**[C++11 并发编程] 17 超时等待 - clock和duration**

原创 2015年09月09日 14:59:32

* 标签：
* [C++](http://so.csdn.net/so/search/s.do?q=C++&t=blog) /
* [duration](http://so.csdn.net/so/search/s.do?q=duration&t=blog) /
* [wait\_for](http://so.csdn.net/so/search/s.do?q=wait_for&t=blog) /
* [ratio](http://so.csdn.net/so/search/s.do?q=ratio&t=blog)

之前我们看到的所有等待机制都是不会超时的，也就是说，等待某个同步事件的线程会一直挂起。有些情况下，我们希望设置一个最长等待时间，使得程序可以继续与用户进行交互，使得用户可以取消这个操作。我们先来看看C++11提供的时钟类clock：

**clock**

clock提供了如下四种信息：

* 当前时间
* 存放从clock获取到的时间的类型
* 时钟每个tick的周期
* 每个tick的周期是否固定，固定则为“steady”时钟

**Member functions**

|  |  |
| --- | --- |
| [**now**](http://en.cppreference.com/w/cpp/chrono/system_clock/now)  [static] | returns a [std::chrono::time\_point](http://en.cppreference.com/w/cpp/chrono/time_point) representing the current point in time  (public static member function) |
| [**to\_time\_t**](http://en.cppreference.com/w/cpp/chrono/system_clock/to_time_t)  [static] | converts a system clock time point to [std::time\_t](http://en.cppreference.com/w/cpp/chrono/c/time_t)  (public static member function) |
| [**from\_time\_t**](http://en.cppreference.com/w/cpp/chrono/system_clock/from_time_t)  [static] | converts [std::time\_t](http://en.cppreference.com/w/cpp/chrono/c/time_t) to a system clock time point  (public static member function) |

下面是一个简单的实例，计算不同长度向量中元素的和并打印程序执行的时间：

**[cpp]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. #include <iostream>
2. #include <vector>
3. #include <numeric>
4. #include <chrono>
6. volatile int sink;
7. int main()
8. {
9. for (auto size = 1ull; size < 1000000000ull; size \*= 100) {
10. // record start time
11. auto start = std::chrono::system\_clock::now();
12. // do some work
13. std::vector<int> v(size, 42);
14. sink = std::accumulate(v.begin(), v.end(), 0u); // make sure it's a side effect
15. // record end time
16. auto end = std::chrono::system\_clock::now();
17. std::chrono::duration<double> diff = end-start;
18. std::cout << "Time to fill and iterate a vector of "
19. << size << " ints : " << diff.count() << " s\n";
20. }
21. }

程序执行结果如下：

**[plain]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. Time to fill and iterate a vector of 1 ints : 2.93e-06 s
2. Time to fill and iterate a vector of 100 ints : 2.94e-06 s
3. Time to fill and iterate a vector of 10000 ints : 8.9962e-05 s
4. Time to fill and iterate a vector of 1000000 ints : 0.00859845 s
5. Time to fill and iterate a vector of 100000000 ints : 0.948915 s

时钟的tick周期可以通过std::ratio<x,y>来指定，一秒tick25下的时钟周期为std::ratio<1,25>，每2.5秒tick一下的时钟周期为std::ratio<5,2>。

tick周期稳定的时钟被称为稳定时钟，它的is\_steady静态成员变量为true。对于非稳定时钟，在本地时钟发生漂移时，会自动进行调整，这就可能导致后执行的now()操作可能比先执行的now()操作得到的时间更小。在多线程环境下，超时操作需要使用稳定的时钟。可以使用std::chrono::steady\_clock来获得稳定的时钟。std::chrono::system\_clock则被称为实时时钟，可以被转换为time\_t值，也可以通过time\_t转换为system\_clock类型。而std::chrono::high\_resolution\_clock则提供了系统能支持的最高精度的时钟。

**duration**

duration用于指定一个时间段，std::chrono::duration<>类模版的第一个参数指定周期的标示类型（比如int, long或者double），第二个参数指定周期的单位（一个单位代表多少秒）。

例如，以分钟为单位的周期定义如下，一分钟为60秒：

**[cpp]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. std::chrono::duration<short, std::ratio<60, 1>>

以毫秒为单位的周期定义如下，一秒钟为1000毫秒：

**[cpp]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. std::chrono::duration<double, std::ratio<1, 1000>>

此外，标准库在std::chrono名字空间还提供了一系列预定义的周期精度，例如：nanoseconds，microseconds，milliseconds，seconds，minutes和hours。这样如果要用精度为分秒的周期，可以使用如下代码：

**[cpp]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. std::duration<double, std::centi>;

下面一个例子中，定义了多种类型的duration，并在它们之间进行转换：

**[cpp]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. #include <iostream>
2. #include <chrono>
4. int main()
5. {
6. using shakes = std::chrono::duration<int, std::ratio<1, 100000000>>;
7. using jiffies = std::chrono::duration<int, std::centi>;
8. using microfortnights = std::chrono::duration<float, std::ratio<12096,10000>>;
9. using nanocenturies = std::chrono::duration<float, std::ratio<3155,1000>>;
11. std::chrono::seconds sec(1);
13. std::cout << "1 second is:\n";
15. std::cout << std::chrono::duration\_cast<shakes>(sec).count()
16. << " shakes\n";
17. std::cout << std::chrono::duration\_cast<jiffies>(sec).count()
18. << " jiffies\n";
19. std::cout << microfortnights(sec).count() << " microfortnights\n";
20. std::cout << nanocenturies(sec).count() << " nanocenturies\n";
21. }

程序执行效果如下：

**[plain]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. 1 second is:
2. 100000000 shakes
3. 100 jiffies
4. 0.82672 microfortnights
5. 0.316957 nanocenturies

基于duration，等待一个期望35毫秒的实现如下：

**[cpp]** [view plain](http://blog.csdn.net/yamingwu/article/details/48294927) [copy](http://blog.csdn.net/yamingwu/article/details/48294927)

1. std::future<int> f=std::async(some\_task);
3. if(f.wait\_for(std::chrono::milliseconds(35))==std::future\_status::ready)
4. do\_something\_with(f.get());

等待函数会返回一个状态来标示是超时了还是等待的时间发生了。如例子所示，等待的是一个期望，如果超时了，返回值为std::future\_status::timeout，如果时间发生了，返回值为std::future\_status::ready。

基于duration的等待机制使用的是稳定时钟。