Removing RenderWidget

dtapuska@chromium.org
With help from danakj@chromium.org
Feb 4th, 2020

Motivation

The project to move Legacy IPC messages to mojo allows a number of interfaces on the public blink API to be removed. This will leave RenderWidget being much of a shell with implementations of abstractions that could move into blink. The browser will directly interact with blink and there is no need for the content layer to be involved. This document strives to outline a current architecture and end goal architecture.

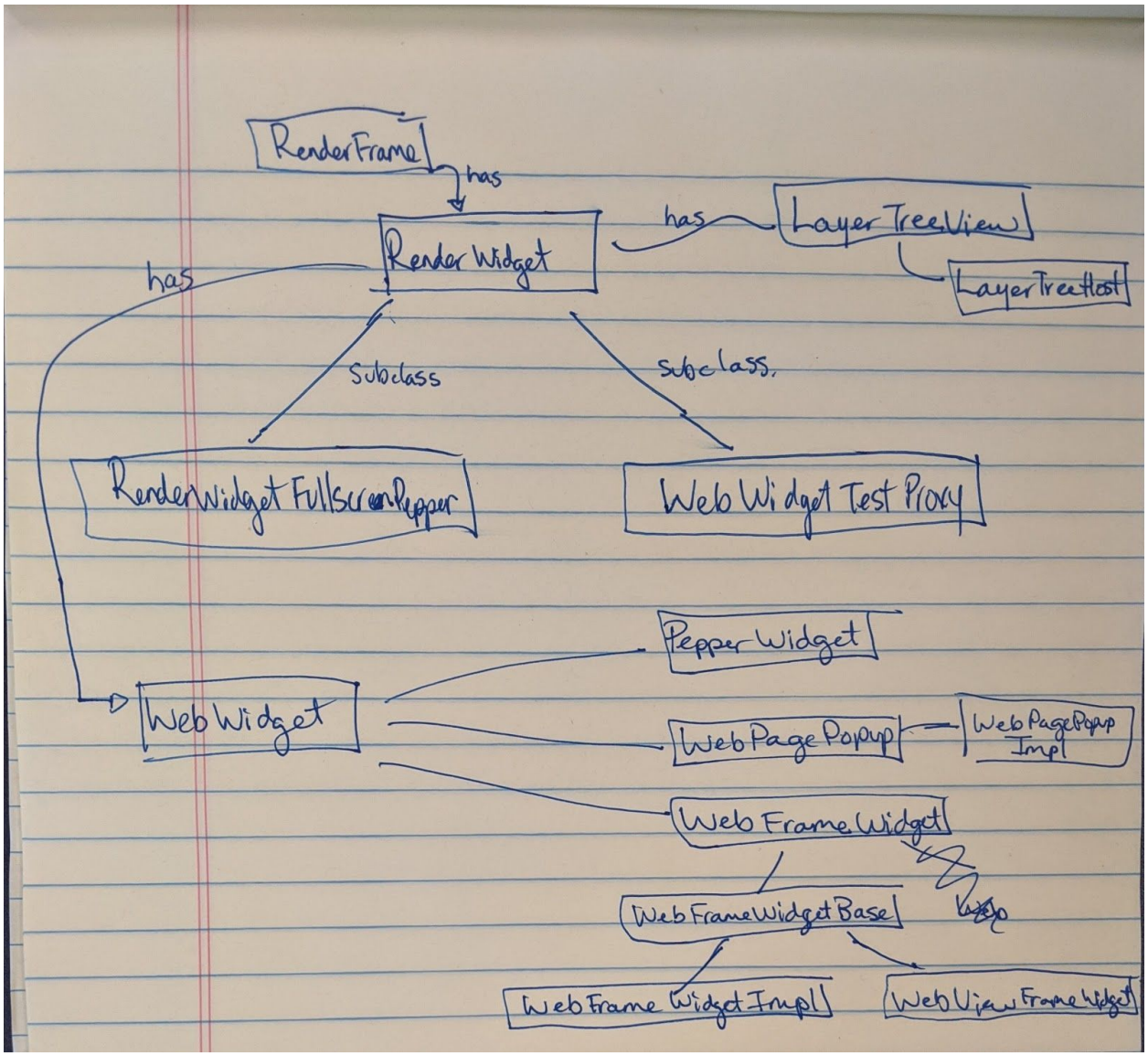
Current Architecture

RenderWidget is one of the main interfaces into blink. It contains a WebWidget which has a few subclasses. WebPagePopup, WebFrameWidget, PepperWidget.

RenderWidget does a number of things based on behalf of these classes.

1) Initializes and owns the compositing tree
2) Supports device emulation (adjusting screen coordinates)
3) Processes input

Many of these services can simply move into blink but must be accommodated somewhere. Blink already depends on cc, input processing is just an abstraction and relies on public blink interfaces. Emulation is a small component of RenderWidget and can easily be handled in blink.



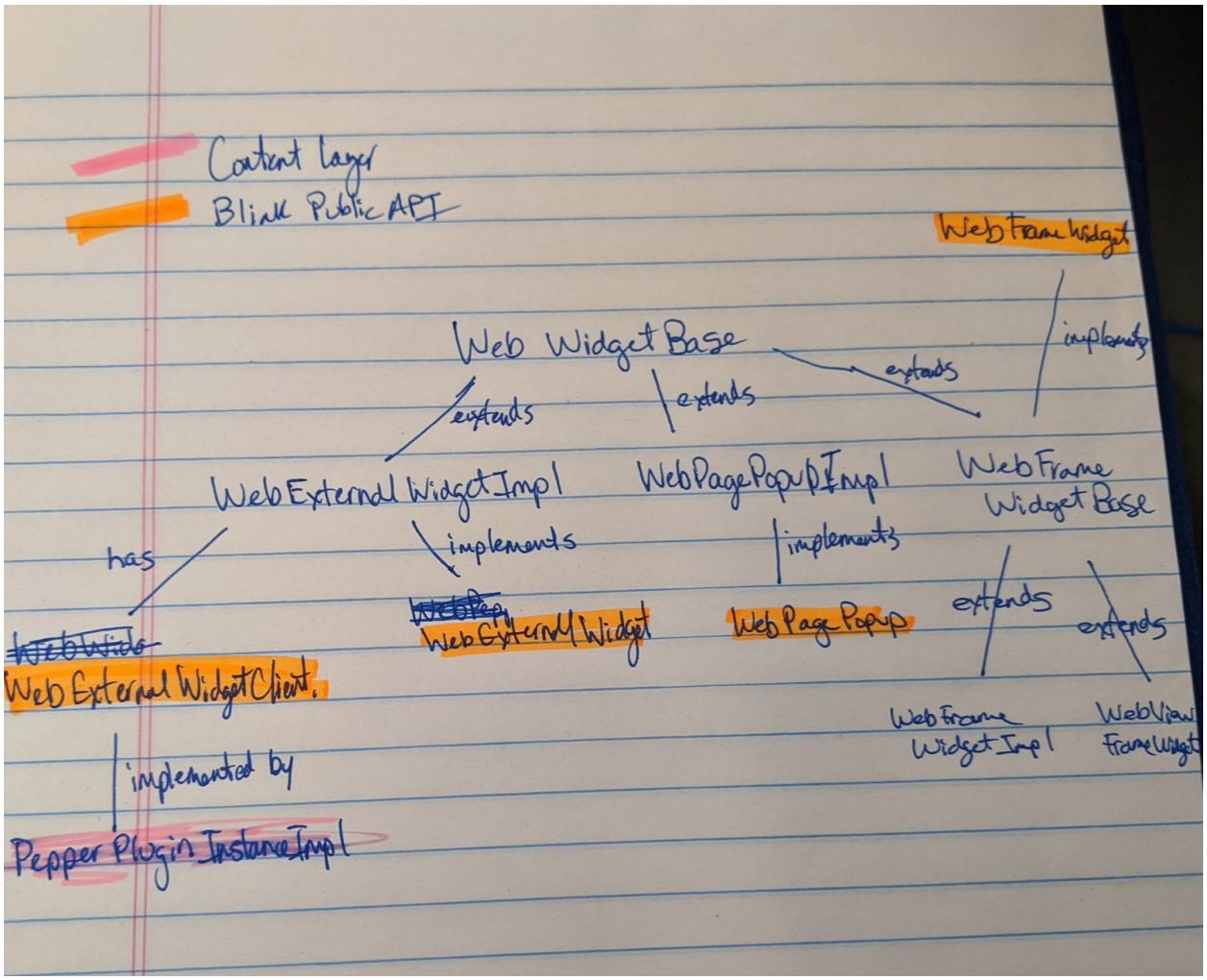
RenderWidget also provides a number of services to content/renderer/ and content/shell/test_runner/. It implements the following interfaces. The users of these services would need to either move into blink along with RenderWidget or go through the blink public API to get to them.

- IPC::Listener
- IPC::Sender
- blink::WebPagePopupClient
- blink::WebWidgetClient
- mojom::Widget
- LayerTreeViewDelegate
- RenderWidgetInputHandlerDelegate
- RenderWidgetScreenMetricsEmulatorDelegate
- MainThreadEventQueueClient

Furthermore, the public API of RenderWidget is currently accessible to content/renderer/ and content/shell/test_runner/ and users of those APIs would require blink public APIs to reach the same code in blink.

Proposed Changes

Proposal is to eventually move compositing, input handling and emulation into a base class called blink::WidgetBase. This class would be entirely implemented in blink's platform/ component and the current implementations of the WebWidget interface would instead extend from this concrete class. RenderFrame could then hold onto WebFrameWidget directly.



The WebPagePopup interface is still exposed for accessibility purposes but could be removed in the future when Accessibility moves inside blink.

A pair of classes WebExternalWidget/WebExternalWidgetClient replace the RenderWidgetFullscreenPepper/PepperWidget classes that exist currently. This is not to expose too much of Pepper into blink since it will eventually be removed due to PPAPI and Flash deprecation. The PepperPluginInstance would hold onto this WebExternalWidget directly and would implement WebExternalWidgetClient.

RenderFrameImpl can implement WebFrameWidgetClient, and WebFrameTestProxy has the ability to override anything in WebFrameWidgetClient for layout tests.

WebWidgetTestProxy also would be removed with its overrides becoming API calls on WebFrameWidget itself.

Browser side will be unaffected by this. RenderWidgetHostImpl will still talk to a widget that is hosted in the render process. The encapsulation of that widget would just end up being inside blink as opposed to inside content/renderer.

For each interface exposed internally in content/ we propose the following:

IPC::Listener

Removed once legacy IPC is removed. IPC messages should move to their appropriate blink side mojo interface.

IPC::Sender

Removed once legacy IPC is removed. IPC messages should move to their appropriate blink side mojo interface.

blink::WebPagePopupClient

This interface is collapsed into the single WidgetBase class. The caller to this in blink is WebPagePopupImpl which is a WebWidget implementation today, so it would become a call on its base class once moved to blink::WidgetBase.

blink::WebWidgetClient

This is a superclass of the WebPagePopupClient, and is exposed for blink to reach RenderWidget. Blink will have even simpler access to the WidgetBase as a parent class of its current WebWidget implementations, so will change calls from WebWidgetImpl->Client->Foo() to be WebWidgetImpl->Foo().

mojom::Widget

This interface moves seamlessly into blink. Any banned types such as std::vector would become blink types automatically as a result of the move.

LayerTreeViewDelegate

The LayerTreeView is owned by RenderWidget in order to hold a different lifetime on the cc::LayerTreeHost and its client pointers, which go back to the LayerTreeView. This will move to blink with RenderWidget and would be owned by WidgetBase instead.
There is a separate consideration of exposing the cc::LayerTreeHost inside the RenderWidget+LayerTreeView to the rest of blink. That conversation remains relatively unchanged by this move, as blink::WidgetBase+LayerTreeView would continue to encapsulate the cc::LayerTreeHost by default. Exposing it would require exposing methods and states on LayerTreeHost that WidgetBase wants to manage such as visibility, and finding another way to maintain clear state control and ownership.

RenderWidgetInputHandlerDelegate

This is the client pointer given to RenderWidgetInputHandler. The RenderWidgetInputHandler would need to move to blink along with RenderWidget, and its delegate would point to the blink::WidgetBase instead.
RenderWidgetInputHandler also has a pointer to the full RenderWidget object, which would naively become a pointer to the blink::WidgetBase object during the move.
To not move these classes at the same time would require introducing public WebRenderWidgetInputHandler with a static create method for WidgetBase to call, and a WebRenderWidgetInputHandlerDelegate pointer back to WidgetBase. It would also require the direct usage of RenderWidget to be moved to the Delegate interface instead, which would be a positive decoupling regardless of reason.

Open Problems:
Moving RenderWidgetInputHandler requires decoupling it from content types such as content::InputEventDispatchType and content::InputEventAck. These types are not accessible to blink and need to be moved out of content or replaced with other types, both for RenderWidget and RenderWidgetInputHandlerDelegate.
The RenderWidgetInputHandler also depends on the content::MainThreadEventQueueClient which is accessed through RenderWidget. This type would not be accessible and needs to be moved out of content, converted to another type, or moved along with RenderWidget at the same time. It has a larger set of transitive dependencies on content types, so moving it is not trivial, and these would need to be decoupled first.

RenderWidgetScreenMetricsEmulatorDelegate

This is the client pointer given to RenderWidgetScreenMetricsEmulator. The RenderWidgetScreenMetricsEmulator would need to move along with RenderWidget, similar to RenderWidgetInputHandler, or introduce public Web apis in each direction between blink and content.

Open Problems:
The emulator has very limited content dependencies in the content::ScreenInfo class and and content::ScreenOrientationValues type. This type would need to be replaced with another type or moved out of content in order to make moving the class into blink possible.

MainThreadEventQueueClient

This is the client interface given to MainThreadEventQueue which is owned by RenderWidget and shared with the WidgetInputHandlerImpl, RenderWidgetInputHandler, and FrameInputHandlerImpl. We have above determined that RenderWidgetInputHandler would likely need to move to blink as the same time as RenderWidget (or at least all parts of RenderWidget that use or are used by the class).

Open Problems:
MainThreadEventQueue is coupled to more than a handful of content types around input. Each of these types would need to be moved out of content or replaced in order to move MainThreadEventQueue also. Otherwise, a public Web api, plus a Web client api, would need to be created in order to use the current content implementation.

WebWidgetTestProxy

void BeginMainFrame(base::TimeTicks frame_time) override;
- BeginFrame is removed from WebWidget but moves to WebLocalFrameWidgetClient as a callback.

void RequestDecode(const cc::PaintImage& image,
base::OnceCallback<void(bool)> callback) override;
- Not necessary. Call SetNeedsAnimate if running web test inside blink.

void RequestPresentation(PresentationTimeCallback callback) override;
- Not necessary. Call SetNeedsAnimate if running web test inside blink.

// WebFrameWidgetClient implementation (replacing WebWidgetClient in WebFrameWidget).
void ScheduleAnimation() override;
- Remains

bool RequestPointerLock(blink::WebLocalFrame* requester_frame,
bool request_unajusted_movement) override;
- Should likely need to move to browser to approve pointer lock

void RequestPointerUnlock() override;
- Moved to browser

bool IsPointerLocked() override;
- Detectable in blink

void SetToolTipText(const blink::WebString& text,
blink::WebTextDirection hint) override;
- Add API to retrieve last tooltip text from the Frame (it is stored in the ChromeClient).

void StartDragging(network::mojom::ReferrerPolicy policy,
blink::WebScreenInfo GetScreenInfo() override;

EventSender* event_sender() { return &event_sender_; }
void Reset();
void BindTo(blink::WebLocalFrame* frame);
void EndSyntheticGestures();
void SynchronouslyComposite(bool do_raster);
- Add a

Mojo Changes

```
interface FrameWidget {
  DragEnter() => (DragOperation operation);  //
DragMsg_TargetDragEnter//DragHostMsg_UpdateDragCursor
  DragOver() =>  (DragOperation operation);  //
DragMsg_TargetDragOver//DragHostMsg_UpdateDragCursor
  DragLeave(); // DragMsg_TargetDragLeave
  DragDrop();  // DragMsg_TargetDrop
  DragSourceEnded(); // DragMsg_SourceEnded
  DragSourceSystemEnded(); // DragMsg_SourceSystemDragEnded
};

interface FrameWidgetHost {
  SetHasTouchEventHandlers(); // WidgetHostMsg_HasTouchEventHandlers
}

interface Widget {
  Hide();  // WidgetMsg_WasHidden
  Show(); // WidgetMsg_WasShown
  UpdateVisualProperties();  // WidgetMsg_UpdateVisualProperties
  SetActive();  // WidgetMsg_SetActive
  MouseLockLost();  // WidgetMsg_MouseLockLost
  ShowContextMenu();  // WidgetMsg_ShowContextMenu
  ForceRedraw() => {}; WidgetMsg_ForceRedraw//WidgetHostMsg_ForceRedrawComplete
  UpdateScreenRects() => (); //
WidgetMsg_UpdateScreenRects//WidgetHostMsg_UpdateScreenRects_ACK
  Close() => (); // WidgetMsg_Close//WidgetHostMsg_Close_ACK
};

interface WidgetHost {
  LockMouse() => (bool allowed);  // WidgetMsg_LockMouse_ACK/WidgetHostMsg_LockMouse
  TextInputStateChanged(); // WidgetHostMsg_TextInputStateChanged
  r(); // WidgetHostMsg_IntrinsicSizingInfoChanged
  DidChangeCursor(); // WidgetHostMsg_DidChangeCursor
  SetTooltipText(); // WidgetHostMsg_SetTooltipText
  SetBounds(); // WidgetHostMsg_RequestSetBounds
  SelectionBoundsChanged(); // WidgetHostMsg_SelectionBoundsChanged
};
```

Path forward

1) Create WebWidgetBase and move LayerTreeView into blink
2) Move input into WebWidgetBase
3) Remove RenderWidget cleaning up IPC and direct ownership of WebFrameWidget by RenderFrameImpl.