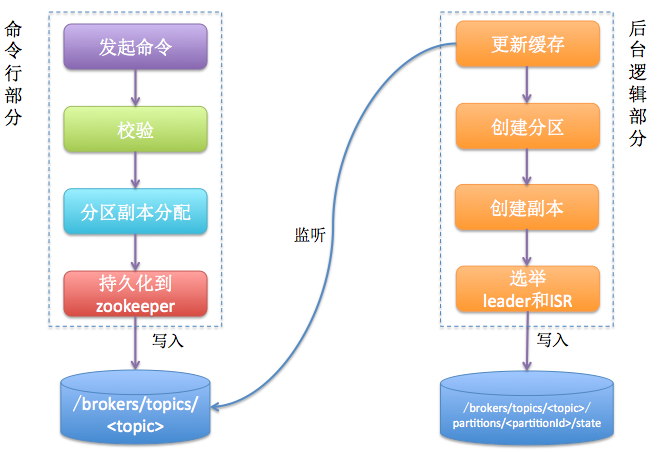
Kafka创建topic命令很简单，一条命令足矣：bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 3 --partitions 3 --topic test

　　这条命令会创建一个名为test的topic，有3个分区，每个分区需分配3个副本。那么在这条命令之后Kafka又做了什么事情呢？本文将对此进行一下梳理，完整地阐述Kafka topic是如何创建的。

　　topic创建主要分为两个部分：命令行部分+后台(controller)逻辑部分，如下图所示。主要的思想就是后台逻辑会监听zookeeper下对应的目录节点，一旦发起topic创建命令，该命令会创建新的数据节点从而触发后台的创建逻辑。



　　简单来说我们发起的命令行主要做两件事情：1. 确定分区副本的分配方案(就是每个分区的副本都分配到哪些broker上)；2. 创建zookeeper节点，把这个方案写入/brokers/topics/<topic>节点下

Kafka controller部分主要做下面这些事情：1. 创建分区；2. 创建副本；3. 为每个分区选举leader、ISR；4.更新各种缓存

下面我们详细说说其中的逻辑。在开始之前，我们假设本例中要创建的topic名字是test，有3个分区，副本因子(replication-factor)也是3。注意：本文只涉及主要的逻辑，一些非默认行为不在此次讨论之中。

命令行部分

　　我们发起topic创建命令之后，Kafka会做一些基本的校验，比如是否同时指定了分区数、副本因子或是topic名字中是否含有非法字符等。值得一提的是，0.10版本支持指定broker的机架信息，类似于Hadoop那样，可以更好地利用局部性原理减少集群中网络开销。如果指定了机架信息(broker.rack), Kafka在为分区做副本分配时就会考虑这部分信息，尽可能地为副本挑选不同机架的broker。当然本例中我们暂时不考虑机架信息对于创建topic的影响。

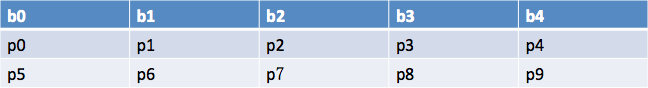
　　做完基本的校验之后，Kafka会从zookeeper的/brokers/ids下获取集群当前存活broker列表然后开始执行副本的分配工作。首先，分区副本的分配有以下3个目标：

* 尽可能地在各个broker之间均匀地分配副本
* 如果分区的某个副本被分配到了一个broker，那么要尽可能地让该分区的其他副本均匀地分配到其他broker上
* 如果所有broker都指定了机架信息，那么尽可能地让每个分区的副本都分配到不同的机架上

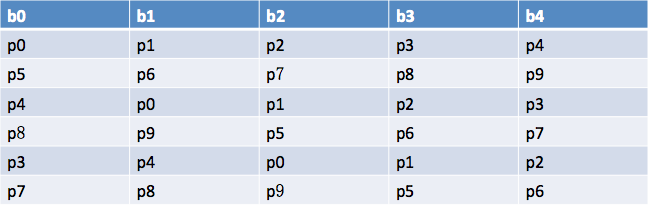
第3个目标目前对于我们没什么用，那前两点是如何做到的？如果直接看源码可能有些晦涩难懂，概括起来就一句话：随机挑选一个broker采用轮询的方式分配每个分区的第一个副本，然后采用增量右移的方式分配其他的副本。好像还是比较难理解，是吧？ 那我举个例子吧：假设你有10个分区p0, p1, p2, ..., p9，每个分区的副本因子都是3，即总共30个副本，要分配在5个broker(b0, b1, b2, b3, b4)上，采用上面的策略就是这样的：

1 Kafka会从5个broker中随机选一个broker，假设它选了b0

2 它会依次采用轮询的方式为所有分区分配第一个副本，如下表所示。即从b0开始依次顺序分配broker给10个分区的第一个副本。



3 目前Kafka已经分配了10个副本，剩下的20个副本Kafka会采用增量右移的方式，比如如果前两行是1，2，3，4，5(第一行) 6，7，8，9，10(第二行)，那么第3行右移1位，变成5，1，2，3，4，第4行右移2位，变成：9, 10, 6, 7, 8，以此类推。那么采用这种方式分配的副本方案如下表所示：



当然，如果考虑机架信息，分配算法会有所调整，但基本上也是满足上面那3个目标的。

对于本文中使用的例子，我们假设分配方案如下：(格式是分区号 -> 副本所在broker Id集合)

0 -> [0，1，2]

1 -> [1，0，2]

2 -> [2，0，1]

　　确定了分区副本分配方案之后，Kafka会把这个分配方案持久化到zookeeper的/brokers/topics/<topic>节点下，类似于这样的信息：{"version":1,"partitions":{"0":[0,1,2],"1":[1,0,2],"2":[2,0,1]}}

okay，至此命令行部分的工作就算完成了，此时你应该可以看到Kafka会返回Created topic "test"给你，表明topic创建成功。但是，千万不要以为Kafka创建topic的工作就完成了，后面还有很多事情要做，即controller要登场了。

后台逻辑部分

　　所谓的后台逻辑其实是由Kafka的controller负责提供的。Kafka的controller内部保存了很多信息，其中有一个分区状态机，用于记录topic各个分区的状态。这个状态机内部注册了一些zookeeper监听器。Controller在启动的时候会创建这些监听器。其中一个监听器(TopicChangeListener)就是用于监听zookeeper的/brokers/topics目录的子节点变化的。一旦该目录子节点数发生变化就会调用这个监听器的处理方法。对于上面的例子来说，由于命令行已将分配方案持久化到/brokers/topics/test下，所以会触发该监听器的处理方法。

　　TopicChangeListener监听器一方面会更新controller的缓存信息(比如更新集群当前所有的topic列表以及更新新增topic的分区副本分配方案缓存等)，另一方面就是创建对应的分区及其副本对象并为每个分区确定leader副本及ISR。

　　至此，整个topic的创建就完成了！

====================================================================================================================

　　显然，刚才关于后台controller逻辑部分几乎就是一笔带过了，没有详细展开。毕竟如果直接讲代码会比较枯燥。一般情况下，我们了解到此程度就可以了。下面将针对代码详细分析下controller是如何创建topic的。

　　上边提到过，controller内部定义了很多数据结构用于记录当前集群的各种状态。在Controller中还分别定义了一个分区状态机(PartitionStateMachine)和副本状态机(ReplicaStateMachine)，分别记录各个分区的状态和状态流转，如下面两张图所示：



咋一看，这两张图似乎差不多，但一个是分区状态流转，一个是副本状态流转。不管是分区还是副本，只有处于Online状态的才能正常工作。当然在设置这个状态之前必须要先完成一些工作。下面详细说说：

1 首先，分区状态机的registerPartitionChangeListener方法会注册一个zookeeper监听器，监听到/brokers/topics下新增了test节点之后，立即处理TopicChangeListener的handleChildChange方法

2 handleChildChange方法的具体逻辑是：

　　2.1 结合controller缓存的topic列表和/brokers/topics目录下的topic列表，找出新增的topic：test。假设controller topic列表是A，/brokers/topics下列表是B，新增topic列表可由A - B求得

　　2.2 使用类似的方法，确定已经被删除的topic集合，即B - A

　　2.3 更新controller缓存的topic列表(把test加进去，把那些已经被删除的topic从缓存中踢出去)

　　2.4 从/brokers/topics/test节点中取出这个topic所有分区的副本分配方案，然后去更新controller对应的这部分信息(其实也是把test的方案加入到缓存中，另外也会把已删除的topic对应的方案也踢出去)

　　2.5 调用onNewTopicCreation开始创建topic

3 onNewTopicCreation：创建topic的回调方法，实现真正的创建topic的逻辑：

　　3.1 注册分区变更监听器——之前说过了分区状态机会注册一些zookeeper监听器，刚刚提到的TopicChangeListener只是其中之一，而这里的监听器是监听topic的分区变化的。该监听器就是PartitionModificationListener类，顾名思义，它负责监听topic下分区的变化情况，具体来说就是监听/brokers/topics/topic节点的数据，一旦发生变化该监听器就会被触发。当然对于创建topic而言，这一步仅仅是注册而已并不会被触发，因为在注册这个监听器之前Kafka已经把数据写入这个节点了。所以此时该监听器不会触发操作，这是为以后修改topic时候使用的。 既然本次不会触发监听器，代码里面就手动调用onNewPartitionCreation来创建分区了

　　3.2 调用onNewPartitionCreation方法创建分区

4 onNewPartitionCreation: 这个方法的目的就是创建topic的所有分区对象，主要涉及4个步骤：

　　4.1 创建分区对象，并设置成NewPartition状态：既然叫分区状态机，必然有个地方要保存Kafka集群下所有topic的所有分区的状态。每当有新topic创建时，就需要把新增topic所有分区加入这部分缓存，以达到同步的效果。新增的分区状态统一设置成NewPartition

　　4.2 为每个分区创建对应的副本对象：Kafka首先从controller缓存中找出这个分区对应的分配方案(还记得吧，controller有个地方保存了所有topic的分区副本分配方案，就是从这里找)，然后把这个分区下的所有副本都设置成NewReplica状态——具体来说Kafka是怎么做的呢？首先，它会尝试去获取zookeeper中/brokers/topics/test/partitions/<partitionId>/state节点的数据，该节点保存了每个分区的leader副本和ISR信息。不过对于创建topic来说，目前这个topic的所有分区都没有leader和ISR信息，所以该节点应该还不存在，应该是空——这是正常的，因为后面会开始选举！所以这里Kafka仅仅是更新副本状态机的状态缓存就可以了(忘了说了，既然分区状态机有个缓存保存集群中所有分区的状态，那么副本状态机自然也有类似的缓存来保存集群中所有topic下所有分区的副本的状态，所以此时还需要更新这部分缓存)

　　4.3 前2步创建了分区对象和副本对象，并分别设置成了NewPartition和NewReplica状态。那么这一步就要把分区状态转换到OnlinePartition，只有处于此状态才可以正常使用。这也是这一步需要做的事情：leader选举！ 代码写的很冗长，但简单来说就是选取副本集合中的第一个副本作为leader副本，并把整个副本集合作为ISR。举例来说，对于test的分区0，它的副本集合是0,1,2，那么分区0的leader副本就是0，ISR就是[0，1，2]。之后Kafka会把这些信息连同controller的epoch和leader的epoch(多说一句，controller epoch值表示controller被易主的次数，leader epoch也是同理)一同写入zookeeper的/brokers/topics/test/partitions/0/state节点下，之后更新controller的leader缓存。(再多说一句，controller有个地方记录了topic所有分区的leader和ISR信息)。 okay，现在新增topic的所有分区都选好了leader和ISR，那么就需要让集群中其他broker知晓—— 因此需要发送UpdateMetadataRequest给当前所有broker——具体的发送方法其实就是将分区的leader和ISR信息打包封装进一个map然后为map中的每一项都构造一个UpdateMetadataReuqest对象并通过controller的sendRequest方法发给所有存活着的broker(为什么要发送给所有broker？因为LeaderAndIsr请求是唯一一个所有broker都能立即响应而不需要求助于leader broker的请求！) 具体的发送逻辑由于涉及了Kafka底层网络协议及KafkaApi机制，等以后有机会再详谈吧。。。

　　4.4 设置副本对象为OnlineReplica：目前所有的分区都已经选好了leader和ISR并已经持久化到zookeeper中，当然还都传播到了其他broker上。那么这最后一步就是将副本状态机中缓存的副本状态从NewReplica转换到OnlineReplica

　　okay，至此一个topic就完整地创建出来了~~