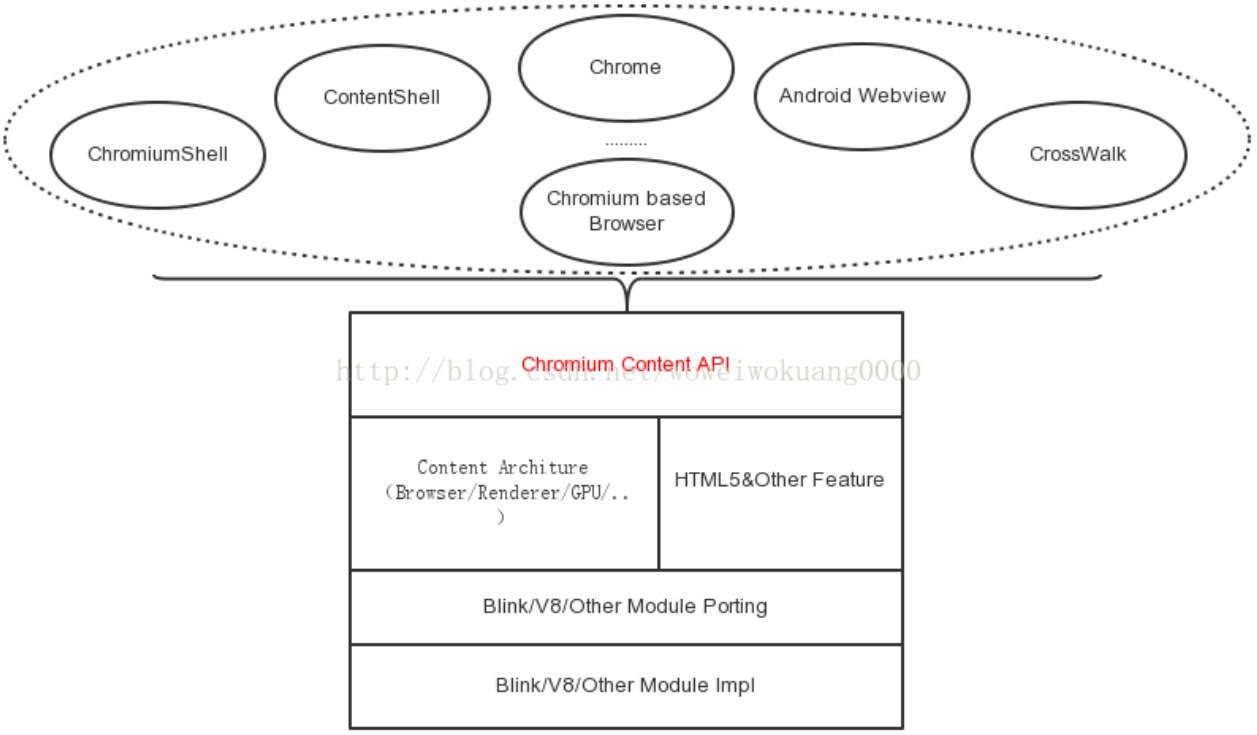
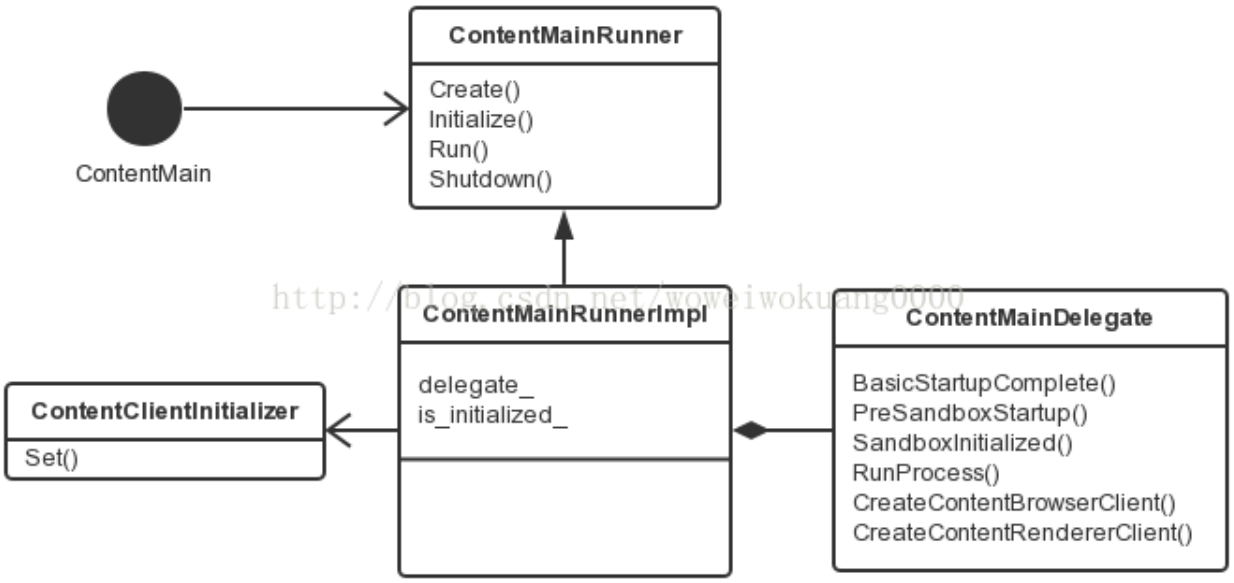
Chromium Content模块初始化 - woweiwokuang0000的专栏 - 博客频道

简单的说，Chromium Content模块就是Chromium内核核心功能实现，基于这些实现有提供了公开和稳定的接口，即ContentAPI；其他浏览器或类似功能开发者可以基于这些API进行二次开发，而不需要关心这个内核的底层实现细节，并且同时可以在其产品中展现Chromium内核的很多特性，如多进程架构、HTML5支持、硬件加速渲染甚至是沙盒机制等：



这里以content模块的初始化入口即ContentMain进行简单的总体分析：

首先是各个进程公共部分ContentMain：



实际上ContentMain方法（此处位非Android：content/app/content_main.cc, Android类似：content/app/android/content_main.cc）比较简单，仅仅数行代码：

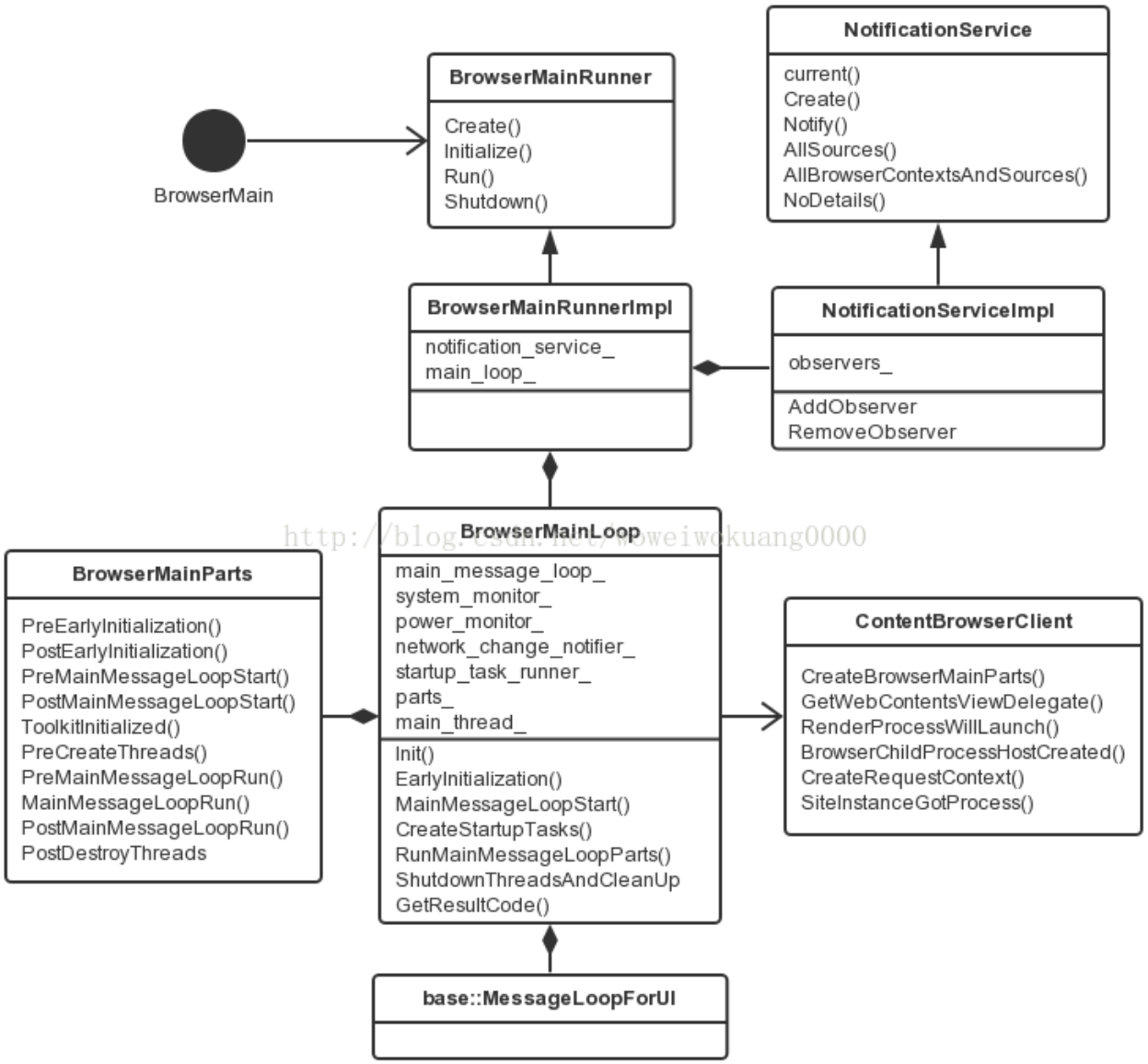
```
1. int ContentMain(const ContentMainParams& params) {
2.   scoped_ptr<ContentMainRunner> main_runner(ContentMainRunner::Create());
3.   int exit_code = main_runner->Initialize(params);
4.   if (exit_code >= 0)
5.     return exit_code;
6.   exit_code = main_runner->Run();
7.   main_runner->Shutdown();
8.   return exit_code;
9. }
```

其中参数中包含了一个重要的对象指针，即ContentMainDelegate子对象，这个对象根据平台或运行环境需要自行实现，如Crosswalk中为xwalk::XWalkMainDelegate delegate，而ContentShell中为content::ShellMainDelegate delegate等，其中实现了几个重要的方法，在不同进程初始化时需要进行回调完成相应的操作；ContentMainRunnerImp::Initialize方法中主要完成进程相关的操作，如调用ContentClientInitializer::Set（）方法根据进程类型(Browser进程类型为空)创建个子的ContentClient对象；ContentMainRunnerImplement::Run方法则是调用RunNamedProcessTypeMain（）方法根据进程类型创建主线程：

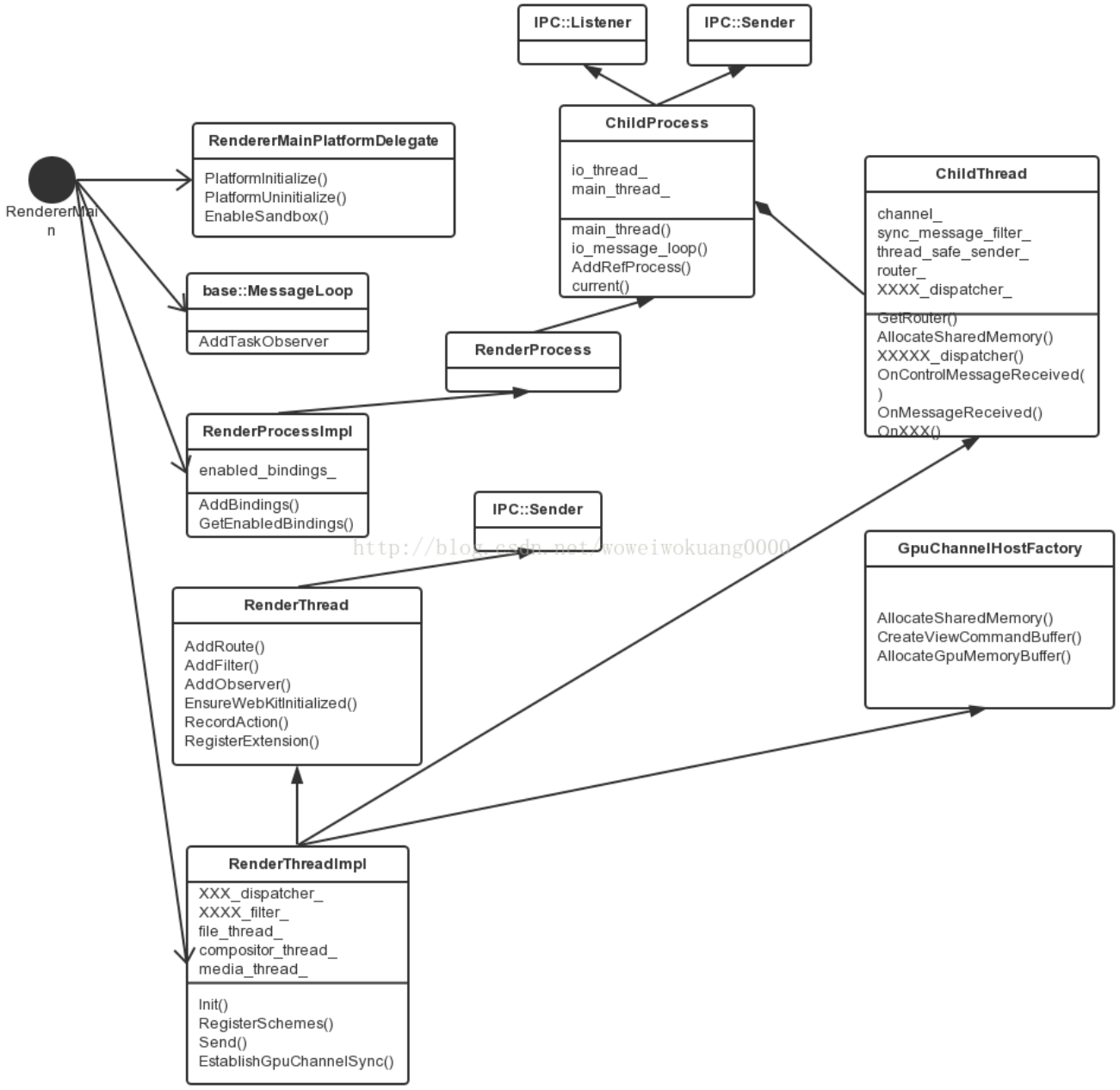
```
1. <span style="color:#333333;">#if !defined(CHROME_MULTIPLE_DLL_CHILD)
2.     { "",                       BrowserMain },
3. #endif
4. #if !defined(CHROME_MULTIPLE_DLL_BROWSER)
5. #if defined(ENABLE_PLUGINS)
6. #if !defined(OS_LINUX)
7.     { switches::kPluginProcess,      PluginMain },
8. #endif
```

```
    { switches::kWorkerProcess,      WorkerMain },
    { switches::kPpapiPluginProcess, PpapiPluginMain },
    { switches::kPpapiBrokerProcess, PpapiBrokerMain },
#endif  // ENABLE_PLUGINS
    { switches::kUtilityProcess,     UtilityMain },
    { switches::kRendererProcess,    RendererMain },
    { switches::kGpuProcess,         GpuMain },
#endif  // !CHROME_MULTIPLE_DLL_BROWSER
  };
  for (size_t i = 0; i < arraysize(kMainFunctions); ++i) {
      .....
      return </span><span style="color:#ff0000;">kMainFunctions[i].function(main_function_params);
    </span><span style="color:#333333;">
  }</span>
```

下面分别为Browser进程和Renderer进程初始化阶段的主要类图，入口分别为BrowserMain和RendererMain：Browser进程：

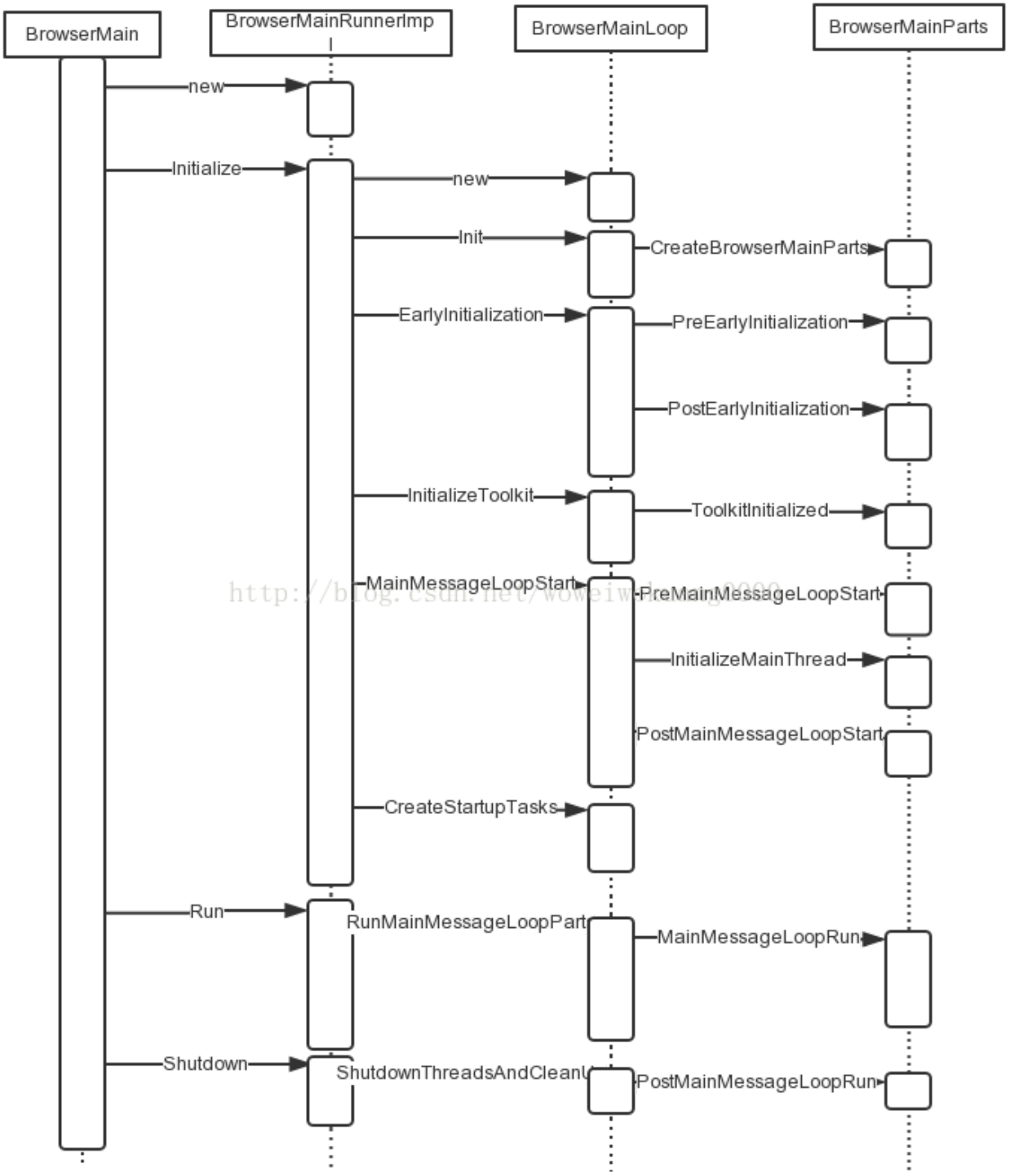


其中BrowserMainRunnerImpl类主要完成主消息循环BrowserMainLoop的创建、初始化及运行；BrowserMainParts类为虚基类，主要定义了主线程初始化各个阶段的回调操作，具体实现需要各个运行环境自己提供，后面会对这些阶段进行归类。Renderer进程：



其中，RenderThreadImpl实际上就相当于是RenderView的后台线程，主要负责与Browser进程进行IPC通信，并将接收到的IPC消息经过不同的*_dispatcher_转发给不同的模块或子线程；RendererMainPlatformDelegate类为抽象类，具体实现与平台相关。

下面为Browser进程的主线程初始化过程的各个阶段调用的顺序关系，此处不关注具体实现细节：



上图中BrowserMainParts在各个运行环境中都有相应的实现，如Contentshell，chrome等，其中方法可以看做是回调各个平台自己相应的实现完成所需要的功能；这几个部分分别位于如下几个文件中：content/browser/browser_main.cc, content/browser/browswer_main_runner.cc, content/browser/browser_main_loop.cc;其中BrowserMainLoop::CreateStartupTasks()方法看以看做是批量操作，其中创建了若干个任务StartupTask添加到任务队列startup_task_runner_，并通过调用startup_task_runner_->StartRunningTasksAsync()或startup_task_runner_->RunAllTasksNow()方法执行这些任务，主要完成其他线程的启动，其中Browser进程的其他线程如DB线程、File线程、Cache线程、IO线程等都是在这些任务之一的BrowserMainLoop::CreateThreads()忠完成创建和启动的：

1.
   for (size_t thread_id = BrowserThread::UI + 1;

```
 2.       thread_id < BrowserThread::ID_COUNT;
 3.       ++thread_id)</span><span style="color:#333333;"> {
 4.     scoped_ptr<BrowserProcessSubThread>* thread_to_start = NULL;
 5.     base::Thread::Options* options = &default_options;
 6.     switch (thread_id) {
 7.       case BrowserThread::DB:
 8.         TRACE_EVENT_BEGIN1("startup",
 9.             "BrowserMainLoop::CreateThreads:start",
10.             "Thread", "BrowserThread::DB");
11.         thread_to_start = &db_thread_;
12.         break;
13.       case BrowserThread::FILE_USER_BLOCKING:
14.         TRACE_EVENT_BEGIN1("startup",
15.             "BrowserMainLoop::CreateThreads:start",
16.             "Thread", "BrowserThread::FILE_USER_BLOCKING");
17.         thread_to_start = &file_user_blocking_thread_;
18.         break;
19.       case BrowserThread::FILE:
20.         TRACE_EVENT_BEGIN1("startup",
21.             "BrowserMainLoop::CreateThreads:start",
22.             "Thread", "BrowserThread::FILE");
23.         thread_to_start = &file_thread_;
24. #if defined(OS_WIN)
25.         options = &ui_message_loop_options;
26. #else
27.         options = &io_message_loop_options;
28. #endif
29.         break;
30.       case BrowserThread::PROCESS_LAUNCHER:
31.         TRACE_EVENT_BEGIN1("startup",
32.             "BrowserMainLoop::CreateThreads:start",
33.             "Thread", "BrowserThread::PROCESS_LAUNCHER");
34.         thread_to_start = &process_launcher_thread_;
35.         break;
36.       case BrowserThread::CACHE:
37.         TRACE_EVENT_BEGIN1("startup",
38.             "BrowserMainLoop::CreateThreads:start",
39.             "Thread", "BrowserThread::CACHE");
40.         thread_to_start = &cache_thread_;
41.         options = &io_message_loop_options;
42.         break;
43.       case BrowserThread::IO:
```

```
44.         TRACE_EVENT_BEGIN1("startup",
45.             "BrowserMainLoop::CreateThreads:start",
46.             "Thread", "BrowserThread::IO");
47.         thread_to_start = &io_thread_;
48.         options = &io_message_loop_options;
49.         break;
50.       case BrowserThread::UI:
51.       case BrowserThread::ID_COUNT:
52.       default:
53.         NOTREACHED();
54.         break;
55.     }
56.     BrowserThread::ID id = static_cast<BrowserThread::ID>(thread_id);
57. </span><span style="color:#ff0000;">    if (thread_to_start) {
58.       (*thread_to_start).reset(new BrowserProcessSubThread(id));
59.       (*thread_to_start)->StartWithOptions(*options);
60.     }</span><span style="color:#333333;"> else {
61.       NOTREACHED();
62.     }
63.     TRACE_EVENT_END0("startup", "BrowserMainLoop::CreateThreads:start");
64.   }
65.   created_threads_ = true;
66.   return result_code_;
67. }
68. </span>
```

这里只是描述了大体的流程分析，详细流程还需要结合具体的产品（如contentshell，crosswalk等）进行分析。

顶

0