Keyed Service Scheduling (DRAFT)

alexclarke@
Promises design Prototype

Background

The HelloChrome Hackathon identified opportunities for cutting 30% off the start up time of chrome on low end Android phones. This is a complex space but a large part of it is the way KeyedServices are initialized. There are several hundred services which are eagerly constructed during Profile init. On high end machines this is reasonably quick (~100ms) but on low end android devices it can be painfully slow (2000+ms).

<Insert trace & analysis>
<insert dump from DependencyGraph::DumpAsGraphviz>

A common scenario with poor user experience is switching away from a web page and back again. On low end devices that involves a restart due to the OOM killer. It's likely the majority of KeyedServices are not required to issue the navigation but they're currently instantiated first.

Design

We propose to defer as much initialization as possible that is not on the critical path. We propose to do this by replacing the dependency system backing KeyedServices with one based on Promises and to embrace lazy loading of services.

We anticipate lazy loading if applied everywhere may regress some systems, so we also plan to support background service construction. Once the SequenceManager has been ported from blink to base we intend to make Idle and Budget based task runners available in the browser UI thread. We will use those to try and proactively instantiate lower priority services ahead of time.

Some browser startup code (e.g. ProfileInit and TabHelpers::AttachTabHelpers) synchronously create many dozens of services. Where possible we will lazily construct these instead. It may make sense to defer the call to ProfileInit and AttachTabHelpers until the required dependant services are ready using Promises::All().

A new way to get services

We will add the following API to content::BrowserContext:

```
// Returns a ServicePromise for Service
template <typename Service>
ServicePromise<Service> GetServicePromise();

// Returns a pointer to Service if it has been constructed.
template <typename Service>
Service* GetServiceIfExists();

template <typename Service>
void ResgisterServiceForTest(Service*);

// An extension of base::Promise with some syntactic sugar for use
// with services.
template <typename Service>
class ServicePromise<Service> : public base::Promise<Service> {
 public:
  // Returns true if the promise has resolved and is non-null
  bool operator() const;

  Service* operator*();
  Service* operator->();

  // Uses a nested messageloop to wait for a service to be constructed.
  // Used in the prototype but we hope to largely avoid this.
  Service* Await();
};
```

This will use the following factory template to construct the service promise.

```
template <typename Service>
struct ProfileServiceFactory {
  static constexpr bool is_refcounted =
      base::internal::IsRefCountedType<Service>::value;
  using PromiseType = std::conditional_t<is_refcounted,
                                         base::Promise<scoped_refptr<Service>>,
                                         base::Promise<Service*>>;

  static PromiseType CreateServicePromise(Profile* profile);
  static PromiseType CreateBrowserContextServicePromise(
      content::BrowserContext* context);
};
```

Example specialization:

```
template <>
```

```
ServicePromise<BookmarkModel*>
ProfileServiceFactory<BookmarkModel>::CreateServicePromise(Profile* profile) {
  user_prefs::PrefRegistrySyncable* registry = profile->GetPrefsRegistry();
  if (registry)
    bookmarks::RegisterProfilePrefs(registry);

  return base::Promises::All(
             profile->GetServicePromise<BookmarkUndoService>(),
             profile->GetServicePromise<bookmarks::ManagedBookmarkService>(),
             profile->GetServicePromise<bookmarks::StartupTaskRunnerService>())
      .ThenOnCurrent(
          FROM_HERE,
          base::BindOnce(
              [](Profile* profile, BookmarkUndoService* bookmark_undo_service,
                 bookmarks::ManagedBookmarkService* managed_bookmark_service,
                 bookmarks::StartupTaskRunnerService*
                     startup_task_runner_service) -> BookmarkModel* {
                BookmarkModel* bookmark_model =
                    new BookmarkModel(std::make_unique<ChromeBookmarkClient>(
                        profile, managed_bookmark_service));
                bookmark_model->Load(
                    profile->GetPrefs(), profile->GetPath(),
                    startup_task_runner_service->GetBookmarkTaskRunner(),
                    content::BrowserThread::GetTaskRunnerForThread(
                        content::BrowserThread::UI));
                bookmark_undo_service->Start(bookmark_model);
                return bookmark_model;
              },
              base::Unretained(profile)));
}
```

We will initially start converting services with no dependencies, and as we go along we will modify the various ::GetForProfile and ::GetForProfileIfExists methods to use the APIs above.

Then we will start converting services whose dependencies are all new style. Eventually all services will be converted. As a cleanup we'd like to remove all factory classes in favor of the new APIs except for those where the GetForProfile calls take extra parameters. These will require special handling and will likely remain.

In addition we'd like to modify the service constructors to accept pointers to all dependant services rather than just the profile. We think this will make testing easier and will eliminate some redundant lookups.

TODO: design support for shutdown observers
TODO: what about WebContents KeyedServices, should we replace those too?

Results

On linux with the patch 103 services where lazily loaded when booting chrome. By comparison normally 217 services are loaded.

Perf measurements on a Nokia TA-1060 don't look terribly exciting. It did reduce ProfileManager::GetProfile from 62.105ms to 42.925ms thread time but getting that gain entails significant engineering effort. Currently there are many lower hanging fruit.