[public]

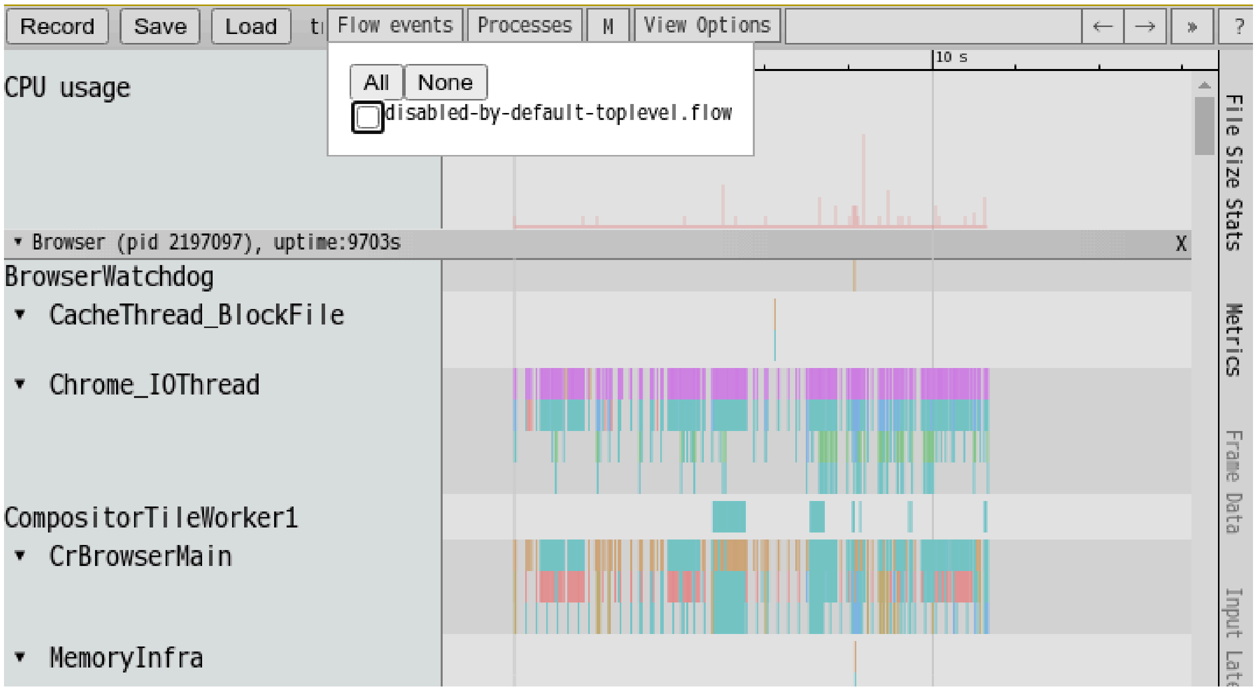
[Code reading] SequenceManager

chikamune@
Status: WIP
Last Update: 2020-08-06

This document describes how SequenceManager works.

The trace logs

Chrome has trace log functionality. You can open chrome://tracing and press the “Record” button, and check the “toplevel.flow” checkbox, and press the “Record” button. As you can see, there are many sub processes, and there are many threads inside the process.



In many cases, each thread has its own SequenceManager, and there is a similar task management mechanism.

The stack traces

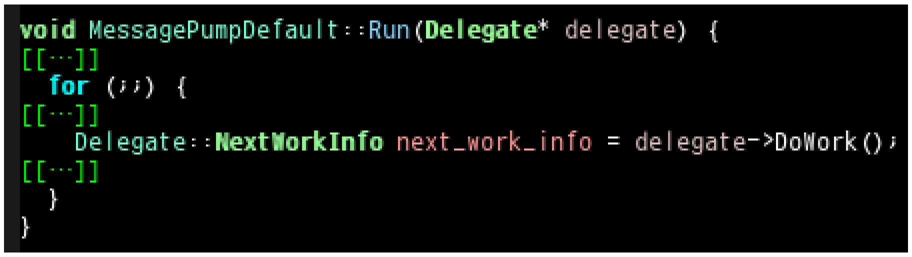
When you check the stack traces, you probably see the common pattern which looks like below. There is a “RunLoop”, and there are function calls of “Run()”, “DoWork()” and “DoWorkImpl()”. This is a common architecture of the SequenceManager.

```
...
```

```
base::TaskAnnotator::RunTask()
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::DoWorkImpl()
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::DoWork()
...
base::sequence_manager::internal::ThreadControllerWithMessagePumpImpl::Run()
base::RunLoop::Run()
...
```

RunLoop::Run()

The actual implementation of RunLoop::Run() is implemented in ThreadControllerWithMessagePumpImpl class. It calls base::MessagePump::Run(Delegate* delegate) and this function creates an infinite loop to run the tasks.

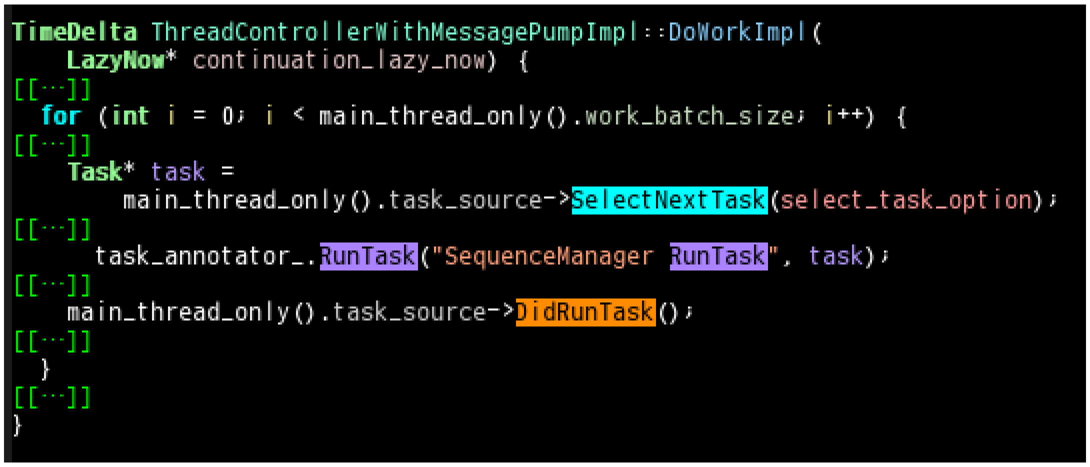


```
void MessagePumpDefault::Run(Delegate* delegate) {
[[…]]
  for (;;) {
[[…]]
    Delegate::NextWorkInfo next_work_info = delegate->DoWork();
[[…]]
  }
}
```

DoWork()

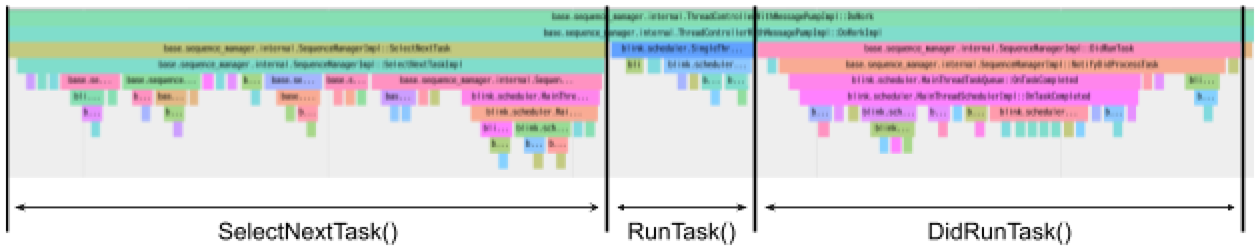
The DoWork{Impl} function runs these steps.

1. SelectNextTask()
   This function selects the next task.
2. RunTask()
   Run the selected task.
3. DidRunTask()
   This function notifies that the previously obtained from SelectNextTask() has been completed.



```
TimeDelta ThreadControllerWithMessagePumpImpl::DoWorkImpl(
    LazyNow* continuation_lazy_now) {
[[…]]
  for (int i = 0; i < main_thread_only().work_batch_size; i++) {
[[…]]
    Task* task =
        main_thread_only().task_source->SelectNextTask(select_task_option);
[[…]]
      task_annotator_.RunTask("SequenceManager RunTask", task);
[[…]]
    main_thread_only().task_source->DidRunTask();
[[…]]
  }
[[…]]
}
```

The below trace shows that there are three blocks under the DoWork() process.

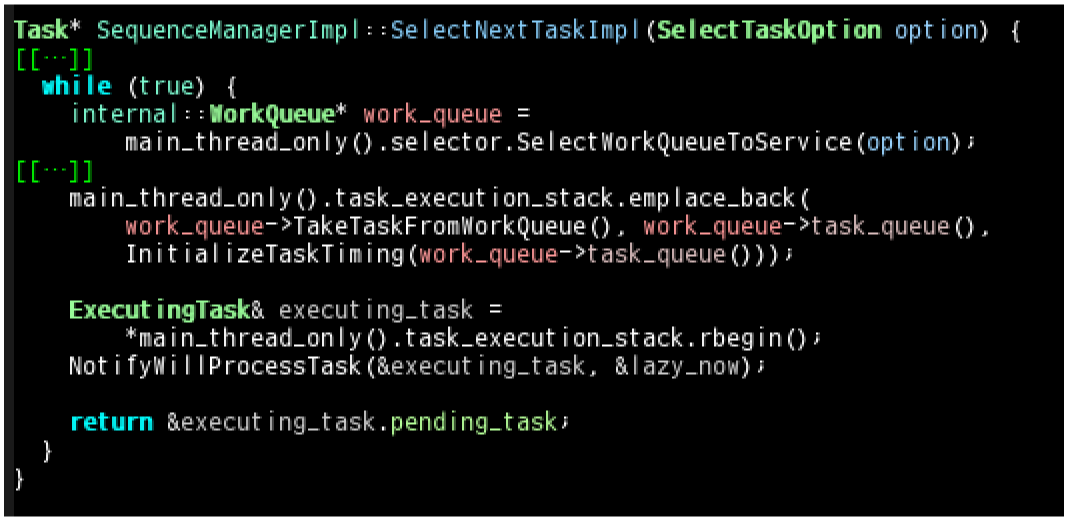


So, basically it selects the next task, and runs the task, and calls DidRunTask() repeatedly.

[Caveats]
The trace log above looks like that SelectNextTask() and DidRunTask() takes longer time than RunTask(). But this is not correct. Writing trace logs is taking time.

SequenceManager::SelectNextTask()

As described above, SelectNextTask() selects the next task. The SelectNextTask() function is implemented in the SequenceManagerImpl class. And the important logic is implemented in SelectNextTaskImpl().



```
Task* SequenceManagerImpl::SelectNextTaskImpl(SelectTaskOption option) {
[[···]]
  while (true) {
    internal::WorkQueue* work_queue =
        main_thread_only().selector.SelectWorkQueueToService(option);
[[···]]
    main_thread_only().task_execution_stack.emplace_back(
        work_queue->TakeTaskFromWorkQueue(), work_queue->task_queue(),
        InitializeTaskTiming(work_queue->task_queue()));

    ExecutingTask& executing_task =
        *main_thread_only().task_execution_stack.rbegin();
    NotifyWillProcessTask(&executing_task, &lazy_now);

    return &executing_task.pending_task;
  }
}
```

SequenceManagerImpl::SelectNextTask() is doing these steps below.

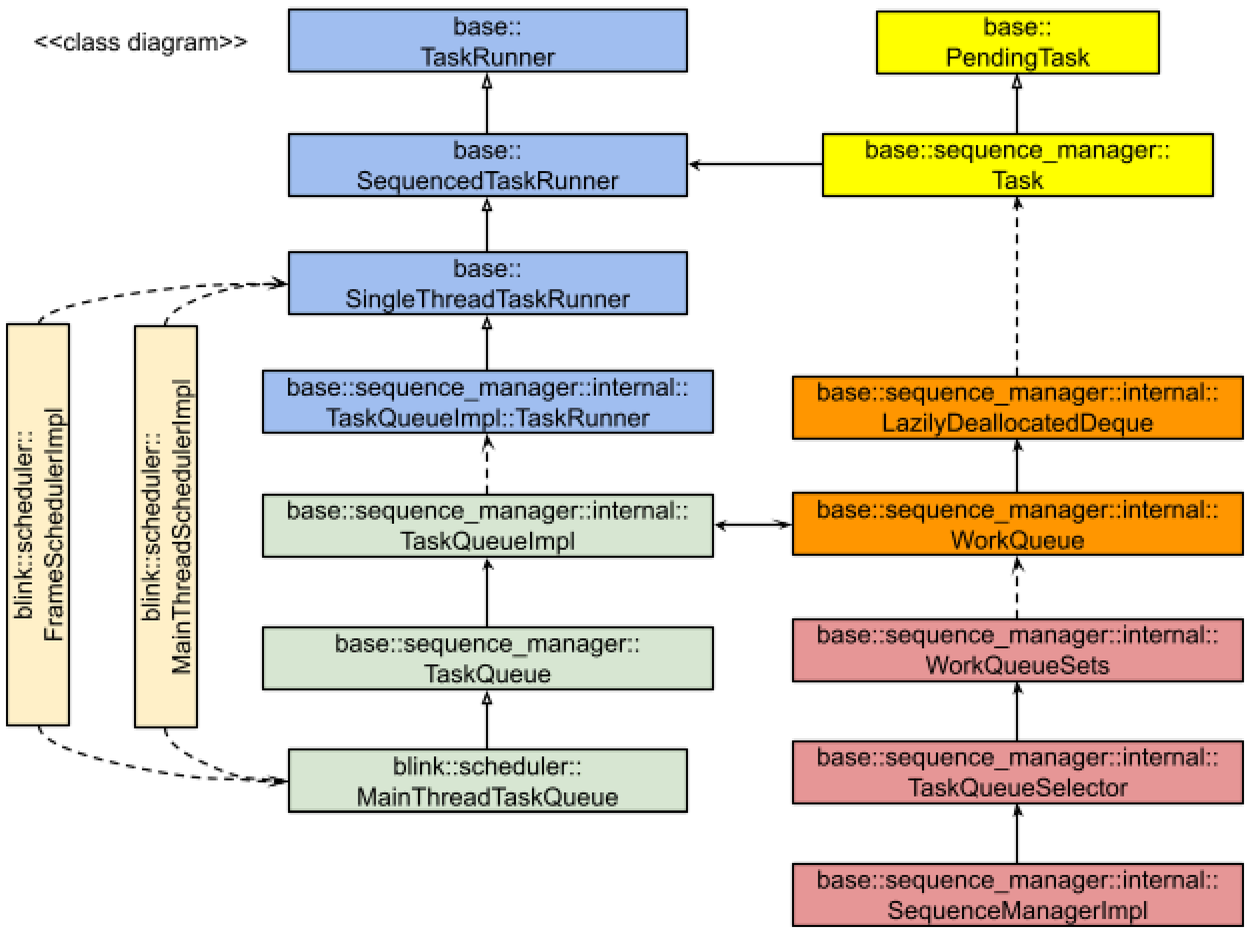
1. Select a WorkQueue for the next Task from WorkQueueSet.
   WorkQueueSet keeps track of which queue (with a given priority) contains the oldest task.
2. Take a task from WorkQueue, and store the task to task_execution_stack.
3. Get an ExecutingTask from task_execution_stack.
4. Return the Task.

WorkQueue

The WorkQueue class keeps track of tasks which are due to run. WorkQueue has LazilyDeallocatedDeque (the alias of this is named as TaskDeque). LazilyDeallocatedDeque is a deque implementation which holds the tasks. LazilyDeallocatedDeque is a special implementation for this purpose. WorkQueue is conceptually referenced by SequenceManager{Impl}, and it's also referenced by TaskQueue{Impl}.

Class diagram

There are related classes around the WorkQueue class. The class diagram below shows the important classes.



Group of Classes	Description
Task	The runnable object which holds a closure object.
WorkQueue	The queue which directly tracks tasks.
SequenceManagerImpl	This class provides SequenceManager::SelectNextTask().
TaskQueue	Almost the same as WorkQueue, but this manages TaskRunner.
TaskRunner	This class provides task posting APIs.
Scheduler	These classes hold TaskQueues and TaskRunners, and control the priorities based on many signals which affect the priorities.

How to post the task

There are two different types of TaskRunners.

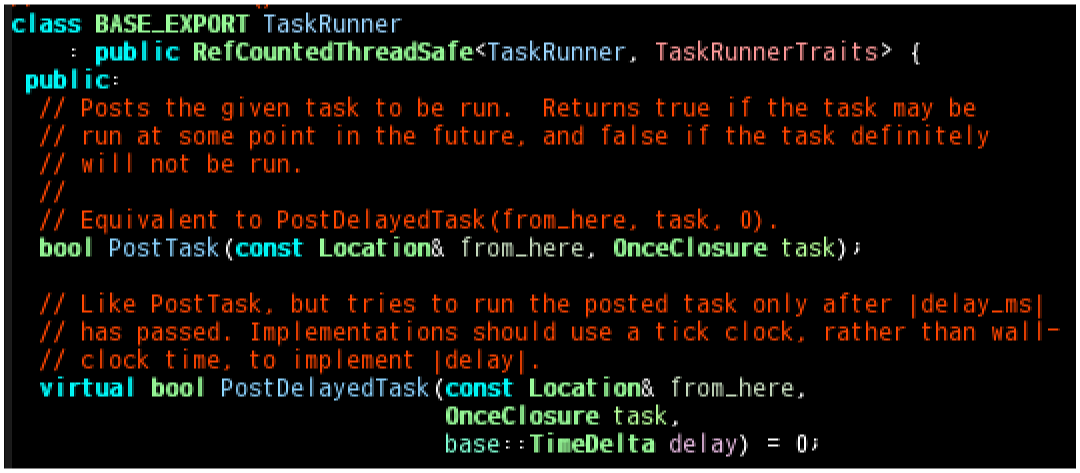
- Thread-global TaskRunner
  Thread-global TaskRunners are held by MainThreadSchedulerImpl.

- Per-frame TaskRunner
  Per-frame TaskRunners are held by FrameSchedulerImpl.

We can get per-frame TaskRunners from FrameScheduler::GetTaskRunner(TaskType). There are several aliases as well.
Aliases:

- LocalFrame::GetTaskRunner
- WebLocalFrame::GetTaskRunner
- RenderFrame::GetTaskRunner
- ExecutionContext::GetTaskRunner

TaskRunner has PostTask() and PostDelayedTask() functions. We can post tasks via these functions. PostTask() just calls PostDelayedTask(from_here, task, /*delay=*/0). The posted OnceClosure will be held by the Task object.



```
class BASE_EXPORT TaskRunner
    : public RefCountedThreadSafe<TaskRunner, TaskRunnerTraits> {
 public:
  // Posts the given task to be run.  Returns true if the task may be
  // run at some point in the future, and false if the task definitely
  // will not be run.
  //
  // Equivalent to PostDelayedTask(from_here, task, 0).
  bool PostTask(const Location& from_here, OnceClosure task);

  // Like PostTask, but tries to run the posted task only after |delay_ms|
  // has passed. Implementations should use a tick clock, rather than wall-
  // clock time, to implement |delay|.
  virtual bool PostDelayedTask(const Location& from_here,
                               OnceClosure task,
                               base::TimeDelta delay) = 0;
```