Scheduler architecture 2.0

Tracking bug: crbug.com/803853
altimin@, haraken@, yutak@
2018 Jan

platform/scheduler/ has a bunch of incoherent abstraction layers (partly because historically platform/scheduler/ was living in //content/ and moved to Blink). The class hierarchy is really complex and has been slowing down engineering velocity. This document proposes a rearchitecture plan of platform/scheduler/.

Directory structure

Currently platform/scheduler/ has base/, child/, renderer/, util/, utility/ and test/.

- platform/scheduler/base/ should move to //base/.
- The distinction between child/ and renderer/ don't make sense because platform/scheduler/ is now in Blink.
- utility/ does not make sense because the utility thread is not a component of Blink. utility/ should move back to //content/.

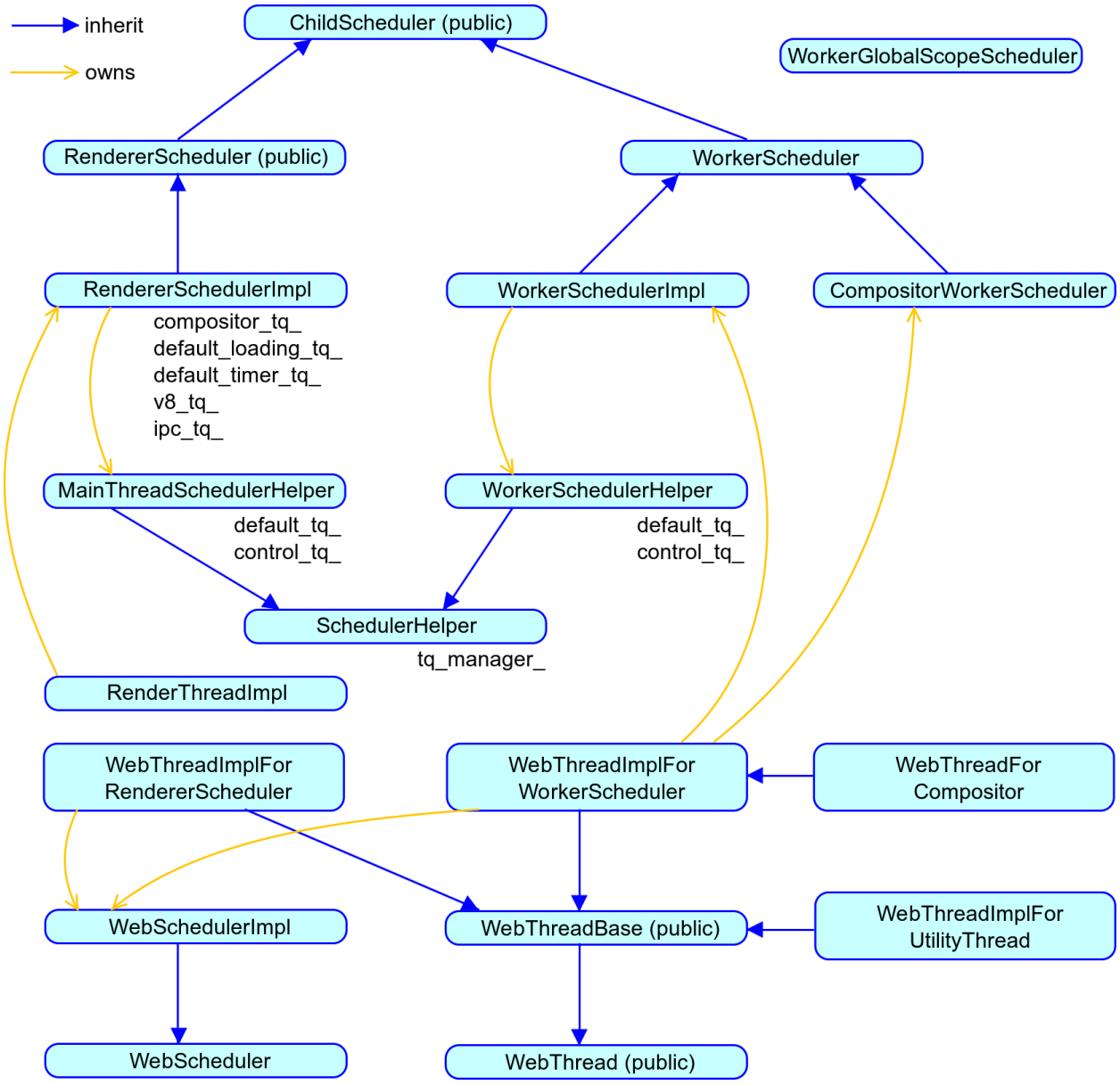
The remaining files should be reorganized with the following directory structure:

- **public/** : Classes that should be exposed to the rest of Blink. public/ should contain only MainThreadScheduler, WorkerThreadScheduler, Timer and ThreadCPUThrottler.
- **main_thread/** : Classes about main thread scheduling.
- **worker/**: Classes about worker scheduling. Note that workers mean non-main threads, not web workers.
- **common/** : Classes shared between the main thread and workers.
- **util/**: Helper utility classes used by scheduler (mostly for tracing and metrics). They should not depend on the rest of the scheduler. (Note from altimin@: loading/ code uses some metric helpers. It may make sense to expose util/ directory to the rest of the blink. ThreadCPUThrottler could go here too. public/util/ is possible too).
- **test/**: Provides mocks and dummy implementations of scheduling primitives for testing. Test-only, can be referenced from outside the scheduler.

Class design

Around ChildScheduler

Today we have the following class relationship:

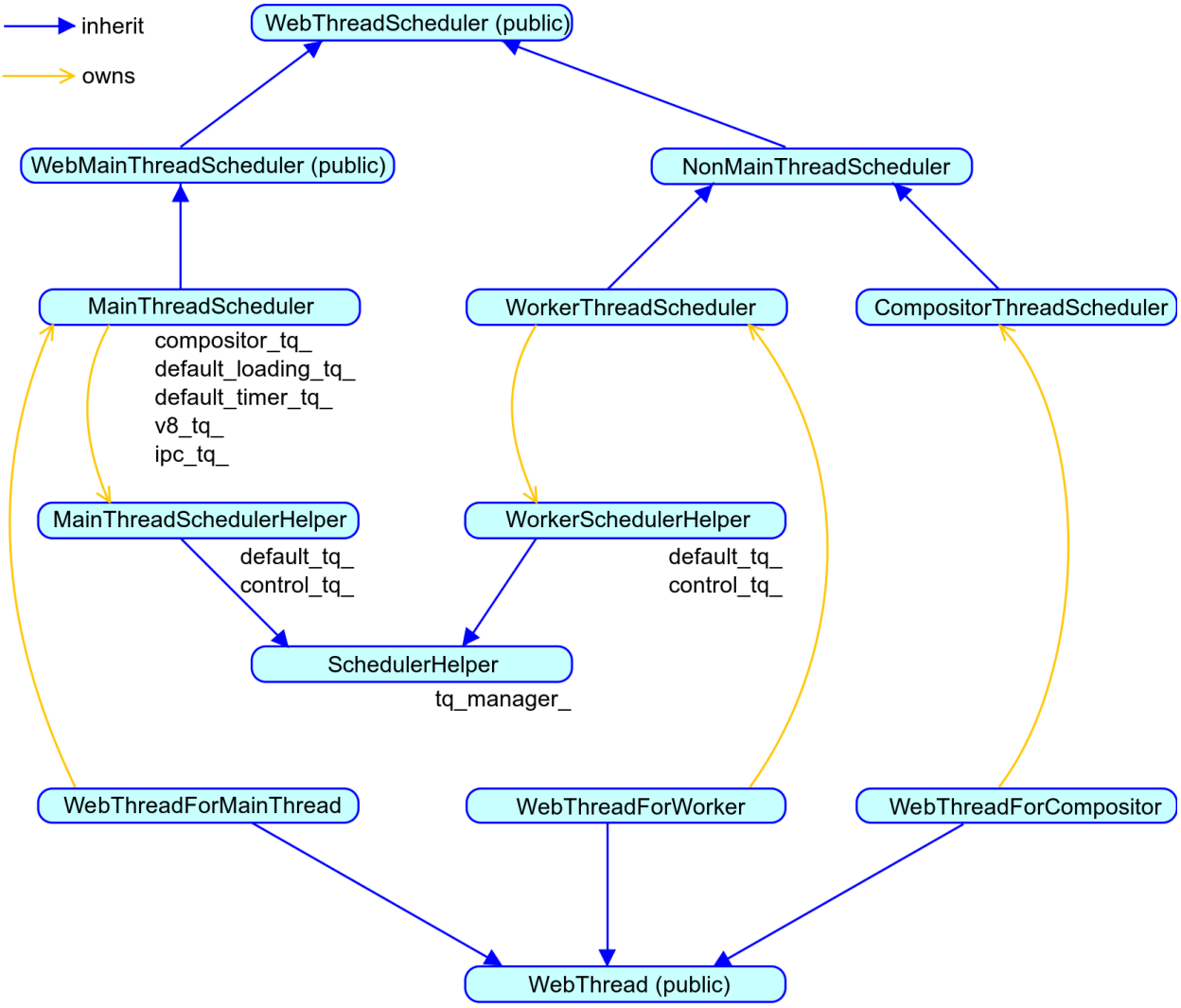


There are a couple of points that may confuse developers:

- Public classes are not prefixed with "Web". Not-public classes are prefixed with "Web".

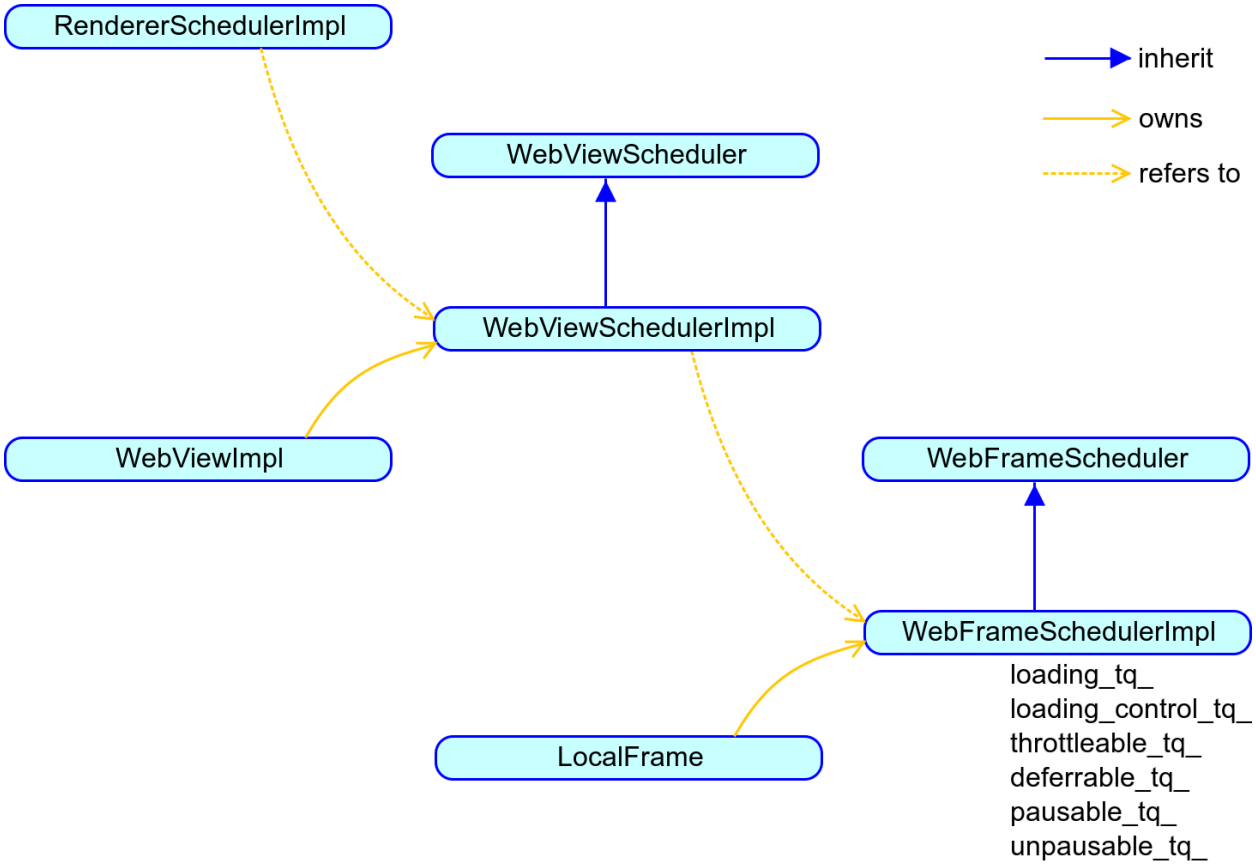
- WebScheduler is used to expose ChildScheduler and RendererScheduler to Blink. WebScheduler should be merged into RendererScheduler and WorkerScheduler.
- WorkerGlobalScopeScheduler is isolated. It should be merged into WorkerScheduler.
- WebThreadForCompositor does not inherit from WebThreadBase. WebThreadForCompositor does not own CompositorWorkerScheduler.
- It's strange that WebThread"Base" inherits from WebThread. WebThreadBase should be removed.
- WebThreadImplForRendererScheduler does not own RendererSchedulerImpl.
- The utility thread is not a component of Blink. WebThreadImplForUtilityThread should move back to //content/.

In summary, I'd propose reorganizing the class relationship as follows:



Around WebViewScheduler

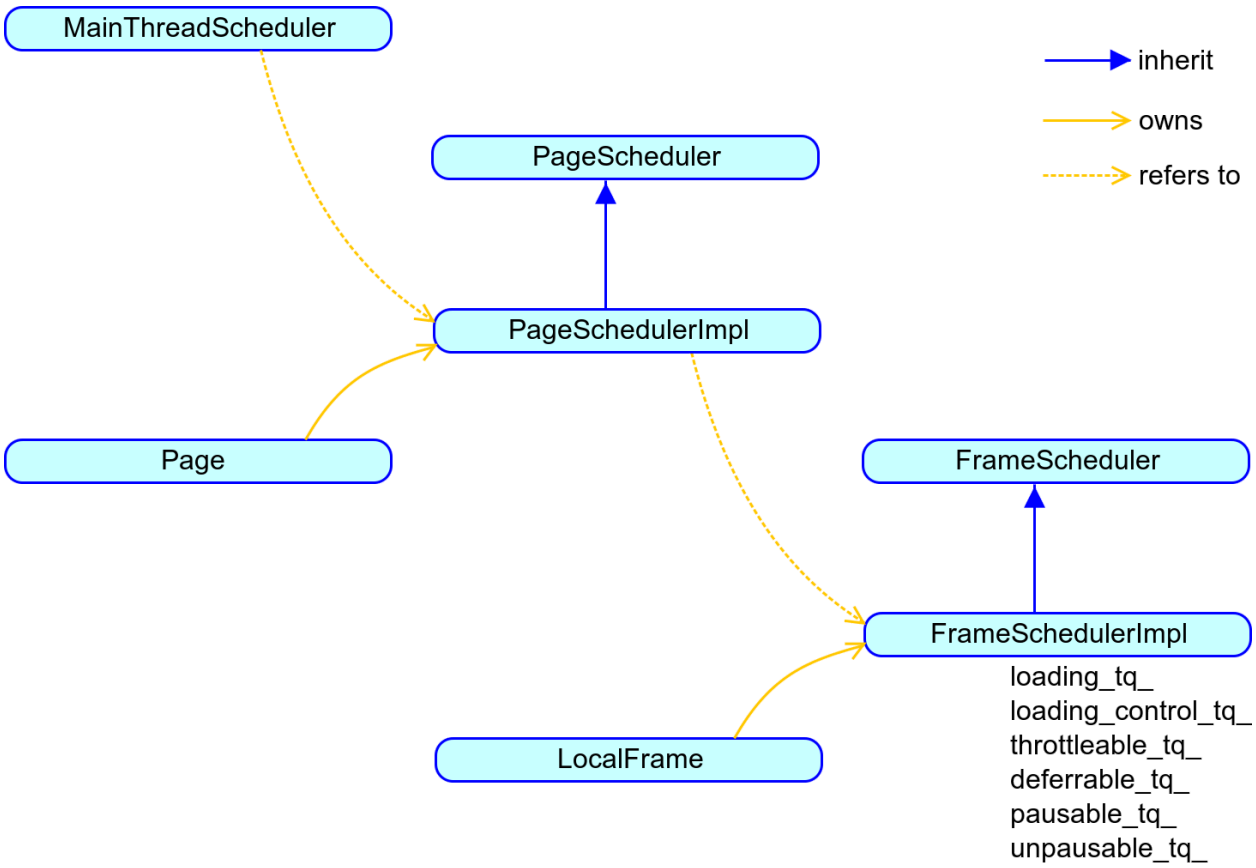
Today we have the following class relationship:



RendererScheduler : WebViewScheduler : WebFrameScheduler = 1 : N : M

RendererScheduler is per main thread. WebViewScheduler is per Page. WebFrameScheduler is per Frame.

I'd propose reorganizing the class relationship as follows:



MainThreadScheduler : PageScheduler : FrameScheduler = 1 : N : M

Exposed APIs

Scheduler will expose two sets of APIs:

- **public/platform/scheduler/** facing content/ (and cc/), referred to as *public API.*
- **platform/scheduler/public/** facing blink, referred to as *Blink API.*

Public API

This API provides interfaces to plumb signals from browser process and compositor to blink and provides main per-thread task runners (e.g. IPC) to be used in content layer.

The classes are:

WebThreadScheduler provides basic support for all threads with blink, namely per-process task runners for the majority of the purposes. A static method is provided to ensure that any thread can easily use the appropriate task runner. This class is renamed from ChildScheduler.

```
class WebThreadScheduler {
  TaskRunner IPCTaskRunner();
  TaskRunner V8TaskRunner();

  static WebThreadScheduler* Current();
};
```

WebMainThreadScheduler provides a way to notify blink scheduler about the signals needed to schedule main thread (e.g. notifications about input events from the compositor) and the renderer process in general (e.g. renderer process visibility state changes). Main-thread specific task runners are available. A thread-safe static method is provided to ensure that code can post main-thread task and use appropriate task runner. This class is renamed from RendererScheduler.

```
class WebMainThreadScheduler : public WebThreadScheduler {
  void OnProcessBackgrounded(bool);
  // Some other signals.

  static WebMainThreadScheduler* GetWebMainThreadScheduler();
};
```

WebCompositorThreadScheduler provides a way to notify blink scheduler about the signals originating from compositor (impl) thread. This class is the compositor-thread specific part of RendererScheduler.

```
class WebCompositorThreadScheduler : public WebThreadScheduler {
  void DidAnimateForInput();
  // Some other signals.

  static WebCompositorThreadScheduler* GetWebCompositorThreadScheduler();
};
```

WebWorkerThreadScheduler will provide way to plumb signals needed for scheduling worker threads. It’s not needed at the moment, but it’s very likely that it will be needed when smarter scheduling for worker threads is implemented.

```
class WebWorkerThreadScheduler : public WebThreadScheduler {
  static WebWorkerThreadScheduler* Current();
};
```

Blink API

This API provides

ThreadScheduler is the counterpart of WebThreadScheduler exposed to Blink. It provides additional blink-specific functionality like idle task support. The existing parts of WebScheduler valid for all threads will be moved here.

```
class ThreadScheduler {
  bool ShouldYieldForHighPriorityWork();

  static ThreadScheduler* Current();
};
```

MainThreadScheduler is the Blink-exposed counterpart of WebMainThreadScheduler. It provides additional Blink-specific functionality, with most important being creation of PageSchedulers and FrameSchedulers. The main-thread specifc parts of WebScheduler will be moved to this class.

```
class MainThreadScheduler : public ThreadScheduler {
  std::unique_ptr<PageScheduler> CreatePageScheduler(...);
  …

  static MainThreadScheduler* GetMainThreadScheduler();
};
```

PageScheduler is the renamed WebViewScheduler. The ownership of PageScheduler should be moved to Page and accessors should be exposed both on WebView and Page classes.

```
class PageScheduler {
  void SetPageVisible(bool);

  std::unique_ptr<FrameScheduler> CreateFrameScheduler();
}
```

FrameScheduler is the renamed WebFrameScheduler. The creation should be simplified to avoid going through ChromeClientImpl, more on that below.

```
class FrameScheduler {
  void SetFrameVisible(bool);
  TaskRunner GetTaskRunner(TaskType);
};
```

CompositorThreadScheduler is the counterpart of WebCompositorThreadScheduler. No methods are exposed to Blink at the moment, but it's added for consistency.

```
class CompositorThreadScheduler {
  static CompositorThreadScheduler* GetCompositorThreadScheduler();
};
```

WorkerThreadScheduler is the merge of WorkerScheduler and WorkerGlobalScopeScheduler. Used to provide scheduling support for web workers.

```
class WorkerThreadScheduler {
  TaskRunner GetTaskRunner(TaskType);
  static WorkerThreadScheduler* Current();
};
```

Implementation

Three directories (platform/scheduler/common, platform/scheduler/main_thread, platform/scheduler/worker) contain the implementations of the interfaces above.

Note that public API and blink API are disjoint, so each implementation class should implement both interfaces. Scheduler-only static accessors are available to be used inside scheduler.

```
class MainThreadSchedulerImpl : public WebMainThreadScheduler, MainThreadScheduler {
  TaskRunner CompositorTaskRunner() override;

  static MainThreadSchedulerImpl* GetMainThreadSchedulerImpl();
}
```

Design rationale and other notes

WebScheduler

Existing WebScheduler class implements both methods common for all threads (i.e. V8TaskRunner) and main-thread-only (i.e. CreateWebViewScheduler). The responsibilities of this class will be split between ThreadScheduler and MainThreadScheduler with a clear understanding which methods belong to which thread.

Getting hold of a scheduler

Getting a scheduler for the right thread is non-trivial now. The most popular (and ~only) option is Platform::Current()->CurrentThread/MainThread()->Scheduler(), which exposes only

WebScheduler without any thread-specific functionality. This results in constructions like “static_cast<scheduler::WebThreadImplForWorkerScheduler&>(thread)”.

Given that the 99% of callers of Platform::Current()->CurrentThread() are interested in getting scheduler, it is proposed to expose it with a static method. This allows to have the same accessors inside the scheduler (exposing *Impl classes). This also allows to expose correct type of the scheduler (main thread, worker or common) without adding unneeded classes like WebMainThread or WebCompositorThread.

Initialisation

There are two different scenarios -- main thread initialisation and child thread initialisation. Main thread initialisation is performed by content::RenderThread to ensure that the scheduler is initialised first. Child thread initialisation is performed by the scheduler as a part of thread startup -- scheduler (still) provides CreateWorkerThread/CreateCompositorThread methods backed by CompositorThreadImpl / WorkerThreadImpl (nee WebThreadImplFor*).

Also it is proposed to replace string name for web thread with an enum, use it as a parameter in Platform::CreateThread and ensure that WorkerThreadScheduler knows its type when it’s created.

PageScheduler & creation of FrameSchedulers

At the present moment FrameScheduler is created via ChromeClientImpl (because LocalFrame constructor doesn’t have access to WebView?). WebViewScheduler should be renamed to PageScheduler and ownership should be transferred to Page class. Given that WebView has a reference to Page, an accessor will be exposed both in Page and WebView.

WebThreadBase

WebThreadBase class should be removed. The methods which are not present in WebThread aren’t really needed:

- PostIdleTask() is used only by a unittest. GetIdleTaskRunner() is used only by PostIdleTask().
- AddTaskObserver/RemoveTaskObserver/AddTimeTaskObserver/RemoveTaskTimeObserver methods should be deprecated, but for now they could be moved to ThreadScheduler directly and used in the following fashion: ThreadScheduler::Current()->AddTaskObserver
- GetSingleThreadTaskRunner is needed only for TLS initialisation.
- WebThreadBase::Init and TLS initialisation should be moved into Create*Thread methods provided by the sched uler (instead of being implemented by the platform)