Merging wtf/ into platform/

haraken@chromium.org

2016 Oct 3

STATUS: DRAFT, PUBLIC

Summary: This document proposes merging wtf/ into platform/ as a part of the platform/ componentization of the Onion Soup project (design doc & slide by esprehn@). It will significantly simplify the Blink architecture and reduce the technical debt.

Recap: A big picture of Onion Soup

Fig.1 and Fig.2 illustrates a big picture of Onion Soup.

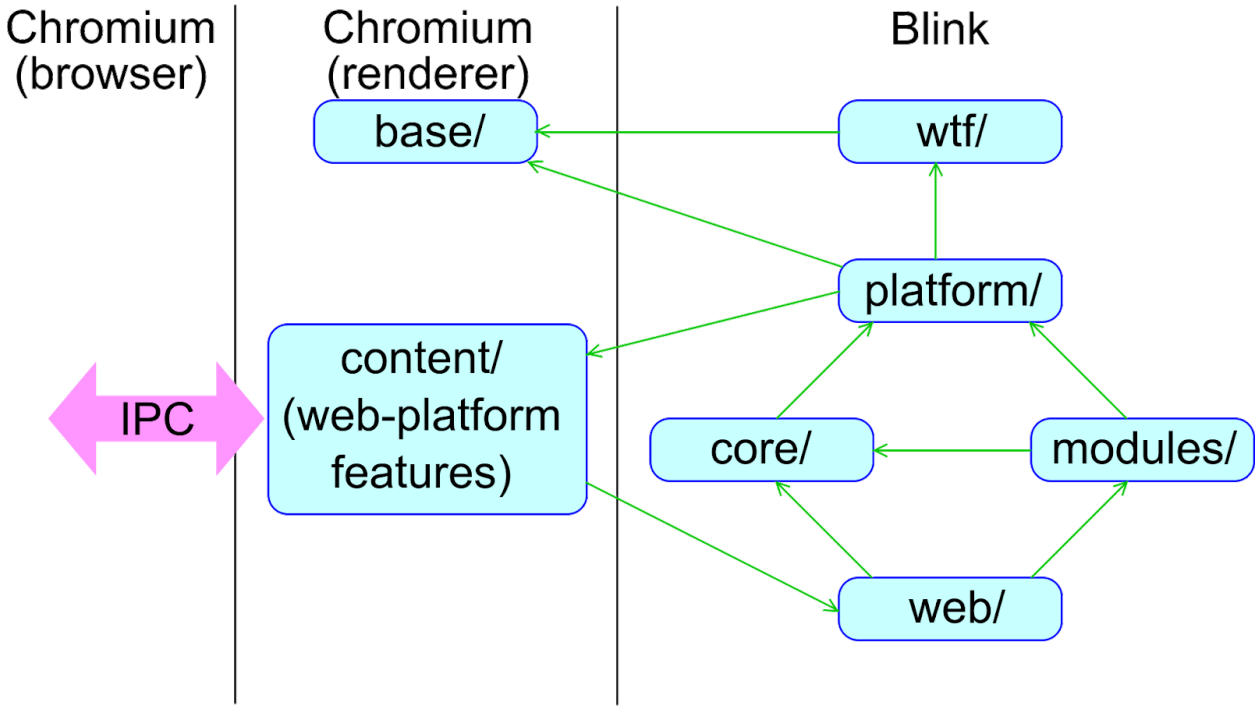

Fig.1 Before Onion Soup

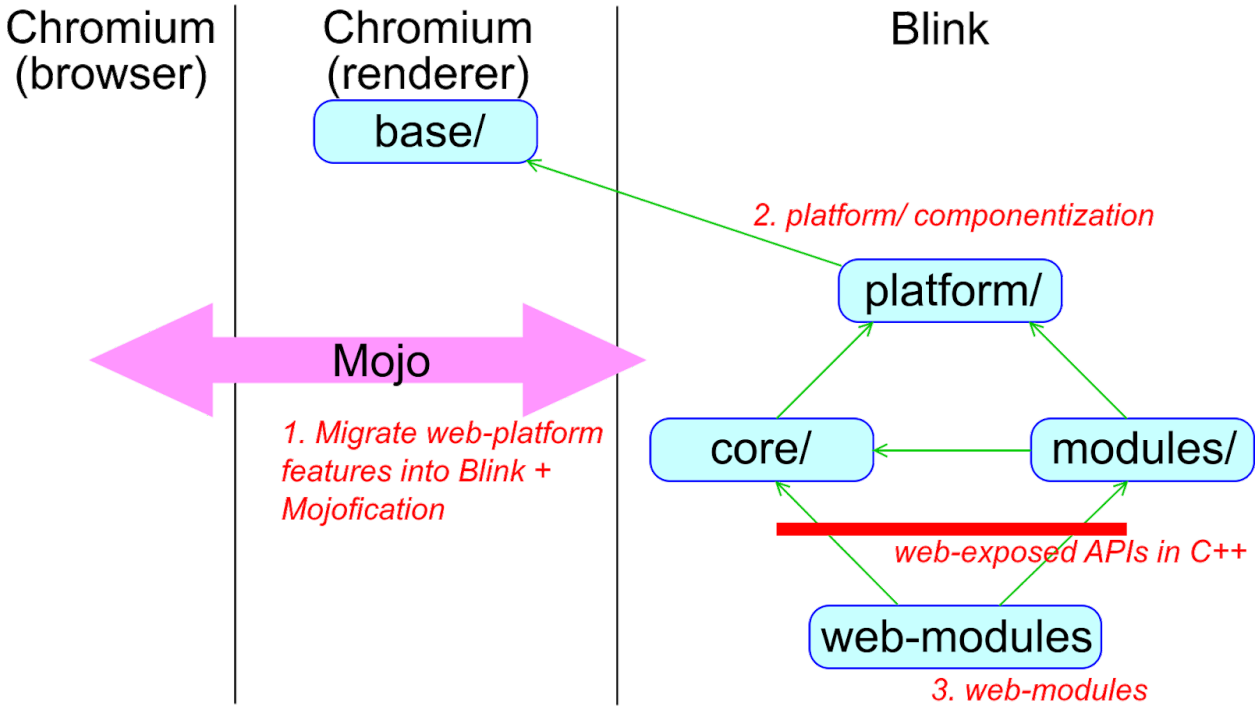

Fig.2 After Onion Soup

Onion Soup consists of the following three parts (See the design document for more details):

1. **Mojofication:** Move web-platform features from content/ to Blink and let the features talk with the browser process directly with Mojo. Deprecate the old IPC. Hopefully most of the public/ APIs will be gone.
2. **platform/ componentization:** Componentize platform/ into modularized components (e.g., platform/{memory, scheduler, graphics, network, loading, bindings etc}). wtf/ is merged into platform/. platform/X is allowed to depend on platform/Y via DEPS rules. Each platform/ directory should have a clear owner team.
3. **web-modules:** Build an infrastructure that enables to implement high-level web features on top of existing web-exposed APIs but in C++ (for performance reasons). The C++ APIs will be exposed to web-modules from core/ and modules/ through Web IDL files. The web/ layer should eventually be replaced with web-modules. Blink-in-JS should be replaced with web-modules.

This document proposes merging wtf/ into platform/ as a part of the platform/ componentization.

Why we want to merge wtf/ into platform/

Merging wtf/ into platform/ will solve a bunch of architectural issues that have led to technical debts. Here are a couple of examples:

- Currently Oilpan is in platform/ but PartitionAlloc is in wtf/. There are a lot of hacks to connect Oilpan with containers in wtf/ (Vector, HashTable etc). This is nasty but there is no easy way to put Oilpan and PartitionAlloc in one place because of the forbidden dependency from wtf/ to platform/: 1) we cannot move Oilpan to wtf/ because Oilpan depends on platform/ stuff; 2) On the other hand, we cannot move PartitionAlloc to platform/ because containers in wtf/ need to depend on PartitionAlloc. If we merge wtf/ into platform/, we can just put Oilpan and PartitionAlloc in platform/memory/. Also we can put PageAllocator, MemoryCoordinator and memory-infra in platform/memory/ together. That looks much nicer.
- There are a lot of files in wtf/ and platform/ whose ideal location is unclear. For example, currently files related to threading primitives are distributed between wtf/ and platform/. It's nicer to just put them together in platform/threading/.
- Since wtf/ cannot depend on platform/, we currently register a bunch of function pointers from platform/ to wtf/ to bypass the dependency. We can get rid of the complexity by merging wtf/ into platform/.
- wtf/ and platform/ can depend on base/. core/ and modules/ cannot depend on base/. If core/ and modules/ want to depend on base/, we need to create a thin wrapper class in wtf/ or platform/. However, it's sometimes not clear on which directory we should create the wrapper class. Merging wtf/ into platform/ will solve the problem.

- Removing the wtf/ layer will make Blink's architectural model simpler.

Discussions

Do we plan to create platform/wtf/?

Yes. Even if we merge wtf/ into platform/, there will still be a lot of primitive things such as String, Vector, HashTable, RefPtr, SpinLock, macros etc. It will make sense to keep these primitive things in one specific directory -- that is platform/wtf/.

Do we plan to remove the WTF:: namespace?

Yes. If we put Oilpan, PartitionAlloc, PageAllocator, MemoryCoordinator and memory-infra in platform/memory/, it won't make sense to mix the WTF namespace and the blink namespace in platform/memory/. Rather, it will make more sense to use a blink::memory namespace. **Our plan is to use a blink::X namespace for platform/X.** platform/wtf/ will have a blink::wtf namespace.

What's a conceptual difference between platform/, core/ and modules/?

core/ and modules/ are a building unit to implement web exposed APIs. All Web IDL files and their implementations should go to core/ and modules/. All other low-level features and libraries should go to platform/X. **We prefer minimizing core/ and modules/ by moving as many WebAPI-independent things to platform/X as possible.**

Regarding the difference between core/ and modules/, there is no distinction in the spec. Conceptually core/ and modules/ are one thing. Hence, the distinction is just for implementational convenience. If a feature X is tightly coupled with core/ things, X should go to core/. Otherwise, X should go to modules/. Remember that modules/X is allowed to depend on modules/Y via DEPS rules. (Note: We discussed if we should merge modules/ into core/ but have decided not to do that at least in short term because the benefit is not clear.)

Specifically, what files are going to be migrated to what directory?

yutak@ is now creating a spreadsheet to clarify the concrete migration plan, so stay tuned :) Basically the idea is to move all files in platform/ and wtf/ into well-modularized platform/X. In particular, files that are currently scattered over the top level of platform/ and wtf/ will find a better home. We are planning to create platform/{memory, scheduler, network, loading, graphics, threading, frame, bindings, wtf, mojo, audio etc}.

Will the merge affect build performance?

I don't think so. wtf/ and platform/ are not a big link unit. Merging the link units won't have a big impact on the build performance.

Any other questions?

…