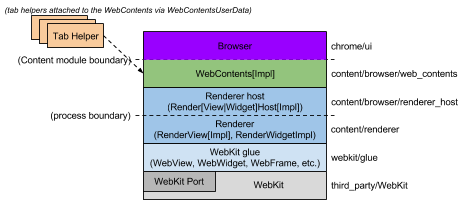
Chromium框架整理

亚星提到几个问题重点了解一下, 我就从这几个方面的解答入手尝试解释Chromium的结构. 多数内容来自[官网文档](http://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome)和[这个人的博客](http://blog.csdn.net/milado_nju?viewmode=contents).

* 渲染过程

WebKit是一个把原始内容转成图片的渲染引擎, chromium调用WebKit把原始内容转成图像显示在窗口上.



[官网文档](http://www.chromium.org/developers/design-documents/displaying-a-web-page-in-chrome)描述了Chromium调用WebKit的过程(从下往上)涉及到把WebKit难用的接口封装成WebKit port, 使用WebKit glue(貌似现在还在移植中?)和渲染层对接, 渲染进程和浏览器进程通过命名通道交流(内容和渲染结果). 而WebKit怎样将Chromium传给它的原始内容转化成图片在[这篇文章](http://blog.csdn.net/milado_nju/article/details/7292131)中被描述为一个解析DOM树中可见内容为Render树, Render树按各种样式划分为RenderLayer树, (通常情况下)最后以RenderLayer为单位渲染成位图显示在窗口上. [官网关于GPU加速的介绍](http://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome)的第一部分也有详细讲基本渲染流程. GPU加速和硬件合成加速会在下面讲到.

* 跨进程通信

Chromium使用IPC命名通道做进程间通信.

回答Chromium如何进行跨进程通信就要先解释一下Chromium的多进程模型. Chromium支持四种进程体系, 一个tab一个进程, 一个域名一个进程, 一个域名实例一个进程, 单进程. 默认是域名实例一个进程, 就是用户每次在地址栏输入一个新的域名会开启一个新进程. 这样做的理由是安全和响应迅速, 一个渲染进程挂了不会拖累整个浏览器. 多进程模型可以简单地想象成一个浏览器进程(老大进程)和多个渲染进程(可以理解成一个tab)通信, 有时候会加上插件进程. 从这篇[讲述多进程架构](http://www.chromium.org/developers/design-documents/multi-process-architecture)和这篇[讲述四种进程模型](http://www.chromium.org/developers/design-documents/process-models)的官方文档中可以看到chromium使用命名通道来实行进程间的通信, 在浏览器进程中的I/O工作线程和渲染进程中的主进程分别负责命名通道的两端. 官方设计文档[Inter-Process communication](http://www.chromium.org/developers/design-documents/inter-process-communication)有讲述怎样使用Chromium的IPC::Channel类实现通信, 包括消息的类型, 消息的创建(发送时创建)和销毁(IPC销毁), 以及消息的处理. 通信的消息按类型分为”routed”(发给指定的view, 需要带有id)和”control”(全局消息)两种, 按方向分为View消息(Browser to view)和ViewHost消息(view to browser)两种, 同理还有, 如果是和插件进程通信就按方向分为PluginProcess消息(browser to plugin)和PluginProcessHost消息(plugin to browser). 这些通信指的都是异步通信(因为不在同一个进程中). 需要同步通信的时候(调用方法需要返回值的时候). 浏览器到渲染进程的IPC是不允许同步的, 所以只有渲染进程到浏览器进程的消息有可能需要同步, 值得一提的是渲染进程的发出消息的线程(非IPC线程, WebKit线程通过IPC线程发送消息, 此处指的是WebKit线程), 在发出需要返回值的同步消息之后就会被挂起, 直到消息返回才会解锁, 与此同时渲染进程的IPC线程继续运行, 接收浏览器进程发过来的其他消息, 但是WebKit线程处理不了啊, 这些消息会放在WebKit线程的消息队列里, 等WebKit线程解锁了再挨个处理. 这里提到了渲染进程中的一个IPC线程和一个WebKit线程, 在官方的[多线程文档](http://www.chromium.org/developers/design-documents/threading)中讲到chromium在线程封装类中使用MessageLoop而非共享内存对象在线程间通信来尽可能地减少对线程的block, 关于MessageLoop我觉得这个人的[这篇文章](http://blog.csdn.net/milado_nju/article/details/8539795)和[这篇文章](http://blog.csdn.net/milado_nju/article/details/8027625)对理解也很有帮助.

* 插件体系

Chromium使用NPAPI插件. 调用插件的一方和封装插件的一方分别需要实现一套NPAPI接口.

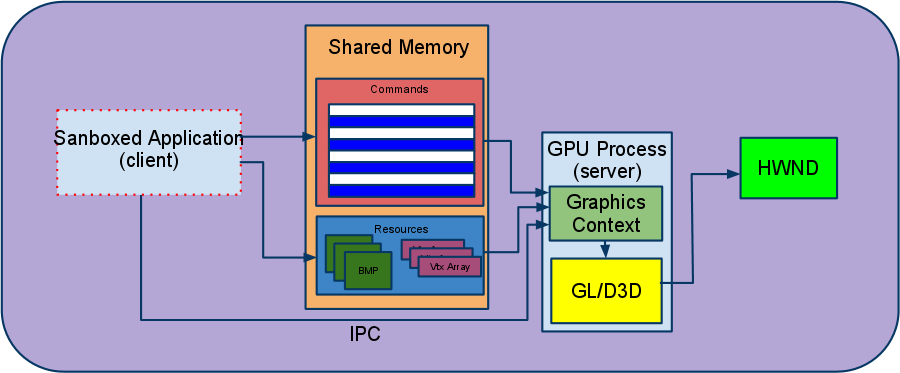
渲染进程解析到一个需要使用插件来渲染的元素时, 会调用插件. Chromium支持在渲染进程内和渲染进程外使用插件. 在进程外使用插件的话, 浏览器进程会为每个插件创建一个进程, 多个渲染进程调用同一个插件进程, 插件进程内会为多个渲染进程创建不同的插件实例供调用. NPAPI接口在插件进程中实现. 插件进程和渲染进程仍然使用IPC通信. [这篇文档](http://www.chromium.org/developers/design-documents/plugin-architecture)和上面同一个人的[这篇文章](http://blog.csdn.net/milado_nju/article/details/7216136)有详细讲述NPAPI插件使用.

插件的绘制分有窗口的和无窗口的两种: (待勘误)有窗口的插件直接在分给它的一块绘制上下文中绘制, 渲染成什么样子完全由插件说了算; 无窗口的插件可能需要绘制在网页内容上层, 有透明效果, 即最终绘制结果由插件和原始内容一起决定, 这时候就不能给它绘制上下文, 渲染进程会保存一个插件绘制的backing store, 发给插件, 插件会保留一份这个backing store的备份, 在这个备份上绘制, 绘制的结果通过共享内存transport API发回给渲染进程.

* 渲染的硬件加速合成和GPU加速

从[软件渲染过程](http://blog.csdn.net/milado_nju/article/details/7455348)即大部分网页内容的渲染过程中渲染进程和浏览器进程通过一个叫做TransportDIB的共享内存(此处有误, 待矫正), 异步地更新浏览器窗口的图像, 浏览器进程告诉渲染进程显示内容(或者渲染进程自己要更新), 渲染进程把TansportDIB作为绘制目标渲染, 渲染完成后告诉浏览器进程, 浏览器进程从TransportDIB中读取内容到自己的backing store中并更新窗口, 在渲染进程通知浏览器进程渲染完毕之前, 浏览器会显示backingstore中也就是上一次更新的内容. 共享内存即用即删, 不会一直存在. backstore才是浏览器端始终保存的内容, 这个backing store是整个网页的图像, 而不是某个render layer的图像, 所以更新的时候, 虽然不会重新渲染所有层但是会检查一遍渲染层是否和更新区域有重叠, 有的话就会把这个render layer的东西都重新绘制过一遍.

理解了软件渲染过程之后再来看WebKit所支持的(需要启用”加速合成”)[硬件加速](http://blog.csdn.net/milado_nju/article/details/7292174)合成, WebKit会为有需要的Render Layer(一部分)创建backing surfaces (因为是WebKit创建的, 所以在渲染进程中), 这些层会把自己绘制到backing surface上而不是目标位图上, 这样在更新的时候, 虽然我们仍然需要从Render Layer Tree生成目标位图, 但是不必重绘每个和目标区域重合的Render Layer, (内容没变的话)直接使用它们在backing surface中的内容就可以了, 所以叫做”合成”(compositing)加速.



软件渲染过程中渲染进程通过IPC和共享内存把绘制的位图传给浏览器进程, 浏览器进程再绘制到窗口上, 而图形处理本来是GPU的强项. 出于和多进程模型同样的安全性考虑(也因为处于沙盒的渲染进程不能直接调用3D接口), GPU的使用也放在单独的进程中. GPU进程可以直接和渲染进程对话, 把渲染进程的渲染结果绘制到窗口上. 渲染进程把渲染结果和发给GPU的命令都放在共享内存中, 通过IPC通道告诉GPU进程要做的事情在哪里, 放在共享内存中的具体的命令在[GPU加速的文档](http://www.chromium.org/developers/design-documents/gpu-accelerated-compositing-in-chrome)和[这里](http://blog.csdn.net/milado_nju/article/details/7216145)有详细介绍.

* Window/Windowless 绘制的渲染加速区别

这块没找到资料, 尝试对比了一下Window/Windowless插件在软件绘制, 硬件组合加速和GPU加速时的流程, 没有发现什么问题...

结合前面两点, 有窗口插件在直接在渲染进程的绘制上下文渲染, 无窗口插件通过共享内存发送给渲染进程渲染结果, 联系在使用硬件合成加速时, 渲染进程会把一些层绘制在backing surface中(而不是绘制在渲染进程和浏览器进程的共享内存transpor DIB中), 在使用GPU加速时, 会直接把共享内存中的内容绘制在窗口上,

(软件渲染过程)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 插件进程 --> 渲染进程 | 渲染进程 --> 浏览器进程 |  |
| 有窗口 | 插件绘制在目标绘制上下文中 | 共享内存 | 插件绘制在渲染进程和浏览器进程的共享内存中 |
| 无窗口 | 插件绘制在插件和渲染进程的共享内存中 | 共享内存 | 渲染进程把它和插件的共享内存中的内容搬到它和浏览器的共享内存中, 大丈夫 |

(硬件加速合成)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 插件进程 --> 渲染进程 | 渲染进程 --> 浏览器进程 |  |
| 有窗口 | 插件在渲染进程创建的, 某个Render layer的backing surface上绘制, 即渲染进程的某块内存上 | (待勘误)共享内存 | 插件绘制在渲染进程的内存上, 渲染进程组合一下搬到它和浏览器进程的共享内存上 |
| 无窗口 | 插件绘制在插件和渲染进程的共享内存中 | 共享内存 | 插件绘制在共享内存上, 渲染器从它和插件的共享内存搬到自己Render Layer的内存中, 再组合一下放在它和浏览器的共享内存中  可能会有Render Layer太多, 插件和渲染进程需要开启很多的共享内存的问题? |

(GPU加速)

|  |  |  |  |
| --- | --- | --- | --- |
|  | 插件进程 --> 渲染进程 | 渲染进程 --> GPU进程 |  |
| 有窗口 | 插件在渲染进程创建的, 某个Render layer的backing surface上绘制, 即渲染进程的某块内存上 | 共享内存  GPU直接绘制在窗口上 | 插件绘制在渲染进程的内存上, 渲染进程组合一下搬到它和GPU进程的共享内存上 |
| 无窗口 | 插件绘制在插件和渲染进程的共享内存中 | 共享内存  GPU直接绘制在窗口上 | 插件绘制在共享内存上, 渲染器从它和插件的共享内存搬到自己Render Layer的内存中, 再组合一下放在它和GPU的共享内存中.  没看出来这里有什么问题... |