(19)中华人民共和国国家知识产权局

(12)发明专利申请

(10)申请公布号 CN 104391732 A
(43)申请公布日 2015.03.04

(21)申请号 201410668924.9

(22)申请日 2014.11.20

(71)申请人 中国船舶重工集团公司第七二六研究所
地址 201108 上海市闵行区金都路 5200 号

(72)发明人 姜洪宇 杜啸晓 刘杰 张孝华

(74)专利代理机构 上海汉声知识产权代理有限公司 31236
代理人 郭国中

(51)Int.Cl.
G06F 9/45(2006.01)

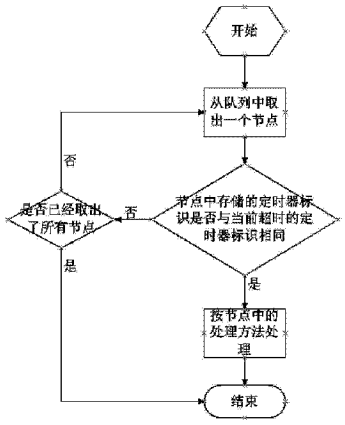
权利要求书1页 说明书4页 附图2页

(54)发明名称

软件多定时器动态管理方法

(57)摘要

本发明提供了一种软件多定时器动态管理方法，首先，定义数据结构，存储定时器标识和定时器超时处理方法；然后，定义队列，存储多定时器信息，每打开一个定时器，将此定时器的标识和处理方法放到队列中；每关闭一个定时器，就将此定时器的标识和处理方法从队列中删除；开启定时器，将定时器信息加入队列；从队列中取出一个节点，判断节点中存储的定时器标识是否与当前超时的定时器标识相同，若相同则该节点对应的定时器超时，按照节点中对应的处理方法处理；否则，判断是否已经取出所有节点，若是，则结束操作；否则，重新提取节点判断。本发明减少了多定时器动态开启关闭时对未开启定时器的非必要查询，减少了 CPU 开销，提高了系统运行速度。



CN 104391732 A

1. 一种软件多定时器动态管理方法，其特征在于，包括以下步骤：

步骤 1：定义数据结构 TimerData，存储定时器标识 TimerID 和定时器超时处理方法 TimerFun；

步骤 2：定义队列 TimerQueue，存储多定时器信息，该队列的节点的数据结构形式为 TimerData，每打开一个定时器，将此定时器的标识和处理方法放到队列中，队列的节点数加 1，；每关闭一个定时器，就将此定时器的标识和处理方法从队列中删除，队列的节点数减 1；

步骤 3：开启定时器，将定时器信息加入队列；

步骤 4：从队列中取出一个节点，判断节点中存储的定时器标识是否与当前超时的定时器标识相同，若相同则执行步骤 5，否则，若不同，则执行步骤 6；

步骤 5：节点中存储的定时器标识与当前超时的定时器标识相同，该节点对应的定时器超时，按照节点中对应的处理方法处理；

步骤 6：判断是否已经取出所有节点，若是，则结束操作；否则，重复步骤 4。

2. 根据权利要求 1 所述的软件多定时器动态管理方法，其特征在于，步骤 2 中，所述队列 TimerQueue 中定义 TimerQueue[n] 表示 TimerQueue 的第 N 个节点；

TimerQueue[n].TimerID 表示 TimerQueue 的第 N 个节点中存储的定时器标识；

TimerQueue[n].TimerFun 表示 TimerQueue 的第 N 个节点中存储的定时器超时处理方法。

3. 根据权利要求 2 所述的软件多定时器动态管理方法，其特征在于，步骤 4 通过定时器响应函数判断节点中存储的定时器标识是否与当前超时的定时器标识相同，具体为：在定时器响应函数中，依次找出队列中所有的定时器标识 TimerQueue[n].TimerID，与当前定时器响应函数中的定时器标识 nID 相比较，如果相同，则当前为 TimerQueue[n].TimerID 所对应的定时器超时。

软件多定时器动态管理方法

技术领域

[0001] 本发明涉及计算机领域，具体地，涉及一种软件多定时器动态管理方法。

背景技术

[0002] 定时器，从字面理解，就是一个时钟，在设定了定时时间之后，经历了设定的时间之后，就会通知设置定时器的人员，时间已经到了，可以进行下一步工作了，在日常生活中，我们经常能接触到定时器，例如，半自动洗衣机中就有定时器，从开始洗衣，旋动定时旋钮，定时器就开启，到定时旋钮回到0，定时器时间到，洗衣结束，可以进行下一步放掉脏水等工作了。我们这里讲的定时器是在做计算机软件编程时，使用到的一种软件定时器，计算机软件逻辑也经常会出现需要设定一个或多个时间，在时间到了之后开始做下一步工作。例如全自动洗衣机，里面有一个计算机控制板，这个计算机控制板中运行的计算机软件就有我们所说的定时器，软件会自动设置定时器，开始第一步工作，在定时器时间到了之后在设置定时器开始第二步工作等。

[0003] 在嵌入式系统中，使用Microsoft Visual C++编写软件代码同时对多个通信节点进行通信超时监视时，经常会使用到定时器，一般定时器的使用方法是使用SetTimer(UINT nIDEvent,UINT nElapse,void(CALLBACK EXPORT*lpfnTimer)(HWND,UINT,UINT,DWORD))函数，第一个参数表示设置的定时器的标识，在定时器响应函数中做定时器超时的处理。一般需要首先判断当前超时的定时器是否是等待需要处理的定时器，如果是，则处理，如果不是则不处理。

[0004] 如图1所示，在Microsoft Visual C++中，常用的定时器方案为，使用SetTimer(UINT nIDEvent,UINT nElapse,void(CALLBACK EXPORT*lpfnTimer)(HWND,UINT,UINT,DWORD))函数设置定时器，一般指定定时器的标识，在时间响应函数OnTimer(UINT nIDEvent)中，判断当前超时的定时器是否为设置的定时器，即OnTimer(UINT nIDEvent)中nIDEvent是否等于SetTimer(UINT nIDEvent,UINT nElapse,void(CALLBACK EXPORT*lpfnTimer)中的nIDEvent。如果相同则处理。如果当前定时器需要关闭，则调用函数KillTimer(int nIDEvent)关闭定时器标识为nIDEvent的定时器，所以在定时器开启时，必须记录开启的定时器标识nIDEvent，在后面关闭定时器时用。这种做法的缺点是：所能设的定时器不能太多，否则，当一个线程的定时器过多时，需要在定时器处理函数中做大量的判断，判断当前是哪一个定时器超时，这种判断所带来的CPU开销在资源有限的嵌入式系统中是非常可观的，而且这种做法只能设置预先规定好定时器标识符的定时器，对于数量随时变化，各个定时器的开启和关闭也随时变化的情况，这种做法就非常难以实现。

[0005] 常用定时器的语句举例如下：

[0006] SetTimer(101,300,NULL)；

[0007] 设置定时器标识为101的定时器，定时时间为300毫秒。

[0008] SetTimer(200,50,NULL)；

[0009] 设置定时器标识为 200 的定时器，定时时间为 50 毫秒。

[0010] 本例程当前时刻开启了两个定时器，但本例程中，假设可能用到 100 个定时器。则一般定时器响应函数如下：

[0011]

```
void CMainPrjDlg::OnTimer(UINT nIDEvent)
{
    switch(nIDEvent)
    {
        case 101:
            DealTime101();
            break;
        case 102:
            DealTime102();
            break;
        .........
        case 200:
            DealTime200();
            break;
        default:
            break;
    }
}
```

[0012]

[0013] 上面描述的使用定时器的方案举例中，有 100 个定时器，则需在 OnTimer(UINT nIDEvent) 函数中判断当前是哪一个定时器时间到了，而且，定时器越多，则需要判断的次数可能就越多，如果当前时间超时的定时器为定时器 200，则在 OnTimer(UINT nIDEvent) 函数中，需要判断 100 次才能判断出当前的定时器。而且，如果当前最短的定时时间为 50ms，则 50ms 之内，计算机至少要进入 OnTimer(UINT nIDEvent) 函数一次，做 100 次判断，CPU 的开销很大。

[0014] 本发明要解决在使用 Microsoft Visual C++ 开发软件过程中，多定时器情况下，定时器数量可变，各个定时器开启关闭状态也随时变化时，造成的定时器难以管理，CPU 开销大的问题。

发明内容

[0015] 针对现有技术中的缺陷，本发明的目的是提供一种软件多定时器动态管理方法。

[0016] 根据本发明的一个方面，提供一种软件多定时器动态管理方法，包括以下步骤：

[0017] 步骤 1：定义数据结构 TimerData，存储定时器标识 TimerID 和定时器超时处理方法 TimerFun；

[0018] 步骤 2：定义队列 TimerQueue，存储多定时器信息，该队列的节点的数据结构形式为 TimerData，每打开一个定时器，将此定时器的标识和处理方法放到队列中，队列的节点数加 1，；每关闭一个定时器，就将此定时器的标识和处理方法从队列中删除，队列的节点数减 1；

[0019] 步骤 3：开启定时器，将定时器信息加入队列；

[0020] 步骤 4：从队列中取出一个节点，判断节点中存储的定时器标识是否与当前超时的定时器标识相同，若相同则执行步骤 5，否则，若不同，则执行步骤 6；

[0021] 步骤 5：节点中存储的定时器标识与当前超时的定时器标识相同，该节点对应的定时器超时，按照节点中对应的处理方法处理；

[0022] 步骤 6：判断是否已经取出所有节点，若是，则结束操作；否则，重复步骤 4。

[0023] 优选地，步骤 2 中，队列 TimerQueue 中定义 TimerQueue[n] 表示 TimerQueue 的第 N 个节点；TimerQueue[n].TimerID 表示 TimerQueue 的第 N 个节点中存储的定时器标识；TimerQueue[n].TimerFun 表示 TimerQueue 的第 N 个节点中存储的定时器超时处理方法。

[0024] 优选地，步骤 4 通过定时器响应函数判断节点中存储的定时器标识是否与当前超时的定时器标识相同，具体为：在定时器响应函数中，依次找出队列中所有的定时器标识 TimerQueue[n].TimerID，与当前定时器响应函数中的定时器标识 nID 相比较，如果相同，则当前为 TimerQueue[n].TimerID 所对应的定时器超时。

[0025] 与现有技术相比，本发明具有如下的有益效果：本发明通过使用队列记录当前开启的定时器标识和定时器处理方法来避开传统的静态查询的方式，减少在多定时器动态开启关闭时对未开启定时器的非必要查询，减少了 CPU 开销，提高了系统的运行速度。

附图说明

[0026] 通过阅读参照以下附图对非限制性实施例所作的详细描述，本发明的其它特征、目的和优点将会变得更明显：

[0027] 图 1 为常规多定时器响应流程图；

[0028] 图 2 为本发明软件多定时器动态管理方法流程原理图。

具体实施方式

[0029] 下面结合具体实施例对本发明进行详细说明。以下实施例将有助于本领域的技术人员进一步理解本发明，但不以任何形式限制本发明。应当指出的是，对本领域的普通技术人员来说，在不脱离本发明构思的前提下，还可以做出若干变形和改进。这些都属于本发明的保护范围。

[0030] 一种软件多定时器动态管理方法，包括以下步骤：

[0031] 步骤 1：定义数据结构 TimerData，存储定时器标识 TimerID 和定时器超时处理方法 TimerFun。

[0032] 步骤 2 :定义队列 TimerQueue,存储多定时器信息,该队列的节点的数据结构形式为 TimerData,每打开一个定时器,将此定时器的标识和处理方法放到队列中,队列的节点数加 1, ;每关闭一个定时器,就将此定时器的标识和处理方法从队列中删除,队列的节点数减 1。

[0033] 队列刚定义时应该是一个空的队列,即节点数为 0。进一步地,定义 TimerQueue[n] 表示 TimerQueue 的第 N 个节点, TimerQueue[n].TimerID 表示 TimerQueue 的第 N 个节点中存储的定时器标识, TimerQueue[n].TimerFun 表示 TimerQueue 的第 N 个节点中存储的定时器超时处理方法。

[0034] 步骤 3 :开启定时器,将定时器信息加入队列。

[0035] 步骤 4 :从队列中取出一个节点,判断节点中存储的定时器标识是否与当前超时的定时器标识相同,若相同则执行步骤 5,否则,若不同,则执行步骤 6。

[0036] 步骤 5 :节点中存储的定时器标识与当前超时的定时器标识相同,该节点对应的定时器超时,按照节点中对应的处理方法处理。

[0037] 具体地,步骤 4 中通过定时器响应函数判断节点中存储的定时器标识是否与当前超时的定时器标识相同,在定时器响应函数中,依次找出队列中所有的定时器标识 TimerQueue[n].TimerID,与当前定时器响应函数中的定时器标识 nID 相比较,如果相同,则说明当前为 TimerQueue[n].TimerID 所对应的定时器超时,则步骤 5 中使用对应的处理方法 TimerQueue[n].Fun 处理。

[0038] 步骤 6 :判断是否已经取出所有节点,若是,则结束操作 ;否则,重复步骤 4。

[0039] 在本发明的一个较佳实施例中, 最多可能使用到 64 个定时器,每个定时器的定时时间有 100ms,300ms,500ms 三种,本实例的大部分时间只开启一个定时器,而且,每个定时器的处理方法都不尽相同,采用本发明的方案,大部分时间,定时器响应函数中,每次只需判断一次就能完成相应处理了,相比传统的定时器处理方法,速度提高了很多。

[0040] 以上对本发明的具体实施例进行了描述。需要理解的是,本发明并不局限于上述特定实施方式,本领域技术人员可以在权利要求的范围内做出各种变形或修改,这并不影响本发明的实质内容。

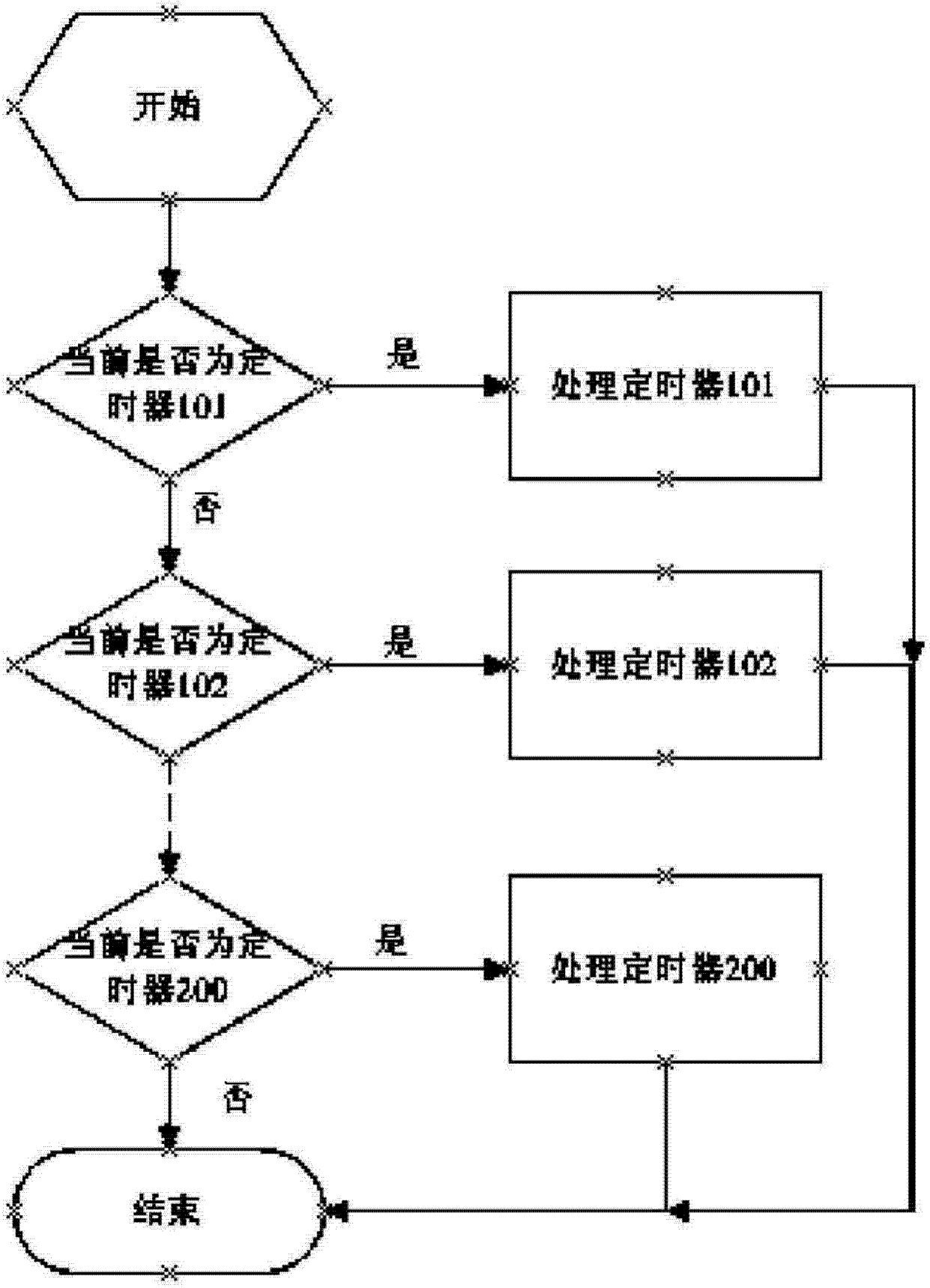

开始
当前是否为定时器101
是
处理定时器101
否
当前是否为定时器102
是
处理定时器102
当前是否为定时器200
是
处理定时器200
否
结束

图 1

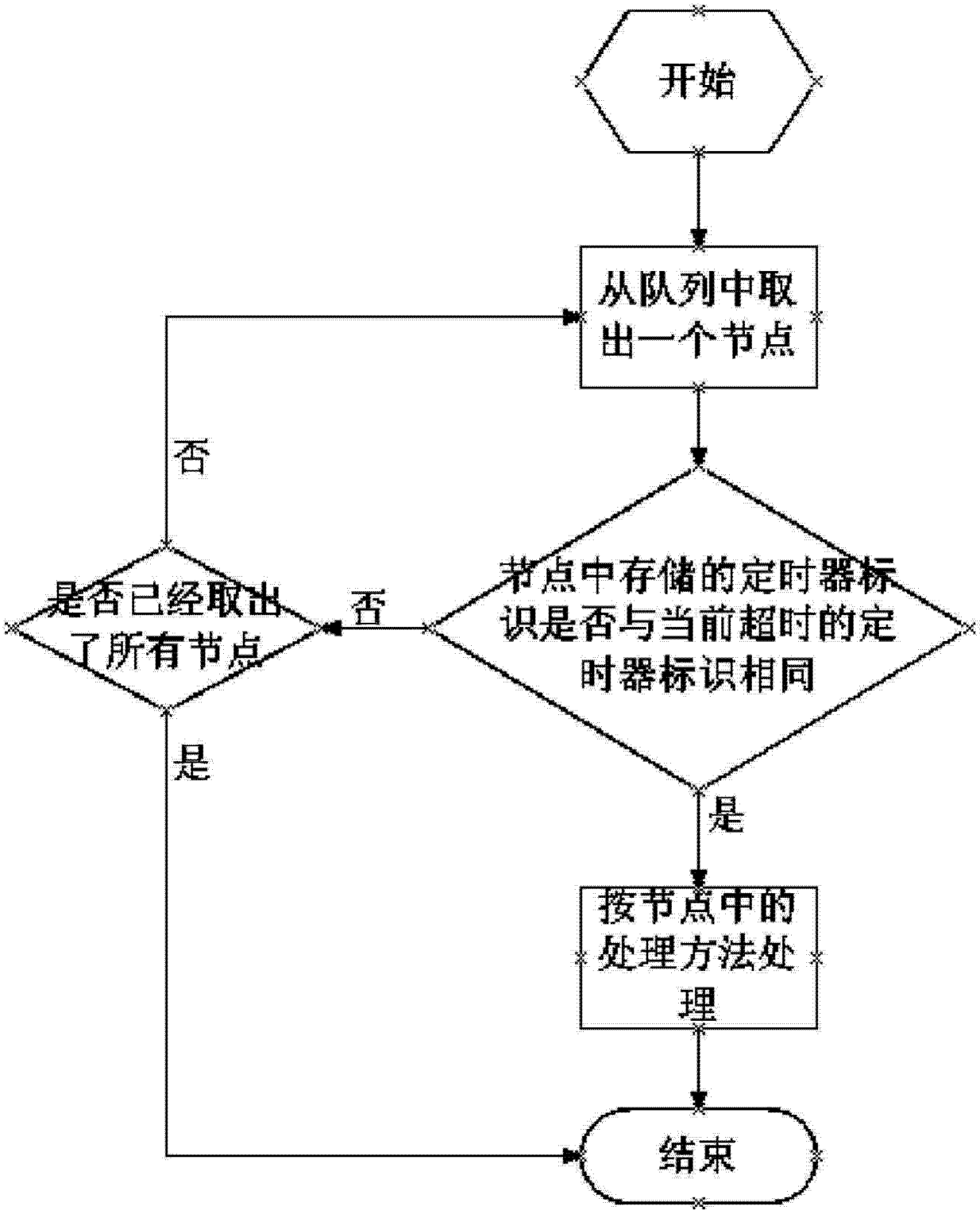

开始
从队列中取出一个节点
节点中存储的定时器标识是否与当前超时的定时器标识相同
否
是否已经取出了所有节点
否
是
是
按节点中的处理方法处理
结束

图 2