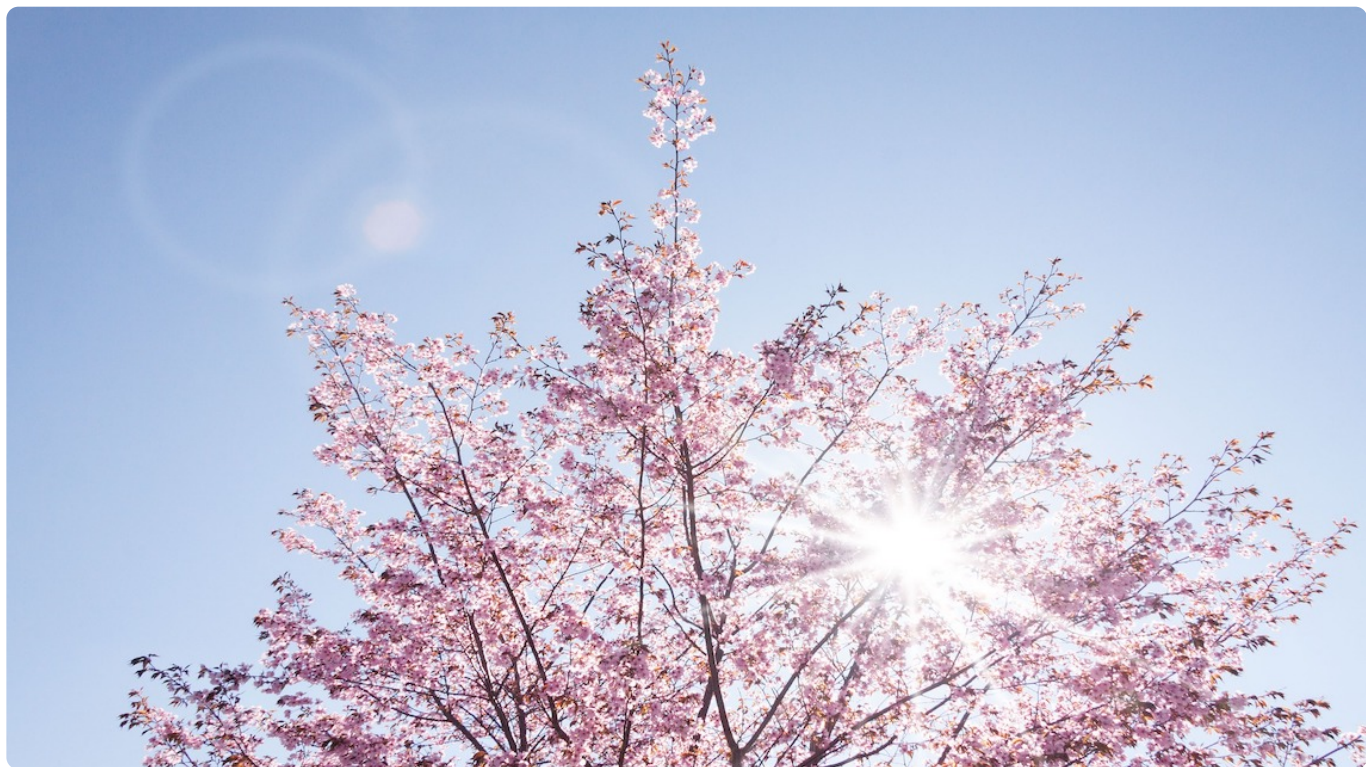


27 | 消费者组元数据（上）：消费者组都有哪些元数据？

2020-06-30 胡夕

Kafka核心源码解读

[进入课程 >](#)



讲述：胡夕

时长 22:10 大小 20.31M



你好，我是胡夕。从今天这节课开始，我们进入到最后一个模块的源码学习：消费者组管理模块。

在这个模块中，我将会带你详细阅读 Kafka 消费者组在 Broker 端的源码实现，包括消费者组元数据的定义与管理、组元数据管理器、内部主题 `__consumer_offsets` 和重要的组件 `GroupCoordinator`。

我先来给你简单介绍下这 4 部分的功能，以方便你对消费者组管理有一个大概的理解。



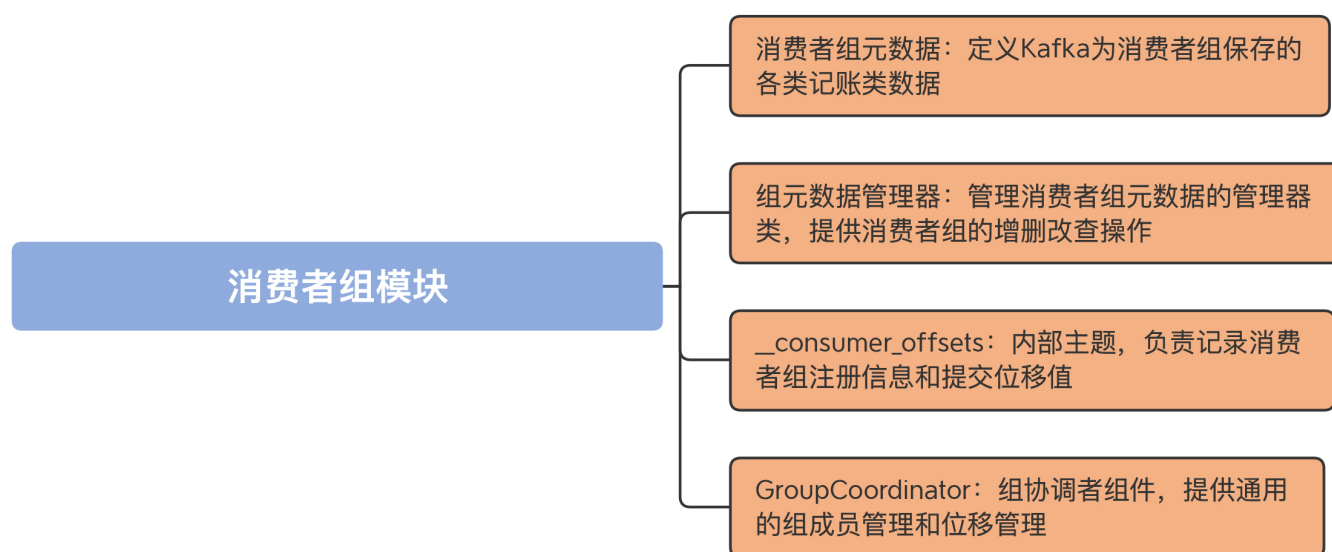
消费者组元数据：这部分源码主要包括 `GroupMetadata` 和 `MemberMetadata`。这两个类共同定义了消费者组的元数据都由哪些内容构成。

组元数据管理器：由 GroupMetadataManager 类定义，可被视为消费者组的管理引擎，提供了消费者组的增删改查功能。

__consumer_offsets：Kafka 的内部主题。除了我们熟知的消费者组提交位移记录功能之外，它还负责保存消费者组的注册记录消息。

GroupCoordinator：组协调者组件，提供通用的组成员管理和位移管理。

我把这 4 部分源码的功能，梳理到了一张思维导图中，你可以保存下来随时查阅：



今天，我们首先学习消费者组元数据的源码实现，这是我们理解消费者组工作机制和深入学习消费者组管理组件的基础。除此之外，掌握这部分代码对我们还有什么实际意义吗？

当然有了。我想你肯定对下面这条命令不陌生吧，它是查询消费者组状态的命令行工具。我们在输出中看到了 GROUP、COORDINATOR、ASSIGNMENT-STRATEGY、STATE 和 MEMBERS 等数据。实际上，这些数据就是消费者组元数据的一部分。

复制代码

```
1 bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --gr
2
3 GROUP          COORDINATOR (ID)  ASSIGNMENT-STRATEGY  STATE      #MEMBERS
4 mygroup        172.25.4.76:9092 (0)    range                Stable     2
```

所以你看，我们日常使用的命令和源码知识的联系是非常紧密的，弄明白了今天的内容，之后你在实际使用一些命令行工具时，就能理解得更加透彻了。

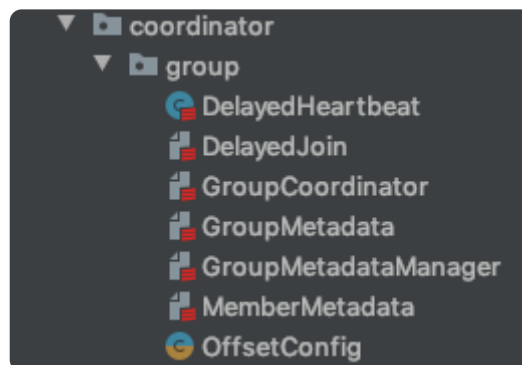
好了，我们现在就正式开始今天的学习吧。

就像我前面说的，元数据主要是由 GroupMetadata 和 MemberMetadata 两个类组成，它们分别位于 GroupMetadata.scala 和 MemberMetadata.scala 这两个源码文件中。从它们的名字上也可以看出来，前者是保存消费者组的元数据，后者是保存消费者组下成员的元数据。

由于一个消费者组下有多个成员，因此，一个 GroupMetadata 实例会对应于多个 MemberMetadata 实例。接下来，我们先学习下 MemberMetadata.scala 源文件。

组成员元数据 (MemberMetadata)

MemberMetadata.scala 源文件位于 coordinator.group 包下。事实上，coordinator.group 包下的所有源代码，都是与消费者组功能息息相关的。下图是 coordinator.group 包的源码文件列表，你可以看到，MemberMetadata.scala 和稍后我们要学到的 GroupMetadata.scala，都在其中。



从这个包结构图中，我们还可以发现后面几节课中要学习的源码类（比如 GroupCoordinator、GroupMetadataManager）也都在里面。当然了，你一定也发现了，coordinator 包下还有个 transaction 包，里面保存了 Kafka 事务相关的所有源代码。如果你想深入学习事务机制的话，可以去阅读下这个包下的源代码。

现在，我们聚焦到 MemberMetadata.scala 文件，包括 3 个类和对象。

MemberSummary 类：组成员概要数据，提取了最核心的元数据信息。上面例子中工具行命令返回的结果，就是这个类提供的数据。

MemberMetadata 伴生对象：仅仅定义了一个工具方法，供上层组件调用。

MemberMetadata 类：消费者组成员的元数据。Kafka 为消费者组成员定义了很多数据，一会儿我们将会详细学习。

按照难易程度，我们从最简单的 MemberSummary 类开始学起。

MemberSummary 类

MemberSummary 类就是组成员元数据的一个概要数据类，它的代码本质上是一个 POJO 类，仅仅承载数据，没有定义任何逻辑。代码如下：

复制代码

```
1 case class MemberSummary(  
2     memberId: String,                // 成员ID, 由Kafka自动生成  
3     groupId: Option[String],         // Consumer端参数group.instance.id值  
4     clientId: String,               // client.id参数值  
5     clientHost: String,             // Consumer端程序主机名  
6     metadata: Array[Byte],          // 消费者组成员使用的分配策略  
7     assignment: Array[Byte])        // 成员订阅分区
```

可以看到，这个类定义了 6 个字段，我来详细解释下。

memberId：标识消费者组成员的 ID，这个 ID 是 Kafka 自动生成的，规则是 consumer- 组 ID-< 序号 >-。虽然现在社区有关于是否放开这个限制的讨论，即是否允许用户自己设定这个 ID，但目前它还是硬编码的，不能让你设置。

groupId：消费者组静态成员的 ID。静态成员机制的引入能够规避不必要的消费者组 Rebalance 操作。它是非常新且高阶的功能，这里你只要稍微知道它的含义就可以了。如果你对此感兴趣，建议你去官网看看 group.instance.id 参数的说明。

clientId：消费者组成员配置的 client.id 参数。由于 memberId 不能被设置，因此，你可以用这个字段来区分消费者组下的不同成员。

clientHost：运行消费者程序的主机名。它记录了这个客户端是从哪台机器发出的消费请求。


metadata: 标识消费者组成员分区分配策略的字节数组，由消费者端参数 partition.assignment.strategy 值设定，默认的 RangeAssignor 策略是按照主题平均分配分区。

assignment: 保存分配给该成员的订阅分区。每个消费者组都要选出一个 Leader 消费者组成员，负责给所有成员分配消费方案。之后，Kafka 将制定好的分配方案序列化成长字节数组，赋值给 assignment，分发给各个成员。

总之，MemberSummary 类是成员概要数据的容器，类似于 Java 中的 POJO 类，不涉及任何操作逻辑，所以还是很好理解的。

MemberMetadata 伴生对象

接下来，我们学习 MemberMetadata 伴生对象的代码。它只定义了一个 plainProtocolSet 方法，供上层组件调用。这个方法只做一件事儿，即从一组给定的分区分配策略详情中提取出分区分配策略的名称，并将其封装成一个集合对象，然后返回：

 复制代码

```
1 private object MemberMetadata {  
2     // 提取分区分配策略集合  
3     def plainProtocolSet(supportedProtocols: List[(String, Array[Byte])]) = supp  
4 }
```

我举个例子说明下。如果消费者组下有 3 个成员，它们的 partition.assignment.strategy 参数分别设置成 RangeAssignor、RangeAssignor 和 RoundRobinAssignor，那么，plainProtocolSet 方法的返回值就是集合[RangeAssignor, RoundRobinAssignor]。实际上，它经常被用来统计一个消费者组下的成员到底配置了多少种分区分配策略。

MemberMetadata 类

现在，我们看下 MemberMetadata 类的源码。首先，我们看下**该类的构造函数以及字段定义**，了解下一个成员的元数据都有哪些。

 复制代码

```
1 @nonthreadsafe  
2 private[group] class MemberMetadata(  
3     var memberId: String,  
4     val groupId: String,
```



```

5  val groupId: Option[String],
6  val clientId: String,
7  val clientHost: String,
8  val rebalanceTimeoutMs: Int, // Rebalance操作超时时间
9  val sessionTimeoutMs: Int, // 会话超时时间
10 val protocolType: String, // 对消费者组而言, 是"consumer"
11 // 成员配置的多套分区分配策略
12 var supportedProtocols: List[(String, Array[Byte])] {
13 // 分区分配方案
14 var assignment: Array[Byte] = Array.empty[Byte]
15 var awaitingJoinCallback: JoinGroupResult => Unit = null
16 var awaitingSyncCallback: SyncGroupResult => Unit = null
17 var isLeaving: Boolean = false
18 var isNew: Boolean = false
19 val isStaticMember: Boolean = groupId.isDefined
20 var heartbeatSatisfied: Boolean = false
21 .....
22 }

```

MemberMetadata 类保存的数据很丰富，在它的构造函数中，除了包含 MemberSummary 类定义的 6 个字段外，还定义了 4 个新字段。

rebalanceTimeoutMs: Rebalance 操作的超时时间，即一次 Rebalance 操作必须在这个时间内完成，否则被视为超时。这个字段的值是 Consumer 端参数 `max.poll.interval.ms` 的值。

sessionTimeoutMs: 会话超时时间。当前消费者组成员依靠心跳机制“保活”。如果在会话超时时间之内未能成功发送心跳，组成员就被判定成“下线”，从而触发新一轮的 Rebalance。这个字段的值是 Consumer 端参数 `session.timeout.ms` 的值。

protocolType: 直译就是协议类型。它实际上标识的是消费者组被用在了哪个场景。这里的场景具体有两个：第一个是作为普通的消费者组使用，该字段对应的值就是 `consumer`；第二个是供 Kafka Connect 组件中的消费者使用，该字段对应的值是 `connect`。当然，不排除后续社区会增加新的协议类型。但现在，你只要知道它是用字符串的值标识应用场景，就足够了。除此之外，该字段并无太大作用。

supportedProtocols: 标识成员配置的多组分区分配策略。目前，Consumer 端参数 `partition.assignment.strategy` 的类型是 `List`，说明你可以为消费者组成员设置多组分配策略，因此，这个字段也是一个 `List` 类型，每个元素是一个元组 (Tuple)。元组的第一个元素是策略名称，第二个元素是序列化后的策略详情。

除了构造函数中的 10 个字段之外，该类还定义了 7 个额外的字段，用于保存元数据和判断状态。这些扩展字段都是 var 型变量，说明它们的值是可以变更的。MemberMetadata 源码正是依靠这些字段，来不断地调整组成员的元数据信息和状态。

我选择了 5 个比较重要的扩展字段，和你介绍下。

assignment：保存分配给该成员的分区分配方案。

awaitingJoinCallback：表示组成员是否正在等待加入组。


awaitingSyncCallback：表示组成员是否正在等待 GroupCoordinator 发送分配方案。

isLeaving：表示组成员是否发起“退出组”的操作。

isNew：表示是否是消费者组下的新成员。

以上就是 MemberMetadata 类的构造函数以及字段定义了。它定义的方法都是操作这些元数据的，而且大多都是逻辑很简单的操作。这里我选取 metadata 方法带你熟悉下它们的编码实现风格。你可以在课后自行阅读其他的方法代码，掌握它们的工作原理。

我们看下 metadata 方法的代码：

 复制代码

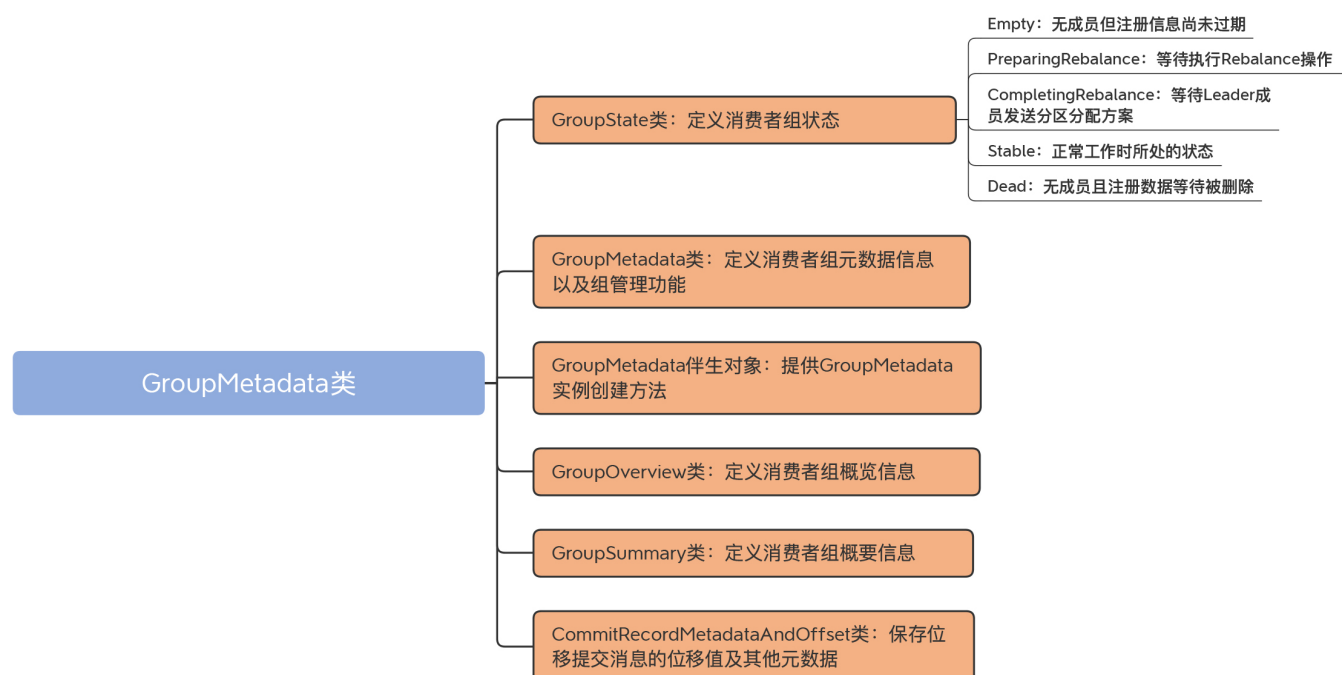
```
1 def metadata(protocol: String): Array[Byte] = {  
2   // 从配置的分区分配策略中寻找给定策略  
3   supportedProtocols.find(_. _1 == protocol) match {  
4     case Some((_, metadata)) => metadata  
5     case None =>  
6       throw new IllegalArgumentException("Member does not support protocol")  
7   }  
8 }
```

它实现的逻辑是：从该成员配置的分区分配方案列表中寻找给定策略的详情。如果找到，就直接返回详情字节数组数据，否则，就抛出异常。怎么样，是不是很简单？

组元数据 (GroupMetadata 类)

说完了组成员元数据类，我们进入到组元数据类 GroupMetadata 的学习。它位于 coordinator.group 包下的同名 scala 文件下。

GroupMetadata 管理的是消费者组而不是消费者组成员级别的元数据，所以，它的代码结构要比 MemberMetadata 类复杂得多。我先画一张思维导图帮你梳理下它的代码结构。



总体上来看，GroupMetadata.scala 文件由 6 部分构成。

GroupState 类：定义了消费者组的状态空间。当前有 5 个状态，分别是 Empty、PreparingRebalance、CompletingRebalance、Stable 和 Dead。其中，Empty 表示当前无成员的消费组；PreparingRebalance 表示正在执行加入组操作的消费组；CompletingRebalance 表示等待 Leader 成员制定分配方案的消费组；Stable 表示已完成 Rebalance 操作可正常工作的消费组；Dead 表示当前无成员且元数据信息被删除的消费组。

GroupMetadata 类：组元数据类。这是该 scala 文件下最重要的类文件，也是我们今天要学习的主要内容。

GroupMetadata 伴生对象：该对象提供了创建 GroupMetadata 实例的方法。

GroupOverview 类：定义了非常简略的消费者组概览信息。


GroupSummary 类：与 MemberSummary 类类似，它定义了消费者组的概要信息。

CommitRecordMetadataAndOffset 类：保存写入到位移主题中的消息的位移值，以及其他元数据信息。这个类的主要职责就是保存位移值，因此，我就不展开说它的详细代码了。

接下来，我们依次看下这些代码结构中都保存了哪些元数据信息。我们从最简单的 GroupState 类开始。

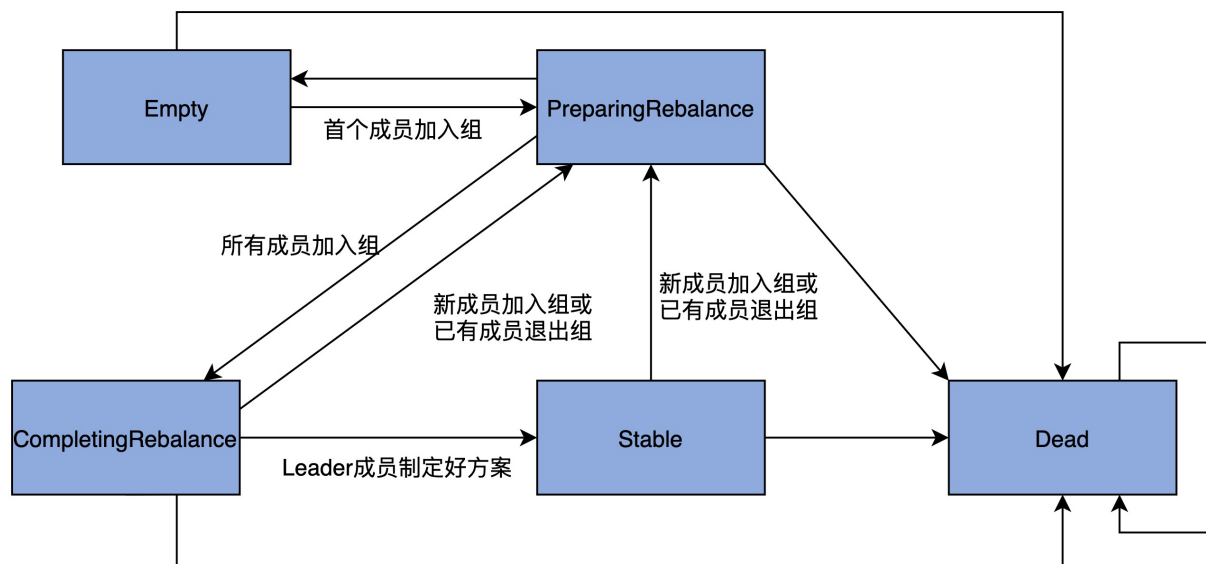
GroupState 类及实现对象

GroupState 类定义了消费者组的状态。这个类及其实现对象 Stable 的代码如下：

 复制代码

```
1 // GroupState trait
2 private[group] sealed trait GroupState {
3   // 合法前置状态
4   val validPreviousStates: Set[GroupState]
5 }
6 // Stable状态
7 private[group] case object Stable extends GroupState {
8   val validPreviousStates: Set[GroupState] = Set(CompletingRebalance)
9 }
10 .....
```

这里我只展示了 Stable 状态的代码，其他 4 个状态 的代码都差不多。为了方便你理解消费者组之间的状态流转，我绘制了一张完整的状态流转图。




你需要记住的是，一个消费者组从创建到正常工作，它的状态流转路径是 Empty -> PreparingRebalance -> CompletingRebalance -> Stable。

GroupOverview 类

接下来，我们看下 GroupOverview 类的代码。就像我刚才说的，这是一个非常简略的组概览信息。当我们在命令行运行 kafka-consumer-groups.sh --list 的时候，Kafka 就会创建 GroupOverview 实例返回给命令行。

我们来看下它的代码：

 复制代码

```
1 case class GroupOverview(  
2   groupId: String,           // 组ID信息，即group.id参数值  
3   protocolType: String,     // 消费者组的协议类型  
4   state: String)           // 消费者组的状态
```

怎么样，很简单吧。GroupOverview 类封装了最基础的组数据，包括组 ID、协议类型和状态信息。如果你熟悉 Java Web 开发的话，可以把 GroupOverview 和 GroupMetadata 的关系，理解为 DAO 和 DTO 的关系。

GroupSummary 类

它的作用和 GroupOverview 非常相似，只不过它保存的数据要稍微多一点。我们看下它的代码：


 复制代码

```
1 case class GroupSummary(  
2   state: String,             // 消费者组状态  
3   protocolType: String,     // 协议类型  
4   protocol: String,         // 消费者组选定的分区分配策略  
5   members: List[MemberSummary]) // 成员元数据
```

GroupSummary 类有 4 个字段，它们的含义也都很清晰，看字段名就能理解。你需要关注的是 **members 字段**，它是一个 MemberSummary 类型的列表，里面保存了消费者组所有成员的元数据信息。通过这个字段，我们可以看到，**消费者组元数据和组成员元数据是 1 对多的关系**。

GroupMetadata 类

最后，我们看下 GroupMetadata 类的源码。我们先看下该类构造函数所需的字段和自定义的扩展元数据：

 复制代码

```
1 @nonthreadsafe
2 private[group] class GroupMetadata(
3     val groupId: String, // 组ID
4     initialState: GroupState, // 消费者组初始状态
5     time: Time) extends Logging {
6     type JoinCallback = JoinGroupResult => Unit
7     // 组状态
8     private var state: GroupState = initialState
9     // 记录状态最近一次变更的时间戳
10    var currentStateTimestamp: Option[Long] = Some(time.milliseconds())
11    var protocolType: Option[String] = None
12    var protocolName: Option[String] = None
13    var generationId = 0
14    // 记录消费者组的Leader成员，可能不存在
15    private var leaderId: Option[String] = None
16    // 成员元数据列表信息
17    private val members = new mutable.HashMap[String, MemberMetadata]
18    // 静态成员Id列表
19    private val staticMembers = new mutable.HashMap[String, String]
20    private var numMembersAwaitingJoin = 0
21    // 分区分配策略支持票数
22    private val supportedProtocols = new mutable.HashMap[String, Integer]().with
23    // 保存消费者组订阅分区的提交位移值
24    private val offsets = new mutable.HashMap[TopicPartition, CommitRecordMetadata]
25    // 消费者组订阅的主题列表
26    private var subscribedTopics: Option[Set[String]] = None
27    .....
28 }
```

GroupMetadata 类定义的字段非常多，也正因为这样，它保存的数据是最全的，绝对担得起消费者组元数据类的称号。

除了我们之前反复提到的字段外，它还定义了很多其他字段。不过，有些字段要么是与事务相关的元数据，要么是属于中间状态的临时元数据，不属于核心的元数据，我们不需要花很多精力去学习它们。我们要重点关注的，是上面的代码中所展示的字段，这些是 GroupMetadata 类最重要的字段。

currentStateTimestamp：记录最近一次状态变更的时间戳，用于确定位移主题中的过期消息。位移主题中的消息也要遵循 Kafka 的留存策略，所有当前时间与该字段的差值

超过了留存阈值的消息都被视为“已过期”（Expired）。

generationId：消费组 Generation 号。Generation 等同于消费者组执行过 Rebalance 操作的次数，每次执行 Rebalance 时，Generation 数都要加 1。

leaderId：消费者组中 Leader 成员的 Member ID 信息。当消费者组执行 Rebalance 过程时，需要选举一个成员作为 Leader，负责为所有成员制定分区分配方案。在 Rebalance 早期阶段，这个 Leader 可能尚未被选举出来。这就是，leaderId 字段是 Option 类型的原因。

members：保存消费者组下所有成员的元数据信息。组元数据是由 MemberMetadata 类建模的，因此，members 字段是按照 Member ID 分组的 MemberMetadata 类。

offsets：保存按照主题分区分组的位移主题消息位移值的 HashMap。Key 是主题分区，Value 是前面讲过的 CommitRecordMetadataAndOffset 类型。当消费者组成员向 Kafka 提交位移时，源码都会向这个字段插入对应的记录。

subscribedTopics：保存消费者组订阅的主题列表，用于帮助从 offsets 字段中过滤订阅主题分区的位移值。

supportedProtocols：保存分区分配策略的支持票数。它是一个 HashMap 类型，其中，Key 是分配策略的名称，Value 是支持的票数。前面我们说过，每个成员可以选择多个分区分配策略，因此，假设成员 A 选择[“range”，“round-robin”]、B 选择[“range”]、C 选择[“round-robin”，“sticky”]，那么这个字段就有 3 项，分别是：<“range”，2>、<“round-robin”，2> 和 <“sticky”，1>。

这些扩展字段和构造函数中的字段，共同构建出了完整的消费者组元数据。就我个人而言，我认为这些字段中最重要的就是 **members 和 offsets**，它们分别保存了组内所有成员的元数据，以及这些成员提交的位移值。这样看的话，这两部分数据不就是一个消费者组最关心的 3 件事吗：**组里面有多少个成员、每个成员都负责做什么、它们都做到了什么程度。**

总结

今天，我带你深入到了 GroupMetadata.scala 和 MemberMetadata.scala 这两个源码文件中，学习了消费者组元数据和组成员元数据的定义。它们封装了一个消费者组及其成员的所有数据。后续的 GroupCoordinator 和其他消费者组组件，都会大量利用这部分元数据执行消费者组的管理。

为了让你更好地掌握今天的内容，我们来回顾下这节课的重点。

消费者组元数据：包括组元数据和组成员元数据两部分，分别由 GroupMetadata 和 MemberMetadata 类表征。

MemberMetadata 类：保存组成员元数据，比如组 ID、Consumer 主机名、协议类型等。同时，它还提供了 MemberSummary 类，封装了组成员元数据的概要信息。

GroupMetadata 类：保存组元数据，包括组状态、组成员元数据列表，等等。

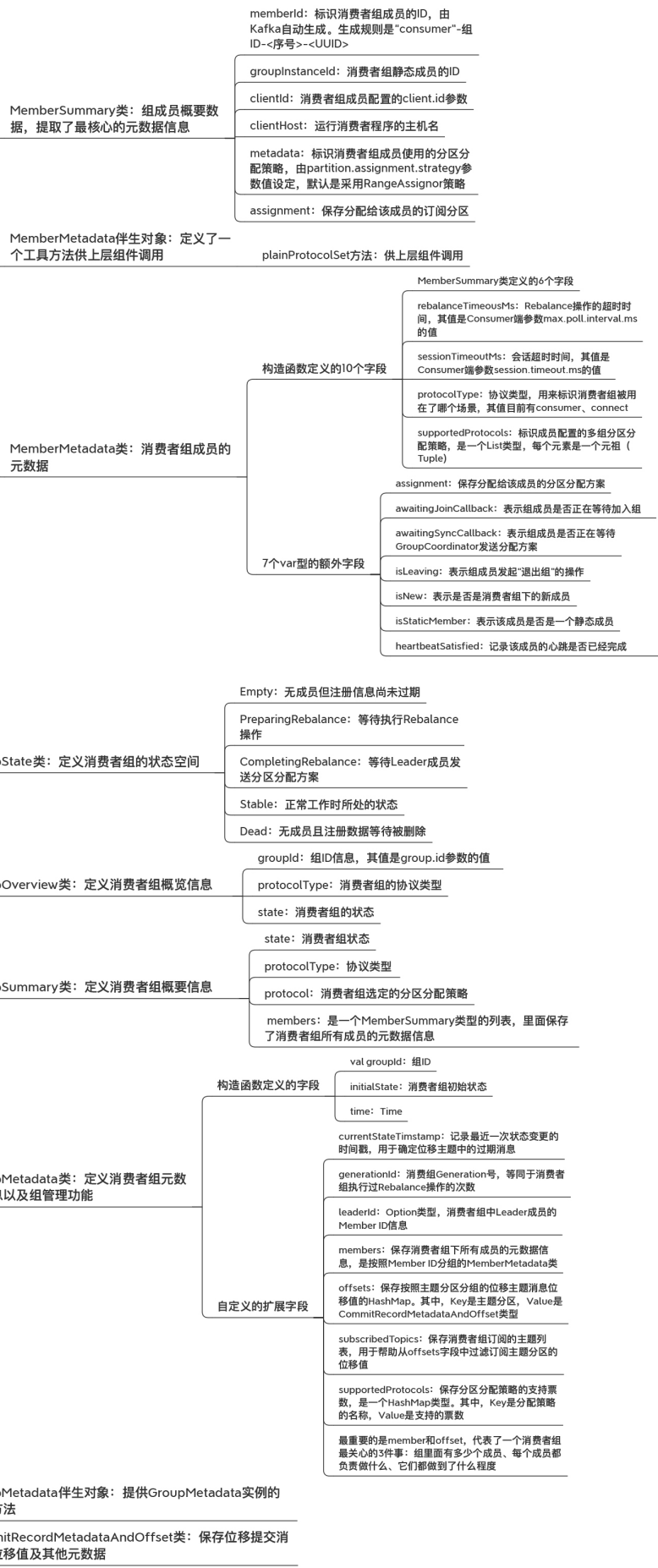
1 对多关系：组元数据与组成员元数据是 1 对多的关系。这是因为每个消费者组下存在若干个组成员。

消费者组元数据

MemberMetadata: 保存消费者下组成员的元数据

1对多, 因为每个
消费者组下存在若干个组成员

GroupMetadata: 保存消费者组的元数据



今天这节课的逻辑不是特别复杂，我们重点学习了消费者组元数据的构成，几乎未曾涉及元数据的操作。在下节课，我们将继续在这两个 scala 文件中探索，去学习操作这些元数据的方法实现。

但我要再次强调的是，今天学习的这些方法是上层组件调用的基础。如果你想彻底了解消费者组的工作原理，就必须先把这部分基础“铺平夯实”了，这样你才能借由它们到达“完全掌握消费者组实现源码”的目的地。

课后讨论

请你思考下，这节课最开始的工具行命令输出中的 ASSIGNMENT-STRATEGY 项，对应于咱们今天学习的哪一项元数据呢？

欢迎在留言区写下你的思考和答案，跟我交流讨论，也欢迎你把今天的内容分享给你的朋友。

更多课程推荐

从0开始学架构

—— 前阿里P9技术专家的
实战架构心法 ——

李运华 前阿里P9技术专家



涨价倒计时 🕒

今日秒杀 **¥79**，7月1日涨价至 **¥129**

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

精选留言 (1)

写留言



伯安知心

2020-06-30

这是在groupstate中定义的属性GroupState(group: String, coordinator: Node, assignmentStrategy: String, state: String, numMembers: Int)。assignmentStrategy是string类型，通过方法collectGroupsState得到并且封装，partitionAssignor就是消费者组的分区策略。

展开 ∨

