetMap构建失败（为什么会失败没有搞明白），offsetMap构建失败之后broker不会承认自己是coordinator。
消费者组找coordinator的逻辑很简单：abs(consumer_groupName.hashCode) % __consumer_offset.partition.num对应的partition所在的broker就是这个group的coordinate，一旦这

个broker的offsetMap构建失败，那么这个broker就不承认自己是这个group的coordinate，这个group的消费就无法继续进行，会出现Marking Coordinator Dead错误。
此时需要删除过期的位移主题的文件（根据文件名很容易确定哪个是最新的），重启broker。重启过程中需要关注log cleaner是否会再次挂掉。

PS：上述问题在broker重启和正常运行中都有可能遇到。

作者回复: 是个很好的梳理思路~

共 7 条评论>

60

Eco
2020-01-15

有个问题想请教一下，这个位移主题，Consumer是像消费其他主题的分区的内容一样去获取数据的话，那么这本身不也得有个位移，那这个位移又保存到哪里的呢？这样下去不就陷入了一个死循环了吗？要么就不是像正常的消费消息那样去从位移主题获取当前消费者对于某个主题的分区的位移？

作者回复: 好问题！其实Kafka并不太关注__consumer_offsets消费的情况，不过Coordinator的确会在JVM中把所有分区当前已提交的最新位移缓存起来，并且通过这个缓存来决定哪个consumer当前消费到了哪个位移。

共 7 条评论>

28

mellow
2019-07-09

老师能讲一下，同一个group下的consumer启动之后是怎么去offset topic 拿到该group上次消费topic每个partition的最新offset呢？是根据key来定位offset topic的partition吗，然后拿到所有消息得到最新的offset吗

作者回复: 它会去寻找其Coordinator Leader副本对应的broker去拿。根据group.id找到对应Coordinator的分区数

共 5 条评论>

23

王藝明
2019-10-14

老师好！
为什么位移主题写入消息时，不直接替换掉原来的数据，像 HashMap 一样呢？而是要堆积起来，另起线程来维护位移主题

作者回复: 位移主题也是主题，也要遵循Kafka底层的日志设计思路，即append-only log

共 3 条评论> 21

sharpdeep
2020-03-26

有个困惑: 位移主题用来记住位移，那么这个位移主题的位移由谁来记住呢?

作者回复: 位移主题的位移由Kafka内部的Coordinator自行管理

共 2 条评论> 15

2019-08-12

对GroupId 还有疑惑，假设一个Group下有 3 个Consumer，那这三个Consumer 对应的groupid 应该是一样的。这样的话怎么做key做唯一区分呢

作者回复: 每个client都有自己的member id和client id用于区分彼此

共 3 条评论> 14

HZ
2020-02-24

老师 有两点不太清楚 1. 位移主题里面，对于同一个consumer group的位移提交记录，是分布在50个partitions中吗？ 2. 如果把位移主题当作kafka里面的一个普通主题，那么写入这个主题的数据可以保证不丢失吗？ 写入是ack=all？ 同时，broker端的min.insync.replicas的设置有影响吗?

作者回复: 1. 同一个group的位移记录只保存在一个partition上
2. 没错，写入就是acks=all的设置
3. min.insync.replicas对位移主题依然适用

共 4 条评论> 13

西边一抹残阳
2020-01-04

胡老师，消费者提交的位移消息，保存到位移主题分区是随机的吗？就是某一个消费者的提交第一个位移数据保存在位移主题的A分区里面，第二个位移数据保存在位移主题的B分区里面还有消费者是怎样获取已消费的位移数据

作者回复: 不是随机的。通常来说，同一个group下的所有消费者提交的位移数据保存在位移主题的同一个分区下

 12

谦寻
2019-07-15

consumer端 日常业务发版呢，那每次发版需要重启consumer不是也会导致Rebalance，这个如何规避

作者回复: 可以考虑使用standalone consumer，否则group机制无法避免

共 2 条评论> 12

Coder4
2019-07-11

老师好，前几年一直有个说法，说kafka不适合创建过多topic，请问现在的新版还有这个问题么？

作者回复: topic过多其实是指分区数过多。会有两个可能的问题：1. controller无法管理这么多分区；2. 分区数过多导致broker物理随机IO增加，减少吞吐量。

第一个问题社区算是修复了吧，目前单controller能够支持20w的分区数，况且社区也在考虑做多controller方案；第二个问题目前没有太多直接的修复举措，只能说具体问题具体分析吧

共 4 条评论> 9