13.13 实现一个计时器¶

问题¶

你想记录程序执行多个任务所花费的时间

解决方案¶

`time` 模块包含很多函数来执行跟时间有关的函数。 尽管如此，通常我们会在此基础之上构造一个更高级的接口来模拟一个计时器。例如：

```
import time

class Timer:
    def __init__(self, func=time.perf_counter):
        self.elapsed = 0.0
        self._func = func
        self._start = None

    def start(self):
        if self._start is not None:
            raise RuntimeError('Already started')
        self._start = self._func()

    def stop(self):
        if self._start is None:
            raise RuntimeError('Not started')
        end = self._func()
        self.elapsed += end - self._start
        self._start = None

    def reset(self):
        self.elapsed = 0.0

    @property
    def running(self):
        return self._start is not None

    def __enter__(self):
        self.start()
        return self

    def __exit__(self, *args):
        self.stop()
```

这个类定义了一个可以被用户根据需要启动、停止和重置的计时器。 它会在 `elapsed` 属性中记录整个消耗时间。 下面是一个例子来演示怎样使用它：

```
def countdown(n):
    while n > 0:
        n -= 1

# Use 1: Explicit start/stop
t = Timer()
t.start()
```

```
t.start()
countdown(1000000)
t.stop()
print(t.elapsed)

# Use 2: As a context manager
with t:
    countdown(1000000)

print(t.elapsed)

with Timer() as t2:
    countdown(1000000)
print(t2.elapsed)
```

讨论¶

本节提供了一个简单而实用的类来实现时间记录以及耗时计算。 同时也是对使用with语句以及上下文管理器协议的一个很好的演示。

在计时中要考虑一个底层的时间函数问题。一般来说， 使用 `time.time()` 或 `time.clock()` 计算的时间精度因操作系统的不同会有所不同。 而使用 `time.perf_counter()` 函数可以确保使用系统上面最精确的计时器。

上述代码中由 `Timer` 类记录的时间是钟表时间，并包含了所有休眠时间。 如果你只想计算该进程所花费的CPU时间，应该使用 `time.process_time()` 来代替：

```
t = Timer(time.process_time)
with t:
    countdown(1000000)
print(t.elapsed)
```

`time.perf_counter()` 和 `time.process_time()` 都会返回小数形式的秒数时间。 实际的时间值没有任何意义，为了得到有意义的结果，你得执行两次函数然后计算它们的差值。

更多关于计时和性能分析的例子请参考14.13小节。