构件媒体播放器研究

【基金项目：国家高技术研究发展计划（863 计划）项目（编号 2001AA113400）】

王瑀屏[1]　陈瑢[2]　王小鸽[3]
（1、清华大学深圳研究生院软件工程中心 北京 100084；2、科泰世纪公司 上海 201203；3、清华大学计算机科学与技术系 北京 100084）

摘要：本文分析了使用构件技术实现媒体播放器的优势：可自动替换解码器、可动态升级等。并以 DirectShow 为例，说明构件媒体播放器的实现方法。然后，通过对 DirectShow 的深入分析（同步问题和控制问题），简化改进 DirectShow，提出一套新的基于构件的媒体播放器的接口定义和实现方案。最后，说明这一方案可以支持硬件解码器，而且可以扩展到媒体捕捉方面。

关键词：媒体播放；构件技术；DirectShow；COM；CAR
中图分类号:TP311.52

Research on Media Player with Component Technique

WANG Yu-ping[1]　CHEN Rong[2]　WANG Xiao-ge[3]
(1.Software Engineering Center, ShenZhen Graduate Institute, Tsinghua University, Beijing100084, China; 2.Koretide Co., Shanghai201203, China; 3.Department of Computer Science and Technology, Tsinghua University, Beijing100084)

Abstract: The advantages of media player's implement using component technique, such as automatical replacing decoders and dynamic upgrading, are analysed. And with an example of DirectShow, illuminate how a media player with component techique works. Afterwards, with analysing DirectShow deeply(Synchronization Problem and Controling Problem), simplify and improve DirectShow, show a new set of interface definition and implement scheme of media player. At last, explain that this scheme can also support hardware decoder, and can be expanded to media capture.

Key words:media playback; component technique; DirectShow; COM; CAR

1、引言

随着数字图像和语音存储技术的发展，各种各样适用于不同场合的编解码算法相继出现。这些算法针对不同的语音、声音、图像、运动画面的特点，对大量的多媒体数据进行压缩。从而，各种编解码器也就随着用户的多媒体需求而广泛应用。另一方面，软件工程技术的发展，造就了面向组件技术。它不仅能够像面向对象技术那样简化开发过程，更简化了维护升级过程。编解码器这种经常需要更新的软件也就必然应当使用这种组件技术。

在编解码器的组件化过程中，最经常用到的莫过于 Microsoft 的 DirectShow。它是基于 Microsoft 提出的 COM 组件对象模型上实现的。但是 DirectShow 的细节实现相当繁琐，对于编解码器这种需要一定效率的软件来说无疑是一种负担。

本文通过对 DirectShow 结构进行分析，对其进行大胆的简化，使得整个媒体播放过程在保持灵活性的同时，处理更加简便。

本文是为科泰世纪公司的“和欣”操作系统设计媒体播放器和数码摄像机的方案的尝试和讨论。

2、DirectShow 构件结构

DirectShow 是在 Microsoft Windows 平台上针对流媒体的体系结构，以用来完成多媒体流的高品质捕捉和播放。它支持多种格式，这些格式可以根据文件动态替换选择。不仅如此，DirectShow 简化了媒体的播放、格式转换、和捕捉任务。同时，它为应用程序提供了对于底层流控制体系。这样，用户可以编写自己的 DirectShow 组件来支持新的格式或自定的特效。[1]

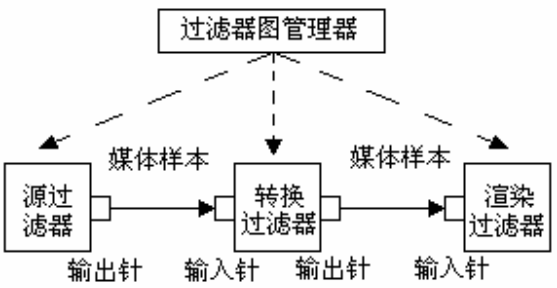

图 1、DirectShow 系统结构图

那么，DirectShow 是如何做到这些的呢？图 1，是 DirectShow 的一个系统结构图。其中最主要的部分就是各个过滤器（filter），DirectShow 能够完成各种各样的媒体操作，都是通过各个过滤器来完成的。各个过滤器，按照它们的表现，可以分成源过滤器（source filter）、转换过滤器（transform filter）、和渲染过滤器（rendering filter），分别是只有输出没有输入、输入输出都有、和只有输入没有输出的过滤器。

DirectShow 就是通过动态的选择和连接不同的过滤器来完成各种媒体操作的。不论我们想要播放的是.avi 文件还是.asf 文件，只要更换一个解码过滤器，事情就完成了。如果文件从网络得来，只需要 更换一个源过滤器，就和本地播放没有什么区别。而且，这些过程并不需要用户自己来控制，用户需要做的，只是给出一个要播放的文件名或者 URL。可以说，这种模块的可自动替换性才是 DirectShow 最重要的意义。

另外，DirectShow 除了播放和捕捉的定式给程序设计人员带来方便，还允许用户自行向过滤器图中加入过滤器，并将它们连接起来。这样，我们可以利用已有的过滤器组件像拼装玩具一样拼出自己想要完成的工作。这是由于 DirectShow 是基于 COM 组件模型的多媒体框架。而 COM 构件技术是 Windows 平台上的二进制构件模型[2]，它支持动态升级。这一优点，只需与非构件化的播放器做一个比较就可以了（参考[3]）。

一个写好的播放器程序，我们不再需要重新修改代码就可以使其支持更多的文件格式和编码格式。当一种新的文件格式或者编码格式出现的时候，只需要安装一个新的过滤器，DirectShow 就会找到这个过滤器，并在适当的时候使用该过滤器。这样，编写播放器程序时就可以将更多的精力放在如何使其更加美观、更加有个性等方面。

3、同步问题

在媒体播放的过程中，一个很重要的问题是视频和音频的同步问题。有时我们在计算机上看到的影片中的图像和声音会出现“延迟”，导致的影片中一个人张了嘴，可是说出的话已经听到或者过了一会才听到。另外，由于网络传输、系统 CPU 忙等问题，可能导致数据流中一些帧不能按时产生，这更增加了同步问题的难度。这时，应该有一个原则：那就是图像可以丢，但是声音不可以丢。原因就是人眼对于连续运动的图像并不敏感，每秒 25 帧的图像在人脑中已经可以行程连续图像，即使其中缺几帧也不会有很大的影响；而声音则不同，人耳对于缺失 1/25 秒的声音会有很强烈的感觉。这样，又有一个问题，如果丢了图像，如何让后续的图像能够随着声音的时间播放。这就是经典的“对口型”问题。

3.1、DirectShow 的解决方案

在 DirectShow 的这种层次结构下想要解决这一问题更加复杂。DirectShow 的**每两个相连的过滤器之间**都有一块缓冲区，用于两个过滤器写入读出以传输数据。这个缓冲区中存储的是叫做媒体样本（Media Sample）的对象，而所有的缓冲区则通过叫做内存分配器（Memory Allocator）的对象来管理。每一个媒体样本中除了媒体数据之外，还要保存该帧的时间片（time stamp），标记该数据产生的时刻或者应当被渲染的时刻。整个 DirectShow 有一个系统基准时钟（System Reference Clock）对象，提供标准时间。不论媒体样本中的时间还是播放的时间都是以这个时钟为基准进行操作的。

在播放过程中，当网络繁忙或者 CPU 繁忙时，渲染过滤器会将缓冲区中的媒体样本依次播出，直到样本缓冲被取空，并通知视频解码器以加快模式运行。这种模式会有更快的解码速度，但是会丢失一些帧的信息。通过这样丢掉一些图像的策略可以保证用户看到的效果不会有明显的下降。与之对应，当媒体样本中缺少了声音样本的时候，视频渲染过滤器和音频渲染过滤器都会停下来等待下一个声音样本。只有在这种情况下，用户才会看到播放器停顿，直到下面的数据到来。其它情况下将不会有很强的感觉。

3.2、改进的解决方案

值得注意的是这些缓冲区。虽然每个过滤器处理数据的速度是不一样的，但是，它们之间的同步并不一定需要缓冲区来完成，甚至有些过程是可以串行处理而不影响效率的。通过下面讨论可以看出，去掉一些缓冲区不会影响世纪效果，反而可以提高执行效率。

从本地读取文件时，文件系统本身为了提高读写效率，就会在内存中设置一个缓冲区，当读写操作超出这个范围时才去读取磁盘或类似的相对慢速的大容量存储器。那么从内存读取文件内容的过程是可以忽略的，完全可以当后面的解码器用到数据的时候读取。因此，读取文件和解码过程是完全可以串行进行的。过程就是读取文件，然后解码，再读取文件，再解码。比较一下多线程处理的情形：两个线程分别在做读文件块和解码，当解码器需要数据时会向缓冲区中取得，如果缓冲区中没有数据没有，解码器会等待并切换线程，反过来如果读文件线程在读文件的时候发现缓冲区满了，它也只有等待并切换线程。两者都可以将进程时间完全占用，但是后者却还需要在线程切换上消耗资源。仔细的读者会发现，刚才的分析过程中，

已经将 DirectShow 在文件源过滤器和文件分裂器合在一起考虑。其原因也和上面类似，不再重复阐述。

同样，即使文件从远程读取，ISO 的七层网络模型或者 TCP/IP 的五层模型都会保证到达上层的数据是完整的，可以说网络对于读取文件来说应当是透明的。而不论是 TCP 的握手协议还是使用 Sliding Window 技术，它能保证内容正确不丢失的方法也就是在内存中设置一段缓冲区，等待数据确认正确完整时再将该缓冲区提供给上层用户使用。即当网络忙碌或者传输有错误的时候，网络协议会帮助将数据暂时读入内存中，正如文件系统将磁盘文件读入内存一样。现在，光缆传输正在推广，将来的网络出错会很小，而传输速度会变得更快。这样，网络的读取文件过程可以说越来越透明，对于应用来说，也就更加没有必要将本地文件系统和网络文件系统区别对待。

4、控制问题

在媒体播放过程中，用户给出了一个想要播放的文件的位置（不管它在本地还是远程）之后，需求并没有完成。在播放过程中，用户会提出开始、结束、暂停、恢复、搜索等命令。

在媒体应用程序中，用户命令的优先级应该是最高的，用户提出的要求一定要在一个很短的时间给出反应。另一方面，媒体处理过程中，一般需要尽量利用 CPU 时间，提前缓冲一些媒体数据。那么当用户提出暂停命令时，有两种策略：一种是立即停止操作，不再进行编码解码工作，另一种则不停止编码解码操作，而只停止播放图像声音或写文件直到缓冲区填满。前者处理简单一些，但是有可能造成 CPU 时间的浪费，后者则有及时恢复的问题。而对于搜索命令，应当向播放文件的提供者请求搜索，应当注意有些情况下是不允许进行搜索的。当搜索被允许时，应当尽快停止各个过滤器中的数据处理过程，搜索文件中对应时间的数据提供给后面的过滤器处理。

同时，由于待播放的时间点已经有所改变，应当同时注意到同步时钟的调节问题。

4.1、DirectShow 的解决方案

对于控制命令，DirectShow 通过过滤器图管理器提供接口接收用户的命令，然后将这些命令转发给所有的过滤器。开始命令将带一个时间片参数，指示所有的过滤器以该时间片的数据开始进行处理；停止命令则使所有的过滤器停止并清空缓冲区，恢复最初的状态；暂停命令使所有的过滤器停止，但它们仍能处理一帧，也就是能将发出暂停命令的时刻读出的第一帧处理完，应用将看到这一帧静止在屏幕上显示出来；恢复命令和开始命令在 DirectShow 中是同一个命令；搜索命令则指示渲染过滤器进行搜索，然后通过渲染过滤器的连接逐级向前来通知所有的过滤器进行搜索，与此同时，保持相互间的同步。

从上面的说明中可以看出，DirectShow 的这种做法按照上面说到的第一种做法，即当用户要求暂停时立即暂停所有过滤器的操作。其实，按照 DirectShow 的结构，理论上是完全可以实现第二种做法的。但是，它并没有这么做，其中一个原因就是当渲染过滤器暂停时，如果前面的过滤器仍然在工作，且不说期间的媒体样本缓冲区的处理，当缓冲区被添满后如何使这些过滤器尽量不占用 CPU 资源就成为一个难于解决的问题。DirectShow 在回避这个问题的同时，造成了恢复后浪费一定的 CPU 时间的可能性。

在播放的搜索处理上，DirectShow 的方法则更是无可奈何。实际的搜索操作应当从文件源开始搜索，就像找到一个文件头重新开始播放一样。但是 DirectShow 却通过位于过滤器图末端的渲染过滤器开始向前传递要进行搜索的消息，这一过程将是缓慢的。虽然，要进行搜索这一命令有必要让所有的过滤器知道（因为有调整同步时钟的问题），DirectShow 的做法并不能称为合理。不直接向源过滤器提出搜索命令的另一个原因是如何找到源过滤器的问题。在过滤器图建立起来之后，过滤器图管理器尽量不干扰过滤器间的数据处理过程。但是遇到暂停命令的时候，需要将这一命令转发给所有的过滤器，这一过程需要从渲染过滤器开始进行，以使暂停命令能够得到最快的响应。也就是说 DirectShow 为了将一条消息发给所有的过滤器，需要按照过滤器图的相互联系，自后向前逐层寻找。这种方法在搜索问题上露出了破绽。但是，为了系统的简单性，DirectShow 不能为了一个不太常用的搜索命令而设置一条自前向后传递消息的过程。

4.2、改进的解决方案

出现上述问题的根本原因有两条：第一，DirectShow 想要支持的过滤器之间的连接方式太多，从而达到完成任意功能的目标。造成一种无法一起控制所有正在使用的过滤器，而只能通过遍历过滤器图来找到所有的过滤器。第二，这些过滤器太没有“个性”，对于它们的处理几乎是等同的。而事实上，它们的功能有着本质的差别。

基于以上分析，提出了一下改进方案。按照比较常用的播放和捕捉的要求，将过滤器之间的连接关系锁定下来。让各个“过滤器”引出不同的接口，并将这些接口指针统一保存起来使用。这样，暂停和恢复时可以找到渲染过滤器，搜索的时候也可以找到源过滤器。再加上上一节提到的将某些过滤器间的缓冲区删掉，可以将同步问题集中在一个很小的范围内解决。这时，我们就有能力实现前述的实现暂停恢复的两种策略，而这可以当成一种选项给用户选择。

5、方案说明

综合以上的想法，得出以下的整体设计。对于媒体播放流程，有如图 2 的结构：

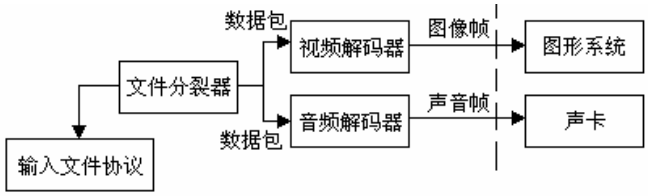

图 2、媒体播放流程结构图

整体上，是参考 DirectShow 的过滤器体系，将播放和捕捉过程分成若干个模块进行。但是，这里有如下几点不同：

1. 源过滤器换成了输入协议构件，这一构件仅被文件分裂器调用。如果需要缓冲，可以在输入协议构件内部进行，不增加整个系统同步控制的负担。
2. 虚线左边的各个模块不再独自处理、相互传递数据，而由播放器引擎（与过滤器图对应）调用，中间的数据缓冲被舍弃。
3. 相对的，虚线处的缓冲区保留。原因是解码器的速度相对不稳定，不易控制。
4. 结构相对固定。但其中每个模块仍然可以根据不同的需要更换。虽然损失了一定的灵活性，但是仍能够完成需要的功能及灵活性。

其中的每一个模块引出自己的接口，下面给出一个简要的功能性的定义：

```
interface ISplitter {
   Set_Protocol();          //设文件来源（本地，http，等）
   Get_StreamNum_Audio();//获取文件中音频流数量
   Get_StreamNum_Video();//获取文件中视频流数量
   Get_Stream_Audio();   //获取指定音频流信息
   Get_Stream_Video();   //获取指定视频流信息
   Read_Header();        //读出文件头信息
   Read_Packet();        //读出一个数据包
   Read_Seek();          //支持搜索情况下，进行搜索
   Get_Extension();      //获取该构件支持的文件扩展名
   Get_Parameter();      //获取文件的版权等信息
};
interface IVideoDecoder {
   SetContext();         //设置解码参数
   Decode();             //从一个数据包中解出一帧图像
   HurryUp();            //加速模式
   Flush();              //清空缓冲（以支持停止）
};
interface IAudioDecoder {
   SetContext();         //设置解码参数
   Decode();             //从一个数据包中解出一帧声音
   HurryUp();            //加速模式
   Flush();              //清空缓冲（以支持停止）
};
```

有了这些构件的支持，播放功能的结构已经很完整。剩下的控制问题参照前面的思路变得很容易，已经是细节问题，这里就不详细说明了。另外，在科泰世纪公司的 CAR 构件模型的支持下，写好的众多解码器构件、文件分裂器构件构件将可以通过类别（Category）构件找到，满足实现一个动态升级的播放器应用的要求（参考[4]）。

6、扩展与结论

现在，随着媒体编码标准的出台，主要是 MPEG 标准的制定，各种硬件编解码器也纷纷出现。它们利用硬件速度更快的优势，有效的完成编解码工作。如果有可能，使用这类的硬件进行编解码，则会达到很好的效果。那么，在设计构件化的编解码器的时候，应当考虑到对这类硬件的支持。

现有的思路是，即使有硬件编解码的存在，它们还是需要通过软件来完成编解码功能（比如，通过驱动程序）。那么，可以将硬件编解码器按照设计好的编解码器接口写成一个构件，插入到上面讲到的媒体播放中去，就可以了。

同时，将这一结构反向（读文件变成写文件，解码器换为编码器，同时将图左右翻转），可以实现媒体捕捉的功能。至于各种格式转换功能，由于不涉及实时性要求，实现更为简单。这样，DirectShow 能够完成的，这里也就都可以而且相对简洁地完成了。

总之，对于 Windows 操作系统上的各式媒体应用来说，绝大多数都是调用 DirectShow 提供的 API 来实现的。相对的，在其它的平台上，就都面临着如何完成这一目前最经常用到的任务的题目。文中提出的这种结构，简单明了，又不失灵活，可以用于支持组件技术的任何操作系统。

目前，笔者正在将这一结构应用于科泰世纪公司的“和欣”操作系统上，播放器已经初成模样。

参考文献

[1]DirectX Documentation for C++, MSDN [J/OL]. Microsoft Co. http://msdn.microsoft.com.
[2]潘爱民.COM 本质论[M].北京：清华大学出版社.2001
[3]FFmpeg Documentation[EB/OL]. Free Software Foundation. http://ffmpeg.sourceforge.net/ffmpeg-doc.html.
[4]上海科泰世纪公司.基于类别创建驱动构件对象实现设备驱动程序多态的方法[P].中国专利：02159488.0，2002 年 12 月.美国专利：10/747,058，2002 年 12 月

作者简介：
王瑀屏（1984－），男，辽宁沈阳人，硕士研究生，主要研究方向：DirectX 研究开发；
陈瑢（1957－），男，科泰世纪公司 CEO、首席科学家，主要研究方向：网络操作系统，构件技术；
王小鸽（1957－），女，山西，副教授，博士，主要研究方向：高性能计算技术，中间件技术