AgentSchedulingGroup-bound BrowserInterfaceBroker [PUBLIC]

talp@

Last Modified	Feb 2, 2021
Status	Implemented
Reviewers	kouhei@

Update (Oct 25, 2021)

ASG-BIB has been implemented and may be used. However, we currently only have a single ASG per renderer process, so the ASG-BIB effectively behaves the same as the process global BIB. One approach to make the ASG-BIB useful could be instantiating multiple ASGs (ASG per SiteInstance) while **sharing the process's IPC channel** [cl]. This will cause minimal behavioral changes (compared to separate IPC channel per ASG), while still allowing documents from different ASGs to share interfaces only amongst themselves.

Background

Several ongoing projects would benefit from the ability of attributing Mojo interfaces to specific AgentSchedulingGroups. Currently, this association can only be done indirectly via an ExecutionContext. However, due to a variety of reasons, not all interfaces can be scoped to an ExecutionContext, requiring those interfaces to be bound to the entire renderer. This document details the creation of a BrowserInterfaceBroker scoped to AgentSchedulingGroups, that would enable us to bind interfaces to an AgentSchedulingGroup, thus solving this problem.

Anatomy of a BrowserInterfaceBroker

BrowserInterfaceBroker(BIB) is a Mojo interface that enables the renderer process to generically obtain a Remote to a browser-implemented interface, and binds the Receiver to the appropriate implementation in the browser process.
In the renderer process, usage is very simple:

```
Remote<SomeMojoInterface> remote;
execution_context->GetBrowserInterfaceBroker().GetInterface(
  remote.BindNewPipeAndPassReceiver());
```

```
Remote<SomeProcessGlobalMojoInterface> remote2;
Platform::Current()->GetBrowserInterfaceBroker()->GetInterface(
  remote2.BindNewPipeAndPassReceiver());
```

On the browser side, whenever a new "context" is created (RenderFrameHost or one of the different worker hosts) it instantiates a BrowserInterfaceBrokerImpl. The BIBImpl then triggers registration of interface binders in its mojo::BinderMap[1]. Then, whenever a request for an interface is received by the BIB it searches for an appropriate binder in its map and runs it with the PendingReceiver sent from the renderer process.

> Note: Some of the content/ binder registration functions call out to the ContentBrowserClient to allow it to register its own binders. At the moment, we don't see a need for embedders to register ASG-scoped binders, so this will not be provided at this time.

ASG-BIB design

As much as possible, we try to follow the same design as the existing BIB.

Browser process

Ownership, lifetime, etc.

Similar to the way in which a new BIBImpl is created (and owned by) a new RenderFrameHostImpl [cs], AgentSchedulingGroupHost will have its own BIBImpl<ASGH, ASGH> member, which will register its interfaces in new functions added in browser_interface_binders.cc.
The ASGH's Receiver<BIB> will be bound as part of ASGH::SetUpIPC by using a message/API on the AgentSchedulingGroup Mojo interface. Binding it via the ASG interface guarantees that the BIB will be reset once the ASG is destroyed.

Interface registration

Two new functions will be added to browser_interface_binders.cc (in the content::internal:: namespace):

```
void PopulateBinderMap(AgentSchedulingGroupHost*, mojo::BinderMap*);
void PopulateBinderMapWithContext(
  AgentSchedulingGroupHost*,
  mojo::BinderMapWithContext<AgentSchedulingGroupHost*>);
```

These will be called by the BIBImpl constructor "automatically".

---

[1] It also has a mojo::BinderMapWithContext, but for our purposes the two can be treated the same way.

Renderer process

Ownership, lifetime, etc.

In the renderer process the Remote<BIB> would need to be accessible by both content/renderer and Blink code. Some ways this can be achieved:

1. Store the BIB on blink::AgentGroupScheduler - This is simple to implement, but the BIB is not directly related to scheduling.
2. Store the BIB on a new blink::AgentSchedulingGroup - We would need a place to store ASG-bound state at some point, might as well do it now ^_^. This would also be the natural ASG-related object to implement MojoBindingContext. If we do this, though, it might be good to reconsider the (already confusing) naming scheme.
3. Store the BIB on content::AgentSchedulingGroup, and access it from Blink using a client/delegate.

My personal preference is for option 2, as it seems "cleanest", but all 3 are viable.

Obtaining an ASG-BIB

From Blink:
We expect most code that instantiates per-ASG interfaces would have access to a Page(Scheduler), Frame(Scheduler), Document, etc., from which it is possible to reach the AgentGroupScheduler. For code in platform/ that does not have access to any of those, some context would have to be plumbed in[2] from calling code.

From content/renderer:
Most code that would use the per-ASG BIB would probably have access to a Render{Frame,View,Widget}, that have access to the ASG they are associated with.

Should I use the ASG-BIB?

With the addition of the ASG-BIB, we would have 3 different browser interface brokers to choose from when obtaining or registering a Mojo interface. This brings up the question of which of those should be used.
Generally speaking, the preference should be to use the ExecutionContext's BIB where possible. Reasons why this may not be possible include:

- Spec for that feature specifically requires the interface to be at a larger scope (usually Agent).
- Global state in the renderer that can not be easily migrated to be per-ExecutionContext or per-ASG. An example of this is V8 state that is bound to a v8::Isolate - an Isolate is currently bound to a thread, and V8 does not currently have any concept with a scope matching ASG.

[2] Probably MojoBindingContext, which would be implemented by blink::AgentSchedulingGroup.

Interface migration list

This is a list of interfaces we plan to migrate to the ASG-BIB once it is implemented. Note that this is very dynamic, and the list will probably change as we learn more about each specific interface.

- DomStorageProvider - The different storage APIs provided by DomStorageProvider require messages to be ordered within an Agent.
- BrodcastChannelProvider - The broadcast channels spec says a broadcast channel is associated with an agent. Note that broadcast channels also need to be accessed from workers (while ASGs live on the main thread). Since workers are in a separate agent, though, it should be possible to use EC-BIB for workers and ASG-BIB for frames.
- WebDatabaseHost(?) - Probably has similar ordering guarantees as DomStorageProvider. Might be too much work to migrate, though, and the interface is deprecated.
- TimeZoneMonitor(?) - Spec requires that all frames in the same agent be notified of the change atomically. However, this uses v8::Isolate-bound state that would make this migration non-trivial.