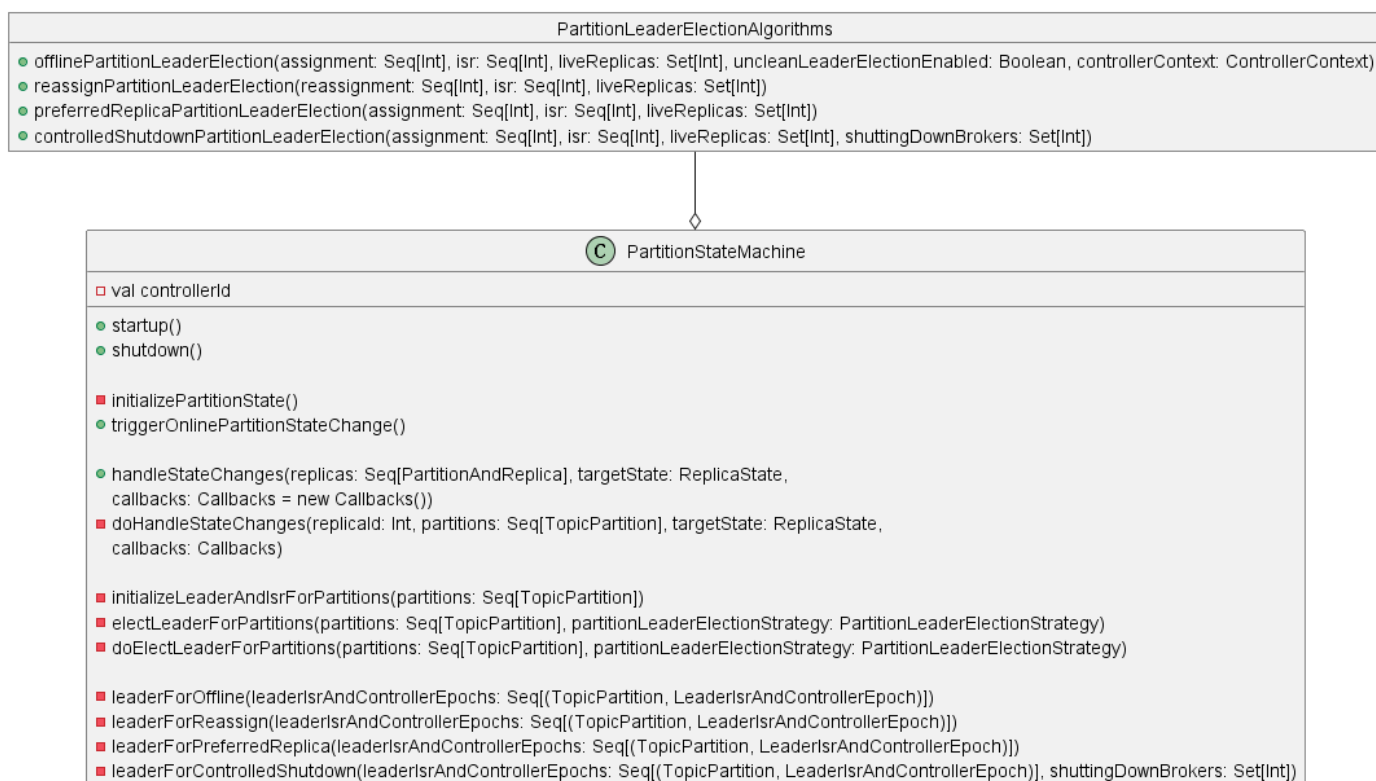


- `partitionStateMachine`
 - `startUp()`
 - `handleStateChanges()`
 - `doHandleStateChanges()`
 - `initializeLeaderAndIsrForPartitions(partitions: Seq[TopicPartition])`
 - `doElectLeaderForPartitions()`

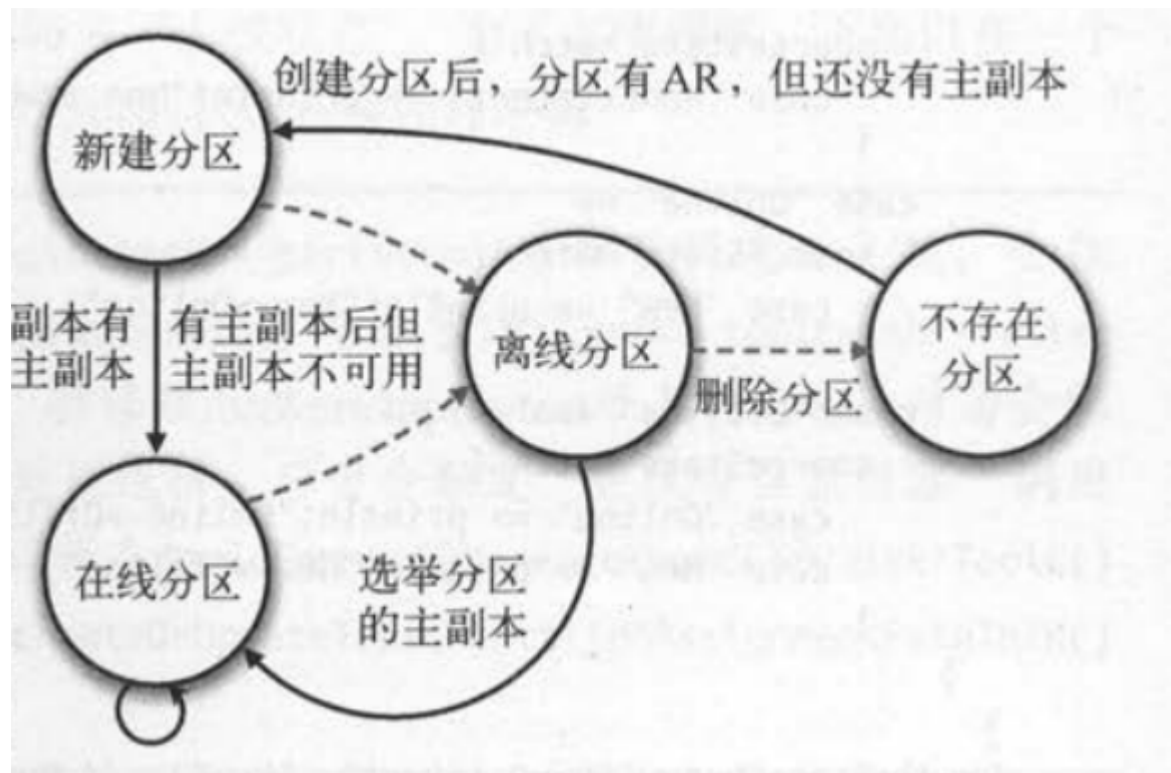
partitionStateMachine

`partitionStateMachine`是`controller`模块中的分区状态机，主要负责分区在`online,offline,new,non_existent`几种状态间进行转化。分区状态机主要负责对于分区进行了副本领导的选举，将无副本的分区下线，更新`controller`的上下文，发送数据给各个代理节点更新等一系列事物。下面将根据`partitionStateMachine`中的一系列函数进行说明

类图如下：



分区状态机：



startUp()

```

def startup() {
  info("Initializing partition state")
  initializePartitionState()
  info("Triggering online partition state changes")
  triggerOnlinePartitionStateChange()
  info(s"Started partition state machine with initial state -> $partitionState")
}

```

startUp函数的主要功能

- `init()`判断分区的主副本是否存在，若存在则为**online**，不存在为**offline**
- `triggerOnlinePartitionStateChange()`将所有的分区状态变为新建或离线(除非分区要被删除)
- 以上的操作进行判断的前提，均从**controller**的上下文中获得

handleStateChanges()

```

def handleStateChanges(partitions: Seq[TopicPartition], targetState:
PartitionState,
                      partitionLeaderElectionStrategyOpt:
Option[PartitionLeaderElectionStrategy] = None): Unit = {
  if (partitions.nonEmpty) {
    try {
      controllerBrokerRequestBatch.newBatch()
      doHandleStateChanges(partitions, targetState,

```

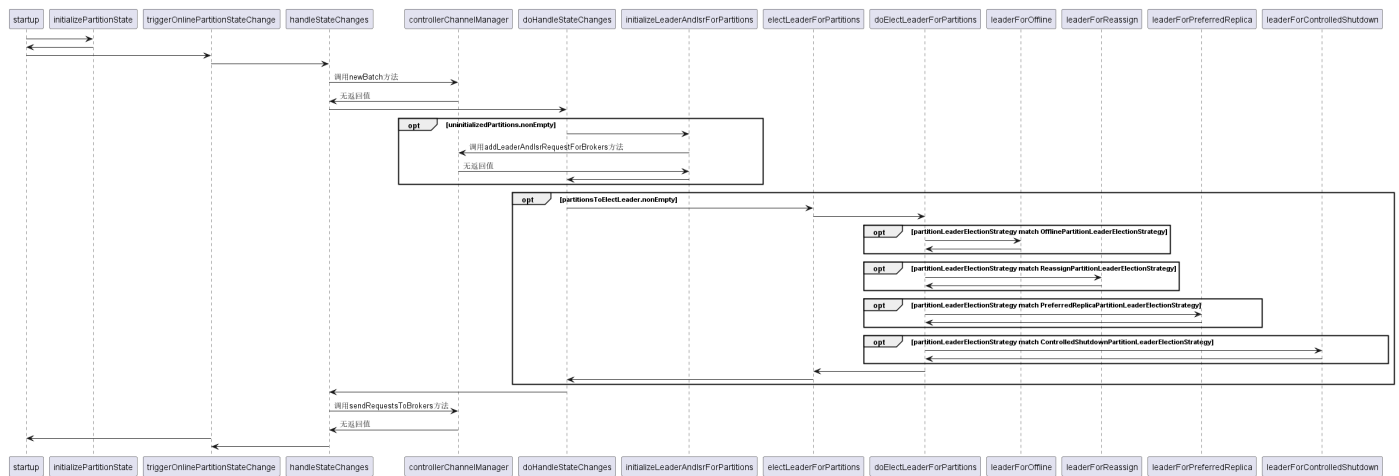
```

partitionLeaderElectionStrategyOpt)
    controllerBrokerRequestBatch.sendRequestsToBrokers(controllerContext.epoch)
  } catch {
    case e: Throwable => error(s"Error while moving some partitions to
    $targetState state", e)
  }
}
}

```

handleStateChanges()首先调用BrokerRequestBatch(ControllerChannelManager中的类)来建立与各个代理节点之间的通道，在中间调用dohandleStateChanges()函数

handleStateChanges()时序图如下：



doHandleStateChanges()

```

private def doHandleStateChanges(partitions: Seq[TopicPartition], targetState:
PartitionState,
                                partitionLeaderElectionStrategyOpt:
Option[PartitionLeaderElectionStrategy]): Unit = {
    val stateChangeLog =
stateChangeLogger.withControllerEpoch(controllerContext.epoch)
    partitions.foreach(partition => partitionState.getOrElseUpdate(partition,
NonExistentPartition))
    val (validPartitions, invalidPartitions) = partitions.partition(partition =>
isValidTransition(partition, targetState))
    invalidPartitions.foreach(partition => logInvalidTransition(partition,
targetState))
    targetState match {
      case NewPartition =>
        validPartitions.foreach { partition =>
          stateChangeLog.trace(s"Changed partition $partition state from
          ${partitionState(partition)} to $targetState with " +
          s"assigned replicas
          ${controllerContext.partitionReplicaAssignment(partition).mkString(",")}")
          partitionState.put(partition, NewPartition)
        }
      case OnlinePartition =>
    }

```

```

        val uninitializedPartitions = validPartitions.filter(partition =>
partitionState(partition) == NewPartition)
        val partitionsToElectLeader = validPartitions.filter(partition =>
partitionState(partition) == OfflinePartition || partitionState(partition) ==
OnlinePartition)
        if (uninitializedPartitions.nonEmpty) {
            val successfulInitializations =
initializeLeaderAndIsrForPartitions(uninitializedPartitions)
            successfulInitializations.foreach { partition =>
                stateChangeLog.trace(s"Changed partition $partition from
${partitionState(partition)} to $targetState with state " +

s"${controllerContext.partitionLeadershipInfo(partition).leaderAndIsr}")
                partitionState.put(partition, OnlinePartition)
            }
        }
        if (partitionsToElectLeader.nonEmpty) {
            val successfulElections =
electLeaderForPartitions(partitionsToElectLeader,
partitionLeaderElectionStrategyOpt.get)
            successfulElections.foreach { partition =>
                stateChangeLog.trace(s"Changed partition $partition from
${partitionState(partition)} to $targetState with state " +

s"${controllerContext.partitionLeadershipInfo(partition).leaderAndIsr}")
                partitionState.put(partition, OnlinePartition)
            }
        }
        case OfflinePartition =>
            validPartitions.foreach { partition =>
                stateChangeLog.trace(s"Changed partition $partition state from
${partitionState(partition)} to $targetState")
                partitionState.put(partition, OfflinePartition)
            }
        case NonExistentPartition =>
            validPartitions.foreach { partition =>
                stateChangeLog.trace(s"Changed partition $partition state from
${partitionState(partition)} to $targetState")
                partitionState.put(partition, NonExistentPartition)
            }
    }
}

```

doHandleStateChanges()步骤

1. 首先根据传进的partitions信息分出validPartitions和invalidPartitions；对于valid的partition如果目标是转成new，offline，nonexist都可以直接进行；若要转成online，则应根据该分区是否是新建而进行领导的选举，选举成功之后才可以转成online状态
2. 在要转为online的分区中，分为uninitializedPartitions和partitionsForLeader，分别对应新建未选举过的分区和已经经历过选举的分区；对于uninitializedPartitions调

用initializedLeaderAndIsrForPartitions(), 对于partitionsForLeader调用
electLeaderForPartitions()函数

initializeLeaderAndIsrForPartitions(partitions: Seq[TopicPartition])

```
private def initializeLeaderAndIsrForPartitions(partitions: Seq[TopicPartition]):  
Seq[TopicPartition] = {  
    val successfulInitializations = mutable.Buffer.empty[TopicPartition]  
    val replicasPerPartition = partitions.map(partition => partition ->  
controllerContext.partitionReplicaAssignment(partition))  
    val liveReplicasPerPartition = replicasPerPartition.map { case (partition,  
replicas) =>  
        val liveReplicasForPartition = replicas.filter(replica =>  
controllerContext.isReplicaOnline(replica, partition))  
        partition -> liveReplicasForPartition  
    }  
    val (partitionsWithoutLiveReplicas, partitionsWithLiveReplicas) =  
liveReplicasPerPartition.partition { case (_, liveReplicas) => liveReplicas.isEmpty  
}  
  
    partitionsWithoutLiveReplicas.foreach { case (partition, replicas) =>  
        val failMsg = s"Controller $controllerId epoch ${controllerContext.epoch}  
encountered error during state change of " +  
            s"partition $partition from New to Online, assigned replicas are " +  
            s"[${replicas.mkString(",")}]", live brokers are  
[${controllerContext.liveBrokerIds}]. No assigned " +  
            "replica is alive."  
        logFailedStateChange(partition, NewPartition, OnlinePartition, new  
StateChangeFailedException(failMsg))  
    }  
    val leaderIsrAndControllerEpochs = partitionsWithLiveReplicas.map { case  
(partition, liveReplicas) =>  
        val leaderAndIsr = LeaderAndIsr(liveReplicas.head, liveReplicas.toList)  
        val leaderIsrAndControllerEpoch = LeaderIsrAndControllerEpoch(leaderAndIsr,  
controllerContext.epoch)  
        partition -> leaderIsrAndControllerEpoch  
    }.toMap  
    val createResponses = try {  
        zkClient.createTopicPartitionStatesRaw(leaderIsrAndControllerEpochs)  
    } catch {  
        case e: Exception =>  
            partitionsWithLiveReplicas.foreach { case (partition, _) =>  
logFailedStateChange(partition, partitionState(partition), NewPartition, e) }  
            Seq.empty  
    }  
    createResponses.foreach { createResponse =>  
        val code = createResponse.resultCode  
        val partition = createResponse.ctx.get.asInstanceOf[TopicPartition]  
        val leaderIsrAndControllerEpoch = leaderIsrAndControllerEpochs(partition)  
        if (code == Code.OK) {  
            controllerContext.partitionLeadershipInfo.put(partition,  
leaderIsrAndControllerEpoch)  
        }  
    }  
    controllerBrokerRequestBatch.addLeaderAndIsrRequestForBrokers(leaderIsrAndControllerEpochs)
```

```

rEpoch.leaderAndIsr.isr,
    partition, leaderIsrAndControllerEpoch,
controllerContext.partitionReplicaAssignment(partition), isNew = true)
    successfulInitializations += partition
  } else {
    logFailedStateChange(partition, NewPartition, OnlinePartition, code)
  }
}
successfulInitializations
}

```

- `initializeLeaderAndIsrForPartitions(partitions: Seq[TopicPartition])`函数首先得到 `partitionWithoutReplicas`和`partitionsWithReplicas`，前者会报错并记录在文件中
- 由`partitionsWithReplicas`出发，调用`LeaderAndISR.api`得到第一个副本与存活副本，并向zk节点创建`createResponses`。函数将`Epoch`更新，并更新上下文，发送给个代理节点完成`initialization`(`successfulinitialization`进行计数)，并根据`createResponses`进行交互

doElectLeaderForPartitions()

```

private def doElectLeaderForPartitions(partitions: Seq[TopicPartition],
partitionLeaderElectionStrategy: PartitionLeaderElectionStrategy):
(Seq[TopicPartition], Seq[TopicPartition], Map[TopicPartition, Exception]) = {
  val getDataResponses = try {
    zkClient.getTopicPartitionStatesRaw(partitions)
  } catch {
    case e: Exception =>
      return (Seq.empty, Seq.empty, partitions.map(_ -> e).toMap)
  }
  val failedElections = mutable.Map.empty[TopicPartition, Exception]
  val leaderIsrAndControllerEpochPerPartition =
mutable.Buffer.empty[(TopicPartition, LeaderIsrAndControllerEpoch)]
  getDataResponses.foreach { getDataResponse =>
    val partition = getDataResponse.ctx.get.asInstanceOf[TopicPartition]
    val currState = partitionState(partition)
    if (getDataResponse.resultCode == Code.OK) {
      val leaderIsrAndControllerEpochOpt =
TopicPartitionStateZNode.decode(getDataResponse.data, getDataResponse.stat)
      if (leaderIsrAndControllerEpochOpt.isEmpty) {
        val exception = new StateChangeFailedException(s"LeaderAndIsr information
doesn't exist for partition $partition in $currState state")
        failedElections.put(partition, exception)
      }
      leaderIsrAndControllerEpochPerPartition += partition ->
leaderIsrAndControllerEpochOpt.get
    } else if (getDataResponse.resultCode == Code.NONODE) {
      val exception = new StateChangeFailedException(s"LeaderAndIsr information
doesn't exist for partition $partition in $currState state")
      failedElections.put(partition, exception)
    } else {
      failedElections.put(partition, getDataResponse.resultException.get)
    }
  }
}

```

```

    }
  }
  val (invalidPartitionsForElection, validPartitionsForElection) =
    leaderIsrAndControllerEpochPerPartition.partition { case (partition,
    leaderIsrAndControllerEpoch) =>
      leaderIsrAndControllerEpoch.controllerEpoch > controllerContext.epoch
    }
  invalidPartitionsForElection.foreach { case (partition,
    leaderIsrAndControllerEpoch) =>
    val failMsg = s"aborted leader election for partition $partition since the
    LeaderAndIsr path was " +
      s"already written by another controller. This probably means that the
    current controller $controllerId went through " +
      s"a soft failure and another controller was elected with epoch
    ${leaderIsrAndControllerEpoch.controllerEpoch}."
    failedElections.put(partition, new StateChangeFailedException(failMsg))
  }
  if (validPartitionsForElection.isEmpty) {
    return (Seq.empty, Seq.empty, failedElections.toMap)
  }
  val shuttingDownBrokers = controllerContext.shuttingDownBrokerIds.toSet
  val (partitionsWithoutLeaders, partitionsWithLeaders) =
    partitionLeaderElectionStrategy match {
      case OfflinePartitionLeaderElectionStrategy =>
        leaderForOffline(validPartitionsForElection).partition { case (_,
        newLeaderAndIsrOpt, _) => newLeaderAndIsrOpt.isEmpty }
      case ReassignPartitionLeaderElectionStrategy =>
        leaderForReassign(validPartitionsForElection).partition { case (_,
        newLeaderAndIsrOpt, _) => newLeaderAndIsrOpt.isEmpty }
      case PreferredReplicaPartitionLeaderElectionStrategy =>
        leaderForPreferredReplica(validPartitionsForElection).partition { case (_,
        newLeaderAndIsrOpt, _) => newLeaderAndIsrOpt.isEmpty }
      case ControlledShutdownPartitionLeaderElectionStrategy =>
        leaderForControlledShutdown(validPartitionsForElection,
        shuttingDownBrokers).partition { case (_, newLeaderAndIsrOpt, _) =>
        newLeaderAndIsrOpt.isEmpty }
    }
  partitionsWithoutLeaders.foreach { case (partition, leaderAndIsrOpt,
    recipients) =>
    val failMsg = s"Failed to elect leader for partition $partition under
    strategy $partitionLeaderElectionStrategy"
    failedElections.put(partition, new StateChangeFailedException(failMsg))
  }
  val recipientsPerPartition = partitionsWithLeaders.map { case (partition,
    leaderAndIsrOpt, recipients) => partition -> recipients }.toMap
  val adjustedLeaderAndIsrs = partitionsWithLeaders.map { case (partition,
    leaderAndIsrOpt, recipients) => partition -> leaderAndIsrOpt.get }.toMap
  val UpdateLeaderAndIsrResult(successfulUpdates, updatesToRetry, failedUpdates)
    = zkClient.updateLeaderAndIsr(
    adjustedLeaderAndIsrs, controllerContext.epoch)
  successfulUpdates.foreach { case (partition, leaderAndIsr) =>
    val replicas = controllerContext.partitionReplicaAssignment(partition)
    val leaderIsrAndControllerEpoch = LeaderIsrAndControllerEpoch(leaderAndIsr,
    controllerContext.epoch)
    controllerContext.partitionLeadershipInfo.put(partition,
    leaderIsrAndControllerEpoch)
  }

```



```

controllerBrokerRequestBatch.addLeaderAndIsrRequestForBrokers(recipientsPerPartition(partition), partition,
    leaderIsrAndControllerEpoch, replicas, isNew = false)
}
(successfulUpdates.keys.toSeq, updatesToRetry, failedElections.toMap ++
failedUpdates)
}

```

1. 用`getDataResponse()`拿到每个分区znode节点数据，然后译码产生`EpochOpt`将其加入到`LeaderIsrControllerEpochPerPartition`中，产生`invalid`与`valid Partition`
2. 对以上`valid Partition`分区进行选举，并最终发送给各个代理节点

注意上面的过程，在对`valid Partition`进行选举时，会根据不同的选举逻辑调用不同的`leader`函数，如

```

val (partitionsWithoutLeaders, partitionsWithLeaders) =
partitionLeaderElectionStrategy match {
    case OfflinePartitionLeaderElectionStrategy =>
        leaderForOffline(validPartitionsForElection).partition { case (_,
newLeaderAndIsrOpt, _) => newLeaderAndIsrOpt.isEmpty }
    case ReassignPartitionLeaderElectionStrategy =>
        leaderForReassign(validPartitionsForElection).partition { case (_,
newLeaderAndIsrOpt, _) => newLeaderAndIsrOpt.isEmpty }
    case PreferredReplicaPartitionLeaderElectionStrategy =>
        leaderForPreferredReplica(validPartitionsForElection).partition { case
(_, newLeaderAndIsrOpt, _) => newLeaderAndIsrOpt.isEmpty }
    case ControlledShutdownPartitionLeaderElectionStrategy =>
        leaderForControlledShutdown(validPartitionsForElection,
shuttingDownBrokers).partition { case (_, newLeaderAndIsrOpt, _) =>
newLeaderAndIsrOpt.isEmpty }
}

```

在上述的选举中得到`partitionWithLeaders`，通过不同的四个函数来进行分区领导副本的选举，并最终将结果更新到`controller`的`context`中，将信息发送给各个代理节点