Implementation of Preloading Pipeline

This document is public

Author: kenoss@
Contributors: kouhei@, lingqi@, nhiroki@, taiyo@, <yourname>@,
Status: Draft
Created: May 8, 2024
Last Updated: May 21, 2024

TODO

- Revisit development plan

Scope of this document

This document discusses a design of Automatic fallback from prerender to prefetch, adding DUI prefetch, and unification of `PreloadingDecider` and `PrerendererImpl`.

We emphasize prefetch and prerender, but preconnect is in scope, too.

Prototype: [cl]
Diagrams: Original slide

Motivation

- Prefetch/Prerender/DiskCache Designs
- Concept of Preloading Pipeline
- Unified Prefetch Cache: 1-pager
- Automatic fallback from prerender to prefetch

Currently, navigational preloads are triggered separately except DSE prefetch/prerender. Historically, execution stacks of prefetch and prerender were developed in parallel and we couldn't coordinate them well. Unified prefetch cache developed necessary parts in the prefetch execution stack, so that we can now discuss the remaining coordination parts.

With better coordination of prefetch/prerender, we can consider the following applications:

- Prerender that can be downgradable to prefetch Automatic fallback from prerender to prefetch

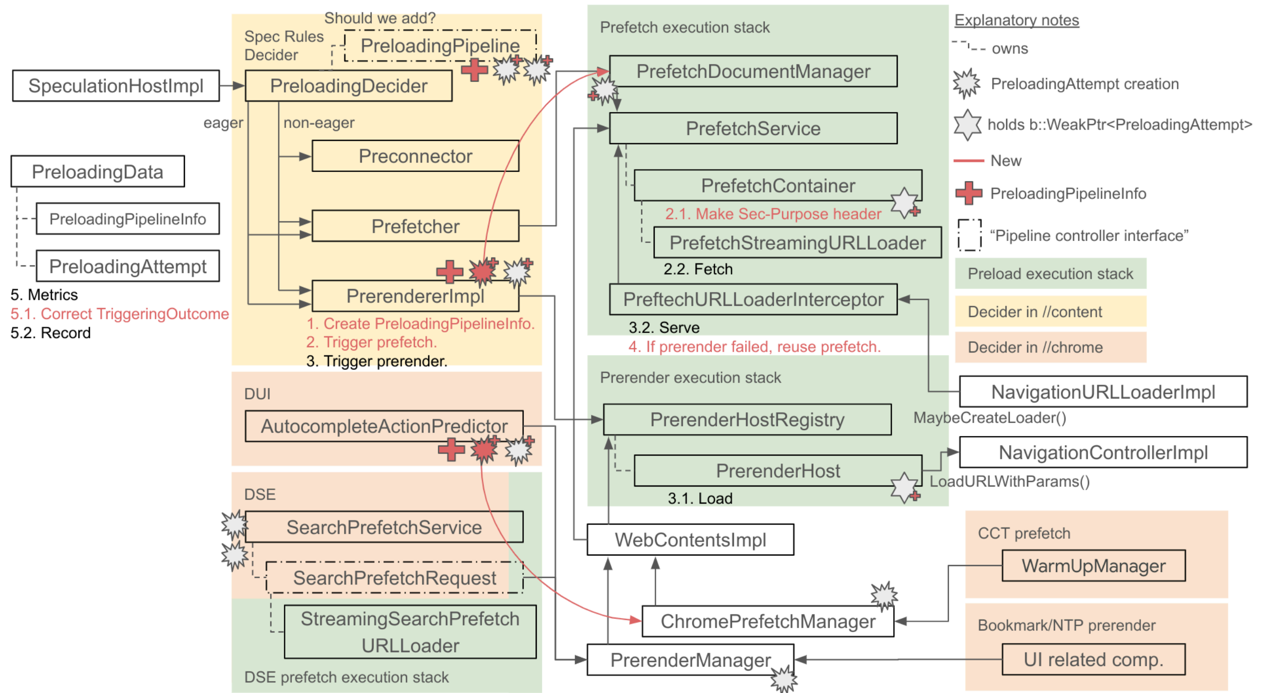
    ○ If Chrome triggers a prerender but the page doesn't support it, Chrome needs to discard the first fetch result and fetch almost the same one again. We can prevent the behavior by triggering prefetch first.
- Starting weaker preloads and upgrading it as confidence increased (everywhere!) Concept of Preloading Pipeline
    ○ DSE prefetch execution stack is dedicated to DSE. We want to use canonical execution stacks for "pipeline" everywhere, e.g. non-eager Spec Rules and DUI.

Important note: Notice that the former is a special case of the latter. We can implement the former by passing full confidence to the "pipeline" in the first place. (See [below].) So, we generalize the issue and discuss the latter.

With these proposal and applications, we expect the following outcomes:

- LCP improvement by increasing prefetch due to downgrading prerender.
- Reducing duplicate fetch due to prerender failure. Web developers can use prerender with fewer server resources.
- LCP improvement by adding DUI prefetch.
- (Possibly) LCP improvement by adding more fine-grained control on triggering non-eager Spec Rules.

Rolls and layering



Before we discuss the preloading pipeline, we sort out rolls that should be necessary for discussing it and correspondence to the classes of current implementations.

Input (not standard wording)

E.g. Speculation Rules and speculation candidates, user input on Omnibox for DUI/DSE, mousehover/mousedown on contents for Document Rules, mousehover/mousedown on bookmark bar/NTP.

Decider (not standard wording)

Decides to trigger preloads from input.

- `Prefetcher` and `PrerendererImpl` trigger preloads for eager Speculation Rules unconditionally.
- `PreloadingDecider` triggers preloads when confidence increases.

Triggering passes to execution stack `PreloadingAttempt`, DevTools observer, `PreloadingPredictor`, and additional data depending on preloading type.

`PreloadingAttempt`

`PreloadingAttempt` is 1. a common part of requests of preloads, 2. an interface to tell and store the status of preload (mainly for metrics).

As 1:

- Decider to execution part [PrerendererImpl -> PrerenderHostRegistry]
- //chrome (decider) to //content (execution stack) via `WebContents` [DUI prerender]

Note that the non common part is e.g. `PrerenderAttributes` [cs].

Speculation candidate reason (not standard wording)

Where speculation candidate came from.

This is a way of using `PreloadingPredictor` with variable name `creating_predictor` [cs]. E.g. `SpeculationRulesFromIsolatedWorld` distinguishes whether a Speculation Rules is injected by extensions or not.

Enacted reason (not standard wording)

Enacted reason is a reason why a decider triggered a preloading attempt.

This is a way of using `PreloadingPredictor` with variable name `enacting_predictor` [cs].

DevTools observer

(Preloading) DevTools observer is a set of callbacks to listen to updates of `PreloadingAttempt`.

There are currently `SpeculativeHostDevToolsObserver` (interface), `Prefetcherer`, `PrerenderDevToolsAttempt`.

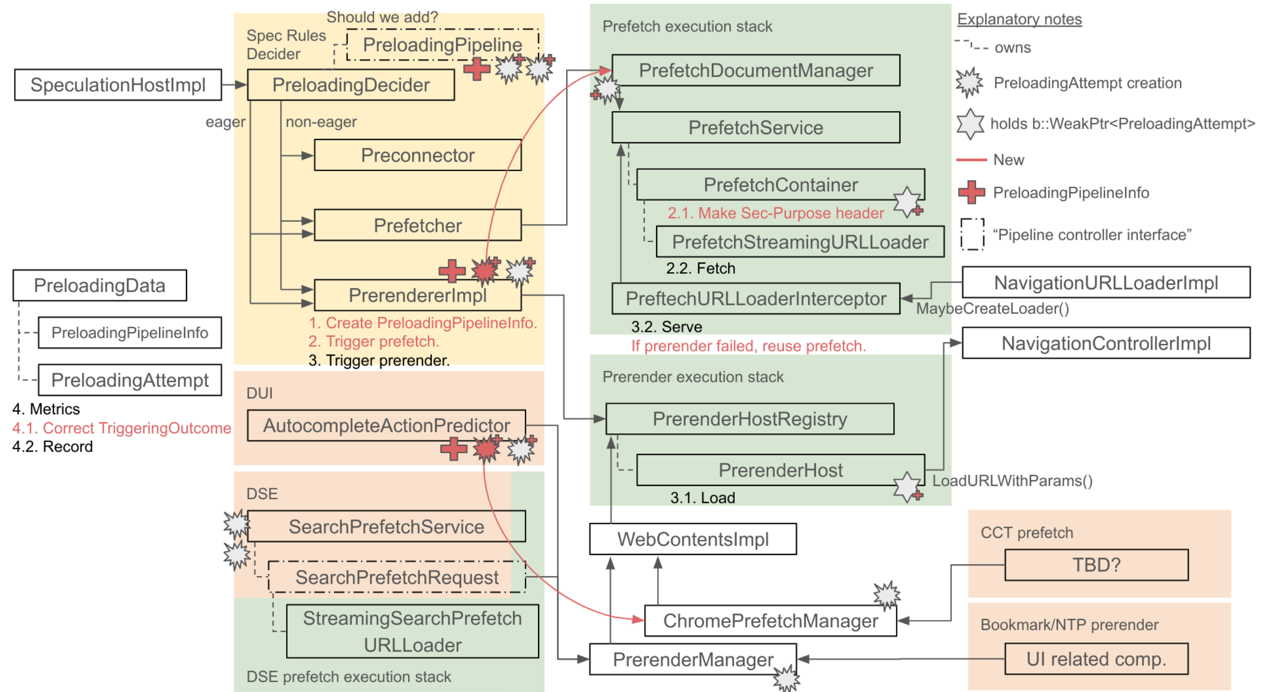
Execution stack of preload (not standard word)

Execution stack of preload run and manage preload and intercept navigation to serve preloads.

- Prefetch: `PrefetchDocumentManager` + `PrefetchService` + `PrefetchContainer` + etc.
- Prerender: `PrerenderHostRegistry` + `PrerenderHost` + etc.
- DSE prefetch: (`SearchPrefetchService` + `SearchPrefetchRequest` +) `SearchPrefetchURLLoader` + `SearchPrefetchURLLoaderInterceptor` + etc. for prefetch + execution stack of prerender.

Note that DSE preloads are the predecessors of preloading pipeline (prefetch -> prerender) and have a slightly different structure from others (Spec Rules prefetch/preneder, DUI prerender, etc.) and structure that will be proposed in this document. See Shareable Search Prefetch Responses for more details.

Design



Summary

In this section, we propose an implementation of a preloading pipeline. Then, we'll describe the rationale and details in the following sections.

We use principles:

- Small changes as much as possible
- Mimic DSE prefetch (`SearchPrefetchService` etc.) as much as possible
- Use non-DSE execution stack
- Correct discordance of layering (E.g. DevTools observer, `PreloadingAttempt` creation timing)

Summary of changes:

- We trigger prefetch/prerender with different `PreloadingAttempt` for each, which share a new class `PreloadingPipelineInfo`. See [below].
- We use `PrefetchDocumentManager` and `PrerenderHostRegistry/PrerenderManager` to execute and manage each preloads associated with `PreloadingAttempt`.
- In metrics recording of `PreloadingAttempt`, we record all preloading attempts (with existing sampling logic). If Chrome tried to upgrade prefetch to prerender and it

succeeded, the preloading attempts of prefetch has triggering outcome `kTriggeredButUpgradedToPrerender`.
- We emit a new CDP event `Preload.preloadingPipelineInfoUpdated` when `PreloadingPipelineInfo` updated and frontend groups the preloads in the pipeline.
- We use `SpeculationHostDevToolsObserver` to track updates of `PreloadingAttempts` (prefetch/prerender) and `PreloadingPipelineInfo`.

PreloadingPipelineInfo

We introduce a lightweight structure `PreloadingPipelineInfo` to bundle `PreloadingAttempt`s.

```
C/C++
// In prototype CL, PreloadingBundle
class PreloadingPipelineInfo {
public:
  // For DevTools
  virtual base::UnguessableToken GetId() const = 0;
  // For Sec-Purpose header
  virtual PreloadingType GetPlannedMaxPreloadingType() const = 0;
  // For DevTools and metrics
  virtual const std::vector<PreloadingAttempt*>& GetPreloadingAttempts() const
= 0;
};
```

We create `PreloadingPipelineInfo` via `PreloadingData` and associate it to `PreloadingAttempt`.

We need such data associated with `PreloadingAttempt` because

- Decider knows how a pipeline should be run. Decider registers this info.
- Execution stacks and `PreloadingData` need to know to control behaviors, e.g. `Sec-Purpose` header, metrics, DevTools.

Alternative: Add `base::WeakPtr<PreloadingPipelineDelegate>` to `PreloadingAttempt`, or pass something like this to the execution stack separately. We think they are almost equivalent (See also [below].) and use the simplest one, `PreloadingPipelineInfo`.

Stronger preloading merely uses outcomes of weaker preloadings

(Thanks to the unified cache effort) Note that just starting weaker preloading ahead of stronger ones works and uses the result of weaker one (e.g. prefetch ahead of prerender), except `Sec-Purpose` header, concurrent preloading limits, metrics, DevTools, etc. For remaining parts, we only need information about the preloading pipeline.

```cpp
// PrerendererImpl::MaybePrerender(), create of pipeline and trigger
prefetch/prerender.

auto* prefetch_document_manager = ...;
std::unique_ptr<PrerenderDevToolsAttempt> devtools_observer = ...; // Create.

// Control additional logics in execution stack etc. by existence of
`PreloadingPipelineInfo`.
PreloadingPipelineInfo* pipeline_info;
if (base::FeatureList::IsEnabled(...)) {
  pipeline_info = preloading_data->AddPreloadingPipelineInfo(
      /* max_planned_preloading_type*/ PreloadingType::kPrerender);
}

auto* prefetch_document_manager = ...;
// Internally calls PreloadingData::AddPreloadingAttempt() with pipeline_info.
prefetch_document_manager->PrefetchAheadOfPrerender(
    pipeline_ info, candidate.Clone(), enacting_predictor, devtools_observer);

// Associates PreloadingAttempt with PreloadingPipelineInfo.
PreloadingAttempt* attempt = preloading_data->AddPreloadingAttempt(
                pipeline_info, // nullptr if attempt is not in pipeline.
                creating_predictor, enacting_predictor,
                PreloadingType::kPrerender, std::move(same_url_matcher),
                web_contents->GetPrimaryMainFrame()->GetPageUkmSourceId()),
                // Emits `Preload.preloadingPipelineInfoUpdated`.
                devtools_observer);
return registry_->CreateAndStartHost(attributes, preloading_attempt);
```

Feature flag and the existence of `PreloadingPipelineInfo`

We control additional logics that we are proposing in this document by the existence of `PreloadingPipelineInfo` for `PreloadingAttempt`.

Effect of `kPrefetchReusable`

- (S1) of [doc] "Prerendering fails before sending a request (e.g., memory limit check)" doesn't require `kPrefetchReusable` as it loads one time.
- (S3) of the doc "Prerendering fails after receiving a response (e.g., disallowed API usage)" requires it as it loads two times.

Rest of the document assumes `kPrefetchReusable`.

Using dedicated `PreloadingAttempt` for each prefetch and prerender

- `SearchPrefetchService` already provides upgrading prefetch to prerender, which uses `PreloadingAttempt` for each and upgrades prefetch to prerender when the loader is about consumed [cs].
- `PreloadingAttempt` has `PreloadingType` [cs]. It's simpler to represent prefetch/prerender and execute each.

Note also that `enacting_predictor` can differ between prefetch and prerender that is triggered by `PreloadingDecider`. (It depends on settings of confidence. Note that all predictors in `PreloadingDecider` always start prerender with current settings.)

`Sec-Purpose` header

We should send a prefetch request with `Sec-Purpose: prefetch;prerender` if the response can be used for prerender. We determine the appropriate header by `PreloadingPipelineInfo::GetPlannedMaxPreloadingType()`. We use `kPrefetch/kPrerender` for Spec Rules as designated and `kPrerender` for DUI etc.

```
C/C++
// PrefetchDocumentManager, set Sec-Purpose header.

char* sec_purpose;
PreloadingPipelineInfo* pipeline_info = attempt_ ?
attempt_->GetPreloadingPipelineInfo() : nullptr;
if (pipeline_info && pipeline_info->GetMaxPlannedPreloadingType() ==
PreloadingType::kPrerender) {
  if (IsProxyRequiredForURL(url) {
    sec_purpose = "prefetch;prerender;anonymous-client-ip";
  } else {
    sec_purpose = "prefetch;prerender";
  }
} else {
```

```
    if (IsProxyRequiredForURL(url) {
      sec_purpose = "prefetch;anonymous-client-ip";
    } else {
      sec_purpose = "prefetch";
    }
  }
  request->headers.SetHeader("Sec-Purpose", sec_purpose);
```

Metrics of `PreloadingAttempt`

- Existing behavior of `PreloandingDecider`: Tries stronger one first: prerender -> prefetch -> preconnect. So, this doesn't need coordination. If creating predictor and enacting predictor differ, metrics are recorded for each.
- Existing behavior of `SearchPrefetchService`: Prefetch/prerender are recorded independently. If prerender started, it is recorded with triggering outcome `kTriggeredButUpgradedToPrerender` [cs].

We have several options. We use A:

- Option A: We just follow the behavior of DSE prefetch/prerender.
  - Implementation: Send upgrade notice from `PrefetchStreamingUrlLoader` to `PrefetchService`.
- Option B: Record metrics by seeing the end state as below.

Triggering outcome of prefetch	Triggering outcome of prerender	Metrics for prefetch	Metrics for prerender
kSuccess	Rejected as not eligible	kSuccess	N/A
kSuccess	kFailure	kSuccess	kFailure
kSuccess	kSuccess	kTriggeredButUpgradedToPrerender	kSuccess

  - Implementation: `PreloadingDataImpl::RecordMetricsForPreloadingAttempts` and `PreloadingAttemptImpl::RecordPreloadingAttemptMetrics` are called when primary page is changed and WebContents is destroyed. At this timing, look at the associated `PreloadingPipelineInfo` and mark prefetch

as `kTriggeredButUpgradedToPrerender` if proceeding prerender succeeded.

```
C/C++
// Metrics, option B.

void PreloadingDataImpl::RecordMetricsForPreloadingAttempts(
    ukm::SourceId navigated_page_source_id) {
  // Correct triggering outcome if needed.
  for (auto& pipeline_info : preloading_pipeline_infos_) {
    bool has_success_prerender = false;
    for (auto* attempt_to_be_casted : pipeline_info.GetPreloadingAttempts) {
      auto* attempt =
static_cast<PreloadingAttemptImpl*>(attempt_to_be_casted);
      if (attempt->GetPreloadingType() == PreloadingType::kPrerender &&
attempt->triggering_outcome() === kSuccess) {
        has_success_prerender = true;
        break;
      }
    }
    if (!has_success_prerender) {
      continue;
    }

    for (auto* attempt : pipeline_info.GetPreloadingAttempts()) {
      if (attempt->GetPreloadingType == PreloadingType::kPrefetch) {
        attempt->SetTriggeringOutcome(kTriggeredButUpgradedToPrerender);
      }
    }
  }

  ...
}
```

DevTools

Currently, DevTools shows prefetch and prerender per Speculation Rules separately. With the pipeline concept, it's nice to show pipelines per Speculation Rules by default and leave an option to users to show associated preloads. I.e.

- If a pipeline contains a preload, show the preload on the Speculation grid.
- If a pipeline contains multiple preloads, show the successful strongest one on the Speculation grid and "Expand" button. If "Expand" is clicked, it shows all preloads associated with the pipeline.

For this, we emit a new CDP event `Preload.preloadingPipelineInfoUpdated` when `PreloadingPipelineInfo` updated.

We'll discuss the details of design with another document.

Limits of number of preloads

Past discussions:

- Preloading limits and triggers
  - Preloading Limit Changes
  - [Public] Better Prerender Limit and Scheduler

There are some limits on the number of preloading.

- Represents resource limit
  - Limit in `PrefetchDocumentManager` [cs] and `PrerenderHostRegistry` [cs]. Limits essentially all prefetch and prerender. (Exception is DSE prefetch managed in `SearchPrefetchService`.)
- More aggressive limit on trigger side
  - E.g. DUI prerender [cs] can run just one prerender (per WebContents).

Preloading pipeline should be counted as the following:

- `PreloadingAttempt` for prefetch and prerender should be counted separately in `PrefetchDocumentManager` and `PrerenderHostRegistry` as it consumes memory for prefetch and prerender.
- We don't have a special policy for limits on the trigger side. We still use just one preloading pipeline (can contain one prefetch and one prerender) for each browser-triggered preload.

Creation timing of `PreloadingAttempt`

We can create a `PreloadingAttempt` by `PreloadingData::AddPreloadingAttempt()` [cs].

Classification of creation timings (See also the above diagram):

- Decider
  - Spec Rules preconnect/prerender (`AnchorElementPreloader`, `Prerenderer`)
  - DUI prerender (`AutocompleteActionPredictor`)

    - DSE prefetch/prerender (`SearchPrefetchService`)
- //chrome-side manager
    - CCT prefetch (`ChromePrefetchManager` [cs])
    - Bookmark/NTP prerender (`PrerenderManager`)
- Prefetch execution stack
    - Spec Rules prefetch (`PrefetchDocumentManager`)
- Exceptions
    - NoStatePrefetch (`NoStatePrefetchManager` [cs])
    - `PrerenderHostRegistry::BackNavigationLikely()` [cs]

This is troublesome because

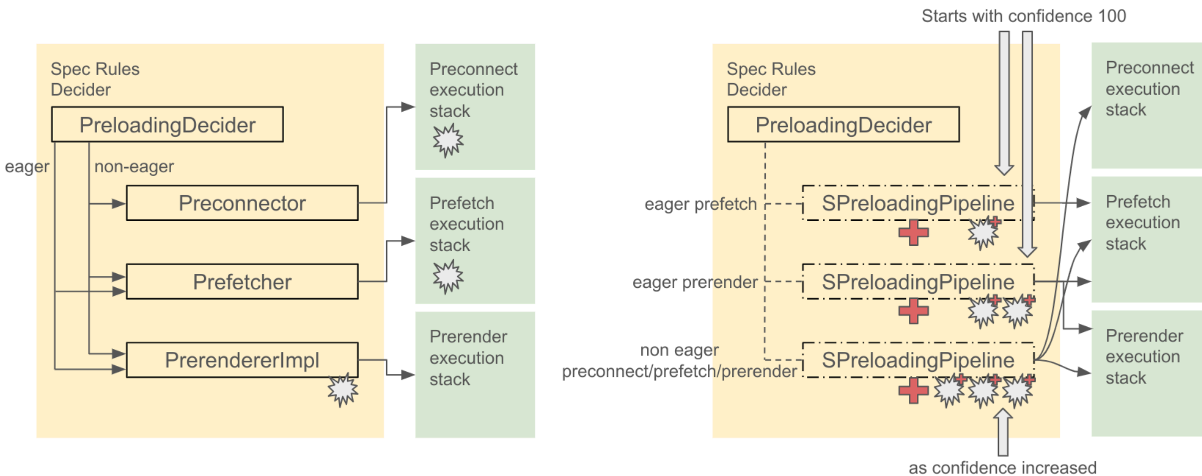
- Not coherent. It's nice to have a concrete layering policy.
- From the pipeline viewpoint, it's easier that `SpeculationPreloadingPipeline` or something like that can control creation of `PreloadingPipelineInfo`, `PreloadingAttempt` and calling `PreloadingData::AddPreloadingPrediction()`. (See `SpeculationPreloadingPipeline::MaybeStartPreloads()` [below].)

Note that there are both cases that deciders in //chrome does/doesn't want to create it. DSE does because it needs fine-grained control. Bookmark doesn't because it doesn't need and want to keep the interface concise.

So, we propose that

- Regard `PreloadingAttempt` as a common part of request of preload from triggering entity to execution stack.
- Layering policy:
    - Create `PreloadingAttempt` in deciders if possible.
    - In //chrome, deciders can delegate managing it to //chrome-side managers `ChromePrefetchManager`/`PrerenderManager`.
    - Exceptions are allowed.
- Task: Move the creation in `PrefetchDocumentManager` to `Prefetcher`/`PrerendererImpl` or `PreloadingDecider`. (It is likely a necessary step to add DUI prefetch. See the above diagram.)

(optional) Controlling interface `SpeculationPreloadingPipeline`



One of the aspects of `SearchPrefetchRequest` [cs] is pipeline.

```
C/C++
class SearchPrefetchRequest {
 public:
  SearchPrefetchRequest(const GURL& canonical_search_url,
                        const GURL& prefetch_url,
                        bool navigation_prefetch,
                        content::PreloadingAttempt*
prefetch_preloading_attempt,
                        base::OnceCallback<void(bool)> report_error_callback);
  ~SearchPrefetchRequest();

  bool StartPrefetchRequest(Profile* profile);
  void CancelPrefetch();
  void MaybeStartPrerenderSearchResult(PrerenderManager& prerender_manager,
                                       const GURL& prerender_url,
                                       content::PreloadingAttempt& attempt);

  ...
};
```

We might mimic this aspect and have yet another class `SpeculationPreloadingPipeline`.

C/C++
```
class SpeculationPreloadingPipeline {
public:
  SpeculationPreloadingPipeline(
    /*might need*/SpeculationCandidatePtr& candidate,
    PreloadingConfidence prefetch_threshold,
    PreloadingConfidence prerender_threshold);

  // Trigger preconnect/prefetch/prerender.
  MaybeStartPreloads(const PreloadingPredictor enacting_predictor,
                     PreloadingConfidence confidence);
};
```

C/C++
```
// preloading_decider.h

class PreloadingDecider {
  ...
private:
  base::flat_map<GURL, std::unique_ptr<PreloadingPipeline> pipelines_;
};
```

C/C++
```
// preloading_decider.cc

void PreloadingDecider::MaybeEnactCandidate(
    const GURL& url,
    const PreloadingPredictor& enacting_predictor,
    PreloadingConfidence confidence,
    bool fallback_to_preconnect) {
  if (!pipelines_.find(url)) {
    pipelines_.insert(
      url,
      std::make_unique<PreloadingPipeline>(PREFETCH_THRESHOLD,
                                           PRERENDER_THRESHOLD));
  }

  auto* pipeline = pipelines_.find(url);
  CHECK(pipeline);

  pipeline->MaybeStartPreloads(candidate);
```

```
}
```

```
C/C++
// speculation_preloading_pipeline.cc

void PreloadingPipeline::MaybeStartPreloads(
  const PreloadingPredictor enacting_predictor,
  PreloadingConfidence confidence) {
  if (last_confidence_ < confidence) {
      AddPreloadingPrediction(url_, enacting_predictor, confidence);
  }
  last_confidence_ = confidence;

  // Not C++, dummy code.
  actions = actions_[last_triggered_index_ + 1:];
  for (auto* action : actions) {
    if (confidence < action->confidence_threshold) {
      break;
    }

    last_triggered_index_ += 1;
    // Trigger preconnect/prefetch/prerender.
    action.Run(pipeline_info_, enacting_predictor, /*etc*/...);
  }
}
```

Also, we might generalize it to `PreloadingPipeline`, which is not specialized to Spec Rules and can be used in other deciders.

Observations:

- For downgradable prerender (to prefetch), `SpeculationPreloadingPipeline` is not necessary.
- Downgradable prerender can be implemented with `SpeculationPreloadingPipeline` with `prefetch_threshold == prerender_threshold`. But note that `PrerendererImpl::MaybePrerender()` is also called from `PreloadingDecider`. We can't simply use it in `PrerenderImpl`, but we need to unify `PreloadingDecider` and `PrerenderImpl`.

- Current Speculation Rules preloads `PreloadingDecider` and `PrerendererImpl` slightly differ from `SpeculationPreloadingPipeline`, but it would be meaningful to refactor and unify them into `PrerenderImpl`.
  - Main difference: Currently, `PreloadingDecider` tries to start prerender first, and then prefetch if prerender is not started [cs]. We should change the order the same as downgradable prerender.
  - `Prefetcher` (and `Preconnector`/`AnchorElementPreloader`? Need more investigation.) are very light weight and we can remove it once we have a unified DevTools observer.
- Current DUI prerender is slightly different from `SpeculationPreloadingPipeline` [cs], which determines a recommended action first and then executes prerender in a different path. It would not be a high priority as it manages just one prerender (and prefetch in the future).

Possible implementation options:

- A. Only introduce `PreloadingPipelineInfo`.
  - Note again that this works for //chrome side deciders, e.g. DUI.
- B. Introduce both `PreloadingPipelineInfo` and `SpeculationPreloadingPipeline` at the same time. Unify `PreloadingDecider` and other preloaders.
  - In the short term, we only use the new `PreloadingDecider` for eager preloads, by replacing [cs]. Then, we replace all.
- A => B. Introduce only `PreloadingPipelineInfo` and have an experiment. Then, refactor.

We (kenoss@) prefer B or A => B.

Note that //chrome side and general `PreloadingPipeline` are out of scope in these options. We revisit it if needed.

Discordance and refactoring

Can we refactor and make `PreloadingAttempt` own `std::unique_ptr<SpeculativeHostDevToolsObserverImpl>`?

It's convenient to make `PreloadingAttempt` abstract a common part (among preconnect/prefetch/prerender) of Chrome-internal requests from other components to execution stacks of preload. It's also convenient that `PreloadingAttempt` has callbacks to report changes of `PreloadingAttempt` and `PreloadingPipelineInfo`. Currently, we manage them separately [prefetch][prerender].

Prefetch: Currently, PrefetchDocumentManager sets base::WeakPtr<SpeculativeHostDevToolsObserver> to PrefetchContainer [cs], and the pointer is Prefetcher [cs], which has the same lifetime to PreloadingDecider and initiator document [cs].

Prerender: PrerenderHost doesn't outlive the initiator document.

So, we can refactor in this way.

Remaining questions

Lifetime of PreloadingPipelineInfo

Options: PreloadingData has std::unique_ptr<PreloadingPipelineInfo> or share it by scoped_refptr<PreloadingPipelineInfo>.

Lifetime of PreloadingData and PreloadingAttempt is similar to the initiator document.

Prefetch: PrefetchContainer can outlive the initiator document [cs] as prefetch is available until it expires. Should we care about this case?

Prerender: PrerenderHost doesn't outlive the initiator document. Both options work.

Development plan

We propose a development plan as the following.

1. Support downgradable prerender to prefetch for eager Speculation Rules. Platform: all.
    a. Update: PreloadingPipelineInfo, metrics, full DevTools support, removing Prerenderer.
    b. Expected outcome: LCP improvement for prerender failure case.
2. Support low confidence prefetch for DUI.
    a. Update: Minimal changes.
    b. Expected outcome: LCP improvement.
3. Support downgradable prerender to prefetch for non-eager Speculation Rules. (I.e. use the same confidence threshold for prefetch and prerender.) Platform: all.
    a. Update: Unifying PreloadingDecider and PrerendererImpl. Maybe introduce PreloadingPipeline.
    b. Expected outcome: LCP improvement for prerender failure case.
4. (Maybe) Support prefetch for low confidence with ML model.

Trashbox

~~Observations~~

~~Concepts around preloading pipeline~~

- ~~Information of preloading pipeline~~
  - ~~Consists of id, type, associated preloading attempts, max preloading type (See below).~~
  - ~~Must be accessible from `PreloadingAttemptImpl`, e.g. for metrics.~~
  - ~~Maybe it would be nice to have callbacks for completion/cancellation of preloadings.~~
- ~~Controller of preloading pipeline~~
  - ~~Initiate/upgrade/downgrade/cancel control interface for single URL. See Concept of Preloading Pipeline.~~
  - ~~`AutocompleteActionPredictor` [cs] + `PrerenderManager` [cs] is the nearest ones.~~
- ~~Set of preloading pipelines~~
  - ~~We might want to hold multiple pipelines for lower confidence e.g. DUI.~~