对于 Filter 的把握，一般认为要掌握以下三方面的内容：Filter 之间 Pin 的连接、Filter 之间的数据传输以及流媒体的随机访问（或者说流的定位）。下面就开始分别进行阐述。

所谓的 Filter Pin 之间的连接，实际上是 Pin 之间 Media Type（媒体类型）的一个协商过程。连接总是从输出 Pin 指向输入 Pin 的。

要想深入了解具体的连接过程，就必须认真研读 SDK 的基类源代码（位于 BaseClasses\amfilter.cpp，类 CBasePin 的 Connect 方法）。

连接的大致过程为，枚举欲连接的输入 Pin 上所有的媒体类型，逐一用这些媒体类型与输出 Pin 进行连接，如果输出 Pin 也接受这种媒体类型，则 Pin 之间的连接宣告成功；如果所有输入 Pin 上枚举的媒体类型输出 Pin 都不支持，则枚举输出 Pin 上的所有媒体类型，并逐一用这些媒体类型与输入 Pin 进行连接。如果输入 Pin 接受其中的一种媒体类型，则 Pin 之间的连接到此也宣告成功；如果输出 Pin 上的所有媒体类型，输入 Pin 都不支持，则这两个 Pin 之间的连接过程宣告失败。

有一点需要注意的是，上述的输入 Pin 与输出 Pin 一般不属于同一个 Filter，典型的是上一级 Filter（也叫 Upstream Filter）的输出 Pin 连向下一级 Filter（也叫 Downstream Filter）的输入 Pin。

当 Filter 的 Pin 之间连接完成，也就是说，连接双方通过协商取得了一种大家都支持的媒体类型之后，即开始为数据传输做准备。这些准备工作中，最重要的是 Pin 上的内存分配器的协商，一般也是由输出 Pin 发起。在 DirectShow Filter 之间，数据是通过一个一个数据包传送的，这个数据包叫做 Sample。Sample 本身是一个 COM 对象，拥有一段内存用以装载数据，Sample 就由内存分配器（Allocator）来统一管理。已成功连接的一对输出、输入 Pin 使用同一个内存分配器，所以数据从输出 Pin 传送到输入 Pin 上是无需内存拷贝的。而典型的数据拷贝，一般发生在 Filter 内部，从 Filter 的输入 Pin 上读取数据后，进行一定意图的处理，然后在 Filter 的输出 Pin 上填充数据，然后继续往下传输。

下面，我们就具体阐述一下 Filter 之间的数据传送。

首先，大家要区分一下 Filter 的两种主要的数据传输模式：推模式（Push Model）和拉模式（Pull Model）。

所谓推模式，即源 Filter（Source Filter）自己能够产生数据，并且一般在它的输出 Pin 上有独立的子线程负责将数据发送出去，常见的情况如代表 WDM 模型的采集卡的 Live Source Filter；

所谓拉模式，即源 Filter 不具有把自己的数据送出去的能力，这种情况下，一般源 Filter 后紧跟着接一个 Parser Filter 或 Splitter Filter，这种 Filter 一般在输入 Pin 上有个独立的子线程，负责不断地从源 Filter 索取数据，然后经过处理后将数据传送下去，常见的情况如文件源。推模式下，源 Filter 是主动的；拉模式下，源 Filter 是被动的。而事实上，如果将上图拉模式中的源 Filter 和 Splitter Filter 看成另一个虚拟的源 Filter，则后面的 Filter 之间的数据传输也与推模式完全相同。

数据到底是怎么通过连接着的 Pin 传输的呢？

首先来看推模式。在源 Filter 后面的 Filter 输入 Pin 上，一定实现了一个 IMemInputPin 接口，数据正是通过上一级 Filter 调用这个接口的 Receive 方法进行传输的。值得注意的是

（上面已经提到过），数据从输出 Pin 通过 Receive 方法调用传输到输入 Pin 上，并没有进行内存拷贝，它只是一个相当于数据到达的“通知”。

再看一下拉模式。拉模式下的源 Filter 的输出 Pin 上，一定实现了一个 IAsyncReader 接口；其后面的 Splitter Filter，就是通过调用这个接口的 Request 方法或者 SyncRead 方法来获得数据。Splitter Filter 然后像推模式一样，调用下一级 Filter 输入 Pin 上的 IMemInputPin 接口 Receive 方法实现数据的往下传送。

研读 SDK 的基类源代码（BaseClasses\source.cpp 和 pullpin.cpp）。

下面，我们来讲一下流的定位（Media Seeking）。在 GraphEdit 中，当我们成功构建了一个 Filter Graph 之后，我们就可以播放它。在播放中，我们可以看到进度条也在相应地前进。当然，我们也可以通过拖动进度条，实现随机访问。要做到这一点，在应用程序级别应该可以知道 Filter Graph 总共要播放多长时间，当前播放到什么位置等等。那么，在 Filter 级别，这一点是怎么实现的呢？

我们知道，若干个 Filter 通过 Pin 的相互连接组成了 Filter Graph。而这个 Filter Graph 是由另一个 COM 对象 Filter Graph Manager 来管理的。通过 Filter Graph Manager，我们就可以得到一个 IMediaSeeking 的接口来实现对流媒体的定位。在 Filter 级别，我们可以看到，Filter Graph Manager 首先从最后一个 Filter（Renderer Filter）开始，询问上一级 Filter 的输出 Pin 是否支持 IMediaSeeking 接口。如果支持，则返回这个接口；如果不支持，则继续往上一级 Filter 询问，直到源 Filter。

一般在源 Filter 的输出 Pin 上实现 IMediaSeeking 接口，它告诉调用者总共有多长时间的媒体内容，当前播放位置等信息。（如果是文件源，一般在 Parser Filter 或 Splitter Filter 实现这个接口。）

如果我们写源 Filter，就要在 Filter 的输出 Pin 上实现 IMediaSeeking 这个接口；

如果写中间的传输 Filter，只需要在输出 Pin 上将用户的获得接口请求往上传递给上一级 Filter 的输出 Pin；

如果写 Renderer Filter，需要在 Filter 上将用户的获得接口请求往上传递给上一级 Filter 的输出 Pin。

进一步的了解，请认真研读 SDK 的基类源代码（位于\BaseClasses\transfrm.cpp 的类方法 CTransformOutputPin::NonDelegatingQueryInterface 实现和 ctlutil.cpp 中类 CPosPassThru 的实现）。

如何写自己的 Filter

首先，从 VC++的项目开始（请确认你已经给 VC++配置好了 DirectX 的开发环境）。写自己的 Filter，第一步是使用 VC++建立一个 Filter 的项目。由于 DirectX SDK 提供了很多 Filter 的例子项目（位于 DXSDK\samples\Multimedia\DirectShow\ Filters 目录下），最简单的方法就是拷贝一个，然后再在此基础上修改。但如果你是 Filter 开发的初学者，笔者并不赞成这么做。

自己新建一个 Filter 项目也很简单。使用 VC++的向导，建立一个空的”Win32 Dynamic-

link Library”项目。注意，几个文件是必须有的：.def 文件，定义四个导出函数；定义 Filter 类的.cpp 文件和.h 文件，并在.cpp 文件中定义 Filter 的注册信息以及两个 Filter 的注册函数：DllRegisterServer 和 DllUnregisterServer。（注：Filter 的注册信息是 Filter 在注册时写到注册表里的内容，格式可以参考 SDK 的示例代码，Filter 相关的 GUID 务必使用 GuidGen.exe 产生。）接下去进行项目的设置（Project->Settings…）。此时，你可以打开一个 SDK 的例子项目进行对比，有些宏定义完全可以照抄，最后注意将输出文件的扩展名改为.ax。

上一讲曾经提到过，在写 Filter 之前，选择一个合适的 Filter 基类是至关重要的。为此，你必须对几个 Filter 的基类有相当的了解。

在实际应用中，Filter 的基类并不总是选择 CBaseFilter 的。相反，因为我们绝大部分写的都是中间的传输 Filter（Transform. Filter），所以基类选择 CTransformFilter 和 CTransInPlaceFilter 的居多。如果我们写的是源 Filter，我们可以选择 CSource 作为基类；如果是 Renderer Filter，可以选择 CBaseRenderer 或 CBaseVideoRenderer 等。

总之，选择好 Filter 的基类是很重要的。当然，选择 Filter 的基类也是很灵活的，没有绝对的标准。能够通过 CTransformFilter 实现的 Filter 当然也能从 CBaseFilter 一步一步实现。

对 Filter 基类的选择提出几点建议供大家参考。

首先，你必须明确这个 Filter 要完成什么样的功能，即要对 Filter 项目进行需求分析。请尽量保持 Filter 实现的功能的单一性。如果必要的话，你可以将需求分解，由两个（或者更多的）功能单一的 Filter 去实现总的功能需求。

其次，你应该明确这个 Filter 大致在整个 Filter Graph 的位置，这个 Filter 的输入是什么数据，输出是什么数据，有几个输入 Pin、几个输出 Pin 等等。你可以画出这个 Filter 的草图。弄清这一点十分重要，这将直接决定你使用哪种“模型”的 Filter。比如，如果 Filter 仅有一个输入 Pin 和一个输出 Pin，而且一进一处的媒体类型相同，则一般采用 CTransInPlaceFilter 作为 Filter 的基类；如果媒体类型不一样，则一般选择 CTransformFilter 作为基类。

再者，考虑一些数据传输、处理的特殊性要求。比如 Filter 的输入和输出的 Sample 并不是一一对应的，这就一般要在输入 Pin 上进行数据的缓存，而在输出 Pin 上使用专门的线程进行数据处理。这种情况下，Filter 的基类选择 CSource 为宜（虽然这个 Filter 并不是源 Filter）。

当 Filter 的基类选定了之后，Pin 的基类也就相应选定了。接下去，就是 Filter 和 Pin 上的代码实现了。有一点需要注意的是，从软件设计的角度上来说，应该将你的逻辑类代码同 Filter 的代码分开。下面，我们一起来看一下输入 Pin 的实现。你需要实现基类所有的纯虚函数，比如 CheckMediaType 等。在 CheckMediaType 内，你可以对媒体类型进行检验，看是否是你期望的那种。因为大部分 Filter 采用的是推模式传输数据，所以在输入 Pin 上一般都实现了 Receive 方法。有的基类里面已经实现了 Receive，而在 Filter 类上留一个纯虚函数供用户重载进行数据处理。这种情况下一般是无需重载 Receive 方法的，除非基类的实现不符合你的实际要求。而如果你重载了 Receive 方法，一般会同时重载以下三个函数 EndOfStream、BeginFlush 和 EndFlush。我们再来看一下输出 Pin 的实现。一般情况下，你

要实现基类所有的纯虚函数，除了 CheckMediaType 进行媒体类型检查外，一般还有 DecideBufferSize 以决定 Sample 使用内存的大小，GetMediaType 提供支持的媒体类型。最后，我们看一下 Filter 类的实现。首先当然也要实现基类的所有纯虚函数。除此之外，Filter 还要实现 CreateInstance 以提供 COM 的入口，实现 NonDelegatingQueryInterface 以暴露支持的接口。如果我们创建了自定义的输入、输出 Pin，一般我们还要重载 GetPinCount 和 GetPin 两个函数。

Filter 框架的实现大致就是这样。你或许还想知道怎样在 Filter 上实现一个自定义的接口，以及怎么实现 Filter 的属性页等等。这些问题都能在 SDK 的示例项目中找到答案。其他的，关于在实际编程中应该注意的一些问题，笔者整理了一下，供大家参考。

1. 锁（Lock）问题

DirectShow 应用程序至少包含有两条线程：一条主线程和一条数据传输线程。既然是多线程，肯定会碰到线程同步的问题。Filter 有两种锁：Filter 对象锁和数据流锁。

Filter 对象锁用于 Filter 级别的如 Filter 状态转换、BeginFlush、EndFlush 等；数据流锁用于数据处理线程内，比如 Receive、EndOfStream 等。如果这两种锁没有搞清楚，很容易产生程序的死锁，这一点特别需要提醒。

2. EndOfStream 问题

当 Filter 接收到这个“消息”，意味着上一级 Filter 的数据都已经发送完毕。在这之后，如果 Receive 再有数据接收，也不应该去理睬它。如果 Filter 对输入 Pin 上的数据进行了缓存，在接收到 EndOfStream 后应确保所有缓存的数据都已经处理过了才能返回。

3. Media Seeking 问题

一般情况下，你只需要在 Filter 的输出 Pin 上实现 NonDelegatingQueryInterface 方法，当用户申请得到 IID_ImediaPosition 接口或 IID_IMediaSeeking 接口时将请求往上一级 Filter 的输出 Pin 上传递。当 Filter Graph 进行 Mediaseeking 的时候，一般会调用 Filter 上的 BeginFlush、EndFlush 和 NewSegment。如果你的 Filter 对数据进行了缓存，你就要重载它们，并做出相应的处理。如果你的 Filter 负责给发送出去的 Sample 打时间戳，那么，在 Mediaseeking 之后应该重新从零开始打起。

4. 关于使用专门的线程

如果使用了专门的线程进行数据的处理和发送，你需要特别小心，不要让线程进行死循环，并且要让线程处理函数能够去时时检查线程命令。应该确保在 Filter 结束工作的时候，线程也能正常地结束。有时候，你把 GraphEdit 程序关掉，但 GraphEdit 进程仍在内存中，往往就是因为数据线程没有安全关闭这个原因。

5. 如何从媒体类型中获取信息

比如，你想在输入 Pin 连接的媒体类型中，获取视频图像的宽、高等信息，你应该在输

入 Pin 的 CompleteConnect 方法中实现，而不要在 SetMediaType 中。

DirectX 媒体对象（DirectX Media Objects，简称 DMOs），是微软提供的另一种流数据处理 COM 组件。与 DirectShow filter 相比，DMO 有很多相似之处。对 filter 原理的熟悉，将会大大帮助你对 DMO 的学习。另外，DMO 也因其结构简单、易于创建和使用而倍受微软推崇。