Platform Architecture

IndexedDB: Onion Soup

cmp@chromium.org

Last updated: 2019-09-04

Tracking bug/717812 status: fixed, later work tracked in bug/843764.

Background

IndexedDB is a local storage system, accessible to user Javascript in web pages and workers. IndexedDB was previously mojofied in crbug.com/627484, but has not yet been "onion souped." Completing the onion soup work was blocked on mojofying Blob storage, which later on was completed as part of crbug.com/611935.

This work is part of the Chromium Onion Soup effort. I plan to:

1. DONE - Move IDB Factory implementation off of a channel-associated interface
2. DONE - Create a WebIDBFactory instance for each IdbFactory
3. DONE - Remove the IOThreadHelper classes and IO thread hopping from the renderer code

4. DONE - Stop passing io_runner around in the renderer code
5. DONE - Remove callback_runner hopping in callbacks objects
6. DONE - Stop passing callback_runner around in the renderer code
7. DONE - Move IndexedDB Mojo file to Blink (part 1)
8. DONE - Move IndexedDB Mojo file to Blink (part 2)
9. DONE - Move IndexedDB Mojo file to Blink (part 3)
10. DONE - Merge //content/common/indexed_db/ into Blink
11. DONE - Upgrade WebIndexKeys type to WebIDBIndexKeys
12. DONE - Mojo: Fix Blink-specific export_define_blink directive
13. DONE - Mojo: Add ArrayTraitsWebVector helpers
14. DONE - (Andreas) Land getDatabaseNames() support
15. DONE - Use UnionTraits for IDBKeyData
16. DONE - Merge //content/renderer/indexed_db/ into Blink
17. SKIPPED - Add IDBKey kNullType support
18. DONE - Revert merge change
19. DONE - Add stable DB compatibility tests
20. DONE - Change Mojo IDBValue.bits type from string to array<uint8>
21. DONE - Use ArrayDataView to convert IDBKey binary to std::string
22. DONE - Move content/renderer/indexed_db/ to Blink, take 2
    a. -- At this point, IndexedDB onion souping line in Onion Soup sheet can be marked done
23. DONE - Remove redundant Web* enums
24. DONE - Remove UMA metric gathering from callbacks
25. DONE - Remap Blink variant types onto renderer/modules/indexeddb/ classes
26. DONE - Correct origin security checks
27. DONE - Diagnose perf regression in IOThreadHelper removal
28. SKIPPED - Update Mojo Factory object to be associated to Database
29. DONE - Remove IO thread hopping in browser process
30. PENDING WORK FROM HERE ON TRACKED IN *bug/843764*
31. Use native Mojo callbacks
32. Merge Impl classes in Blink renderer/modules/indexeddb
33. Remove transaction IDs from interface

34. -- At this point reach 1.0 OKR --
35. Remove "cpp_only = true" from content/common/BUILD.gn mojo target
36. Update IndexedDBCallbacksImpl to pass cursor by value
37. Change UMA_HISTOGRAM_ENUMERATION so kTypeEnumMax can be removed
38. Map IDBValue.bits directly to std::string / WebData
39. Add exhibition DB to verify IndexedDB maintains binary compatibility
40. Consider additional testing pattern for Mojom interfaces using InterceptorForTesting

Proposal

1. DONE - Move IDB Factory implementation off of a channel-associated interface

The previous mojofication effort in crbug.com/627484 assumed the browser and renderer had to maintain message ordering wrt the legacy IPC channel. This was due to IDB's dependency on blob storage and the blob storage system's usage of the same renderer<->browser IPC channel for implicit ordering of its interface messages.

In crbug.com/611935, blob storage was fixed to remove its message ordering requirement and was also updated to be mojofied. Since IndexedDB had no other message order requirements of its own, now we can update IndexedDB from a channel-associated interface to a normal interface.

crrev.com/c/1146335

2. DONE - Create a WebIDBFactory instance for each IdbFactory

crrev.com/c/1150914

3. DONE - Remove the IOThreadHelper classes and IO thread hopping from the renderer code

a. DONE - WebIDBFactoryImpl - crrev.com/c/1150724
b. DONE - WebIDB{Database,Cursor}Impl - crrev.com/c/1155794

4. DONE - Stop passing io_runner around in the renderer code.

Right now IndexedDBCallbacksImpl and IndexedDBDatabaseCallbacksImpl need io_runner so they can create a new WebIDBDatabaseImpl or WebIDBCursorImpl on certain success results. Now that both WebIDBDatabaseImpl and WebIDBCursorImpl no longer rely on io_runner, we can remove io_runner from their constructors and the callbacks objects.

crrev.com/c/1155830

5. DONE - Remove callback_runner hopping in callbacks objects

crrev.com/c/1162921

6. DONE - Stop passing callback_runner around in the renderer code

crrev.com/c/1164471

Issues dealt with:

- Had to remove callback_runner from args and private members everywhere at once.

7. DONE - Move IndexedDB Mojo file to Blink (part 1)

In this step, we will move code from **//content/common/indexed_db/** to **Blink** to simplify the current indirection from Blink to Chromium (the current Chromium implementation is injected into Blink to allow it to call back into Chromium). The new code location within Blink for most code will be with the types from IDL, **//third_party/blink/public/common/indexeddb/** which will connect to the browser process directly via Mojo.

Making the Mojo file live in Blink gives these advantages:

- Allows use of IDB Mojo types directly from within Blink
- Allows use of backing classes that the Mojo class uses directly from within Blink

Moving the Mojo file from **//content/common/indexed_db/** to Blink requires that the classes and structs that are typemapped in Mojo also be moved to Blink. Code that refers to these classes only exists in **//content/** at the moment, so the backing classes and structs can be moved to Blink first and one at a time. After the classes and structs are moved, the Mojo file can be moved and references to the Mojo types can be updated from indexed_db.mojom.* to blink.mojom.IDB*.

Since all of this work taken together would be very large in a single CL, I plan to split this work up over multiple CLs.

crrev.com/c/1173713

Issues dealt with:

- Design plan to in-place migrate Content IDB mojo items to Blink IDB mojo items.
- Figure out where Blink-exposed APIs should live (in public/common/).
- Fix Blink common build target so that code isn't double-included on Android.
    - Required crrev.com/c/1176070. (Thanks @mek!)
- Don't use "using" in header files.
- Don't prefill the Blink IDB mojom file with types that haven't been converted yet.

8. DONE - Move IndexedDB Mojo file to Blink (part 2)

crrev.com/c/1185790

Issues dealt with:

- Use forward-decls where possible, then include the header only if necessary.
- Don't use "using" in header files. (hit again, by accident?, wish for a presubmit check)
- Refactor struct traits switch for readability.

9. DONE - Move IndexedDB Mojo file to Blink (part 3)

crrev.com/c/1189134

Issues dealt with:

- Had to convert all IDB Mojo interfaces at once since they refer to each other in their args.
- Manual changes would have taken a while, used grep and sed to update automatically.
    - Example:

        - git grep -E '\sFactory' content/{browser,common,renderer}/indexed_db
        - for file in `git grep -E '\sFactory' content/{browser,common,renderer}/indexed_db|sed -E 's/([^:]+):.*/\1/'`; do sed -i 's/\sFactory/IDBFactory/' $file; done
    - Example:
        - for file in `git grep ::blink::mojom::IDB|sed -E 's/([^:]+):.*/\1/'`; do sed -i 's/::blink::mojom::IDB/blink::mojom::IDB/' $file; done
- Use of "Continue" in mojom file triggered an Android compile issue in mojom bindings.
    - /b/swarming/w/ir/tmp/t/tmpooaK41/java/org/chromium/blink/mojom/IdbCursor.java:32: error: <identifier> expected
    - void continue(
    - ^
    - /b/swarming/w/ir/tmp/t/tmpooaK41/java/org/chromium/blink/mojom/IdbCursor.java:32: error: '(' expected
    - void continue(
    - ^
    - /b/swarming/w/ir/tmp/t/tmpooaK41/java/org/chromium/blink/mojom/IdbCursor.java:32: error: illegal start of type
    - void continue(
    - ^
- Remove global namespace prefix (::) from mojo references.
    - This was used previously due to a namespace conflict on indexed_db and that ended up leading to it being reused during the conversion.
- Try not to use "using" in .cc files. If needed, only use it in an anonymous namespace.
- Don't bring in the cpp_only = true removal in this CL to avoid a future potential issue causing this change to be reverted.

10. DONE - Merge //content/common/indexed_db/ into Blink

crrev.com/c/1194877

In this step, we will move remaining code from **//content/common/indexed_db/** to **Blink** to simplify the current indirection from Blink to Chromium (the current Chromium implementation is injected into Blink to allow it to call back into Chromium). The new code location within Blink for most code will be with the types from IDL, **//third_party/blink/public/common/indexeddb/** which will connect to the browser process directly via Mojo.

It turns out that after moving the Mojo types and backing classes and structs to Blink, what remains in //content/common/indexed_db is the indexed_db_constants.h file. This file has 3 parts to consider:

kNoDatabase

Not referenced anywhere. Delete.

kMaxIDBMessageOverhead

Arbitrary limit, referenced in //content, will move to Blink IDB mojom file.

kMaxIDBMessageSizeInBytes

References IPC::Channel::kMaximumMessageSize.

This variable was originally used to ensure messages that were too large aren't sent through the IPC channel. When IDB was Mojofied (around 1-2 years ago), we continued to use the same IPC value, even though Mojo has a differently sized limit. The Mojo limit is ~2x higher than the IPC limit.

Options:

- No equivalent is available/in use today for Mojo. Code doesn't expect to deal with this sort of limit directly. The Mojo code will directly verify that the data being sent isn't too large. I found no examples of code handling this directly:
  - Reviewed `git grep -i mojo | grep -i size` and `git grep -i mojo | grep -i max` output
- mek@ suggested that if code uses/checks a limit like this then it's not using Mojo correctly. This statement is meant to imply that the right way to interact with Mojo is to make smaller calls with reasonably sized arguments. There is a limit to Mojo message size and so code that expects that it might return a very large number of results in a single message should have a mechanism for chunking that reply across multiple messages.
- Alternative approach: let GetAll() return results piecemeal back from the browser to Blink over multiple calls. Probably has a performance implication on the current code.
- Simple approach: This code has been in place for 1-2 years and appears to also be an arbitrary limit. I don't propose to remove the check, but just to capture the old IPC limit value in Blink and update code to use that previous limit value.

Issues dealt with:

- Need better handling of large Mojo messages - filed https://crbug.com/879222
- Mojo doesn't support math expressions - filed https://crbug.com/879227

11. DONE - Upgrade WebIndexKeys type to WebIDBIndexKeys

https://crrev.com/c/1252912

12. DONE - Mojo: Fix Blink-specific export_define_blink directive

https://crrev.com/c/1282209

13. DONE - Mojo: Add ArrayTraitsWebVector helpers

https://crrev.com/c/1282211

14. DONE - (Andreas) Land getDatabaseNames() support

https://crrev.com/c/1228492

This work was done in parallel with the onion soup CL. We need to sequence these into the tree and ensure features aren't lost in whichever order they're landed in.

15. DONE - Use UnionTraits for IDBKeyData

https://crrev.com/c/1284390

Notes about using a span in mojom traits:

A span is a data structure that provides a sequence of values without copying referring to the original input. The span will be invalid if the original input becomes invalid. It's important to keep this in mind since changing to a span here may introduce a bug if the span is used incorrectly. I dug a little on this since it's my first use of this class (span) in this context (traits):

- The original input comes from a passed argument `key`. `key` is type const IndexedDBKey&.
- key.binary() returns type const base::string&.

- This binary() function here originally created a new std::vector, copied the raw data out of key.binary(), and returned the vector to the caller by value. (This may use copy elision / NRVO - named return value optimization - under the hood.)
- The string() function below takes a `key` const-ref and returns a const base::string& from `key.string(),` return type const base::string16&.

From this, I believe the original inputs for all of these are and will be valid for the length of time needed to satisfy mojom. Returning a span should be as safe as string() below returning a const-ref.

So what's happening in the system? Actually unwrapping all of the layers above this we have an IDB call in either content/renderer/ or content/browser/ which will trigger a call on a Mojo interface or send an async return result containing an IndexedDBKey argument. Mojo will serialize that argument and then post a task with that data for a Mojo receiver to take and deserialize on a different process/thread.

The IndexedDBKey instance only has to be valid for as long as the Mojo code needs to serialize the instance, which is by design short. Once the instance is serialized, it's safe for the instance to go away.

Here are some notes on traits getters (ie. "getters should be simple"):

https://chromium.googlesource.com/chromium/src/+/master/docs/security/mojo.md#structtraits-getters-should-be-simple

Here's documentation saying that getters for all fields are called exactly once:

https://cs.chromium.org/chromium/src/mojo/public/cpp/bindings/struct_traits.h?l=55

Here's a span instance being returned in a traits getter:

https://cs.chromium.org/chromium/src/device/gamepad/public/cpp/gamepad_mojom_traits.cc?sq=package:chromium&dr=CSs&g=0&l=200

Special thanks to Oksana for talking this through with me and for passing along pointers on this subject!

16. DONE - Merge //content/renderer/indexed_db/ into Blink

https://crrev.com/c/1265900

In this step, we will move code from **//content/renderer/indexed_db/** to **Blink**. The new code location within Blink for most code (including the cursor prefetching logic) will be in **//third_party/blink/renderer/modules/indexeddb/**.

Updates post-landing:

- Broke jumbo builds - https://crrev.com/c/1301434 and https://crrev.com/c/1301458
- Broke compatibility with Chrome stable IDB instances - https://crbug.com/899446
- Increased crash rate in mojo::core::NodeChannel::WriteChannelMessage - https://crbug.com/901269

17. SKIPPED - Add IDBKey kNullType support

https://crrev.com/c/1278141 (now deleted)

Victor's list of related resources:

- Handling Large Values in IndexedDB
- IndexedDB data path
- Blob processing in IDBObjectStore::put()

Skipping this CL. In theory this should be unnecessary once the Blink variant is mapped to the renderer/modules/indexeddb/ types. In practice, parts of this CL were necessary later on, specifically around IDBValue's use of injected primary keys when the key type is null.

18. DONE - Revert merge change

The merge CL introduced a binary incompatibility somehow with Chrome stable's IDB instances. See https://crbug.com/899446 for more info about that bug.

Revert "Remove using directives in web_idb_cursor_impl_unittest.cc" https://crrev.com/c/1305395

Revert "Remove using declaration for mojom::blink::IDBCursor" https://crrev.com/c/1305245
Revert "Merge" https://crrev.com/c/1305243

19. DONE - Add stable DB compatibility tests

https://crrev.com/c/1320693

These tests allow us to catch when a change is introduced that introduces an incompatibility with our existing on-disk representation. Verified that with this test in place and with the previous CL that introduced the incompatibility, the tests fail. After landing the Mojo IDBValue.bits change, these tests pass.

20. DONE - Change Mojo IDBValue.bits type from string to array<uint8>

https://crrev.com/c/1316444

This was the cause of the IndexedDB instance incompatibility. Fixed it before re-landing the merge CL.

21. DONE - Use ArrayDataView to convert IDBKey binary to std::string

https://crrev.com/c/1321164

Improves performance of serializing IndexedDBKey and WebIDBKey instances out of Mojo by:

- not creating a throwaway byte vector
- not passing a std::string/WebData by reference and creating a copy of that reference in the key ctor

22. DONE - Move content/renderer/indexed_db/ to Blink, take 2

This is the second pass at landing the big IndexedDB move from content to Blink.

Here are the CLs for this change:

- https://crrev.com/c/1318374 IndexedDB: Mojom updates: rename files, update return types, move target
- https://crrev.com/c/1316830 IndexedDB: Move content/renderer/indexed_db/ to Blink, take 2

Follow-up:

- (Raphael) WebIDBKey: Stop inlining ReleaseIdbKey()'s implementation CL at https://crrev.com/c/1352184

23. DONE - Remove redundant Web* enums

https://crrev.com/c/1303634

Convert enumerated structs to Mojo types.
After this is done, remove web_idb_types.h.

- This file now contains only: KeyType, KeyPathType, and kIDBOperationTypeCount
- KeyType and KeyPathType encode all of the key types because these aren't stored as enums in Mojo
- kIDBOperationTypeCount is necessary because Mojo doesn't auto-create a "count" of valid enums (ie. max + 1)

After this landed, the Deterministic Linux build broke due to a missing build dependency. The change was reverted at https://crrev.com/c/1337609 .

Relanding occurred in 2 separate CLs:

- https://crrev.com/c/1338201 Blink generated bindings: Add missing build dependencies
- https://crrev.com/c/1338379 IndexedDB: Remove redundant Web* enums, take 2

24. DONE - Remove UMA metric "open time" gathering from callbacks

https://crrev.com/c/1336429

What was being gathered?
UMA data is generated for 4 items from the browser process's IndexedDBCallbacks:

1. OnError(): if set, time from IndexedDBCallback creation to call
2. OnBlocked(): if set, time from IndexedDBCallback creation to call
3. OnUpgradeNeeded(): if set, time from IndexedDBCallback creation to call
4. OnSuccess(): if set, time from IndexedDBCallback creation to call

The connection_open_start_time_ member variable is set in the factory's Open() call. The member variable isn't set by any other path, so it's unset when callbacks are passed and created through other paths.

Where was this added? commit ed0b94 cmumford
Is anyone using it? Pinged cmumford.
Pros of removing? Reduces number of collected UMA stats. Not required for most callback removals, may be required for one or two. Would be nice cleanup.
Cons of removing? Could remove some important metrics data. From bugs it appears it served its purpose.

25. DONE - Remap Blink variant types onto renderer/modules/indexeddb/ classes

The intent of this project was to remove the Web* types that were once needed by the content layer. Now that the Renderer side of the Mojo pipe is bound within Blink, these Web* types can go away after the Mojo interfaces are bound to the native Blink types.

The IndexedDB design and implementation is layered on top of these classes, with one Web* type wrapping another Web* type. At the root of these structures are "web-safe" types like WebString, WebData, and WebVector. The next layer contains instances, pointers to instances, or references to instances of these types.

The non-Web* types in the IndexedDB Blink module mimic/mirror this design, but don't use any of the Web* types internally. I evaluated switching the outermost layer first from its Web* type to non-Web* type. This approach was complicated due to the number of references that had to be updated at once: the outermost layer before the change used WebString, WebIDBKey, WebIDBKeyArray, WebIDBValue, and moving at once required updating all codesites to switch to String, IDBKey, Vector<IDBKey>, IDBValue. Based on the complexity of this approach, this option was the least favored.

The next considered approach was to move the base types over, then the next types based on those types, etc, until all of the types were converted. This approach won out due to the ability to separate changes into simpler CLs, mark them as dependent on each other, and quickly adapt later CLs to changes that came up during authoring and review.

Here are the CLs that came about during this work:

- https://crrev.com/c/1344222 IndexedDB: Remap Blink variant metadata typemaps to modules types
- https://crrev.com/c/1352585 IndexedDB: Remove IDBDatalessKeyType

- https://crrev.com/c/1352622 IndexedDB: Convert key types to mojom enum
- https://crrev.com/c/1352623 IndexedDB: Convert key path types to mojom enum
- https://crrev.com/c/1352624 IndexedDB: No-op move of files to renderer/modules/indexeddb/
- https://crrev.com/c/1354334 IndexedDB: Map mojom key to blink IDBKey
- https://crrev.com/c/1357535 IndexedDB: map key range mojom to blink IDBKeyRange
- https://crrev.com/c/1359862 IndexedDB: Convert WebIDBValue usage to blink IDBValue
- https://crrev.com/c/1364301 IndexedDB: Remove WebData usage from frontend
- https://crrev.com/c/1364302 IndexedDB: Add Mojom typemap for IDBValue
- https://crrev.com/c/1364226 IndexedDB: Update Put() to take a std::unique_ptr<IDBValue>
- https://crrev.com/c/1361831 IndexedDB: Stop using WebString and WebVector
- https://crrev.com/c/1364342 IndexedDB: Remove WebIDBKeyPath
- https://crrev.com/c/1364343 IndexedDB: Rename WebIDBNameAndVersion to IDBNameAndVersion
- https://crrev.com/c/1366478 IndexedDB: Update IDBObserverChanges to stop using WebIDBObservation
- https://crrev.com/c/1366440 IndexedDB: Stop using WebIDBObservation
- https://crrev.com/c/1366271 IndexedDB: Remove unnecessary uses of "using"
- https://crrev.com/c/1366380 IndexedDB: Rename WebIDBDatabaseError to IDBDatabaseError

Notes I found useful for this work:

- Conceptually, the conversions were less like moving from a Web* type to a non-Web* type, and more like unwrapping the Web* type's inner non-Web* type instance to pass around in its place. eg. WebIDBKey held a private std::unique_ptr<IDBKey> instance, and to match the design constraints correctly, it was best to update the WebIDBKey types in function signatures to std::unique_ptr<IDBKey>.
- Mojo can map to unique_ptr<T> and scoped_refptr<T> just fine, with some caveats. For mapping to unique_ptr<T>, the type map must be move-only, so add [move_only] declarators in typemap. Use special handling to create the std::unique_ptr<T> instance. Here are some notes investigating a compile error with IDBKey, unique_ptr, and vector.
- Mojom doesn't know how to handle certain types, like Blink's GC'd types. For these types the right way to handle the conversion is to deal with the "Ptr" type and use a TypeConverter<> to handle serializing and deserializing, notes on how to typemap Mojom types to GC'd and shared ptr types. Also see https://crbug.com/700180.
- Our code had special enums defined outside of Mojom for key and key path types, and these were defined in multiple places (enums for WebIDBKey type, enums for IDBKey type), reduce these as much as possible.

- Mojom unions contain enums the way that Mojom enums do, but Mojom enums also contain max values which are needed in our code.
- There were "in-place" conversions that happened to Mojom types not previously defined in our typemap, collecting these into the typemap simplifies our code. eg. mojom::blink::IDBValuePtr usage with in-place conversions to/from WebIDBValue became a mojom::blink::IDBValue typemap with traits conversions to blink::IDBValue.
- Once all top references to the Web* types were removed, it was safe to simplify move the Web* types to the Blink modules folder where they could be updated to switch to non-Web* type compositions internally, or to intermixed with non-Web* types
- Move non-simple types as much as possible to avoid the cost of copies
- Write Mojom tests to verify that complex conversion code works correctly, we have WPT tests that can catch issues but take longer to build + run compared to a content_browsertest or blink_unit_test

26. DONE - Correct origin security checks

crbug.com/467150

Move the browser-side IDB Factory interface from RenderProcessHost to an origin-providing context.
Write-up: 2018/12/19 Securing IndexedDB against improper origin access

Here are the CLs needed for this work:
- https://crrev.com/c/1384849 IndexedDB: Update factory to use origin supplied by browser process
- https://crrev.com/c/1387552 IndexedDB: Remove origin from IDBFactory Mojo calls
- https://crrev.com/c/1399822 IndexedDB: Check for valid origin before binding
- https://crrev.com/c/1403351 IndexedDB: Skip ReportBadMessage() at bind-time

Verify:
- [DONE] AddObserver() change from RenderProcessHostImpl is included (still uses RPH, so no change needed)
- [DONE] Verify service worker and shared worker tests exist for IndexedDB
    - Yes, here:
    - ls -1 *worker* # in web_tests
    - basics-shared-workers-expected.txt
    - basics-shared-workers.html

    - basics-workers-expected.txt
    - basics-workers.html
    - cursor-advance-workers-expected.txt
    - cursor-advance-workers.html
    - deletedatabase-delayed-by-open-and-versionchange-workers-expected.txt
    - deletedatabase-delayed-by-open-and-versionchange-workers.html
    - index-basics-workers-expected.txt
    - index-basics-workers.html
    - objectstore-basics-workers-expected.txt
    - objectstore-basics-workers.html
    - observer-worker.html
    - observer-workers.html
    - open-twice-workers-expected.txt
    - open-twice-workers.html
    - pending-activity-workers-expected.txt
    - pending-activity-workers.html
    - transaction-complete-workers-expected.txt
    - transaction-complete-workers.html
    -
    - ls -1 *worker* # in wpt
    - idb-explicit-commit.any.worker-expected.txt
    - idb_webworkers.htm
    - idbworker.js
    -
    - -- dmurph points out that any test with ".any.js" as a suffix will also test workers
- [IGNORE] Add test to verify renderer with bad origin is isolated, see IsolatedOriginTest and this CL
    - Don't think this can be tested, the change fixes the code so the origin is no longer passed

When done:

- [DONE] Remove IndexedDB component from https://crbug.com/467150
- [DONE] Update Site Isolation vs Compromised Renderers section on IndexedDB to mark resolved

27. IN PROGRESS - Use native Mojo callbacks

Use Mojo's native callback mechanism so we can start moving away from the complicated IDBCallbacks interface and instead use a simpler and easier to reason about async response return approach.

Example
Before:

```
  Put(int64 transaction_id,
      int64 object_store_id,
      IDBValue value,
      IDBKey key,
      IDBPutMode mode,
      array<IDBIndexKeys> index_keys,
      associated IDBCallbacks callbacks);
```

After

```
struct IDBDatabasePutResponse {
    <return data>
};

  Put(int64 transaction_id,
      int64 object_store_id,
      IDBValue value,
      IDBKey key,
      IDBPutMode mode,
      array<IDBIndexKeys> index_keys) => (IDBDatabasePutResponse response);
```

Here are the CLs necessary for this change:

- https://crrev.com/c/1336660 IndexedDB: Move helpers out of IndexedDBCallbacks so they're accessible
- TODO https://crrev.com/c/1368310 IndexedDB: Move IDBKeyRangeBuilder::Build to IDBKeyRange::Create

- …
- EXPLORATORY For IDBCursor.Advance(): https://crrev.com/c/1336662

IDBDatabaseCallbacks is used by Factory.open() and is a special case. Don't deal with that here, it will be a separate project someone takes on.

If all IDBCallbacks uses are removed, then the Callbacks interface can go away.

Todo:

- ~~How do existing async return values work?~~

How do existing async return values work? Example using web_bluetooth:

- web_bluetooth.mojom RequestDevice
  - interface WebBluetoothService {
  -   RequestDevice(WebBluetoothRequestDeviceOptions options)
  -     => (WebBluetoothResult result, WebBluetoothDevice? device);
- bluetooth.cc RequestDeviceCallback
  - void Bluetooth::RequestDeviceCallback(
  -     ScriptPromiseResolver* resolver,
  -     mojom::blink::WebBluetoothResult result,
  -     mojom::blink::WebBluetoothDevicePtr device) {
  -   if (!resolver->GetExecutionContext() ||
  -       resolver->GetExecutionContext()->IsContextDestroyed())
  -     return;
  -
  -   if (result == mojom::blink::WebBluetoothResult::SUCCESS) {
  -     BluetoothDevice* bluetooth_device =
  -         GetBluetoothDeviceRepresentingDevice(std::move(device), resolver);
  -     resolver->Resolve(bluetooth_device);
  -   } else {
  -     resolver->Reject(BluetoothError::CreateDOMException(result));
  -   }

    - }
- bluetooth.cc blink service_->RequestDevice() call
    - Script Bluetooth::requestDevice()
    - ...
    -   service_->RequestDevice(
    -       std::move(device_options),
    -       WTF::Bind(&Bluetooth::RequestDeviceCallback, WrapPersistent(this),
    -                 WrapPersistent(resolver)));
    - ...
- web_bluetooth_service_impl.cc RequestDevice() definition
    - void WebBluetoothServiceImpl::RequestDevice(
    -     blink::mojom::WebBluetoothRequestDeviceOptionsPtr options,
    -     RequestDeviceCallback callback) {
    -   RecordRequestDeviceOptions(options);
    -
    -   if (!GetAdapter()) {
    -     if (BluetoothAdapterFactoryWrapper::Get().IsLowEnergySupported()) {
    -       BluetoothAdapterFactoryWrapper::Get().AcquireAdapter(
    -           this, base::Bind(&WebBluetoothServiceImpl::RequestDeviceImpl,
    -                            weak_ptr_factory_.GetWeakPtr(),
    -                            base::Passed(&options), base::Passed(&callback)));
    -       return;
    -     }
    -     RecordRequestDeviceOutcome(
    -         UMARequestDeviceOutcome::BLUETOOTH_LOW_ENERGY_NOT_AVAILABLE);
    -     std::move(callback).Run(
    -         blink::mojom::WebBluetoothResult::BLUETOOTH_LOW_ENERGY_NOT_AVAILABLE,
    -         nullptr /* device */);
    -     return;
    -   }
    -   RequestDeviceImpl(std::move(options), std::move(callback), GetAdapter());
    - }
- web_bluetooth_service_impl.cc OnGetDeviceSuccess() calling callback
    - void WebBluetoothServiceImpl::OnGetDeviceSuccess(

```
    RequestDeviceCallback callback,
    blink::mojom::WebBluetoothRequestDeviceOptionsPtr options,
    const std::string& device_address) {
  device_chooser_controller_.reset();

  const device::BluetoothDevice* const device =
      GetAdapter()->GetDevice(device_address);
  if (device == nullptr) {
    DVLOG(1) << "Device " << device_address << " no longer in adapter";
    RecordRequestDeviceOutcome(UMARequestDeviceOutcome::CHOSEN_DEVICE_VANISHED);
    std::move(callback).Run(
        blink::mojom::WebBluetoothResult::CHOSEN_DEVICE_VANISHED,
        nullptr /* device */);
    return;
  }

  const WebBluetoothDeviceId device_id =
      allowed_devices().AddDevice(device_address, options);

  DVLOG(1) << "Device: " << device->GetNameForDisplay();

  blink::mojom::WebBluetoothDevicePtr device_ptr =
      blink::mojom::WebBluetoothDevice::New();
  device_ptr->id = device_id;
  device_ptr->name = device->GetName();

  RecordRequestDeviceOutcome(UMARequestDeviceOutcome::SUCCESS);
  std::move(callback).Run(blink::mojom::WebBluetoothResult::SUCCESS,
                          std::move(device_ptr));
}
```

Todo:

- Convert a single instance of callbacks usage to return values

Start with Advance:

- Browser process: indexed_db_cursor.cc: IndexedDBCursor::Advance

```
void IndexedDBCursor::Advance(uint32_t count,
                              scoped_refptr<IndexedDBCallbacks> callbacks) {
  IDB_TRACE("IndexedDBCursor::Advance");

  if (closed_) {
    callbacks->OnError(CreateCursorClosedError());
    return;
  }

  transaction_->ScheduleTask(
      task_type_,
      BindWeakOperation(&IndexedDBCursor::CursorAdvanceOperation,
                        ptr_factory_.GetWeakPtr(), count, callbacks));
}
```

- Browser process: indexed_db_cursor.cc: IndexedDBCursor::CursorAdvanceOperation

```
leveldb::Status IndexedDBCursor::CursorAdvanceOperation(
    uint32_t count,
    scoped_refptr<IndexedDBCallbacks> callbacks,
    IndexedDBTransaction* /*transaction*/) {
  IDB_TRACE("IndexedDBCursor::CursorAdvanceOperation");
  leveldb::Status s = leveldb::Status::OK();

  if (!cursor_ || !cursor_->Advance(count, &s)) {
    cursor_.reset();
    if (s.ok()) {
      callbacks->OnSuccess(nullptr);
      return s;
    }
    Close();
```

    - callbacks->OnError(IndexedDBDatabaseError(
    - blink::kWebIDBDatabaseExceptionUnknownError, "Error advancing cursor"));
    - return s;
    - }
    -
    - callbacks->OnSuccess(key(), primary_key(), Value());
    - return s;
    - }
- Browser process: indexed_db_callbacks.cc calls SuccessCursorContinue
    - void IndexedDBCallbacks::IOThreadHelper::SendSuccessCursorContinue(
    - const IndexedDBKey& key,
    - const IndexedDBKey& primary_key,
    - blink::mojom::IDBValuePtr value,
    - const std::vector<IndexedDBBlobInfo>& blob_info) {
    - DCHECK_CURRENTLY_ON(BrowserThread::IO);
    - if (!callbacks_)
    - return;
    - if (!dispatcher_host_) {
    - OnConnectionError();
    - return;
    - }
    -
    - if (!value || CreateAllBlobs(blob_info, &value->blob_or_file_info))
    - callbacks_->SuccessCursorContinue(key, primary_key, std::move(value));
    - }

Steps:

- Create indexed_db_return_value.cc, move ConvertReturnValue here
- Move the success path callbacks code to indexed_db_cursor.cc

28. Merge Impl classes in Blink renderer/modules/indexeddb

Now that content/renderer/indexed_db/ has moved into Blink, there's a layer of abstraction that is no longer needed. Example: IDBFactory talks to WebIDBFactoryImpl directly rather than getting an instance through BlinkRenderThreadImpl. We can remove some of these and let Blink renderer/modules/indexeddb/ classes hold the Mojo instances for the interfaces directly. This will improve readability for moving through the IDB layers in Blink (and then to browser, and back).

Possibly 1-2 weeks, should be scoped work, not likely to sprawl.

- Cursor
- Database
- Factory

29. Remove transaction IDs from interface

Add IDBTransaction interface to indexeddb.mojom.
Look for `transaction_id` in indexeddb.mojom, pick a method to copy to IDBTransaction.
Build that method in IDBTransaction, then update calling site to use the IDBTransaction interface, instead.
Delete the old method.
Repeat for remaining `transaction_id` interfaces.

Example path:

- IDBRequest::CreateWebCallbacks()
    - std::unique_ptr<WebIDBCallbacks> IDBRequest::CreateWebCallbacks() {
    -   DCHECK(!web_callbacks_);
    -   std::unique_ptr<WebIDBCallbacks> callbacks =
    -       WebIDBCallbacksImpl::Create(this);
    -   web_callbacks_ = callbacks.get();
    -   return callbacks;
    - }
- WebIDBCallbacksImpl::Create()

    - std::unique_ptr<WebIDBCallbacksImpl> WebIDBCallbacksImpl::Create(
    -     IDBRequest* request) {
    -   return base::WrapUnique(new WebIDBCallbacksImpl(request));
    - }
    - … sets request, which later is used …
- [WebIDBCallbacksImpl::OnSuccess()]()
    - void WebIDBCallbacksImpl::OnSuccess(WebIDBCursor* cursor,
    -                                     WebIDBKey key,
    -                                     WebIDBKey primary_key,
    -                                     WebIDBValue value) {
    -   if (!request_)
    -     return;
    -
    -   probe::AsyncTask async_task(request_->GetExecutionContext(), this, "success");
    -   std::unique_ptr<IDBValue> idb_value = value.ReleaseIdbValue();
    -   idb_value->SetIsolate(request_->GetIsolate());
    -   request_->HandleResponse(base::WrapUnique(cursor), key.ReleaseIdbKey(),
    -                            primary_key.ReleaseIdbKey(), std::move(idb_value));
    - }
- [IDBRequest::HandleResponse()]()
    - void IDBRequest::HandleResponse(std::unique_ptr<WebIDBCursor> backend,
    -                                 std::unique_ptr<IDBKey> key,
    -                                 std::unique_ptr<IDBKey> primary_key,
    -                                 std::unique_ptr<IDBValue> value) {
    -
    - …
    -     return EnqueueResponse(std::move(backend), std::move(key),
    -                            std::move(primary_key), std::move(value));
    - ...
- [IDBRequest::EnqueueResponse()]()
    - void IDBRequest::EnqueueResponse(std::unique_ptr<WebIDBCursor> backend,

  - std::unique_ptr<IDBKey> key,
  - std::unique_ptr<IDBKey> primary_key,
  - std::unique_ptr<IDBValue> value) {
  - …
  - switch (cursor_type_) {
  - case indexed_db::kCursorKeyOnly:
  - cursor = IDBCursor::Create(std::move(backend), cursor_direction_, this,
  - source, transaction_.Get());
  - break;
  - case indexed_db::kCursorKeyAndValue:
  - cursor = IDBCursorWithValue::Create(std::move(backend), cursor_direction_,
  - this, source, transaction_.Get());
  - break;
  - default:
  - NOTREACHED();
  - }
  - ...
- IDBCursor::Create()
  - IDBCursor* IDBCursor::Create(std::unique_ptr<WebIDBCursor> backend,
  - WebIDBCursorDirection direction,
  - IDBRequest* request,
  - const Source& source,
  - IDBTransaction* transaction) {
  - return new IDBCursor(std::move(backend), direction, request, source,
  - transaction);
  - }
  - … which later has Delete() called on it …
- deleteMethod()
  - static void deleteMethod(const v8::FunctionCallbackInfo<v8::Value>& info) {
  - ExceptionState exceptionState(info.GetIsolate(), ExceptionState::kExecutionContext, "IDBCursor", "delete");
  -

  - IDBCursor* impl = V8IDBCursor::ToImpl(info.Holder());
  -
  - ScriptState* scriptState = ScriptState::ForRelevantRealm(info);
  -
  - IDBRequest* result = impl->Delete(scriptState, exceptionState);
  - if (exceptionState.HadException()) {
  -   return;
  - }
  - // [NewObject] must always create a new wrapper. Check that a wrapper
  - // does not exist yet.
  - DCHECK(!result || DOMDataStore::GetWrapper(result, info.GetIsolate()).IsEmpty());
  - V8SetReturnValue(info, result);
  - }
- IDBCursor::Delete()
  - IDBRequest* IDBCursor::Delete(ScriptState* script_state,
  -                              ExceptionState& exception_state) {
  - IDB_TRACE("IDBCursor::deleteRequestSetup");
  - IDBRequest::AsyncTraceState metrics("IDBCursor::delete");
  - if (!transaction_->IsActive()) {
  -   exception_state.ThrowDOMException(
  -       DOMExceptionCode::kTransactionInactiveError,
  -       transaction_->InactiveErrorMessage());
  -   return nullptr;
  - }
  - if (transaction_->IsReadOnly()) {
  -   exception_state.ThrowDOMException(
  -       DOMExceptionCode::kReadOnlyError,
  -       "The record may not be deleted inside a read-only transaction.");
  -   return nullptr;
  - }
  - if (IsDeleted()) {

```
    exception_state.ThrowDOMException(DOMExceptionCode::kInvalidStateError,
                                      IDBDatabase::kSourceDeletedErrorMessage);
    return nullptr;
  }
  if (!got_value_) {
    exception_state.ThrowDOMException(DOMExceptionCode::kInvalidStateError,
                                      IDBDatabase::kNoValueErrorMessage);
    return nullptr;
  }
  if (IsKeyCursor()) {
    exception_state.ThrowDOMException(DOMExceptionCode::kInvalidStateError,
                                      IDBDatabase::kIsKeyCursorErrorMessage);
    return nullptr;
  }
  if (!transaction_->BackendDB()) {
    exception_state.ThrowDOMException(DOMExceptionCode::kInvalidStateError,
                                      IDBDatabase::kDatabaseClosedErrorMessage);
    return nullptr;
  }

  IDBRequest* request = IDBRequest::Create(
      script_state, this, transaction_.Get(), std::move(metrics));
  transaction_->BackendDB()->Delete(
      transaction_->Id(), EffectiveObjectStore()->Id(),
      WebIDBKeyView(IdbPrimaryKey()), request->CreateWebCallbacks().release());
  return request;
}
```

Paths in:

- IDBObjectStore::Create(), takes an IDBTransaction*, one path is IDBTransaction::createObjectStore()

Questions:

- What's transaction() in IDBObjectStore::createIndex()?
- When does IDBTransaction.id_ get set? Anytime this is set to a real value, we should be able to also set a Mojo instance.
    - There's IDBTransaction::CreateNonVersionChange(), this is called in IDBDatabase::transaction()
- IDBDatabase::transaction() calls mojo database CreateTransaction(), how does that work?

Ideas:

- Factory gives us the database, the database should be able to give us a transaction, follow a similar model
- How does factory offer a database mojo instance to Blink?
    - IndexedDBCallbacksImpl::SuccessDatabase()
        - Receives IDBDatabaseAssociatedPtrInfo database_info
        - Creates a WebIDBDatabase* using WebIDBDatabaseImpl, passing database_info
        - Passes the WebIDBDatabase to callbacks_->OnSuccess(), which runs Blink renderer/modules/ code
- Steps
    - Create web_idb_transaction.h, similar to web_idb_database.h
    - Create webidbtransaction_impl.h and webidbtransaction_impl.cc, similar to webidbdatabase_impl.*
        - The impl will hold a blink::mojom::IDBTransactionAssociatedPtr transaction_
    - The renderer/modules/idb_transaction.cc will hold an instance of web_idb_transaction (aka std::unique_ptr<WebIDBTransaction>)
    - … next steps? ...

30. -- At this point reach 1.0 OKR --

31. DONE - Diagnose perf regression in IOThreadHelper removal

crbug.com/868995

Investigated and diagnosed cause of perf regression. Cause was that moving WebIDBFactoryImpl Mojo binding from IO thread to main thread moved all associated interfaces. The explicit thread hopping was still in place during this period, so the renderer was doing extra thread hops to get back to the IO thread which then hopped back to the main thread. All of this together resulted in a perf regression which was reduced/moved back to normal after the IO thread hopping was removed.

(Not needed) Use Forklift to detect if there’s a relative difference between 3505 and 3506.

32. SKIPPED - Update Mojo Factory object to be associated to Database

Earlier, we believed we need to cause the Factory objects to be associated to the Database objects so their messages arrive in relative order.

However, after working on this further, the work is not needed:

- The root of the IndexedDB system is the Factory binding.
- Factory API calls are given *Callbacks interfaces, which are marked associated.
- Passing *Callbacks across the Factory interfaces associates them to the Factory.
- If a Factory interface is called that passes a Database interface back via *Callbacks, the Database interface becomes associated to the *Callbacks interface's main associated binding, which is the Factory interface in this case.
- There are no other interfaces present in the hierarchy that pass a Factory interface, and all other interfaces besides the root Factory interface are marked associated.

33. DONE - Remove IO thread hopping in browser process.

Occurs in content/browser/indexed_db/indexed_db_callbacks.cc and indexed_db_database_callbacks.cc.

Also see https://crbug.com/717798.

The hopping is needed to send a message back across the callbacks and to interact with blobs on disk. An earlier step is focused on removing/slimming down the IDBCallback interface implementation, so the browser side of indexed_db_callbacks.cc will get smaller this way.

Another part of this is that even with the IDBCallback interface usage reduced, there will still be Mojo-native callbacks, and those need to be called on the same thread/sequence that the original Mojo call was triggered on. For some of these callbacks, those occur in cursor_impl.cc. This means that after slimming down IDBCallback usage, there will still need to be IO thread hopping in indexed_db_cursor.cc. Removing the IO thread hopping from here is possible if we can set all of the Mojo interfaces in the browser process to be bound on the IDB sequenced task runner.

It turns out that Hajime is working on adding the ability to bind a Mojo interface on a sequenced task runner. See these in-progress CLs/bugs:

- Use per-frame task runners at mojo https://crbug.com/913912
- DNS: Add task_runner params to some of mojo bindings and use it. https://crrev.com/c/1369499
- Use SequenceLocalStorage and SequencedTaskRunner in mojo/public/cpp https://crrev.com/c/1386068
- Add task_runner params to some of mojo bindings and use it. https://crrev.com/c/1379608/4

34. Remove "cpp_only = true" from content/common/BUILD.gn mojo target

35. Update IndexedDBCallbacksImpl to pass cursor by value

See Marijn's comment:

> nit (and I know this is just code you're moving around) this should really pass the smart pointer by value and std::move to avoid needless ref-count-churning.

36. Change UMA_HISTOGRAM_ENUMERATION so kTypeEnumMax can be removed

Follow-up from https://crrev.com/c/1265900/26/third_party/blink/renderer/modules/indexeddb/idb_key.h#111:
dcheng: Is this enum value critical for something? Maybe we should just remove it (in a followup)?
cmp: |kTypeEnumMax| is used in idb_object_store.cc while in GenerateIndexKeysForValue() and DoPut(). The use appears real and necessary for an EnumerationHistogram.
dcheng: This can actually be removed. There's a two-argument form of UMA_HISTOGRAM_ENUMERATION that autodeduces:
https://cs.chromium.org/chromium/src/base/metrics/histogram_macros.h?rcl=4c342467a1b658a638df41b67c3a04adf7ebb6a7&l=39

(but it requires changing the enum to be scoped, so it's best left for a followup)

37. Map IDBValue.bits directly to std::string / WebData

https://crbug.com/902498

38. Add exhibition DB to verify IndexedDB maintains binary compatibility

Read/write complete DB with all data types.

39. Consider additional testing pattern for Mojom interfaces using InterceptorForTesting

See https://crrev.com/c/611102 StoragePartitonInterceptor / StoragePartitionInterceptor

Worklog

2018-08-28

- Update planned steps. Originally planned this work after the Mojo move to Blink (required in green, cleanup in blue, not needed in red, unknown in yellow):
    - 10. Convert enumerated structs to Mojo types.
        - -> cleanup
    - 11. Diagnose perf regression in IOThreadHelper removal.
    - crbug.com/868995
    -
    - Use Forklift to detect if there's a relative difference between 3505 and 3506.
        - -> cleanup but technically fallout from an earlier change, let's see if we can track it down sooner
    - 12. Use ScopedMessageHandle to communicate across processes/threads.
    - Start with database callbacks.
    - Look at transferable_message_struct_traits.cc for a related pattern.
    - Use ::CreateX() pattern in place of "new X" in Blink code. (re: IDBIndexMetadata)
        - -> I honestly don't remember the context for why we thought this was necessary. From what I can see I can't find any code in content/renderer/indexed_db/ that requires this sort of work.
    - 13. Remove lifecycle Mojo instance management.
    - See UnregisterMojoOwnedDatabaseCallbacks, etc.
        - -> All of this was done as part of callback thread hopping removal.
    - 14. Promote lingering types in //content/common/indexed_db/ to Mojo.

        - -> There are no remaining types or classes in content/common/ to upgrade to Mojo.
- 15. Move items from //content/common/indexed_db/ to //content/browser/ that are only used there.
- Some items in //content/common/indexed_db/ may now only be used in the browser-side implementation. Move those items specifically to //content/browser/indexed_db/.
    - -> There are no classes that are in content/common/ that are only used in content/browser.
- 16. Merge //content/{common,renderer}/ into Blink.
- In this step, we will move code from //content/{common,renderer}/indexed_db/ to Blink to simplify the current indirection from Blink to Chromium (the current Chromium implementation is injected into Blink to allow it to callback into Chromium).
-
- The new code location within Blink for most code will be with the types from IDL, //third_party/blink/public/platform/modules/indexeddb/ which will connect to the browser process directly via Mojo.
-
- The cursor prefetching logic will end up in //third_party/blink/renderer/modules/indexeddb/.
    - -> Required to complete this work.
- 17. Correct origin security checks.
- crbug.com/467150
- Move the browser-side IDB Factory interface from RenderProcessHost to RenderFrameHost/SharedWorkerHost/ServiceWorkerHost.
- Pass origin into interface binding in previous step. (See LockManager::CreateService for an example where origin is associated with interface binding, and LockManager::RequestLock where it is pulled out on method calls.)
- One way this could be done: Expose IDBFactory from RendererInterfaceBinders::InitializeParameterizedBinderRegistry().
- Remove origin from mojo interface.
- Now origin checks are correctly in place (completes IDB component for bug/467150).
- Ensure all Factory, Database, and Cursor pipes are closed on navigation.
- Also see https://chromium.googlesource.com/chromium/src.git/+/01f1a282386443edb7a937ad0b2ace7ea0ae7faf
    - -> cleanup
- 18. Update Mojo Factory object to be associated to Database.
- We determined we need to cause the Factory objects to be associated to the Database objects so their messages arrive in relative order.

    - -> cleanup
  - 19. Replace callback objects with normal mojo instances.
    - -> cleanup
  - 20. Remove transaction IDs from interface.
    - -> cleanup
  - 21. Remove IO thread hopping in browser process.
  - Occurs in content/browser/indexed_db/indexed_db_callbacks.cc and indexed_db_database_callbacks.cc.
    - -> cleanup
  - 22. Remove "cpp_only = true" from content/common/BUILD.gn mojo target.
    - -> cleanup
- Updated list:
  - 10. Diagnose perf regression in IOThreadHelper removal.
  - crbug.com/868995
  -
  - Use Forklift to detect if there's a relative difference between 3505 and 3506.
  - 11. Merge //content/common/indexed_db/ into Blink.
  - 12. Merge //content/renderer/indexed_db/ into Blink.
  - In this step, we will move code from //content/{common,renderer}/indexed_db/ to Blink to simplify the current indirection from Blink to Chromium (the current Chromium implementation is injected into Blink to allow it to callback into Chromium).
  -
  - The new code location within Blink for most code will be with the types from IDL, //third_party/blink/public/platform/modules/indexeddb/ which will connect to the browser process directly via Mojo.
  -
  - The cursor prefetching logic will end up in //third_party/blink/renderer/modules/indexeddb/.
  - 13. Use ScopedMessageHandle to communicate across processes/threads.
  - Start with database callbacks.
  - Look at transferable_message_struct_traits.cc for a related pattern.
  - Use ::CreateX() pattern in place of "new X" in Blink code. (re: IDBIndexMetadata)
  - 14. Correct origin security checks.
  - crbug.com/467150

    - Move the browser-side IDB Factory interface from RenderProcessHost to RenderFrameHost/SharedWorkerHost/ServiceWorkerHost.
    - Pass origin into interface binding in previous step. (See LockManager::CreateService for an example where origin is associated with interface binding, and LockManager::RequestLock where it is pulled out on method calls.)
    - One way this could be done: Expose IDBFactory from RendererInterfaceBinders::InitializeParameterizedBinderRegistry().
    - Remove origin from mojo interface.
    - Now origin checks are correctly in place (completes IDB component for bug/467150).
    - Ensure all Factory, Database, and Cursor pipes are closed on navigation.
    - Also see https://chromium.googlesource.com/chromium/src.git/+/01f1a282386443edb7a937ad0b2ace7ea0ae7faf
    - 15. Convert enumerated structs to Mojo types.
    - 16. Update Mojo Factory object to be associated to Database.
    - We determined we need to cause the Factory objects to be associated to the Database objects so their messages arrive in relative order.
    - 17. Replace callback objects with normal mojo instances.
    - 18. Remove transaction IDs from interface.
    - 19. Remove "cpp_only = true" from content/common/BUILD.gn mojo target.
    - 20. Remove IO thread hopping in browser process.
    - Occurs in content/browser/indexed_db/indexed_db_callbacks.cc and indexed_db_database_callbacks.cc.

2018-10-11

- Unnecessary steps
    - 13. Use ScopedMessageHandle to communicate across processes/threads.
        - Start with database callbacks.
        - Look at transferable_message_struct_traits.cc for a related pattern.
        - Use ::CreateX() pattern in place of “new X” in Blink code. (re: IDBIndexMetadata)

References

IndexedDB spec.

Onion Soup Design Doc.
Mojofy and Onion Soup - File System Messages.
Mojofy and Onion Soup - Cache Storage.
611935 - Content Modularization Project: Blob Service.
627484 - Convert IndexedDB from Chrome legacy IPC to Mojo.
717812 - Onion-soup for IndexedDB.
764130 - [IndexedDB] Have mojo-only integration between IDB and blobs.