Service architecture of the Global Resource Coordinator

oysteine@chromium.org, February 2017

***** PUBLIC DOC ****

Main GRC doc here.

The GRC's goal is centralize the code used for gathering performance metrics and for coordinating Chrome's use of limited system resources, to be able to make informed decisions from a bird's-eye view. The core of the GRC is a Mojo service that does the actual global coordination. While there are a large number of existing and future systems that can/will form parts of this Mojo service, there's a few immediate steps we can do to bootstrap the global resource coordinator. This document describes those steps.

The Plan

The MemoryCoordinator and the memory instrumentation code are our starting point to creating the GRC service. MC is already using mojo interfaces but is not service-fied, and memory-instrumentation is already set up as a service. So bootstrapping the global coordinator service is mostly a matter of moving code around, and starting up the service in the browser process.

The Name Of The Service [In Code]

We will eventually have /services/x/, for some value of X. We discussed a variety of options: resource_coordinator, global_coordinator, coordinator, speed.

We settled on resource_coordinator.

Key First Moves

The first key moves are mainly revolving around servicifying existing code:

- Rename /services/memory_instrumentation to /services/resource_coordinator (this is the memory-infra service). This will be coordinated with existing memory-infra servicification efforts to not block/interfere with their work.
- Move the memory-infra interfaces to /services/resource_coordinator/memory/instrumentation
- Move /content/browser/memory to /content/browser/resource_coordinator/memory (this includes the MemoryCoordinator)
- Move the central controller part of MemoryCoordinator from the /content/browser location above, to /services/resource_coordinator/memory/coordinator, and its public interfaces to /services/resource_coordinator/public
- Once/if /base/perf gets created as part of TracingV2, move the /base parts of the MemoryCoordinator into a subdirectory here.

Interfaces

The interfaces provided by the resource_coordinator service will evolve over time, within the general theme of “measure Chrome’s performance, report metrics from that data, and use it to guide Chrome’s use of system resources”.

Existing interfaces

The initial set of interfaces will be the consolidated set of interfaces we’ll get by the existing code moved into the service:

Memory_instrumentation

Memory_instrumentation has currently two interfaces: The Coordinator which registers callback interfaces that can have memory dumps requested from them and is implemented by the service, and the ProcessLocalDumpManager interface which will be implemented once per process to provide memory dumps.

MemoryCoordinator

The MemoryCoordinator has one interface implemented by the service: the MemoryCoordinatorHandle which each child process uses to register a ChildMemoryCoordinator interface callback.

The above gives us two process-level bindings to the resource_coordinator service, one for memory_instrumentation and one for the MemoryCoordinator. Both of these are process-wide and should be merged at some point as they're basically both doing the same thing.

New interfaces

The first new interfaces are the CoordinationUnitProvider and the CoordinationUnit interfaces. The former is used to request a messagepipe to an internal CU given a CUID, which is a unique identifier that can be re-created anywhere in the codebase (the idea being that you can construct one using a user-defined string, which will then connect to a specific CU in the service). The CoordinationUnitProvider is only usable by the browser process, which is responsible for validating any requests from child processes for CU messagepipes.

The CoordinationUnit interface will have functions for setting up parent/child relationships between themselves, for sending events to a CU (i.e. "i am playing audio now"), and for setting up callbacks for the client of a CU to receive resource usage policies which are calculated based on the received events (i.e. "should this process be backgrounded", "what's my memory limitations" etc).

PerformanceTimeline

The PerformanceTimeline interfaces will receive performance-related events from other parts of Chrome, which systems within the service (and outside, in some cases) can listen for. See the main GRC doc for a high-level overview. Potential event types:

- Memory APIs: OnPurge/OnSuspend, MemoryUsage, MemoryProcessDump, MemoryPressureState
- Task APIs: OnLongTask, WorkQueueSize, TimersThrottled
- Loading APIs: OnPaint, OnStyleRecalc, OnLoad, breakdowns of subsystems (time in script, time in layout, etc).
- Embedder events: OnVisibilityChanged, TabCreated

The general idea is to have a clientlib part of the service which can receive high-frequency/granularity events, and forward/aggregate events to the service as appropriate. See PCU.