# 11.定时器库

定时器库为DPDK执行单元提供定时器服务，使得执行单元可以为异步操作执行回调函数。定时器库的特性如下：

* 定时器可以周期执行，也可以执行一次。
* 定时器可以在一个核心加载并在另一个核心执行。但是必须在调用rte\_timer\_reset()中指定它。
* 定时器提供高精度（取决于检查本地核心的定时器到期的rte\_timer\_manage()的调用频率）。
* 如果应用程序不需要，可以在编译时禁用定时器，并且程序中不调用rte\_timer\_manage()来提高性能。

定时器库使用rte\_get\_timer\_cycles()获取高精度事件定时器（HPET）或CPU时间戳计数器（TSC）提供的可靠时间参考。

该库提供了添加，删除和重新启动定时器的接口。API基于BSD callout()，可能会有一些差异。详细请参考callout手册。

## 11.1.实现细节

定时器以每个逻辑核为基础进行跟踪，一个逻辑核上要维护的所有挂起的定时器，按照定时器到期顺序插入到跳跃表数据结构。

所使用的跳跃表有十个层，表中的每个条目都以1/4的概率显示在每个层上。这意味着所有条目都存在于第0层中，每4个条目中的1个条目存在于第一层，每16个中1个条目存在于第2层，等等。同时，这意味着从逻辑核的定时器列表中添加和删除条目可以在log(n)时间内完成，最多4 ^ 10个条目，即每个逻辑核约有1,000,000个定时器。

定时器结构包含一个称为状态的特殊字段，它是定时器状态（stopped，pending，running，config）及其所有者（lcore id）的联合体。根据定时器状态，我们可以知道定时器当前是否存在于列表中：

* STOPPED：没有所有者，不再链表中。
* CONFIG：由一个逻辑核持有，其他逻辑核不能修改，是否存在于跳表中取决于以前的状态。
* PENDING：由一个逻辑核持有，当前在跳表中。
* RUNNING：由一个逻辑核持有，当前在跳表中，不能由其他逻辑核修改。

不允许在定时器处于CONFIG或RUNNING状态时复位或停止定时器。当修改定时器的状态时，应使用CAP指令来保证状态修改操作（状态+所有者）是原子操作。

在rte\_timer\_manage()函数里面，跳跃表作为常规的链表，通过沿着包含所有计时器条目的第0层链表迭代，直到遇到尚未到期的条目为止。当列表中有条目，但是没有任何条目定时器到期时，为了提高性能，第一个定时器条目的到期时间保存在每个逻辑和计时器列表结构本身内部。在64位平台上，可以直接检查该值，而无需对整个结构进行锁定。（由于到期时间维持为64位值，所以在32位平台上无法直接对该值进行检查，而不使用（CAS）指令或使用锁机制，因此，一旦数据结构被上锁，此额外的检查将被跳过。）在64位和32位平台上，在调用逻辑核的计时器列表为空的情况下，对rte\_timer\_manage（）的调用将直接返回而不进行锁定。=

## 11.2.用例

定时器库用于定期调用，如垃圾收集器或某些状态机（ARP，桥接等）。

## 11.3.参考

* [callout manual](https://link.jianshu.com?t=https://www.daemon-systems.org/man/callout.9.html" \t "_blank) ：唤醒功能，提供定时器到期执行的功能。
* [HPET](https://link.jianshu.com?t=https://en.wikipedia.org/wiki/High_Precision_Event_Timer" \t "_blank) ：有关高精度事件定时器（HPET）的信息。