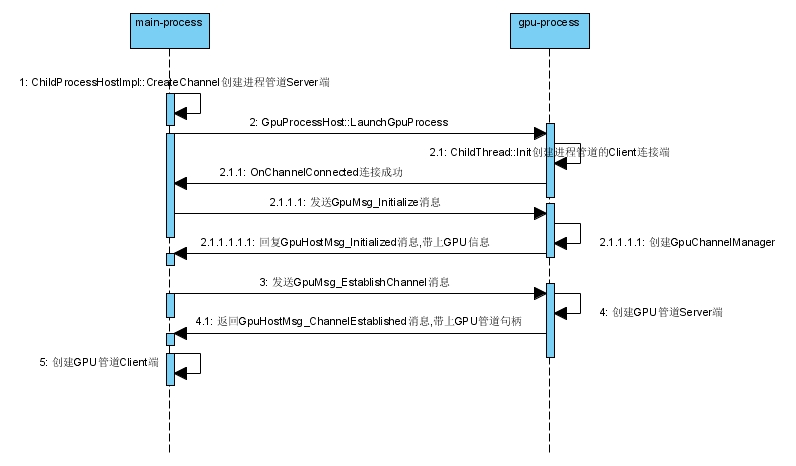
转载请注明出处：http://www.cnblogs.com/fangkm/p/3960327.html

硬件渲染依赖计算机的GPU，GPU种类繁多，兼容这么多种类的硬件，稳定性是个大问题，虽然Chromium内部维护了一个GPU黑名单列表，限定了哪些渲染特性不能在哪些GPU上渲染，但还不足以解决使用GPU过程中的稳定性问题。在Chromium的多进程架构中，不稳定不可控的因素一般都会采取独立进程的渲染方式， 从而保证主进程的稳定性，比如Render进程、插件进程，GPU的使用也一样，采取独立进程的方式运行。

主进程在启动的时候，会调用GpuDataManagerImpl的Initialize初始化方法，在Initialize方法中，会加载预置的GPU黑名单列表，列表信息位于” src\gpu\config\software\_rendering\_list\_json.cc”文件中，由JSON格式来描述，程序中通过kSoftwareRenderingListJson变量来访问。通过分析GPU的黑名单列表，可以获得当前GPU在当前的系统下不知道的渲染特性，当初始化渲染环境时，会从GpuDataManagerImpl维护的禁用渲染特性中查询是否被禁用，从而决定采取硬件渲染还是软件渲染。另外，在GPU进程崩溃超过三次时，会在GpuDataManagerImpl里禁用当前GPU的所有渲染支持，也就是将所有的渲染特性加入到当前的GPU黑名单列表，之后的渲染流程都走软件渲染的逻辑。相关逻辑请参见GpuProcessHost的析构方法。

下面探寻一下GPU进程的启动流程：



      和创建Render进程类似，主进程首先创建一个创建一个命名管道的Server端（参见ChildProcessHostImpl的CreateChannel方法），然后启动GPU进程，将创建好的管道名当命令行参数传递过去（参见GpuProcessHost的LaunchGpuProcess方法），GPU子进程启动后，会从命令行参数中取出管道名，然后创建相应的管道Client端（参见ChildThread的Init方法），然后就完成管道的连接处理。

      与GPU子进程建立连接后，主进程给GPU进程发送GpuMsg\_Initialize消息，GPU进程收到这个消息后，会回复一个GpuHostMsg\_Initialized消息，附上采集到的GPU信息，这个时候GPU会创建好GpuChannelManager对象，管理即将到来的专供GPU渲染的IPC连接列表。主进程在初始化硬件渲染上下文的时候（参见WebGraphicsContext3DCommandBufferImpl的Initialize方法，一般在主线程）,会向GPU子进程发送GpuMsg\_EstablishChannel消息（在IO线程发送），然后Wait住主线程，GPU进程在收到该消息后，会创建一个专供渲染用的管道Server端（参见GpuChannelManager的OnEstablishChannel处理），创建完成后会给主进程回复一个GpuHostMsg\_ChannelEstablished消息，将刚创建好的渲染管道名称传递过去，主进程接收到回复消息后（在IO线程接受处理），将之前Wait的信号授信，主线程继续处理，从而完成硬件渲染通信管道的初始化过程。

      Render进程也需要使用GPU进行硬件渲染，Render进程GPU管道的创建流程委托给主进程，当Render进程初始化3D绘制环境时，会给主进程发送GpuHostMsg\_EstablishGpuChannel同步消息，然后等待答复消息的返回。主进程收到GpuHostMsg\_EstablishGpuChannel消息后的处理过程，就是上面讲到的主进程启动GPU进程的流程，不同的是，当GPU进程返回创建好的IPC管道名后，会将该名称通过GpuHostMsg\_EstablishGpuChannel答复消息返回给Render进程，由Render进程创建相应的IPC管道客户端。

