Virtual Time in Blink

(PUBLIC)

What is virtual time?

Normally immediate tasks are run as soon as dispatched and delayed tasks are run once the requested delay has elapsed. Virtual time allows us to do two things:

1. Suspend and resume execution of tasks
2. Dispatch delayed tasks immediately while preserving the expected order execution

Why do we need this?

The Headless Chrome project (go/ghost-rider) has a number requirements that motivate virtualize time:

- Rendering should be deterministic
- Rendering should be efficient
- Support for rendering N independent requests (tabs) in parallel

Virtual time helps us make the network stack deterministic. We can make sure network requests come back in the order requested and if we stop the world while there are pending requests, we can make sure the various tasks (http parsing, dom timers, etc…) occur in the same order every time (so long as the content is the same).

Many pages run some dynamic js (often heavily backed by dom timers), and usually settle down after a couple of seconds. We can give them a fixed quota of time (e.g. 5 virtual seconds) to settle down before sampling the DOM. This will help make sure we return meaningful results every time independent of machine load (which would skew things if we used real time).

How is virtual time implemented in the scheduler?

A TaskQueue belongs to a TimeDomain which is responsible for providing the current time and scheduling of delayed work. Note it's possible for a TaskQueue to migrate from one TimeDomain to another. There are currently three time domains implemented in the scheduler:

- RealTimeDomain which implements the familiar wall time based scheduling of delayed tasks.
- VirtualTimeDomain which has a clock fully under user control. This is used by the ThrottlingHelper to align tasks for backgrounded tabs to the nearest second.
- AutoAdvancingVirtualTimeDomain a specialization of VirtualTimeDomain, where the virtual time source advances to the next delayed task once the TaskQueueManager has run out of immediate work to do.

Headless chrome will use the AutoAdvancingVirtualTimeDomain for rendered pages.

How will this be exposed to blink?

We propose the following API changes:

WebViewScheduler:

virtual void enableVirtualTime() = 0;
Migrates all loading and timer task queues associated with this WebView to an AutoAdvancingVirtualTimeDomain.

virtual void setAllowVirtualTimeToAdvance(bool) = 0;
Controls whether or not the AutoAdvancingVirtualTimeDomain can advance to the next task if the TaskQueueManager runs out of immediate work.

WebTaskRunner:

virtual double virtualTimeSeconds() const = 0;
Like Platform::currentTimeSeconds but if virtual time has been enabled on the parent WebView, the virtual time base will be used instead of the wall clock.

virtual double monotonicallyIncreasingVirtualTimeSeconds() const = 0;
Like Platform::monotinicallyIncreacingTimeSeconds but if virtual time has been enabled on the parent WebView, the virtual time base will be used instead of the wall clock.

WebView

virtual WebViewScheduler* scheduler() const = 0;
This is just exposing on the interface a member the WebViewImpl already has.

Various things in blink need to use VirtualTime, e.g. TimerBase. For more details see http://go/headless_chrome_virtualtime

Virtual time policies

We will expose via DevTools a number of virtual time policies, which will set WebViewScheduler::setAllowVirtualTimeToAdvance based on the policy and information available inside blink.

Policy	**Description**
Advance	If the scheduler runs out of immediate work, the virtual time base may fast forward to allow the next delayed task (if any) to run.
Pause	The virtual time base may not advance.
Pause if network fetches pending	The virtual time base may not advance if there are any pending resource fetches.