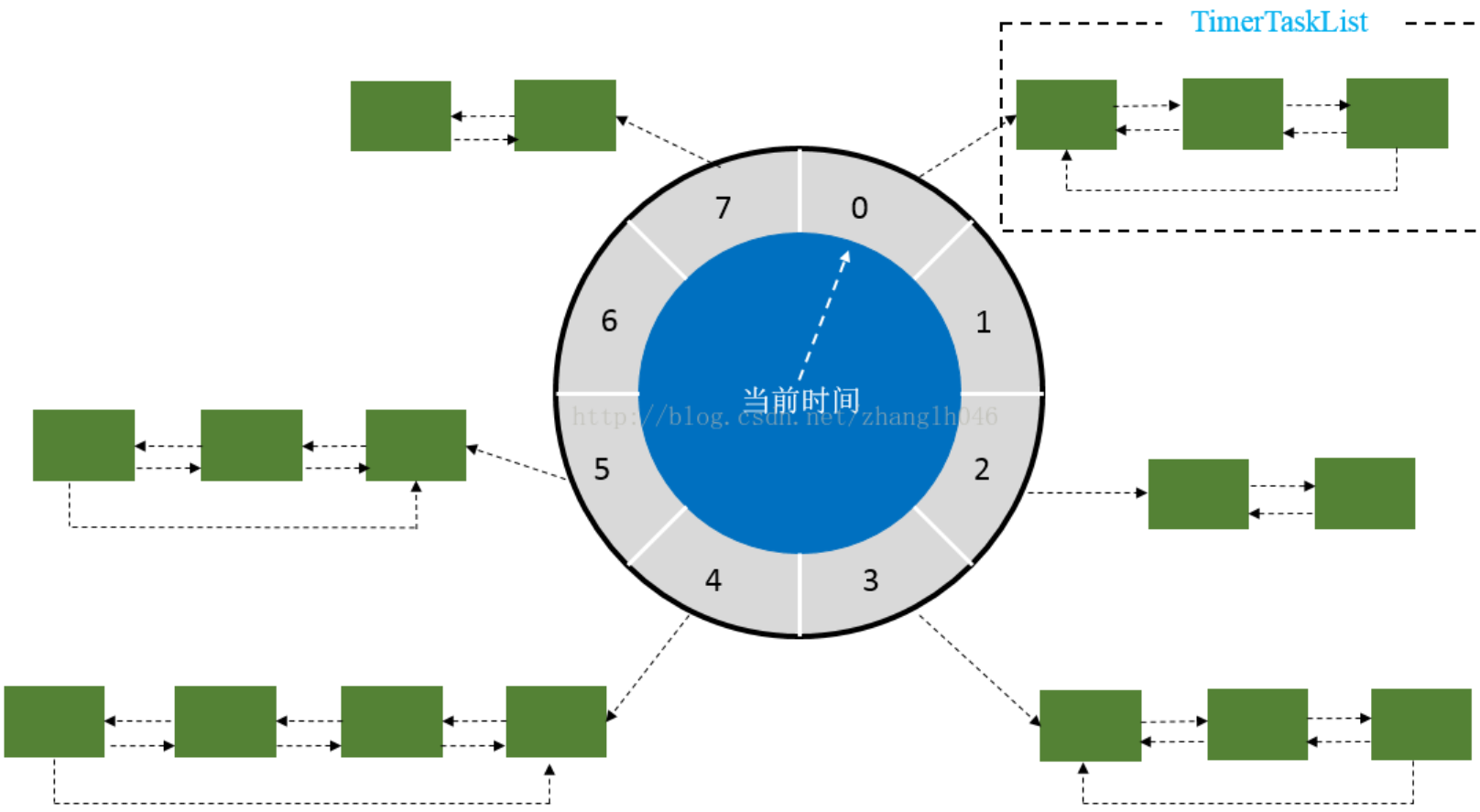
原 TimingWheel[时间轮]介绍

Kafka的延迟操作是一个相对独立的组件，他的主要功能是管理延迟操作，底层依赖于Kafka提供的时间轮实现。JDK本身提供的java.util.Timer也可以实现定时任务，但是如果系统请求量巨大，性能要求很高，他们底层所依赖的数据结构存取操作复杂度都是O(nlog(n))

为了将时间复杂度降为o(1)，一般会使用其他方式的定时任务组件，比如zookeeper的时间桶方式处理session过期，netty也使用Hash

WheelTimer这种时间轮的实现。



Kafka时间轮的实现是TimingWheel，他是一个存储定时任务的环形队列（桶），底层使用数组实现，数组中每一个元素可以存放一个TimerTaskList对象

TimerTaskList是环形双向链表，在其中链表项TimeTaskEntry封装了真正的定时任务TimerTask。TimerTaskList使用expiration字段记录了整个TimerTaskList的超时时间。TimeTaskEntry中的expirationMs字段记录了超时时间戳，timerTask字段指向了对应的TimerTask任务.

TimerTask中的delayMs记录了任务的延迟时间，timerTaskEntry记录了TimerTaskEntry对象

TimingWheel提供了分层的概念，因为年时间跨度比较大，数量很大，单层的时间轮会造成任务的round很大，单个格子链表很长。一般情况，第一层时间跨度是最小的，第二层时间跨度比较大。

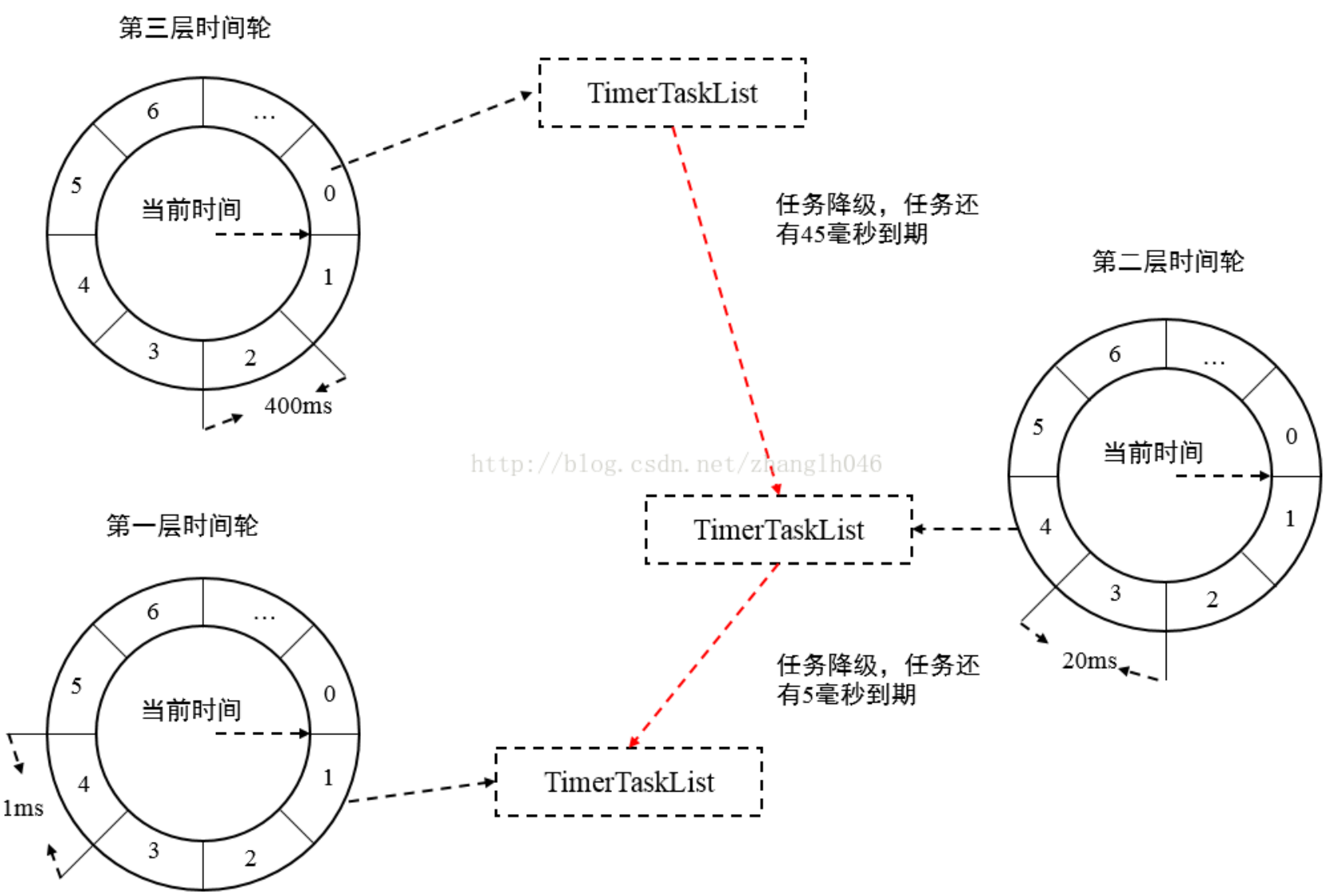
如上图所示：假设编号为0的时间格或者桶保存着到期时间为t,每一个tick的持续时间（tickDuration）为20ms，在这个格子里只能保存着到期时间为[t~t+20]ms的任务，任务到底放在哪一个时间格或者桶里面根据不同的场景可以有不同的算法，假设时间轮的时间格有n个，到期时间为m(ms),那么计算公式m%n = 所在的时间格或者桶，比如n=10,m=34ms,那么他所在桶或者时间格是4

当任务到期时间超出了当前时间所表示的时间范围时，就会尝试加到上一层时间轮，如下图所示：

其中第一层时间轮每个时间格是1ms,整个时间轮跨度是20ms，指针当前时间表示的时间是currentTime，则该时间轮跨度为currentTime

~currentTime+20，只有时间在这段范围内任务才能添加到该层时间轮等待到期。到期时间超出[currentTime~currentTime+20]这个时间范围的任务会尝试添加到上级时间轮中，通过逐层向上级尝试最终找到合适的时间轮层级

整个时间轮表示的时间跨度是不变的，随着指针的不断后移，当前时间轮能处理的时间段也在不断后移，新来的TimerTaskEntry会复用原来的已经到期的TimerTaskList，如下图所示，第一层时间轮跨度始终为20ms,指针表示的时间在不段后移。当指针指向0是时间格的时候，假设currentTime = 100，指向第三个时间格，此时指针表示的时间为当前时间104ms,整个时间轮表示的时间段是[104~ 124]，但是该时间轮的时间跨度依然是20ms。此时间轮中编号为2的时间格表示的时间不再是102~103,而是123~124



假设现在有一个任务在445ms后执行，默认情况下，各个层级的时间轮的时间格个数为20，第一层时间轮每一个时间格跨度为1ms,整个时间轮跨度为20ms,跨度不够。第二层时间轮每一个时间格跨度为20ms,整个时间轮跨度为400ms,跨度依然不够，第三层时间轮每一个时间格跨度为400ms,整个时间轮跨度为8000ms，现在跨度够了，此任务就放在第三层时间轮的第一个时间格对应的TimerTaskList，等待被执行，此TimerTaskList到期时间是400ms,随着时间的流逝，当此TimerTaskList到期时，距离该任务到期时间还有45ms，不能执行该任务，我们将重新提交到时间轮，此时第一层时间轮跨度依然不够，不能执行任务，第二层时间轮时间格跨度为20，整个世间轮跨度为400，跨度足够，放在第三个时间格等待执行，如此往复几次，高层时间轮最终会慢慢移动到低层时间轮上，最终任务到期执行。

一 重要属性

buckets : Array.tabulate[TimerTaskList] 类型，其每一个项都对应时间轮中一个时间格，用于保存TimerTaskList的数组

tickMs： Long 当前时间轮中一个时间格表示的时间跨度

wheelSize: Int 当前时间轮的大小也就是总的时间格数量

taskCounter：AtomicInteger 各层级时间轮中任务的总数

startMs：Long 当期时间轮的创建时间

queue:DelayQueue[TimerTaskList] 整个层级的时间轮公用一个任务队列，其元素类型是TimerTaskList

currentTime：时间轮的指针，将整个时间轮划分为到期部分和未到期部分。在初始化的时候，currentTime被修剪成tickMs的倍数startMs - (startMs % tickMs)

interval：Long 当前时间轮的时间跨度即tickMs * wheelSize,当前时间轮只能处理时间范围在currentTime~currentTime+tickMs*WheelSize之间的定时任务，超过这个范围则需要添加任务到上层时间轮

overflowWheel: TimingWheel 上层时间轮的引用

二 核心方法

2.1 addOverflowWheel 主要用于创建上层时间轮

```
private[this] def addOverflowWheel(): Unit = {
  synchronized {
    if (overflowWheel == null) {
      overflowWheel = new TimingWheel(
        tickMs = interval, // 上层时间轮的时间格跨度等于下一层时间轮的跨度
        wheelSize = wheelSize,// 大小不变
        startMs = currentTime,// 初始化当前时间轮的创建时间
        taskCounter= taskCounter,//时间轮中任务的总数
        queue
      )
    }
  }
}
```

2.2 add 向时间轮中添加定时任务，同时也会检测添加的任务是否已经到期

```
def add(timerTaskEntry: TimerTaskEntry): Boolean = {
  // 获取定时任务的超时时间戳
  val expiration = timerTaskEntry.expirationMs
  // 如果任务已经被取消
  if (timerTaskEntry.cancelled) {
    false // 返回添加失败
  } else if (expiration < currentTime + tickMs) { // 如果时间指针现在指向的时间
    // 举个例子：currentTime=102，时间格跨度为10ms,那么假设添加的任务超时时间戳为10
    false // 返回添加失败
  } else if (expiration < currentTime + interval) {// 如果时间指针现在指向的时
    // 然后把当前任务放进循环数组里面
    // 得到任务到期时间戳/时间格跨度的余数
    val virtualId = expiration / tickMs
    // 获取放在哪一个桶里 (到期时间戳/时间格跨度)%时间轮跨度
    val bucket = buckets((virtualId % wheelSize.toLong).toInt)
    bucket.add(timerTaskEntry)

    // 设置时间格的到期时间
    if (bucket.setExpiration(virtualId * tickMs)) {
      /*
       * 整个时间轮表示的跨度是不变的，随着指针的后移，当前时间轮能够处理的时间段也在不
       * TimerTaskList，此时你需要重新设置TimerTaskList的到期时间，并将桶重新入队
       */
      queue.offer(bucket)
    }
    true
  } else { // 如果超出了时间的跨度范围，则将其添加到上层时间轮来处理
    if (overflowWheel == null) addOverflowWheel()
    overflowWheel.add(timerTaskEntry)
  }
}
```

2.3 advanceClock 尝试推进当前时间轮的指针，同时也会尝试推进上层时间轮的指针，随着当前时间轮的不断推进，上层时间轮指针早晚会被推进成功

```
def advanceClock(timeMs: Long): Unit = {
  // 尝试移动指针currentTime
  if (timeMs >= currentTime + tickMs) {
    currentTime = timeMs - (timeMs % tickMs)

    // 尝试秃顶上层时间轮指针currentTime
    if (overflowWheel != null) overflowWheel.advanceClock(currentTime)
  }
}
```