Random Architecture Stuff

Unifying Style, Reducing Wrappers, More Mojo

Style Unification Update

- Automatically renamed (most) identifiers to match Chrome naming
- 9581 files changed, 601061 insertions(+), 593276 deletions(-)
- 45+ followup patches to fix various issues
- blink-reformat@chromium.org owns 0 bugs =)

Style Unification Update

- Blink C++ style guide is a much shorter list of additions and exceptions from Chrome style
- Blink still allows and encourages mutable reference parameters
- Methods called by generated bindings code (“web-exposed”) should be `namedLikeThis()`.
- Exception: methods specified by `ImplementedAs` should be `NamedLikeThat()`.

Style Unification Update: File Names

- Currently:
  - File names: Document.h, WebLocalFrame.h
  - Include guards: DOCUMENT_H_, WEB_LOCAL_FRAME_H_
- Soon (courtesy of tkent@chromium.org):
  - File names: document.h, web_local_frame.h
  - Include guards: THIRD_PARTY_BLINK_RENDERER_CORE_DOM_DOCUMENT_H_, THIRD_PARTY_BLINK_RENDERER_PUBLIC_WEB_WEB_LOCAL_FRAME_H_
- When adding code, please be consistent with local conventions.
- If moving files, please keep the convention of the source files.

Style Unification Update

- Enable (some) Chrome style plugin checks in Blink code
- Will definitely be enabled:
  - Qualifying `auto` with `&` or `*`
  - Checking `virtual`, `override`, and `final`
- May be enabled:
  - Inlined constructor and destructor
  - Inlined copy constructor / assignment operator

Reducing Wrappers

- Currently: many `//base` types have WTF wrappers
- Problem: awkward to pass these types through the Blink public API
- Goal: allow `//base`, `//ui/gfx/geometry`, and other select directories to be used directly in Blink

Reducing Wrappers

- In progress: dcheng@chromium.org is improving the presubmit checks for allowed / disallowed C++ types
- Once done, expect to also allow:
  - `base::Time`, `base::TimeDelta`, `base::TimeTicks`
  - `base::Optional`
  - `base::span`
- Eventually also allow:
  - `scoped_refptr`, `base::RefCounted`, `base::RefCountedThreadSafe`
  - `base::BindOnce`, `base::BindRepeating`, `base::OnceCallback`, `base::RepeatingCallback`
- And more...

Reducing Wrappers

- Currently, platform and controller don’t have any explicit allow / disallow checks for C++ types
- Going to revisit this, as the current policy is a pretty large blanket exemption…
- Please be careful to IWYU: this lets DEPS act as the primary enforcement
- C++ headers are quite leaky: one header can `#include` many other headers and bring their declarations in scope
- Presubmit check is only a backup to catch accidental violations of DEPS

C++14 and STL

- `std::make_unique`
- `std::enable_if_t`, et cetera
- `std::integer_sequence`
- Generic lambdas and more coming once the CrOS toolchain is updated!

C++14 and STL

- STL container types are still banned: `std::string`, `std::vector`, `std::set`, `std::map`, et cetera
- Other base container types are also banned: `base::FilePath`, `base::StringPiece`, `base::flat_map`, `base::flat_set`, `base::string16`, et cetera
- Exception for tests and code in `//third_party/WebKit/common` and `//third_party/WebKit/Source/platform`

Mojo

- Currently two ways to get interfaces in Blink:
  - Per-frame: `LocalFrame::GetInterfaceProvider()`
  - Process-global: `Platform::GetInterfaceProvider()`
- Confusingly, these two have different signatures
  - Per-frame: returns non-thread-safe `service_manager::InterfaceProvider*`
  - Process-global: returns thread-safe `blink::InterfaceProvider*`
- Plan: consolidate on `service_manager::InterfaceProvider*`

Mojo

- There's also `Platform::GetConnector()`
- Plan: remove this so there's not so many ways to connect to interfaces
- All interfaces will be brokered via the browser process
  - Makes security team happy
  - Makes it easy to mock out for layout tests

Mojo

- Goal #1: associate (almost all) interfaces with a context
- Goal #2: (automatically) revoke interfaces on context disposal
- Goal #3: make it easy to write correct-by-default code

Mojo: associate interfaces with a context

- Currently: workers have no per-worker `InterfaceProvider`
- Instead, all workers use the process-global `blink::InterfaceProvider`
- Requires plumbing `SecurityOrigin`, et cetera through, which is discouraged
- Plan:
  - Get rid of `blink::InterfaceProvider`
  - Add per-worker `InterfaceProvider` instead of overloading `Platform::GetInterfaceProvider()`
  - Add `GetInterfaceProvider()` to `ExecutionContext`

Mojo: revoke interfaces on context disposal

- Add a wrapper (`blink::InterfaceProviderForContext`?) around `service_manager::InterfaceProvider`
- Returns a wrapper around `mojo::InterfacePtr` that automatically calls `mojo::InterfaceProvider::reset()` when the context is disposed
- Add `mojo::Binding` helpers that automatically call `mojo::Binding::Close()` when the context is disposed

Mojo: what is context disposal?

- `LocalFrame` often* does **not** always change on navigation
- However, Mojo encourages interfaces that are attached to a `LocalFrame`
- … but this means origin-bound per-frame interfaces must manually reset themselves on navigation!
- It gets even trickier due to potential IPC races with Mojo:
  - No ordering guarantees between different message pipes
  - So everything ends up associated…

LocalFrame vs LocalDOMWindow vs Document

- Standard navigation

Before	**After**
LocalFrame	Original LocalFrame
LocalDOMWindow	**New** LocalDOMWindow
Document	New Document

LocalFrame vs LocalDOMWindow vs Document

- Standard navigation
- Navigation from the initial empty document:

  If browsingContext's only entry in its session history is the about:blank Document that was added when browsingContext was created, and navigation is occurring with replacement enabled, and that Document has the same origin as the new Document, then do nothing.
  From https://html.spec.whatwg.org/multipage/browsing-the-web.html#initialise-the-document-object

Before	**After**
LocalFrame	Original LocalFrame
LocalDOMWindow	**Original** LocalDOMWindow
Document	New Document

LocalFrame vs LocalDOMWindow vs Document

https://example.com

```
<script>
var w = window.open("https://example.com/window.html");
w.injectedFunction = () => { console.log("Hello world!"); };
</script>
```

https://example/window.html

```
<script>
injectedFunction();
</script>
```

LocalFrame vs LocalDOMWindow vs Document

- Standard navigation
- Navigation from the initial empty document
- Javascript: URL navigation

Before	**After**
LocalFrame	Original LocalFrame
LocalDOMWindow	**Maybe new** LocalDOMWindow
Document	New Document

LocalFrame vs LocalDOMWindow vs Document

- Standard navigation
- Navigation from the initial empty document
- Javascript: URL navigation
- document.open()

Before	**After**
LocalFrame	Original LocalFrame
LocalDOMWindow	**Original** LocalDOMWindow
Document	**Original** Document

LocalFrame vs LocalDOMWindow vs Document

- Standard navigation
- Navigation from the initial empty document
- Javascript: URL navigation
- document.open()
- XSLT?!

Before	**After**
LocalFrame	Original LocalFrame
LocalDOMWindow	**Original** LocalDOMWindow
Document	**New** Document (theoretically undectable)

LocalFrame vs LocalDOMWindow vs Document

- Standard navigation
- Navigation from the initial empty document
- Javascript: URL navigation
- document.open()
- XSLT?!
- Out-of-process iframes

Before	**After**
LocalFrame	**New** LocalFrame
LocalDOMWindow	**New** LocalDOMWindow
Document	**New** Document

Perfect World

- `LocalFrame` and `Document` are 1:1
- `RenderFrameHostImpl`, `RenderFrameImpl`, `WebLocalFrameImpl`, `LocalFrame`, `Document` have matching lifetimes
- Unfortunately there are still weird edge cases to consider...

Almost Perfect World

- Navigations that reuse the Window object:
  - provide a way to steal the Window object
  - what about Mojo interfaces that are already open?
- What happens for javascript: URLs?
- What happens for document.open()?
- What happens for XSLT?

Questions?