Better tracing for Blink Scheduler

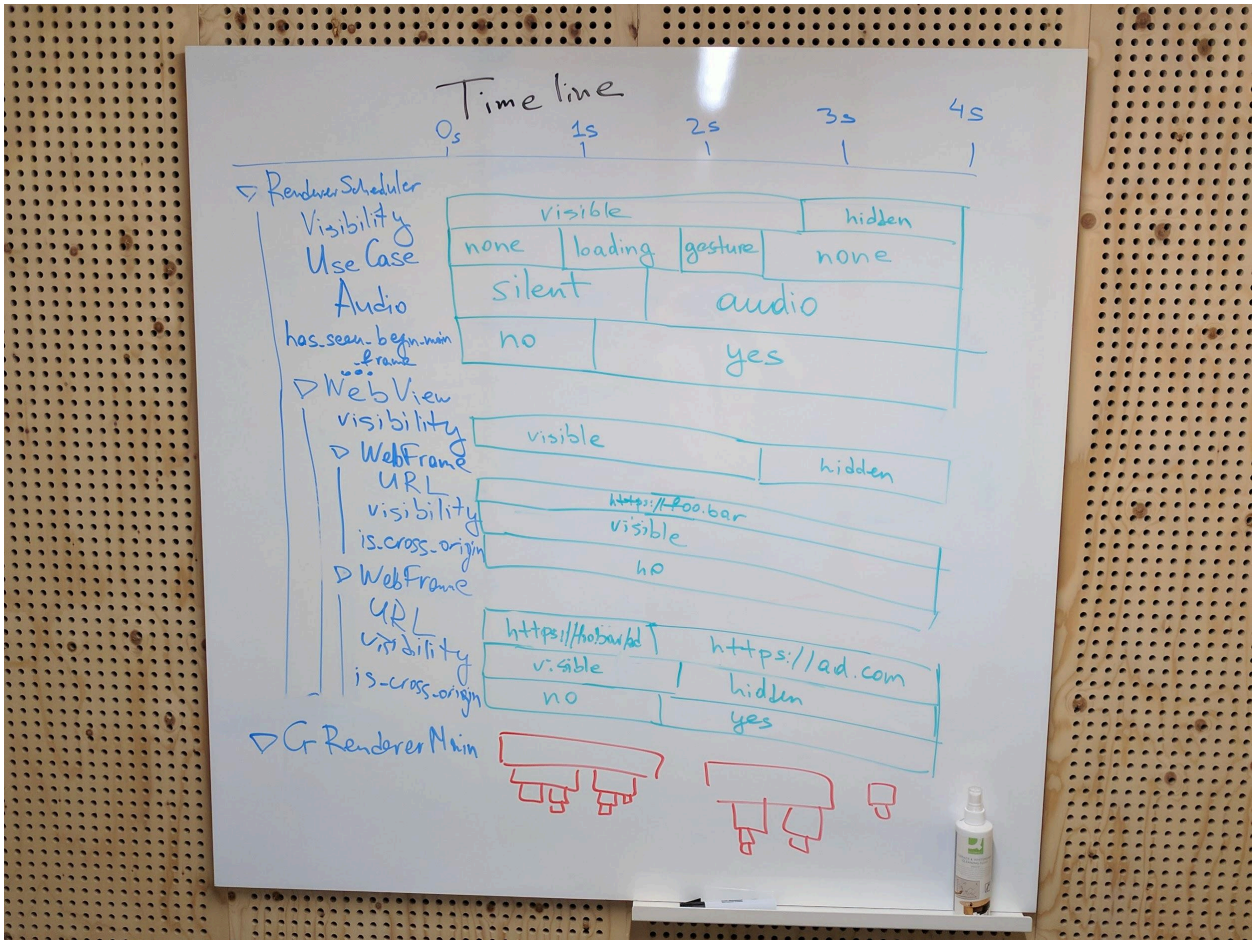
altimin@chromium.org, benjhayden@chromium.org, kraynov@chromium.org
4 September 2017
crbug.com/755128

Current approach to exposing internal scheduler state in tracing relies on snapshotting internal scheduling objects (RendererScheduler, TaskQueueManager). This approach has a number of downsides, including large trace sizes due to duplicated information and a history of a single state (like frame visibility or whether a renderer is playing audio) being hard to read from the trace UI (these values are hidden inside JSON dump of the object state and a manual search through all object dumps is needed to find the place of the change).

This project aims to make scheduling traces easy to read and understand at a glance, exposing the full history of scheduling state for the trace period. This project involves both reworking current approach to tracing in C++ scheduler code and modification of the trace UI to arrange information (i.e. group all information related to a specific frame and expose frame hierarchy).

Desired result

UI mock:



It is proposed to add a custom section to the trace viewer timeline which shows the state of the scheduler and its changes. Granularity can be changed by enabling extra categories:

- renderer.scheduler category provides only basic info (visibility/use case),
- disabled-by-default-renderer.scheduler adds more visual information without increasing trace size (RendererScheduler state and basic frame information — URL and visibility),
- disabled-by-default-renderer.scheduler.verbose adds detailed frame visualisation and object dumps.
- disabled-by-default-renderer.scheduler.debug adds information for queueing tasks.

Milestones

Four new categories

At the moment, three categories are used for tracing scheduler: renderer.scheduler, disabled-by-default-renderer.scheduler and disabled-by-default-renderer.scheduler.debug.

Default renderer.scheduler category includes trace events for the main thread slices, disabled-by-default-renderer.scheduler includes object dumps with most of the information and disabled-by-default-renderer.scheduler.debug includes task metadata (posted_from etc).

It is proposed to rearrange this into following categories:

- *renderer.scheduler*: Main thread flamegraphs (as now) plus visualisation of main states of RendererScheduler (visibility/use_case/audio/etc).
- *disabled-by-default-renderer.scheduler:* Additional information which adds visual noise to the trace viewer but does not increase trace size. At this moment this category will include various counters.
- *disabled-by-default-renderer.scheduler.verbose*: Adds objects dumps while trying to keep trace size reasonable.
- *disabled-by-default-renderer.scheduler.debug:* All information that renderer scheduler has, including queued task metadata with location.

Estimation: 3 days

Audit contents of state dumps and ensure all data is there

Scheduling systems were heavily modified during their lifetime and tracing wasn't always kept up to date. It is required to go though policy making code (UpdatePolicyLocked in RendererScheduler and WebViewScheduler/WebFrameScheduler) and compare it with tracing code (RSI::CreateTraceEventObjectSnaphot and various AsValueInto).

Estimation: 2 days

Visualise RendererScheduler state

Similar to exising use case and visibility (async events for tracing state), visualise other variables of RendererSchedulerImpl::MainThreadOnly. This includes:

- is_audio_playing (default)
- renderer_paused (default)
- touchstart_expected_soon (default)
- {timer, loading}_tasks_seem_expensive (disabled-by-default)
- has_navigated (disabled-by-default)
- Other important bits (to be determined based on the previous step).

Technical challenge here is to make StateTracer work with different categories. It is proposed to use template magic:

```
enum class TracingCategory {
  DEFAULT,
  INFO,
  VERBOSE,
  DEBUG,
};

template <int category>
class StateTracer {
```

```
  const char* GetCategoryName() {
    case (category) {
      DEFAULT: return “renderer.scheduler”;
      ...
    }
  }
};
```

Estimation: 5 days

Visualise frame states

Similar to the previous step, visualise member variables of WebFrameScheduler. This includes:

- frame_visible_ (disabled-by-default)
- page visibility (disabled-by-default)
- frame_paused (disabled-by-default)
- active_connection_count (disabled-by-default)
- cross_origin (disabled-by-default)
- Stretch: URL (extract information from BlameContext)

Note that after this step the complete state of the frame won’t be obvious: frame visibilities will have a separate block, page visibilities will have a separate block, etc. In order to address this temporarily (the proper fix is the last step of this document) it is proposed to add frame_id to every async event.

Estimation: 5 days

Visualise task queues (stretch)

This step involves visualising two things:

- Executed tasks (can be implemented with async events. If task type info is available, task type can be used as a title. Note that this is a slice of async events, completely different from normal thread stacktrace).
- The state of a task queue (enabled/disabled/throttled). This should include counters for things like a number of queued tasks, but this might be blocked by not being able to have different counters with same name (we need to be able to provide object id similar to async events).

Group information in the trace viewer

After the “visualise frame variables” step, trace viewer will contain all necessary information, but it is hard to read due to information from the same frame being scattered. It is proposed to introduce a custom UI component which will arrange async event slices properly.

(Ben, please make sure that I make sense here and please add more specific, I'm being somewhat handwavey)

We need to arrange async slices according to frame ids. It's proposed that each frame-related async event has frame_id argument tying it to a specific frame. A special RendererScheduler container will collect all events with category "renderer.scheduler*" and organize them into AsyncSliceGroups according to frame_ids.

Estimation: 2 weeks+