程序和功能驱动程序。设备的安装程序负责建立这些注册表里的表项，驱动程序的安装，是根据 INF 文件中的指令进行的。注册表中的表项指明了各种驱动程序在数据对象堆栈中的位置，于是 PnP 管理器开始装载最低层的过滤驱动程序，并调用该驱动程序的 AddDevice 函数。该函数在数据对象堆栈中建立一个 FiDO，同时也将前面建立的 PDO 和这个 FiDO 联系在一起。PnP 管理器反复的实现该过程，装载其他位置靠上的低层过滤驱动程序、功能驱动程序、上层过滤驱动程序，直到该堆栈完成。

应用程序对设备的存取通过提交 IO 请求包（IRP）来进行。在操作系统中，对设备的存取过程是这样的：操作系统中的 I / O 管理器接受 I / O 请求（通常是由用户态的应用程序发出的），建立相应的 IRP 来描述它，将 IRP 发送给合适的驱动程序，然后跟踪执行过程，当操作完成后，将返回的状态通知请求的发起者。操作系统中的 I / O 管理器、即插即用管理器、电源管理器都使用 IRP 来与内核模式驱动程序、WDM 驱动程序进行通信，并且，各驱动程序之间的通信也是依靠 IRP。

在 WDM 驱动程序中，IRP 首先从最上层进入，如图 3-1 里右手边的箭头，然后，依次往下传送。在每一层，驱动程序自行决定对 IRP 的处理。有时，一个驱动程序除了把继续 IRP 向下传递外，并不做任何事情。有时，一个驱动程序会完全接管 IRP，不再把它向下传递了。当然，一个驱动程序也可以处理 IRP 后，再把它继续向下传递。这取决于驱动程序的功能和 IRP 的含义。

微软在 WDM 模型中使用分层的驱动程序结构，通过分层的方法，在处理对设备的 I / O 请求时，利用添加合适的驱动程序层的方法，从而非常灵活的改变设备的行为，以实现不同设备的功能。

3.2.2 WDM 驱动程序模块组成

一个 WDM 设备驱动程序的模块组成（如图 3-2 所示）主要包括：初始化自己、创建和删除设备、处理 Win32 打开和关闭文件句柄的请求、处理 Win32 系统的 I/O 请求、对设备的串行化访问、访问硬件、调用其它驱动程序、取消 I/O 请求、超时 I/O 请求、处理一个 PnP 设备被加入或删除的情况、处理电源管理请求、调用 Windows 管理诊断 WMI 向系统管理员报告。其中，“初始化”模块必不可少，所有驱动程序都需要通过分发例程处理用户的 I/O 请求。而 WDM 风格

的设备驱动程序需要有“即插即用（PnP）”模块以及安装标识的 INF 文件。在上述不同模块之间需要交互，其中的一些交互可以直接通过函数调用，而大量信息需要通过数据结构传递（比如在"设备对象"之类的数据结构中可以存储每个设备的信息）。

IRP，每个设备对象有一个内部的 IRP 队列，驱动程序的分发例程把 IRP 插入到这个设备队列中，内核 I/O 管理器从该队列一个个取出 IRP，并传递到驱动程序的 StartIo 例程，StartIo 例程串行处理 IRP，以确保不与其它 IRP 处理程序冲突，但是 StartIo 例程仍然需要通过临界段例程避免与硬件中断发生冲突。

中断是用于停止处理器对一个任务的执行，而被强制运行某个中断去处理代码。中断包括软件中断和硬件中断。中断具有优先级，低优先级中断会被高优先级的中断所中断，以保证重要任务会优先执行。硬件产生的中断总是比软件产生的中断优先级要高。硬件中断类型包括：设备中断请求级处理程序的执行、配置文件定时器、时钟、同步器、处理器之间中断级、电源故障级；软件中断包括异步过程调用执行、线程调度、延迟过程调用执行。至于常规线程执行则没有中断。

图 3-2 WDM 设备驱动程序的模块组

驱动程序的主要的初始化入口点是一个称为 DriverEntry 的例程。多数 WDM 设备对象是由 PnP 管理器调用 AddDevice 入口点创建，该例程在插入新设备和安装 INF 文件指示相对应的驱动程序时被调用，之后，一系列 PnP IRP 被发送到

驱动程序，指示设备应何时启动和查询其功能。为了让应用程序认知驱动设备存在，开发人员必须为每个设备对象创建使 Win32 可见的符号链接，这时可以有两种实现途径：一种是，采用显式的"硬编码"符号链接名，对此，应用程序的源代码中也具有设备名的硬编码；还有一种是采用设备接口，每个设备接口由一个 GUID（全局唯一标识符）标识，对设备注册为具有某个特定的设备接口时就创建了一个符号链接。用户程序可以搜索有特定 GUID 的所有设备。驱动程序的代码是由系统内核通过发送 I/O 的 IRP 请求运行的。IRP 是设备驱动程序的基本结构。

内核所调用的驱动程序中的其它多个例程就是所谓回调例程（Callback），每个回调例程具有一个标准的函数原型，它适用于对应的环境。常用的回调例程包括：Unload（卸载驱动程序）、AddDevice（添加一个新的 PnP 设备）、StartIo（串行处理 IRP）、ISR（中断服务例程）、DpcForIsr（延时过程调用例程）、临界段例程、撤消 IRP 的 Cancel 例程、一个低层驱动程序完成一个 IRP 处理时所调用的 Completion 例程、当 DMA 通道可用时调用的 AdapterControl、Timer（秒级定时器回调例程）、低于 1 秒的超时例程 CustomTimerDpc、用于处理工作队列的 CustomDpc、Reinitialize（用于初始化耗时很长的驱动程序）。

硬件驱动程序应当具有容错性，比如连接电缆断开或缓存区过载，并能够将错误信息反馈到应用程序。可以检查硬件状态位（如打印机缺纸指示），并能够处理快速 I/O 事件。

与其它 Windows 编程类似的是，在编写硬件驱动程序时如何分配或访问内存是一件很讲究细节的技术活。我们知道，Windows 中采用的虚拟内存意味着系统可以使用比物理内存更多的内存。虚拟内存的实现方式是：将每个应用程序的可能地址空间划分成固定大小的块，这些块称为页（x86 处理器的页大小为 4KB，Alpha 处理器的页为 8KB）。页可以驻留在物理内存，也可交换到硬盘。设备驱动程序分配的内存通常有两种：可以是交换型的分页内存，也可以是永久驻留的非分页内存。如果试图在线程调度或更高中断级访问分页内存，就会引起缺页故障，造成内核崩溃。即使是常规线程，如果访问非驻留的分页内存，内核就会造成线程阻塞，直到内存管理器把内存装回内存。问题的关键是，在设计驱动程序时，一方面忌讳滥用非分页内存，另一方面却不得不经常使用非分页内存。因为

要在线程调度或更高中断级访问内存，必须调用非分页内存。当完成内存调用后，应当对所有类型的内存通过 ExFreePool 释放。假如不释放内存，将会减少内存，因为即使当驱动程序卸载后，内核并不收集这些分配了的内存。

没有编写过硬件驱动程序的开发人员，在听到硬件驱动程序编程时，他们从直觉上会联想到需要与硬件资源打交道，包括 I/O 端口、中断、存储器地址和 DMA（直接存储器访问）。其实，这是一种误解，因为有很多驱动程序不需要直接调用任何低层硬件资源，比如 USB 客户端驱动程序就不需要任何硬件资源，所有与硬件相关的琐细工作都有 USB 总线驱动程序完成，USB 客户端驱动程序只是对总线驱动程序发出请求。

一个 WDM 驱动程序的功能模块可由以下几个部分组成：

·驱动程序初始化。

·创建和删除设备。

·处理 Win32 程序打开和关闭句柄的请求。

·处理 Win32 程序输入/输出请求。

·实现对设备的串行化访问。

·访问硬件。

·取消 I/O 请求。

·超时 I/O 请求。

·调用其他驱动程序。

·处理电源管理请求。

·使用 Windows 管理诊断(WMI)向系统管理员报告。

·处理一个可热插拔的设备被加入或删除的情况。

WDM 驱动程序有一个主要的初始化入口点，即一个称为 DriverEntry 的例程,它有一个标准的函数原型，当 WDM 驱动程序被装入时，内核调用 DriverEntry 例程。所有对各种 IRP 的处理例程都在此入口函数中做出定义。大多数的 WDM 设备对象，都是在即插即用管理器调用 AddDevice 例程入口点被创建的。插入新设备后，当系统找到由安装信息所指示的驱动程序时，这个例程调用在此之后，一系列的即插即用 IRP 被发送到驱动程序，设备驱动程序可进行相应的功能处理。

WDM 支持的驱动程序具有分层结构，换言之，对于一种设备而言，它可以具有三种类型的 drivers：总线 driver，或者函数 driver，或者过滤式驱动程序（它可以假定或修正设备的行为值）。为一台设备服务的这些驱动程序链就是所谓驱动程序栈。一个驱动程序栈分阶段地处理用户的请求，这些驱动程序一个个相互叠加在一起，低层的总线驱动程序可用于处理与硬件的所有基本联系，而中间的类驱动程序对整个一类驱动程序提供共同的设施。

微软提供了针对 Windows 的总线驱动程序，并为第三方设备开发商提供有关服务，如枚举设备、对即插即用和 I/O 所需电源的管理，并提供了独立于设备管理方式。设备开发商更多地是提供函数式驱动，其基本内容包括：对设备的操作界面，对设备的读写句柄，对设备电源的管理策略。过滤式驱动程序安装在驱动栈一个或多个设备之上或下端，它可以截获设备、或设备类、或总线的请求，判断这些请求，并可以修改其内容或对其进行响应，例如 USB 键盘的高层过滤驱动程序可以增强加密检查，而适用于鼠标的低层类过滤有助于提高鼠标性能。

函数式驱动是这样一种结构：属于某类设备的常规执行可以通过其类驱动实现，即是说驱动程序在开发时，开发人员的工作只需要写出非常少的驱动代码 minidriver 去与硬件打交道，大部分工作可以通过调用类驱动完成。微软提供的类驱动可以实现常见的系统任务，比如即插即用和电源管理。WDM class drivers 主要内容包括：

（1）流式类驱动，以内核模式支持多媒体内容；

（2）具有支持输入设备的 HID 类驱动；

（3）USB 和 IEEE 1394 总线类驱动；

（4）支持串行和并行方式的存储协议。

Windows 支持 WDM 驱动的各个系统内核组件包括：

（1）Kernel 组件，指基本的同步、性能计算和及时、延缓与 IRQ 控制；

（2）Object Manager 组件，对象说明；

（3）Executive 执行组件，内存分配、互锁及列项操作；

（4）I/O 管理组件，包括 I/O IRP（Request Packet）控制，设备对象，工作项目，注册表访问，系统状态提示，DMA 及中断；

（5）内存管理，虚拟到物理内存映像，物理内存托管和锁定，驱动程序映

像内存锁定，机动 I/O 空间；

（6）处理服务，系统线程生成和删除；

（7）Run-time Library，大容量外存，Unicode 和数据类型转换；

（8）电源管理，电源状态改变，电源 IRP 控制，设备空闲检测；

（9）即插即用子系统，硬件检测和资源分配，PnP（Plug and Play） IRP 控制以及硬件事件；

（10）WMI(Windows Management Instrumentation),用于支持设备测试以及检测指示数据的支撑结构；

（11）内核式流，是连接流数据设备的支撑结构；

（12）硬件提取层，提取平台，访问和调用 I/O 端口及内存映像设备。

电源管理可以是系统级或设备级，前者可以请求整个系统关闭。系统电源有六种状态：不可完全开启、完全关闭、三种休闲状态和一种休眠状态。设备级电源管理则有四种状态：完全开启、完全关闭，加两种休眠状态。一个设备可自行关闭，即使系统其它部分正全速运行。

3.2.3 WDM 的版本问题

虽然 Windows XP/2000/98/Me 都支持 WDM，但是由于历史原因，不同版本 WDM 内容并不相同。当然，新版 WDM 都是旧版 WDM 的超集。跨系统使用的 WDM driver 通常采用 IoIsWdmVersionAvailable 例程去判定当前运行系统支持 WDM 的版本号。按照常理说，保证跨平台兼容性的最简单的方式应该是：写一个驱动程序时仅包括最低版本 WDM 所支持的那些功能。但是，这种思路往往行不通，因为驱动程序不仅要适应不同的 OS，而且还应当具有发挥具体系统特色优势的附加代码。

最新编写出的内核模式的驱动程序应该属于 WDM 类型，其开发平台则应为 Windows XP。任何 WDM drivers 都必须支持 PnP、电源管理，并能执行 WMI。一般的原则是，即使是用于更低版本的驱动程序，也最好首先在 Windows XP 下开发然后做移植，这种策略同样适用于那些并不完全适合 WDM 模式的硬件设备。如果要写出 WDM 驱动程序，开发人员必须通过最新的 Windows DDK 了解不同 Windows 平台的差异，以及总线和设备的相关问题。比如：其一，不同 Windows 平台的驱动程序代码执行会不同，主要由于 WDM 兼容了 Windows XP/ 2000/ 98/Me 中的不同系统结构。在其中一个平台工作正常的 driver，到其它平台时需要全面测试，