Persist Offline Info in an Navigation

Jian Li (jianli@chromium.org)
Last Update: September 22, 2016

Background

Chrome for Android supports saving snapshots of web pages in MHTML archive format such that they can be later accessed when the network is disconnected or in slow/poor conditions. This is done by intercepting the network requests and replacing with the offline content if needed.

Since M54, the new Download Home UI has been introduced. When the user clicks on the Download button, the current web page will be downloaded and saved as a snapshot. Later on, the user can go to Download home page and open it. The saved page should always be opened regardless whether the network is connected or not. To make this happen, a custom offline header, like “X-Chrome-offline: reason=download”, is sent with the request such that offline interceptor can skip the network check and force to load the offline page.

However, the offline header is not persisted with the navigation. Thus when Chrome restarts, the online version could be loaded if the network is connected.

Proposal

In order to bring up the same page as the one before Chrome restarts, we need to persist the offline info embedded in the custom offline header.

Support Persisting Random Data for Session Restore

Originally I proposed to add a new persisted member `offline_page_info_` to `SerializedNavigationEntry` and doing the conversion logic in `ContentSerializedNavigationBuilder`.

In the code review, sky@ proposed a more flexible way to embed the extended data for the restore, which will avoid letting the session core know about offline page persisting logic.

1. Add the following field to `SerializedNavigationEntry`:

```
            std::map<std::string, std::string> extended_info_map_;
```

   Note that extended_info_map_ is not synced at this point since the only usage, offline info, does not need to be synced.

2. Add the following virtual methods to `SerializedNavigationDriver` interface class:

```
// Registers the handler that could be used to read and write the extended
// info stored in NavigationEntry.
virtual void RegisterExtendedInfoHandler(
    const std::string& key,
    std::unique_ptr<ExtendedInfoHandler> handler) = 0;

typedef std::map<std::string, std::unique_ptr<ExtendedInfoHandler>>
    ExtendedInfoHandlerMap;

// Returns all the registered handlers to deal with the extended info.
virtual const ExtendedInfoHandlerMap& GetAllExtendedInfoHandlers() const = 0;
```

Where, `ExtendedInfoHandler` interface is defined as:

```
class ExtendedInfoHandler {
 public:
  virtual ~ExtendedInfoHandler() {}

  virtual std::string GetExtraInfo(
      const content::NavigationEntry& entry) const = 0;
  virtual void RestoreExtraInfo(
      const std::string& extra_info, content::NavigationEntry* entry) = 0;
};
```

3. `ContentSerializedNavigationBuilder` will enumerate all the registered handlers in order to get or set the extended data.

Implementing OfflinePageInfoHandler

In order to persisting offline page related info, we need to implement `ExtendedInfoHandler` interface as `OfflinePageInfoHandler`.

```
class OfflinePageInfoHandler : public ExtendedInfoHandler {
 public:
  // Creates and registers a single instance.
  static void Register();

  OfflinePageInfoHandler();
  virtual ~OfflinePageInfoHandler() override;

  // ExtendedInfoHandler:
  virtual std::string GetExtraInfo(
      const content::NavigationEntry& entry) const override;
  virtual void RestoreExtraInfo(
      const std::string& extra_info, content::NavigationEntry* entry) override;
};
```

Getting/Setting Offline Header out of/in Navigation Entry

`OfflinePageInfoHandler::GetExtraInfo` needs to extract the offline header from the extra request headers stored in a navigation entry and returns it as an extra info.

`OfflinePageInfoHandler::SetExtraInfo` needs to pass the offline header constructed from the extra info as part of the extra request headers to a navigation entry.

Unfortunately the extra headers string is exposed in `NavigationEntryImpl`, but not `NavigationEntry`. We can try to do it via extra data mechanism (SetExtraData/GetExtraData) that is supported by `NavigationEntry`. But maintaining additional info in other field make the whole logic very complicated and we should try to avoiding doing this.

So we propose adding the following method to `NavigationEntry` interface.

```
// Returns the extra headers (separated by \n) to send during the request.
virtual std::string GetExtraHeaders() const = 0;

// Adds more extra headers (separated by \n) to send during the request.
virtual void AddExtraHeaders(const std::string& extra_headers) = 0;
```