Note: this document assumes the reader is familiar with OOPI, WebFrameWidget, and the changes in the Blink public API (third_party/WebKit/public) that are being made to support out-of-process <iframe>s (OOPI). This background information is covered in https://goo.gl/4FsB8N.

Current State

In order to support out-of-process <iframe>s (OOPI), a tab may have one or more frame widgets associated with it to manage input / layout / painting.

Unfortunately, this doesn't mesh well with the old pre-OOPI architecture where WebView inherits from WebWidget. The frame tree now has two distinct types of widgets:

1. The WebWidget that WebView inherits, which is used as the widget for the main frame.
2. The WebFrameWidget used for subframes that need a frame widget.

Problem #1

Logic to operate on frame widgets needs to branch on whether or not the local root is the main frame, since it needs to operate on a different widget implementation for the two cases:

```
if (isMainFrame) {
    webView->doSomething();
} else {
    webFrame->frameWidget()->doSomething();
}
```

Problem #2

There is a lot of code duplication between WebFrameWidgetImpl and WebViewImpl (since it is-a WebWidget). It is not easily possible to share a common base class, due to the diamond inheritance problem. As a result, many WebWidget methods, like handleInputEvent() and resize(), share a lot of copy and pasted code in their WebFrameWidgetImpl/WebViewImpl implementations, but also have differences. This makes it hard to verify that the two implementations behave the same way and can easily lead to accidental copy and paste bugs.

Problem #3

RenderView / WebView holds the Page object, which holds an owning reference to the root of the frame tree. This is what keeps the entire tree of frames alive in each renderer. However, a frame widget is only needed for a frame that is a **LocalFrame**.

Suppose there are multiple renderers processes needed for a tab. By definition, only one renderer has a **LocalFrame** main frame. All the other main frames must be **RemoteFrame** instances that proxy to the main frame renderer host process. Thus, the WebView implementation of WebWidget is completely useless in all the other renderers. This wastes memory and makes it hard to understand the code, since there are unnecessary objects being instantiated.

Solution

In order to solve these problems, the following things will change:

1. The main frame will have its own WebFrameWidget, rather than using WebView's WebWidget inheritance.
2. WebView will no longer inherit from WebWidget, and WebViewClient will no longer inherit from WebWidgetClient.
3. RenderView will no longer inherit from RenderWidget.

Since the current code has a lot of dependencies on the inheritance relationship, this transition must be done in stages:

1. Internally, Blink implements a WebFrameWidget shim (WebViewFrameWidget[1]) that forwards all calls to the equivalent methods on WebView. The Blink public API exposes a way to create a shim that wraps the WebView's WebWidget.
   Note: the shim itself will not be exposed in the public API: it will just appear to be another WebFrameWidget.
2. The embedder is responsible for creating frame widgets[2], so update it to create a WebViewFrameWidget and associate it with the main frame.

   > **Note**: This is currently being implemented in https://codereview.chromium.org/1303773002

3. Update code that calls WebWidget methods through WebView to use the main frame's WebFrameWidget instead. For example:

   ```
   web_view->resize(...);
   ```

   will change to:

   ```
   web_view->mainFrame()->toWebLocalFrame()
       ->frameWidget()->resize(...);
   ```

   It's possible that there will be a convenience helper, to make it less verbose while still highlighting the hidden assumption that the main frame must be local.
4. Remove WebWidget inheritance from WebView.
5. Start unforking logic shared between WebFrameWidgetImpl and WebViewImpl into WebFrameWidgetBase. Both WebFrameWidgetImpl and WebViewFrameWidget will inherit from this base class: the final goal is for WebFrameWidgetBase to implement all widget handling.

---

[1] Implemented in https://codereview.chromium.org/1262443005.

[2] An example of this can be seen in RenderFrameImpl::createChildFrame(): the embedder explicitly creates a widget for the subframe by calling RenderWidget::CreateForFrame().

6. Once WebFrameWidgetImpl is empty, remove it and rename WebFrameWidgetBase to WebFrameWidgetImpl.
7. Similarly, consolidate code from WebViewFrameWidget into WebFrameWidgetBase until there's no special-case code for handling the main frame. At this point, both main frames and subframes should be able to use WebFrameWidget directly.
8. Remove the Blink public API for creating the shim. At this point, all frames with widgets should be using WebFrameWidgetImpl directly.
9. Remove WebWidgetClient inheritance from WebViewClient.
10. Remove RenderWidget inheritance from RenderView and RenderWidget-specific logic from RenderViewImpl.

A Note on RenderWidget

Today, RenderWidget is used for several different purposes:

- RenderViewImpl inherits from RenderWidget, to support input / layout / painting in the pre-OOPI world.
- It is instantiated directly by popup "menus" like the color chooser and date picker.
- Fullscreen mode is implemented by RenderWidgetFullscreen, which inherits from RenderWidget.
- Finally, frame widgets instantiate a RenderWidget directly.

To support this flexibility, all of RenderWidget's members are either public or protected. This is to allow RenderWidget's subclasses to customize it as needed, but also leads to very complex code:

- initialization is inconsistent: many subclasses use RenderWidget::Initialize() to complete initialization, but RenderViewImpl sets all the members itself manually in its own custom initialization method.
- handlers for IPC messages like ViewMsg_OnClose are overridden by subclasses, making it hard to trace control flow
- different types of RenderWidgets have distinct destruction semantics: a comment in RenderWidget::Close() indicates that the call needs to be deferred to the message loop in the common case, but it's not clear why.

Hopefully, once RenderView no longer inherits RenderWidget, RenderWidget can be simplified, but there is no concrete plan on how to do that yet.