[19] 中华人民共和国国家知识产权局

[51] Int. Cl.
G06F 13/10 (2006.01)
G06F 9/44 (2006.01)

[12] 发明专利申请公开说明书

[21] 申请号 200510089608.7

[43] 公开日 2006 年 1 月 4 日

[11] 公开号 CN 1716225A

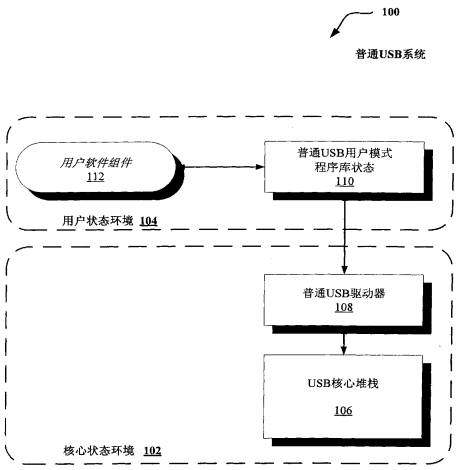
[22] 申请日 2005.4.29
[21] 申请号 200510089608.7
[30] 优先权
[32] 2004. 4.29 [33] US [31] 10/837,444
[71] 申请人 微软公司
地址 美国华盛顿州
[72] 发明人 F·K·本赫萨尼亚 R·E·奥尔
[74] 专利代理机构 上海专利商标事务所有限公司
代理人 沈昭坤
权利要求书 3 页 说明书 35 页 附图 6 页

[54] 发明名称

普通 USB 驱动器

[57] 摘要

这里公开的技术提供一种普通 USB 驱动器。特别是，描述了一种普通 USB 设备驱动器结构，其通过能访问普通核心状态驱动器的用户状态 USB 程序库来促进开发。 在所述的一种实施方式中，所述方法包括响应于设备的插入事件而加载核心状态普通设备驱动器。 用户软件组件通过使用由普通用户状态程序库(例如动态链接库(DLL))提供的多个例程来访问设备，所述普通用户状态程序库与普通设备驱动器进行通信。



1、一种方法，包括：

响应于设备的插入事件而加载核心状态普通设备驱动器；和

用户软件组件通过使用多个由普通用户状态程序库提供的例程来访问设备，所述普通用户状态程序库可通信地耦合到普通设备驱动器，

其中为每个插入设备或者为该设备支持的每项功能来加载单个普通设备驱动器。

2、如权利要求1所述的方法，其中多个例程作为应用编程接口（API）来实施。

3、如权利要求1所述的方法，其中该设备是通用串行总线（USB）设备。

4、如权利要求1所述的方法，其中插入事件由即插即用（PNP）模块来检测。

5、如权利要求1所述的方法，进一步包括加载与提取的设备ID相对应的设备驱动器，所述提取的设备ID对应于插入设备，其中普通设备驱动器简化了加载的设备驱动器与用户软件组件之间的通信。

6、如权利要求5所述的方法，其中所述设备ID由通过数据通信总线耦合到插入设备的核心堆栈来提取。

7、一种方法，包括：

响应于插入事件而提取设备ID；

加载与所提取的设备ID相对应的设备驱动器；和

加载普通设备驱动器来简化所加载的设备驱动器与用户软件组件之间的通信。

8、如权利要求7所述的方法，其中为每个设备ID或者为与所述设备ID相对应的设备所支持的每项功能来加载单个普通设备驱动器。

9、如权利要求7所述的方法，其中所述设备ID标识通用串行总线（USB）设备。

10、如权利要求7所述的方法，其中由核心堆栈执行所述提取。

11、如权利要求7所述的方法，其中由核心堆栈执行所述提取，并且核心堆栈耦合到数据通信总线以与多个设备进行通信。

12、如权利要求 7 所述的方法，其中由核心堆栈执行所述提取，并且至少一个用户软件组件与多个设备通过耦合到核心堆栈的集线器来通信。

13、如权利要求 7 所述的方法，其中在核心状态中加载普通设备驱动器。

14、如权利要求 7 所述的方法，其中用户软件组件置于用户状态。

15、如权利要求 7 所述的方法，其中插入事件由即插即用（PNP）模块来检测。

16、如权利要求 7 所述的方法，其中插入事件由即插即用（PNP）模块来检测，并且所述 PNP 模块将提取的设备 ID 与设备驱动器相匹配。

17、如权利要求 7 所述的方法，进一步包括注册与提取的设备 ID 相对应的唯一 ID。

18、如权利要求 7 所述的方法，其中进一步包括访问与提取的设备 ID 相对应的设备。

19、如权利要求 7 所述的方法，进一步包括用户软件组件通过普通用户状态程序库与普通设备驱动器进行通信。

20、如权利要求 7 所述的方法，进一步包括用户软件组件通过普通用户状态程序库与普通设备驱动器进行通信，所述用户状态程序库包括多个例程来简化与所述设备 ID 相对应的设备之间的通信。

21、如权利要求 20 所述的方法，其中所述例程作为应用编程接口（API）来实施。

22、一种装置，包括：

计算设备；和

耦合到计算设备的设备，

其中响应于将设备耦合到计算设备来加载核心状态普通设备驱动器，并且用户软件组件通过使用普通用户状态程序库提供的多个例程来访问所述设备，所述用户状态程序库可通信地耦合到普通设备驱动器，其中为每个设备或者为该设备支持的每项功能来加载单个普通设备驱动器。

23、如权利要求 22 所述的装置，其中所述多个例程作为应用编程接口（API）来实施。

24、如权利要求 22 所述的装置，其中所述设备是通用串行总线（USB）设备。

25、一个或多个在其上存储有指令的计算机可读介质，当执行所述指令时，引导机器执行的操作包括：

响应于设备的插入事件而加载核心状态普通设备驱动器；和

用户软件组件通过使用由普通用户状态程序库提供的多个例程来访问设备，所述普通用户状态程序库可通信地耦合到普通设备驱动器，

其中为每个插入设备或者为该设备支持的每项功能来加载单个普通设备驱动器。

26、如权利要求25所述的一个或多个计算机可读介质，其中所述多个例程作为应用编程接口（API）来实施。

27、如权利要求25所述的一个或多个计算机可读介质，其中所述设备是通用串行总线（USB）设备。

28、如权利要求25所述的一个或多个计算机可读介质，其中插入事件由即插即用（PNP）模块来检测。

29、如权利要求25所述的一个或多个计算机可读介质，进一步包括加载与提取的设备ID相对应的设备驱动器，所述提取的设备ID对应于插入设备，其中普通设备驱动器简化了加载的设备驱动器与用户软件组件之间的通信。

普通USB驱动器

版权说明

技术领域

本发明一般涉及数字通信，尤其是普通通用串行总线（USB）驱动器。

背景技术

由于计算机变得越来越普及，很多功能都不断并入到单个的计算机系统中。通常，通过给计算机系统添加外部设备来提供附加功能。换句话说，外部设备与计算机系统通信以提供附加功能。

通用串行总线（USB）已经成为将外部设备连接到计算机系统的标准通信信道。USB 允许在提供相对大的带宽的情况下同时将多个设备连接到同一个计算机系统上。例如，USB 的最新通用版本（如 USB2.0）的传输速率可高达480Mbps（兆比特/秒）。

提供USB设备的商家通常也要提供设备的驱动器使其能够访问这些设备。一般来说，所述设备驱动器是一个程序和一组数据，其允许计算机系统访问所述设备（例如从所述设备中读取数据、向所述设备写入数据、向所述设备发送命令、以及从所述设备接收状态数据）。

目前，大多数USB设备商家要为他们的设备提供核心状态设备驱动器。一般来说，所述核心是操作系统（OS）的中心部分，所述核心在OS运行时保持有效并且控制计算机系统。例如，所述核心向计算机系统提供必要的服务（诸如存储和进程管理）。因此，核心状态驱动器在很小的故障（或错误）中就容易引起不可恢复的整个系统的毁坏。另外，要实现核心状态的USB驱动器除了测试核心状态驱动器所需的额外硬件或软件外，还需要在编程复杂的核心状态驱动器上花费时间和精力。而且，当升级OS时，不兼容的或过时的核心状态设备驱动器会引起毁坏并阻碍系统的成功升级。

发明概述

本发明公开了提供普通USB驱动器的技术。尤其是，描述了一种普通USB设备驱动器的结构，它可以促进与普通核心状态驱动器通信的用户状态USB程序库的开发。

在所述的实施方式中，一种方法包括响应于设备的插入事件而加载核心状态普通设备驱动器。用户软件组件通过使用由与普通设备驱动器通信的普通用户状态程序库（如动态链接库（DLL））提供的例程来访问所述设备。

在另一个实施方式中，为每一个插入设备或为所述设备支持的每项功能分别加载单独的普通设备驱动器。

在一些实施方式中，提供一些作为计算机程序产品的生产商品。其中计算机程序产品的一种实施方式提供一种计算机系统可读的计算机程序存储介质和对计算机程序进行编码。另一种计算机程序产品的实施方式可以通过计算机系统和对计算机程序进行编码在载波中配备的计算机数据信号中提供。

而且，计算机程序产品为在计算机系统上执行的计算机进程而编码计算机程序。计算机进程响应于设备的插入事件（可以由即插即用模块检测）来加载核心状态普通设备驱动器。用户软件组件通过使用普通用户状态程序库（如动态连接库（DLL））提供的例程来访问该设备。

在这里还将描述和解释其它的实施方法。

附图的简要说明

下面将参照附图进行详细的描述。在附图中，附图标记最左边的数字表示该附图标记第一次出现的附图号。在不同的附图中使用同样的附图标记来表示相同或相似的部分。

图1表示普通设备驱动器的示意性系统。

图2表示具有用于多个设备的普通设备驱动器的多个实例的示意性系统。

图3表示使用普通设备驱动器的示意性方法。

图4表示处理普通设备驱动器中的I/O的示意性方法。

图5表示用于在普通设备驱动器中提供电源管理的示意性方法。

图6表示一种通用计算机环境，该环境可用于实现这里描述的技术。

详细描述

下面公开的内容描述了提供普通设备驱动器结构的技术。尤其是，公开了

提供普通USB驱动器的技术。特别是，所述结构使用能访问普通核心状态驱动器的用户状态USB程序库。所述结构在一定程度上减少了设备驱动器的开发时间，改善了终端用户的体验（例如不管是在正常操作期间或是升级过程中都限制系统范围的毁坏），并且使测试和调试集中化。所述结构还可用于改善输入—输出（I/O）处理、安全性和电源管理，正如这里即将讨论的。而且，这里讨论的技术还可以用软件、固化软件、硬件和/或它们的组合来实施。

普通驱动器概述

图1表示普通设备驱动器的示意性系统100。图1和2中箭头的方向表示根据所述实施方式的数据流的方向。系统100包括核心状态环境102和用户状态环境104。核心状态环境102包括USB核心堆栈106，其可以是OS的一部分并且可以与USB设备在硬件层通信。USB核心堆栈106可以简化集线器功能，如下面将要参照图2讨论的。USB核心堆栈106耦合到普通USB驱动器108。普通USB驱动器108管理该USB设备的I/O，并且通常都将USB设备设为用户状态环境104。

用户状态环境104包括有普通USB用户状态程序库110和用户软件组件112。相应地，用户软件组件112在一个实施方式中是用户状态软件组件。普通USB用户状态程序库110提供一些例程（或应用程序编程接口（API）），USB用户软件组件112可以用所述例程来与USB设备通信或控制USB设备。在一个实施方式中，普通USB用户状态程序库110是DLL。因此，USB用户软件组件112可以通过在用户状态中而不是核心状态中调用所述例程来从用户状态环境104管理USB设备。下面将在同一标题下进一步讨论多个示意性的API。

设想这样一种实施方式来减少系统范围的毁坏，因为如果USB用户软件组件112（或普通USB用户状态程序库110提供的例程）毁坏了，核心的操作会保持不变。另外，这样一种系统（100）减少了设备驱动器的开发时间并使测试和调试集中化（一定程度上来说，由于只需要一个计算机系统来测试USB接口，也就是说，即使当用户状态软件组件毁坏时，系统也会保持运行）。

在一个实施方式中，将与USB核心堆栈106和普通USB驱动器108之间的通信接口对应的数据以及与USB用户软件组件112和普通USB用户状态程序库110之间的接口有关的信息向开发者公开或公开有效。在一个实施方式中，不需要公开与普通USB驱动器108和普通USB用户状态程序库110之间的接

口有关的信息。

用于多个设备的普通驱动器

图 2 描述了具有用于多个设备的普通设备驱动器的多个实例的示意性系统 200。该系统 200 包括核心状态环境 102、用户状态环境 104、USB 核心堆栈 106、普通 USB 用户状态程序库 110 和 USB 用户软件组件 112。

系统 200 包括用于交换数据的 USB 总线 202（将在下面参照图 6 进一步讨论）。所述 USB 总线只是一个例子，也可以在系统 200 中使用其它类型的数据通信总线（例如有线或无线通信总线，如蓝牙、小型计算机系统接口（SCSI）等）。

如图 2 所示，USB 总线 202 耦合到 USB 核心堆栈 106（如图 1 中所示）。USB 核心堆栈 106 依次耦合到物理设备目标（PDO）来简化与 USB 集线器 206 之间的通信。USB 集线器 206 允许在核心状态环境 102 中处理多个 USB 设备。因此，假设普通 USB 驱动器（108）可以耦合到核心状态环境 102 和用户状态环境 104 之间。

如图 2 所示，可以使用多个 PDO 来简化与各种驱动器（如普通 USB 驱动器 108 和 USB 综合驱动器 208）之间的通信。用户环境 104 可以通过全局唯一标识符（GUID）210 发现和/或识别的功能来与每个普通 USB 驱动器（108）进行通信。每个 GUID210 唯一表示其 USB 接口/功能。在一个示意性实施方式中，OS 注册文件和/或驱动文件（如信息文件（INF））中的 GUID 密钥的格式，例如 Washington 的 Redmond 的微软公司可用的 Window OS，可以如下所示：HKR,,DeviceInterfaceGUID,,"{058815B2-9805-47d3-B7D5-ABC464D3CA06}"。

因此，每个 GUID210 标识 USB 用户状态软件组件 112 所使用的相应设备支持的功能。例如，用户状态软件组件将搜索一些驱动器，这些驱动器公开其相关的 GUID 来寻找支持所需功能的设备。然后 USB 用户软件组件 112 可以通过图 1 中所讨论的普通 USB 用户状态程序库 110 来与 USB 设备进行通信。

假设可以使用多个 USB 集线器（206）来允许与多个 USB 设备或合并到单个 USB 设备中的多个功能进行通信。例如，可以使用 USB 集线器（206）来允许访问在除了具有击键按钮或集成点击设备（如操纵杆和/或触摸垫）外，还具有多媒体命令按钮的键盘中的不同功能。

普通设备驱动器操作

图 3 表示使用普通设备驱动器的示意性方法 300。只要即插即用（PNP）模块（如内置在 OS 中）检测到有插入事件（302）出现，核心堆栈（如图 1 和 2 中的 106）就提取对应于插入设备的设备 ID。所提取的设备 ID 由 PNP 模块（306）提供或检测，并且 PNP 模块将提取的设备 ID 与普通驱动器（308）（如图 2 中的 108）相匹配。

加载匹配的普通驱动器（如图 1 和 2 中的 108）以简化阶段 310 的加载设备驱动器与用户软件组件（如图 1 和 2 中的 112）之间的通信。在阶段 312 中，注册一个唯一 ID（例如参照图 2 的 210 所讨论的）。最后用户软件组件可以通过普通设备驱动器来访问插入设备（314）。

在一个实施方式中，假设 PNP 模块将提取的设备 ID 与普通设备驱动器（例如，这里的普通设备驱动器兼容所述插入设备）相匹配，而不是其它的设备 ID。这就依次消除了加载任何其它的驱动器的需要。因此，在一个实施方式中，响应于设备的插入事件来加载核心状态普通设备驱动器并且用户软件组件通过使用普通用户状态程序库提供的例程来访问所述设备。普通用户状态程序库可通信地耦合到普通设备驱动器。所述例程可以作为 API 来实施（这将在下面的标题“示意性 API”下作进一步讨论）。而且，可以为每一个插入设备（或该设备所支持的功能）分别加载单独的普通设备驱动器。

普通设备驱动器中的 I/O 处理

图 4 表示在普通设备驱动器中处理 I/O 的示意性方法 400。在一个实施方式中，所述方法 400 使得用户软件组件（如图 1 和 2 中的 112）通过启用几个例程将具体硬件 I/O 处理留给普通设备驱动器（如图 1 和 2 中的 110）处理。例如，用户软件组件可以简单地发布由普通 USB 用户状态程序库（如图 1 中的 110）提供的初始化、读/写、和释放命令，并且将具体数据长度考虑、数据碎片和/或数据重新组合留给普通设备驱动器处理。

例如，所述初始化可以将普通 USB 设备驱动器的状态和性能重新设置成默认值并且将普通 USB 设备驱动器准备成处理进一步的操作。所述读取可以从相应设备上的指定端点读取数据，并且所述写入可以向相应设备上的指定端点写入数据。所述释放可以在初始化阶段开始的会话期间清除所有要求的资源。

对于数据长度考虑，USB 输入（也就是，参照图 5 所讨论的从所述设备向主机或计算系统输入数据）端点每次可以从 0 字节到所述端点指定的最大包大

小传送到任何地方，所以需要准备主机系统上的软件来为传送中的每个包进行接收，直到那个最大大小（其中所述传送由多个数据包构成）。如果用户状态软件组件（如图1和2中的112）请求了不是多个这种最大包大小的传送，那么所述普通USB设备驱动器可以自动地调整这个传送的大小。因此，如果用户软件组件请求的数据传送的长度不足以处理所有来自该设备的可能的数据传送大小，那么就增加数据传送的长度。这可以通过将客户端（用户状态软件组件的客户端）的缓冲器分成至少两个单独的传送（这也叫作“分段”）来实现。第一传送大小可以是小于客户端请求的传送大小的最大包大小（这里指“最大数据包”）的最大倍数。用于该第一传送的缓冲器是客户端缓冲器的第一 N*最大包（其中这个值是这个第一子传送的大小），所以不需要存储复制。第二传送是最大包字节。用于这个第二传送的缓冲器是一个在该实施方式中新分配的缓冲器。一旦两个传送都完成了，就将来自第二传送的数据复制到客户端的缓冲器中直到每个客户端缓冲器都满了，或者接到一个短包。如果在复制完成之后有任何来自第二传送的数据遗漏，则基于所述端点的管道安全性，可以保存或不保存下一个传送请求。还假设如果在第一传送中有短包，那么就不会发送第二传送。

而且，USB主机控制器与USB核心堆栈（如图1和2中的106）一起经常只处理传送缓冲器直到指定的大小。这个大小对于不同的主机控制器实施方式来说是不一样的。普通USB设备驱动器考虑这些大小限制，并且如果客户端的传送请求超过了这个限制，所述传送就被分成两个或多个传送，其中所有的传送都小于或等于这个限制大小。在一个实施方式中，由于每个子传送的缓冲器只是客户端原始缓冲器的一部分，所以不需要缓冲器复制。

如图4所示，方法400在阶段402初始化，如通过定义用来访问设备的数据结构。初始化阶段可以将安全设置重新设置成默认值（这将在下面参照示意性API作进一步讨论，诸如WinUsb_Initialize）。这个初始化阶段使普通用户状态程序库（如与图1和2中的110相关的DLL）在一个实施方式中运行。

所述方法400定义管道安全（404），所述管道安全可以定义普通设备驱动器的性能，其包括超时、包末端、从错误中自动恢复、包终止规则、短包处理、部分读取，例如用于读取真实数据之前的包头来预测要发送的数据的大小、丢弃不相关的多余数据、和/或原始数据处理（这表示用户软件组件保证正确的数据处理），这将在下面参照示意性API作进一步讨论，例如

WinUsb_SetPipePolicy。在一个实施方式中，每个端点在普通 USB 驱动器中都可以有单独配置的管道。

这些示意性管道的一些性能设置将在下面参照各种实施方式作进一步讨论：

• 超时-可以对指定的端点设置这个管道安全性，并且为所述端点取消所有的在指定时间内没有完成的传送。

• 包终止规则 — 一些 USB 设备协议需要主机计算系统用短包（小于所述端点指定的门限值最大包大小的包）来标记输出传送（从主机到设备）的末端。如果客户端的传送是多个最大包大小，那么就发送 0 长度包。这可以对用户软件组件（或客户端）自动实施，以便客户端不需要担心最大包大小，或者该传送是否是多个这种大小。

• 从错误中自动恢复 — 如果对于指定端点的指定传送在 USB 总线上出现传送错误，自动恢复机制就试图清除这种错误情况，以便允许发布进一步的传送。还可以假设错误情况是不可恢复的（也就是说，不会出现进一步的传送）。

• 短包处理 — 通常，当从 USB 设备上的指定端点接收到短包时，USB 核心堆栈（如图 1 和 2 中的 106）完成所述传送。有时用户软件组件不打算使用短包来表示传送完成，而宁愿保持从设备进行读取直到接收到指定数量的字节。在一个实施方式中，这可以通过将普通 USB 设备驱动器配置成忽略所述事件并且保持读取数据来进行处理。这可以通过重新发布指向一个字节到最后接收到的字节的缓冲器的过早完成的传送来实现。

• 部分读取 — 一些设备协议包括位于传送的开始位置的包头，所述包头包括与数据有关的信息，包括大小信息、传送类型等。一些客户端可能希望在决定怎样处理所述传送的剩余部分之前仅读取这个包头。为了解决这个问题，客户端可以将相应管道（所述传送指向的管道）的性能设置成允许客户端只读取包头，然后普通 USB 设备驱动器可以保存在数据包的包头之后接收的任何其它数据，以便可以将它加在客户端后续传送的开始位置。

• 丢弃多余数据 — 如果设备在包中发送多于客户端请求的数据，客户端可以将所述端点的普通 USB 设备驱动器的性能配置成丢弃多余的数据，或者把它保存起来用于加在后续传送的开始位置。

• 原始数据 — 为了绕过普通 USB 设备驱动器的自动性能校正，例如，

为了提高与指定设备特性有关的性能，一些客户端宁愿自己管理传送限制。这些客户端可以配置所述端点的普通USB设备驱动器的性能，以便除了校正客户端的行为外，它还校正客户行为。因此，这种方式允许所有的循规蹈矩的客户端请求直接通过到核心USB堆栈。

在一个实施方式中，用户软件组件可以定义个先入先出（FIFO）缓冲器策略（406）（如对于指定管道（端点））并且从设备（408）开始编档FIFO，这将在下面参照示意性API作进一步讨论，如WinUsb_SetFifoPolicy和WinUSB_StarFifo。所述FIFO可以被读取（如通过启用下面要讨论的WinUsb_ReadFifo API）。在一个实施方式中，为每个管道提供一个FIFO。假设可以通过使用WinUsb_ReadPipe来读取数据。因此，不需要FIFO来读取数据。但是，FIFO可以用于设备的指定分类，所述设备可以在任意时间产生数据，或者例如连续流数据。

在阶段410中读取所述管道或向所述管道写入。而且，如下所述，其它API可用于处理I/O，所述I/O使得用户软件组件可以更好的控制I/O功能。例如，在一个实施方式中，如果设备用一个比客户端缓冲器大很多的包进行响应，所述数据就被加到下一个传送的开始位置（如果这个操作是可用的，例如通过Allow_Partial_Reads）。

示意性API

下面将描述一个或多个可用在Microsoft Windows®环境中用于提供参照图1和2所讨论的例程的示意性的API。例如，所述例程（如在图1和2中的普通USB用户状态程序库110中）可以支持下面的动作（将在下面作进一步讨论）：

- 查询USB描述符：设备、配置、接口和字符串；
- 列举与GUID设备接口分类有关的USB设备接口，并通过出让-供给调回例程对其进行过滤；
- 选择性地激活USB设备的一些接口并且保持其它接口未激活；
- 生成标准控制传送请求；和/或
- 发送控制、堆积、中断和同步数据。

下面讨论的示意性API由“WinUSB”开始，以表示它们符合Microsoft Windows® OS的普通USB设备驱动器。假设也使用其它命名约定。同时，下面会突出显示每个API的抽样呼叫、参数和返回值。

1. WinUsb_Initialize

所述 WinUsb_Initialize 函数对 WinUsb 数据结构进行初始化。注意，在调用这个函数时，策略设置被重新设置成默认值。一个调用该 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_Initialize(
IN HANDLE DeviceHandle,
OUT PWINUSB_INTERFACE_HANDLE InterfaceHandle
  );
```

Parameters

DeviceHandle

对设备的处理是通过例程返回来创建一个文件（如 CreateFile）。WinUsb 使用重复 I/O，所以应该在 CreateFile 调用中指定一个标记（如 FILE_FLAG_OVERLAPPED）。

InterfaceHandle

这是用于所有其它 WinUSb API 函数调用的接口处理。这是一种由 WinUSb 生成的不透明的处理。

Return Value

如果成功了，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

2. WinUsb_Free

WinUsb_Free 函数释放 WinUsb_Initialize 分配的所有资源。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_Free(
IN WINUSB_INTERFACE_HANDLE InterfaceHandle
);
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

Return Value

如果成功了，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

3. WinUsb_QueryAlternateInterface

WinUsb_QueryAlternateInterface 函数为特殊接口处理返回第一替换接口描述符。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_QueryAlternateInterface（
IN WINUSB_INTERFACE_HANDLE InterfaceHandle，
IN UCHAR AlternateInterfaceNumber,
OUT PUSB_INTERFACE_DESCRIPTOR
UsbAltInterfaceDescriptor
  );
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

AlternateInterfaceNumber

这是一个表示返回的替换接口的值。0 值表示第一替换接口，1 值表示第二替换接口，等等。

UsbAltInterfaceDescriptor

指向调用-分配 USB_INTERFACE_DESCRIPTOR 结构的指针。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

4. WinUsb_QueryDeviceInformation

WinUsb_QueryDeviceInformation 函数返回与 WINUSB 接口相连的物理设备有关的信息。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_QueryDeviceInformation（
IN WINUSB_INTERFACE_HANDLE InterfaceHandle，
IN ULONG Information Type,
```

IN OUT PULONG BufferLength,

OUT PVOID Buffer

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

InformationType

这是指定检索哪个接口信息值的值。

BufferLength

这是缓冲器的字节长度，或者要读取的字节的最大数量。这个参数可以设置成复制到缓冲器的字节的实际数量。

Buffer

这是一个接收请求值的调用-分配缓冲器。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

注意

下面的列表描述了可能的信息类型（InformationType）值。

DEVICE_SPEED（0x01）

这个请求会基于设备的速度返回下列值中的一个。

低速（LowSpeed）（0x01）

全速（FullSpeed）（0x02）

高速（HighSpeed）（0x03）

PHYSICAL_DEVICE_ID（0x02）

这个值包含设备的物理设备标识符。

5. WinUsb_GetDescriptor

WinUsb_GetDescriptor 函数返回一个请求的描述符。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_GetDescriptor(

IN WINUSB_INTERFACE_HANDLE InterfaceHandle,

IN UCHAR DescriptorType,

IN UCHAR Index,

IN USHORT LanguageID,

OUT PUCHAR Buffer,

IN ULONG BufferLength,

OUT PULONG LengthTransferred

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

DescriptorType

这是一个指定返回的描述符类型的值。对于标准值，可以参考 USB 规范，所述规范在 http://www.usb.org。

Index

这是描述符索引，其编录在 USB 规范中。

LanguageID

这是一个指定语言标识符的值，如果所请求的标识符是字符串标识符的话。

Buffer

这是一个接收所请求的描述符的调用-分配缓冲器。

BufferLength

这是缓冲器的字节长度，或者要读取的字节的最大数量。

LengthTransferred

这个接收复制到缓冲器中的字节的实际数量。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

6. WinUsb_SetCurrentAlternateInterface

WinUsb_SetCurrentAlternateInterface 函数为一个接口选择指定的替换接口。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_SetCurrentAlternateInterface(

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

IN UCHAR InterfaceNumber

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

InterfaceNumber

这是一个包含在 PUSB_INTERFACE_DESCRIPTOR 结构的 bInterfaceNumber 项中的值，其由 WinUsb_QueryAlternateInterface 来填充。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

7. WinUsb_GetCurrentAlternateInterface

WinUsb_GetCurrentAlternateInterface 函数可以获得为一个接口设置的当前替换接口。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_GetCurrentAlternateInterface(

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

OUT PUCHAR InterfaceNumber

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

InterfaceNumber

这是一个指向能接收当前选择的替换接口的 UCHAR 的指针。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

8. WinUsb_QueryPipe

WinUsb_QueryPipe 函数返回与一个接口关联的指定管道的管道信息。注意不会返回默认控制管道。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_QueryPipe(
IN WINUSB_INTERFACE_HANDLE InterfaceHandle，
IN UCHAR AlternateInterfaceNumber,
IN UCHAR PipeIndex,
OUT PWINUSB_PIPE_INFORMATION PipeInformation
);
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

AlternateInterfaceNumber

这是一个指定信息返回到的替换接口的值。

PipeIndex

这是一个指定信息要返回到的管道的值。在一个实施方式中这个值不同于 PipeId。相反，它表示在接口列表中的管道。0 值表示第一管道，1 值表示第二管道，等等。这个值应该小于接口描述符中的 bNumEndpoints。

PipeInformation

这是一个指向调用分配 WINUSB_PIPE_INFORMATION 结构的指针。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

注意

WINUSB_PIPE_INFORMATION 结构的定义如下所示：

```
typedef struct_ WINUSB_PIPE_INFORMATION {
USBD_PIPE_TYPE PipeType;
UCHAR      PipeId;
USHORT      MaximumPacketSize;
```

UCHAR Interval;

} WINUSB_PIPE_INFORMATION,

*PWINUSB_PIPE_INFORMATION;

PipeId 项是 USB8-比特（16 进制的）端点地址，其由 7-比特的地址和一个方向比特位组成。

9. WinUsb_SetPipePolicy

WinUsb_SetPipePolicy 函数设置指定管道（端点）的策略。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_SetPipePolicy（

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

IN UCHAR PipeID,

IN ULONG PolicyType,

IN ULONG ValueLength,

IN ULONG_PTR Value

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是为其设置策略的管道的管道标识符。

PolicyType

这是一个指定要改变的策略参数的值。

ValueLength

这是 Value 指向的缓冲器的字节长度，或者如果 Value 没有指向缓冲器为 0。

Value

由 PolicyType 指定策略参数的新值。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

注意

下面的列表描述了可能的 PolicyType 值：

SHORT_PACKET_TERMINATE（0x01）

如果值是 TRUE（非 0），每个 WRITE（OUT）请求用一个 0 长度包来终止，所述请求是多个用于端点的最大包大小。默认值是 FALSE。

AUTO_CLEAR_STALL（0x02）

如果值是 TRUE（非 0），失速 PID 会被忽略，并且除非正在使用先入先出（FIFO）缓存，否则会返回一个错误。但是在这种情况下，不会停止管道并且数据会继续流动。这个参数对 CONTROL 管道没有影响。要注意主机会自动地清除设备失速条件。默认值是 FALSE。

PIPE_TRANSFER_TIMEOUT（0x03）

该值是时间量，在几毫秒内应该取消传送。0 值表示无限时间量。默认值是无限时间量。

IGNORE_SHORT_PACKETS（0x04）

如果值是 TRUE（非 0），当接收到短包时，读取操作不会完成。相反，只有当读取了指定数量的字节时操作才会完成。如果值是 FALSE，当读取了指定数量的字节时或接收到短包时，读取操作就会完成。默认值是 FALSE。

ALLOW_PARTIAL_READS（0x05）

如果值是 FALSE（0），如果设备返回多于所请求的数据，读取请求就失败。如果值是 TRUE，保存多余的数据，并且将其返回到数据的开始位置以用于后续的读取请求。默认值是 TRUE。

AUTO_FLUSH（0x06）

如果值是 FALSE（0）并且设备返回多于所请求的数据，剩余的数据将被丢弃。如果值是 TRUE，所述行为依赖于 ALLOW_PARTIAL_READS 的值。或者数据将被保存并且返回到数据的开始位置以用于后续的读取请求，或者请求失败。默认值是 FALSE。

Raw_IO（0x07）

这使用户软件组件（例如图 1-2 中的 112）能保证正确的数据处理。如果用户软件组件提供的数据格式不正确，数据会被淹没。

10. WinUsb_GetPipePolicy

WinUsb_GetPipePolicy 函数获得指定管道（端点）的策略。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_GetPipePolicy（
IN WINUSB_INTERFACE_HANDLE InterfaceHandle，
IN UCHAR PipeID,
IN ULONG PolicyType,
IN OUT PULONG ValueLength,
OUT PVOID Value
）;
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是为其得到策略的管道的管道标识符。

PolicyType

这是一个指定要得到的策略参数的值。

ValueLength

这是一个指向 Value 指针指向的缓冲器的长度的指针。在输出时，这个参数接收复制到 Value 缓冲器中的数据长度。

Value

这是一个指向接收指定管道策略值的缓冲器的指针。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

11. WinUsb_SetFifoPolicy

WinUsb_SetFifoPolicy 函数为指定管道（端点）设置安全策略。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_SetFifoPolicy（
```

IN WINUSB_INTERFACE_HANDLE InterfaceHandle,

IN UCHAR PipeID,

IN ULONG PolicyType,

IN ULONG ValueLength,

IN ULONG_PTR Value

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是为其设置策略的管道的管道标识符。

PolicyType

这是一个指定要改变的策略参数的值。

ValueLength

这是 Value 指向的缓冲器的字节长度，或者如果 Value 没有指向缓冲器则为 0。

Value

这是一个用于 PolicyType 指定的策略参数的新值。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

注意

下面的列表描述了可能的 PolicyType 值。

FIFO_SIZE（0x01）

Value 是 FIFO 输入缓冲器的字节大小。默认值是 16×MaxPacketSize。如果达到这个极限，数据就会丢失。

NOTIFICATION_THRESHOLD（0x03）

Value 是在调用回叫函数之前应该在 FIFO 缓冲器中出现的字节的数量。如果 IGNORE_SHORT_PACKETS 是 FALSE 并且遇到短包，无论如何都会调用回叫函数。

12. WinUsb_GetFifoPolicy

WinUsb_GetFifoPolicy 函数得到指定管道（端点）的策略。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_GetFifoPolicy(
IN WINUSB_INTERFACE_HANDLE InterfaceHandle,
IN UCHAR PipeID,
IN ULONG PolicyType,
IN OUT PULONG ValueLength,
OUT PVOID Valur
);
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是为其得到策略的管道的管道标识符。

PolicyType

这是一个指定要得到的策略参数的值。

ValueLength

这是一个指向 Value 指向的缓冲器的长度的指针。在输出时，这个参数接收复制到 Value 缓冲器中的数据长度。

Value

这是一个指向接收指定 FIFO 策略参数的缓冲器的指针。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

13. WinUsb_StartFifo

WinUsb_StartFifo 函数启动从设备读取数据到 FIFO 缓冲器的自动读取机制。当数据被加到缓冲器时，调用 FifoDataNotification 函数，该函数通知客户端数据的存在。然后客户端能发布请求来从 FIFO 缓冲器中读取数据。调用这个

API 的示意性方法是：

BOOL_stdcall

WinUsb_StartFifo(

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

IN UCHAR PipeID,

IN PWINUSB_NOTIFICATION_CALLBACK FifoDataNotification,

IN PVOID FifoDataNotificationContext

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是要影响的管道的管道标识符。

FifoDataNotification

当到达通知门限值时，这是一个要调用的优选回叫函数。

FifoDataNotificationContext

这是一个要传送到 FifoDataNotification 回叫函数的优选上下文。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

注意

PWINUSB_NOTIFICATION_CALLBACK 类型声明如下所示：

typedef

VOID

(*PWINUSB_NOTIFICATION_CALLBACK)(

WINUSB_INTERFACE_HANDLE InterfaceHandle,

UCHAR PipeID,

ULONG NotificationType,

PVOID NotificationParameter,

PVOID Context

);

NotificationType 项的一个可能值是 FifoDataAvailable（0x01）。这个值表示 FIFO 缓冲器包含足够的 NOTIFICATION_THRESHOLD 字节的 ReadFifo 请求的数据来立即完成。这样，NotificationParameter 项就可以被忽略了。

14. WinUsb_StopFifo

WinUsb_StopFifo 函数停止自动读取机制。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_StopFifo (

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

IN UCHAR PipeID,

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是要影响的管道的管道标识符。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

15. WinUsb_ReadFifo

WinUsb_ReadFifo 函数从管道的 FIFO 缓冲器中读取数据。要注意 USB 包的大小不影响读取请求的传送。如果是设备用比客户端缓冲器大得多的包进行响应，并且 ALLOW_PARTIAL_READS 是 TRUE 的话，数据就被加在下一个传送的开始位置。如果 ALLOW_PARTIAL_READS 是 FALSE，读取请求就会失败。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_ReadFifo (

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

IN UCHAR PipeID,

OUT PUCHAR Buffer,

IN ULONG BufferLength,

OUT PULONG LengthTransferred,

IN LPOVERLAPPED Overlapped

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是要读取的管道的标识符。

Buffer

这是要读入数据的调用-分配缓冲器。

BufferLength

这是缓冲器的字节长度，或是读取的字节的最大数量。

LengthTransferred

这是指向接收复制到缓冲器的实际数量的字节的 ULONG 的指针。

Overlapped

这是一个优选的指针，其指向用于异步操作的 OVERLAPPED 结构。如果指定了这个参数，该函数将立即返回，并且当操作完成时用信号通知事件。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

16. WinUsb_ReadPipe

WinUsb_ReadPipe 函数从管道中读取数据。要注意 USB 包的大小不影响读取请求的传送。如果设备用比客户端缓冲器大得多的数据包进行响应，并且 ALLOW_PARTIAL_READS 是 TRUE 的话，数据就被加在下一个传送的开始位置。如果 ALLOW_PARTIAL_READS 是 FALSE，读取请求就会失败。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_ReadPipe (

IN WINUSB_INTERFACE_HANDLE InterfaceHandle,

IN UCHAR PipeID,

IN PUCHAR Buffer,

IN ULONG BufferLength,

OUT PULONG LengthTransferred,

IN LPOVERLAPPED Overlapped

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是要读取的管道的标识符。

Buffer

这是要读入数据的调用-分配缓冲器。

BufferLength

这是缓冲器的字节长度，或是读取的字节的最大数量。

LengthTransferred

这是指向接收复制到缓冲器的实际数量的字节的 ULONG 的指针。

Overlapped

这是一个优选的指针，其指向用于异步操作的 OVERLAPPED 结构。如果指定了这个参数，该函数将立即返回，并且当操作完成时用信号通知事件。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

17. WinUsb_WritePipe

WinUsb_WritePipe 函数向管道写入数据。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_WritePipe (

IN WINUSB_INTERFACE_HANDLE InterfaceHandle,

IN UCHAR PipeID,

IN PUCHAR Buffer,

IN ULONG BufferLength,

OUT PULONG LengthTransferred,

IN LPOVERLAPPED Overlapped

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是要写入的管道的标识符。

Buffer

这是包含写入数据的调用-分配缓冲器。

BufferLength

这是要写入的字节的数量。

LengthTransferred

这是指向接收写入到管道的实际数量的字节的 ULONG 的指针。

Overlapped

这是一个优选的指针，其指向用于异步操作的 OVERLAPPED 结构。如果指定了这个参数，该函数将立即返回，并且当操作完成时用信号通知事件。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

18. WinUsb_ControlTransfer

WinUsb_ControlTransfer 函数在默认的控制端点上发送数据。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_ControlTransfer (

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

IN WINUSB_SETUP_PACKET SetupPacket，

IN PUCHAR Buffer,

IN ULONG BufferLength,

OUT PULONG LengthTransferred,

IN LPOVERLAPPED Overlapped

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

SetupPacket

这是 8 字节的安装包。

Buffer

这是包含要传送的数据的调用-分配缓冲器。

BufferLength

这是要传送的字节数量，不包括安装包。

LengthTransferred

这是指向接收实际数量的传送字节的 ULONG 的指针。

Overlapped

这是一个优选的指针，其指向用于异步操作的 OVERLAPPED 结构。如果指定了这个参数，该函数将立即返回，并且当操作完成时用信号通知事件。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

注意

WINUSB_SETUP_PACKET 结构声明如下所示：

```
typedef struct_WINUSB_SETUP_PACKET{
UCHAR RequestType;
UCHAR Request;
USHORT Value;
USHORT Index;
USHORT Length;
} WINUSB_SETUP_PACKET,*PWINUSB_SETUP_PACKET;
```

19. WinUsb_ResetPipe

WinUsb_ResetPipe 函数重新设置数据触发并清除管道上的失速条件。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_ResetPipe (
IN WINUSB_INTERFACE_HANDLE InterfaceHandle,
IN UCHAR PipeID,
);
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是控制管道的标识符。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

20. WinUsb_AbortPipe

WinUsb_AbortPipe 函数中断管道所有的未决传送。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_AbortPipe (
IN WINUSB_INTERFACE_HANDLE InterfaceHandle,
IN UCHAR PipeID
);
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是控制管道的标识符。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

21. WinUsb_FlushPipe

WinUsb_FlushPipe 函数丢弃比客户端请求的多的作为设备返回结果保存的任何数据。调用这个 API 的示意性方法是：

```
BOOL_stdcall
WinUsb_FlushPipe (
IN WINUSB_INTERFACE_HANDLE InterfaceHandle，
IN UCHAR PipeID
);
```

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PipeID

这是控制管道的标识符。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

电源管理

图 5 表示在普通设备驱动器中提供电源管理的示意性方法 500。在一个实施方式中，当设备（例如参照图 1-4 所讨论的）与计算系统连接时，方法 500 使计算系统（例如参照图 6 所讨论的）进入低功率状态。

电源管理策略是为连接的设备（502）而定义的，如通过普通 USB 用户状态程序库（如图 1 中的 110）。电源策略包括不管是否允许自动暂停、是否可以自动唤醒设备、和/或是否为暂停设备设置了最小延时门限值都提供电源（如参照示意性 API 所讨论的，例如 WinUsb_SetInterfacePowerPolicy）。

具体来说，自动暂停包括当设备变成空闲状态时进行检测，这可能意味着在指定的时间内已经没有传送了，在所述点上，普通 USB 驱动器（如图 1 和 2 中的 108）可以暂停设备以节约能源。在一个实施方式中，当设备需要备份通电时，普通 USB 驱动器将进行自动检测并且在合适的时候进行。

至于自动唤醒计算系统，客户端（如图 1 和 2 中的 112）可以将普通 USB 设备驱动器的性能配置成允许设备从低功率状态（如备用状态或睡眠状态）中唤醒计算系统。在一个实施方式中，普通 USB 设备驱动器将为这个能力处理准备设备和计算系统所需的所有操作。

而且，可以设置延时门限时间段以便客户端（如图 1 和 2 中的 112）可以指定普通USB 设备驱动器保证在最后传送之后在暂停设备之前通过所需的时间量。因此，延时门限值可以指定数据传送之后的一个时间段，计算系统在数据传送之后等待该时间段以在设备进行暂停状态之前通过。

一旦遇到进行暂停状态的情况（如根据有效电源策略）（504），设备就进入低功率状态（506）。这就允许设备连接（如前所述不管是通过有线还是无线）的计算系统进行低功率状态（508）。

假设方法 500 即使在多个设备通过使用这里讨论的普通设备驱动器提供的功能连接到计算系统的情况下，也允许节省功率。也就是说，在没有配置普通设备驱动器的情况下，多个连接到计算系统的设备不会同时进入其暂停状态，或者在足够的时间段内通过计算系统启动全面的功率节省。

在各种实施方式中，下面的列表总结了功率管理状态的行为：

• 对整个电源管理事件保存所有的管道处理、接口处理、锁定和可选设置。

• 进行中的任何传送在低功率状态中都暂停，并且当系统工作时恢复传送。如果客户端（即，用户软件组件 112）正在使用 FIFO 缓冲器，该缓冲器在电源管理调用后重新启动。

• 如果应该恢复设备指定的配置，当系统重新工作时客户端（即，用户软件组件 112）应该这样做。这可以从 WM_POWERBROADCAST 消息中来确定。

• 客户端（即，用户软件组件 112）可以通过调用 WINUSB_SetInterfaceIdle 来指示接口是空闲的，以便支持可选的暂停。调用者可以规定设备在空闲时应该对远程呼叫激活。这个调用没有暗示直接的动作。

而且，如下所述，对于处理电源管理来说其它 API 也是可用的，其可以使用户软件组件更好地控制电源管理功能。

1. WinUsb_SetInterfacePowerPolicy

WinUsb_SetInterfacePowerPolicy 函数为设备设置电源策略。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_SetInterfacePowerPolicy (

WINUSB_INTERFACE_HANDLE InterfaceHandle，

ULONG PolicyType,

ULONG ValueLength,

ULONG_PTR Value

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PolicyType

这是指定要改变的策略参数的值。

ValueLength

这是 Value 指向的缓冲器的字节长度，或者如果 Value 没有指向缓冲器时为 0。

Value

这是 PolicyType 指定的策略参数的新值。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

注意

下面的列表描述了可能的 PolicyType 的值：

AUTO_SUSPEND（0x01）

如果值是 TRUE（非 0），当没有未决传送时设备将暂停。默认值是 TRUE。

ENABLE_WAKE（0x02）

如果设备支持 WAKE 就将 Value 设为 TRUE。默认值是 FALSE。

SUSPEND_DELAY（0x03）

Value 是在所有的传送之后暂停设备之前设备应该等待的最小时间量，以毫秒计。默认值是 5 秒。

2. WinUsb_GetInterfacePowerPolicy

WinUsb_GetInterfacePowerPolicy 函数为设备获得电源策略。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_GepInterfacePowerPolicy (

IN WINUSB_INTERFACE_HANDLE InterfaceHandle，

IN ULONG PolicyType,

IN OUT PULONG ValueLength,

OUT PVOID Value

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

PolicyType

这是指定要获得的策略参数的值。

ValueLength

这是指向 Value 指向的缓冲器的长度的指针。一旦输出，这个参数接收复制到 Value 缓冲器的数据长度。

Value

这是指向接收指定电源策略值的指针。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

3. WinUsb_EnableInterfaceIdle

WinUsb_EnableInterfaceIdle 函数指示接口是空闲的以便操作系统能给设备（USB 暂停）断电。这就是用户状态客户端怎样支持可选暂停。任何对其中一个管道的访问（读取或写入）都会自动将接口带离空闲状态，因此没有提供空闲状态的查询。调用这个函数不能保证设备会暂停；只建议设备当前能支持 USB 暂停状态。如果 FIFO 缓冲器正在运行，它将继续运行直到真正暂停所述设备。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_EnableInterfaceIdle (

WINUSB_INTERFACE_HANDLE InterfaceHandle，

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

4. WinUsb_DisableInterfaceIdle

WinUsb_DisableInterfaceIdle 函数可防止设备暂停，或者如果设备已经暂停就唤醒设备。调用这个 API 的示意性方法是：

BOOL_stdcall

WinUsb_DisableInterfaceIdle (

WINUSB_INTERFACE_HANDLE InterfaceHandle，

);

Parameters

InterfaceHandle

这是由 WinUsb_Initialize 返回的接口处理。

Return Value

如果成功并且填充了结构，这个函数就返回 TRUE。否则，返回 FALSE，并且可以通过调用 GetLastError 来检索已记录的错误。

安全

在一种实施方式中，通过普通设备驱动器 108 中由 IoRegisterDeviceInterface 创建的指定设备对象来提供安全。可以为限制能打开它的符号连接指定安全属性。

在这样一个实施方式中，只有一个用户软件组件（如 112）能有在任何给定时间都开放的连接，因此当软件组件打开连接时，没有其它的软件组件能访问接口，除非软件组件得到实际的处理。

假设用于物理设备的控制管道可以在多个接口（如对于设备支持的不同功

能）之间共享，所以用户软件组件能够向另一个接口发送控制命令。这在实施中可能在控制传送上需要一些限制。另外，可以通过在 PDO（204）上设置安全属性来处理符号连接上的安全访问。

通用计算环境

图 6 表示通用计算机环境 600，其可用于实现这里所描述的技术。例如，计算机环境 600 可用于运行包括普通设备驱动器（108）、普通用户状态程序库（110）和/或用户软件组件（112）的 OS。计算机环境 600 仅仅是计算环境的一个例子，并不打算对计算机和网络结构的使用或功能范围作任何限制。计算机环境 600 也不应该解释成具有与示意性计算机环境 600 中的任何一个组成部分或组合有关的依赖性或需求。

计算机环境 600 包括计算机 602 形式的通用计算设备。计算机 602 的组件包括，但不限于，一个或多个处理器或处理单元 604（可选地包括加密处理器或协处理器）、系统存储器 606 和系统总线 608，所述系统总线将包括处理器 604 的各种系统组件耦合到系统存储器 606。

系统总线 608 表示几种类型总线结构中的一个或多个，包括存储器总线或存储器控制器、外围总线、加速图形端口和处理器或使用多种总线结构中的一些的本地总线。例如，这样的结构包括工业标准结构（ISA）总线，微通道结构（MCA）总线，加强 ISA（EISA）总线，视频电子标准协会（VESA）本地总线和也称为中间层总线的外围组件互连（PCI）总线。

计算机 602 一般包括多种计算机可读介质。所述介质可以是计算机 602 能够访问的任何可用介质，其包括易失和非易失介质、可拆卸和不可拆卸介质。

系统存储器 606 包括易失存储器形式的计算机可读介质，例如随机访问存储器（RAM）610，和/或非易失存储器形式的计算机可读介质，例如只读存储器（ROM）612。基本输入/输出系统（BIOS）614 存储在 ROM612 中，其包含帮助在计算机 602 的组件之间传送信息的基本例程，例如在启动过程中。RAM610 一般包含能被处理单元 604 立即访问和/或不久操作的数据和/或程序模块。

计算机 602 还可以包括其它可拆卸/不可拆卸、易失/非易失的计算机存储介质。举例来说，图 6 描述了用于从不可拆卸、非易失磁性介质（未示出）中读取和写入的硬盘驱动 616，用于从可拆卸、非易失磁盘 620（如软盘）中读取和

写入的磁盘驱动618，和用于从可拆卸、非易失光盘624中读取和写入的光盘驱动622，例如CD-ROM、DVD-ROM或其它光介质。硬盘驱动616、磁盘驱动618和光盘驱动622通过一个或多个数据介质接口626各自连接到系统总线608。可替换地，硬盘驱动616、磁盘驱动618和光盘驱动622可以通过一个或多个接口（未示出）连接到系统总线608。

磁盘驱动和它们相关的计算机可读介质为计算机602提供计算机可读指令、数据结构、程序模块和其它数据的非易失存储。尽管举例说明了硬盘616、可拆卸磁盘620和可拆卸光盘624，也应该理解其它类型的能存储计算机可读数据的计算机可读介质也可以用于实现示意性的计算系统和环境，例如磁带或其它磁性存储设备、闪存卡、CD-ROM、数字化视频光盘（DVD）或其它光存储器、随机访问存储器（RAM）、只读存储器（ROM）、]电可擦除只读存储器（EEPROM）等等。

一些程序模块可以存储在硬盘616、磁盘620、光盘624、ROM612和/或RAM610上，举例来说包括操作系统626、一个或多个应用程序628、其它程序模块630和程序数据632。每个这种操作系统626、一个或多个应用程序628、其它程序模块630和程序数据632（或一些它们的组合）可以实现所有的或部分支持分布式文件系统的内置组件。

用户能通过例如键盘634和点击装置636（如鼠标）的输入设备向计算机602输入命令和信息。其它输入装置638（没有具体给出）包括话筒、操纵杆、游戏盘、圆盘式卫星电视天线、串行端口、扫描仪等等。这些和其它输入装置通过耦合到系统总线608的输入/输出接口640连接到处理单元604，也可以通过其它接口和总线结构连接，例如并行端口、游戏端口、或USB（如参照图1和2所讨论的）。USB端口可用于将摄像机、个人数字助理（PDA）、MP3设备、视频捕捉设备、闪卡读取器等类似设备连接到计算机环境600。

监视器642或其它类型的显示装置也可以经由接口连接到系统总线608，例如视频适配器644。除了监视器642外，其它输出外围设备还包括诸如扬声器（未示出）和经由输入/输出接口640连接到计算机602的打印机646等组件。

计算机602能使用到一个或多个远程计算机的逻辑连接而在联网环境中操作，例如远程计算设备648。举例来说，远程计算设备648可以是个人计算机、便携式计算机、服务器、路由器、网络计算机、对等设备或其它公共网络节点、

游戏控制台和类似设备。远程计算设备 648 作为便携式计算机来描述，其包括这里描述的与计算机 602 相关的许多或所有的组件和特征。

计算机 602 和远程计算机 648 之间的逻辑连接被描述为局域网（LAN）650 和普通光域网（WAN）652。这种网络环境在办公室、企业范围计算机网络、企业内部互联网和因特网中是很常见的。

当在 LAN 联网环境中实施时，计算机 602 经由网络接口或适配器 654 连接到局域网 650。当在 WAN 联网环境中实施时，计算机 602 通常包括调制解调器 656 或其它用于在光域网 652 上建立通信的装置。调制解调器 656 可以是计算机 602 内置的或是外置的，其可以经由输入/输出接口 640 或其它合适结构连接到系统总线 608。应该理解的是所述的网络连接是示意性的，并且可以使用其它用于在计算机 602 和 648 之间建立通信链路的装置。

在联网环境中，诸如所述的计算环境 600 中，所述与计算机 602 相关的程序模块或其中的一部分可以存储在远程存储设备上。举例来说，远程应用程序 658 位于远程计算机 648 的存储器设备上。为了描述，这里所描述的应用程序和诸如操作系统的其它可执行程序组件是分离的模块，尽管承认所述程序和组件在不同时间放置在计算设备 602 的不同存储组件中，并且可由计算机数据处理器来执行。

这里结合通用的计算机可执行指令描述了各种模块和技术，诸如由一个或多个计算机或其它设备执行的程序模块。通常，程序模块包括执行特殊任务或实现特殊抽象数据类型的例程、程序、对象、组件、数据结构等。一般来说，程序模块的功能可以是组合的或分布的，如在各种实施方式中所述的。

这些模块和技术的实施可以存储在计算机可读介质上或通过计算机可读介质发送。计算机可读介质可以是任何计算机能访问的可用介质。举例来说，但不是限制性的，计算机可读介质可以包括“计算机存储介质”和“通信介质”。

“计算机存储介质”包括在方法或技术中使用的用于信息存储的易失性的和非易失性的、可拆卸的和不可拆卸的介质，所述信息是诸如计算机可读指令、数据结构、程序模块或其它数据。计算机存储介质包括，但不限于，RAM、ROM、EEPROM、闪存或其它存储结构、CD-ROM、数字化视频光盘（DVD）或其它光存储器、盒式磁带、磁带、磁盘存储器或其它磁性存储设备、或任何其它可用于存储所需的信息并且能被计算机访问的介质。

“通信介质”一般包括计算机可读指令、数据结构、程序模块或调制数据信号中的其它数据，诸如载波或其它传输机制。通信介质还包括任何信息传递介质。术语“调制数据信号”意思是具有一个或多个以这样一种方式设置或改变它的特征的信号，以便编码信号中的信息。举例来说，但不限于此，通信介质包括有线介质，诸如有线网络或直线连接，和无线介质，诸如音频、射频(RF)、红外线、Wi-Fi、蜂窝网、有效蓝牙、和其它无线介质。上述的任何组合也包括在计算机可读介质的范围内。

在一种实施方式中，信息被访问（例如参照图 1 和 2 所讨论的）的硬件设备可以是任何与通用计算机环境 600 的组件（例如处理单元 604）耦合的设备。硬件设备也可以是通用计算机环境 600 的外围设备（例如鼠标 636、键盘 634、打印机 646 等）

结论

所以，尽管已经用结构特征和/或方法步骤的特定语言描述了本发明，但是也应该理解在附加的权利要求中定义的本发明不必限定于前述的具体特征或步骤。例如，这里描述的技术可以应用到有线或无线通信信道（例如蓝牙、微型计算机系统接口（SCSI）等）。因此，这里公开的具体特征和方法只作为实现所请求保护的本发明的示意性的形式。

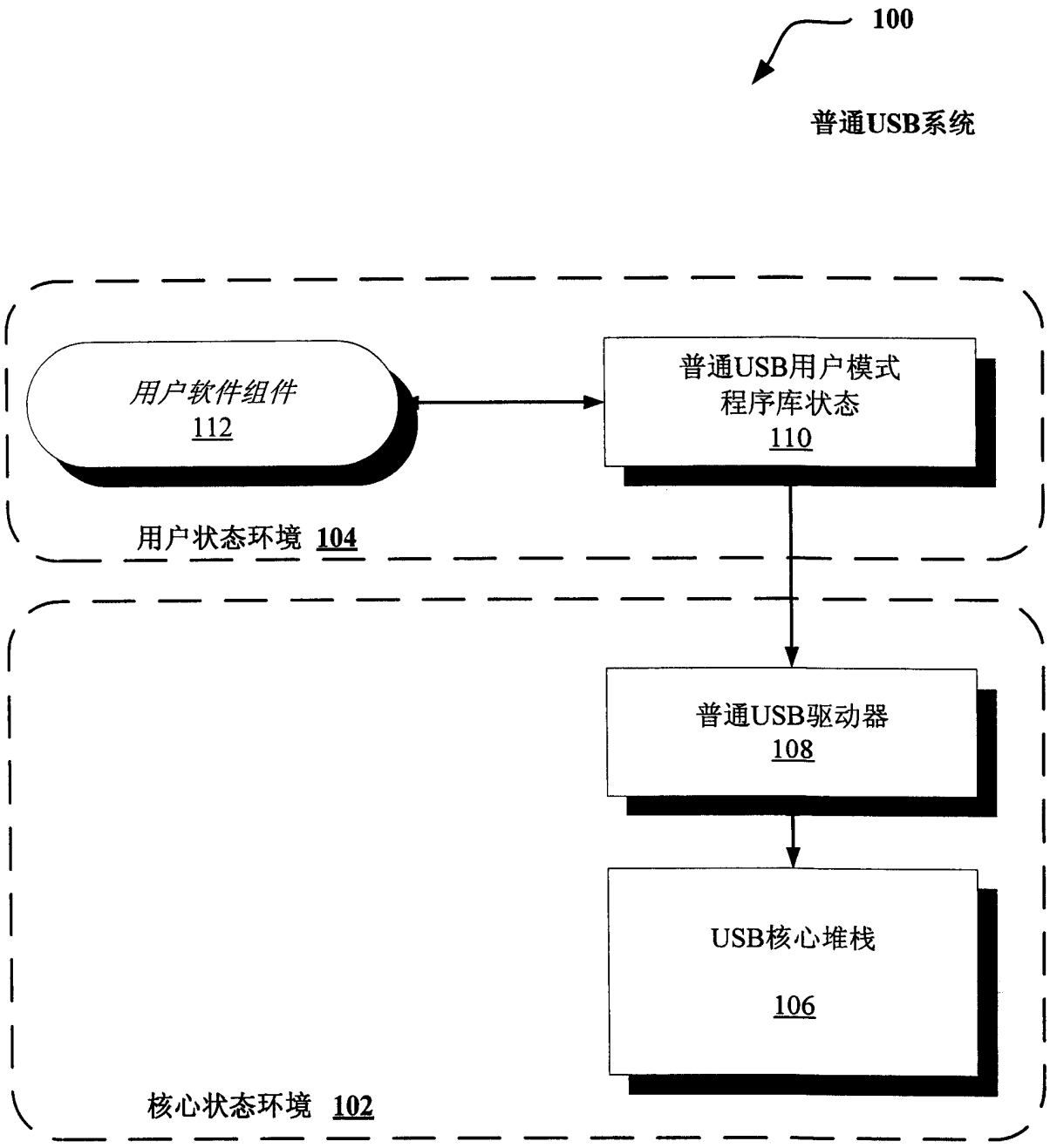

100
普通USB系统
用户软件组件
112
普通USB用户模式
程序库状态
110
用户状态环境 104
普通USB驱动器
108
USB核心堆栈
106
核心状态环境 102

图 1

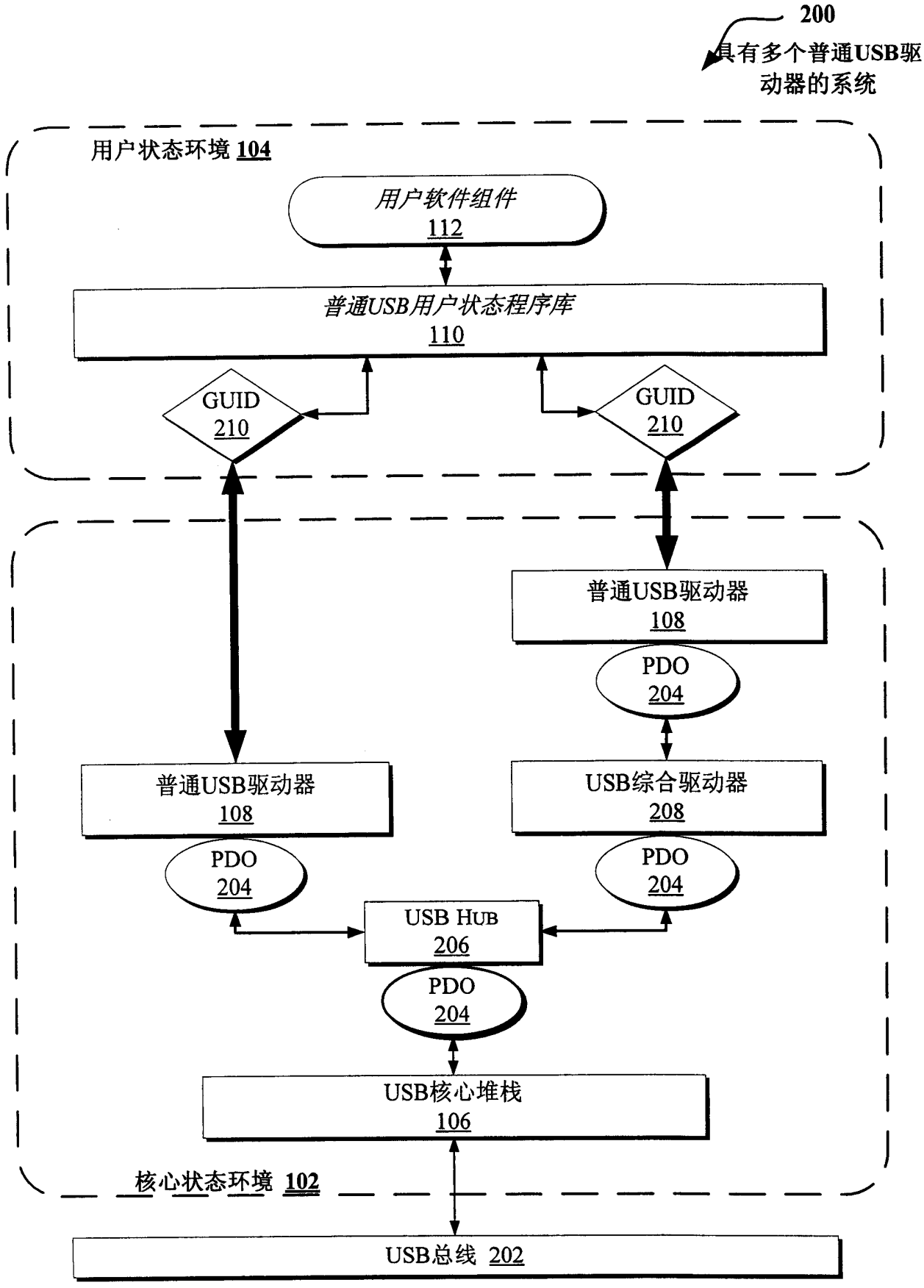

200
具有多个普通USB驱动器的系统
用户状态环境 104
用户软件组件 112
普通USB用户状态程序库 110
GUID 210
GUID 210
普通USB驱动器 108
PDO 204
普通USB驱动器 108
USB综合驱动器 208
PDO 204
PDO 204
USB HUB 206
PDO 204
USB核心堆栈 106
核心状态环境 102
USB总线 202

图 2

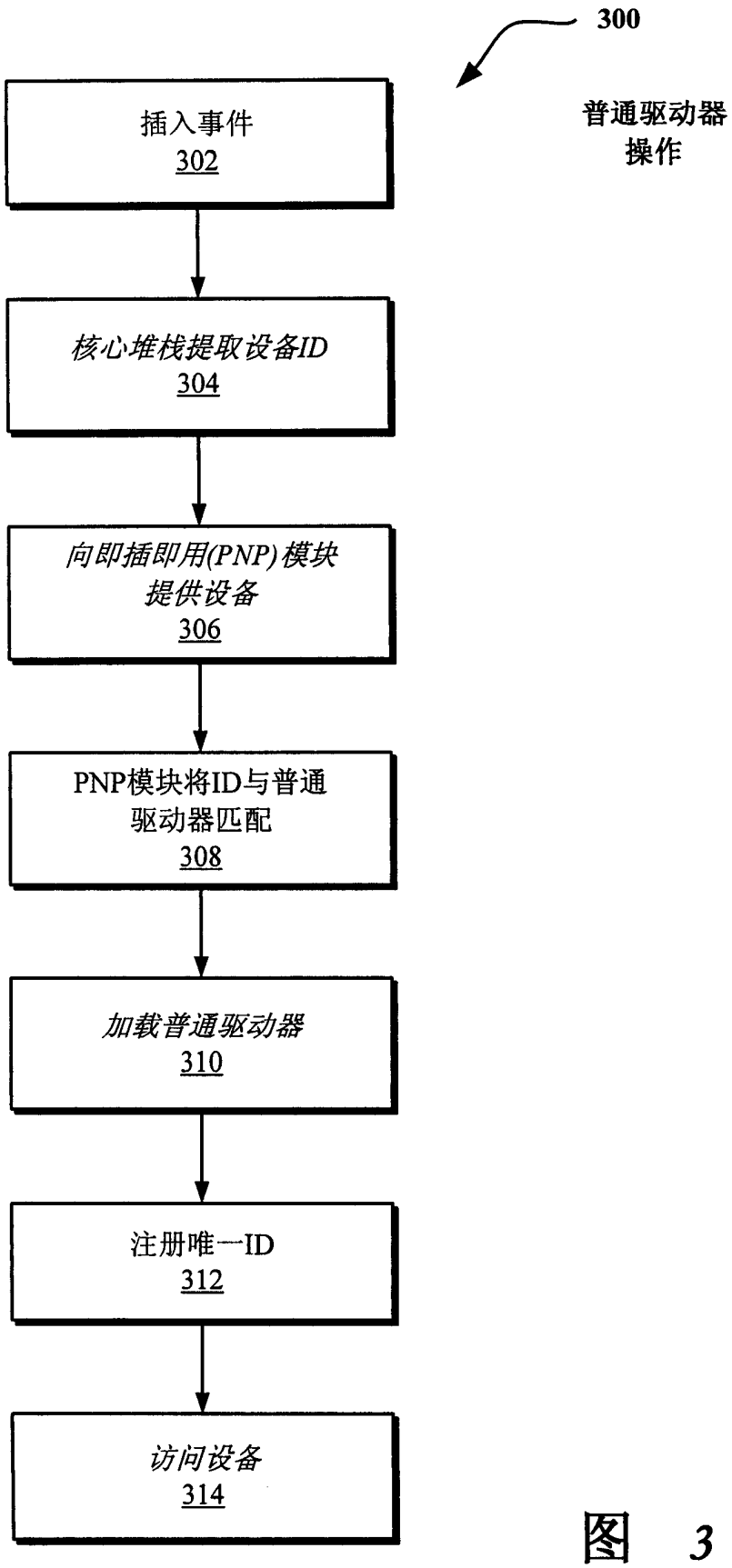

300
普通驱动器
操作
插入事件
302
核心堆栈提取设备ID
304
向即插即用(PNP)模块
提供设备
306
PNP模块将ID与普通
驱动器匹配
308
加载普通驱动器
310
注册唯一ID
312
访问设备
314

图 3

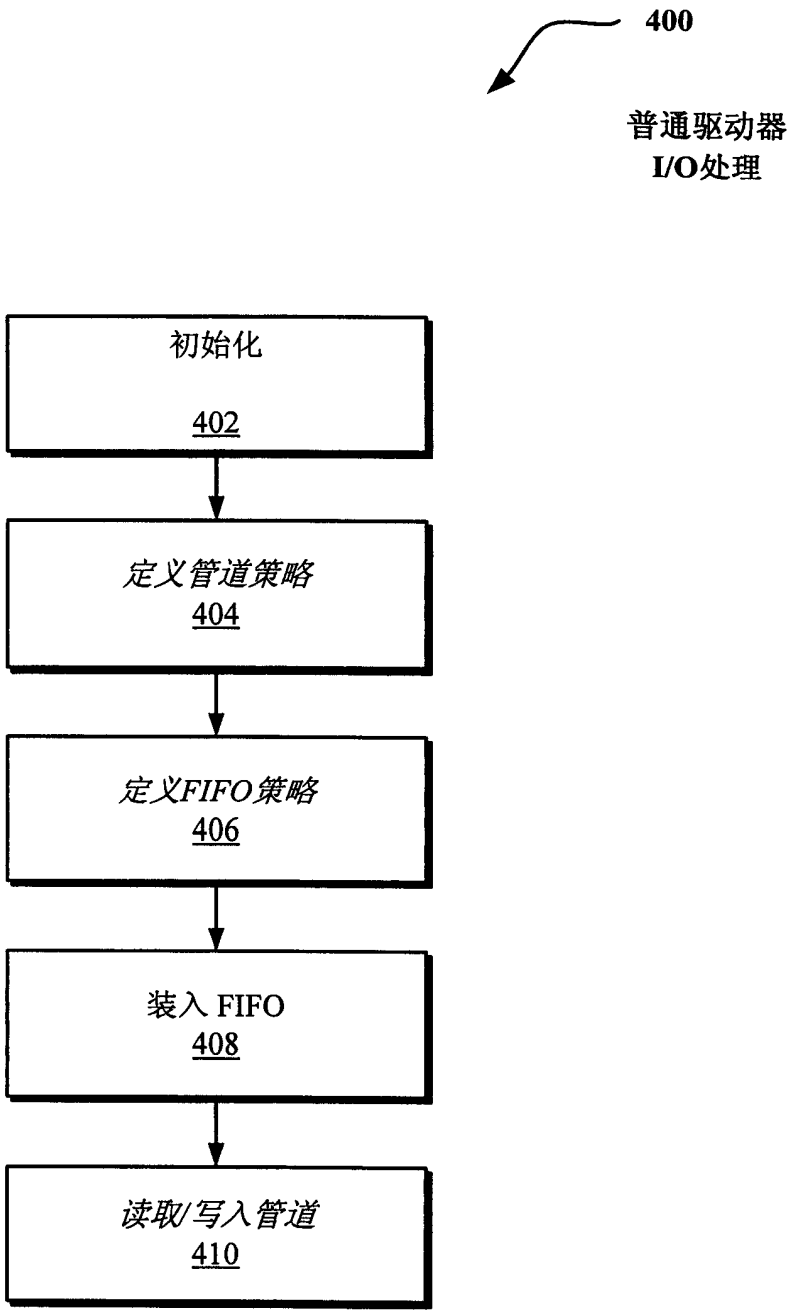

400
普通驱动器
I/O处理
初始化
402
定义管道策略
404
定义FIFO策略
406
装入 FIFO
408
读取/写入管道
410

图 4

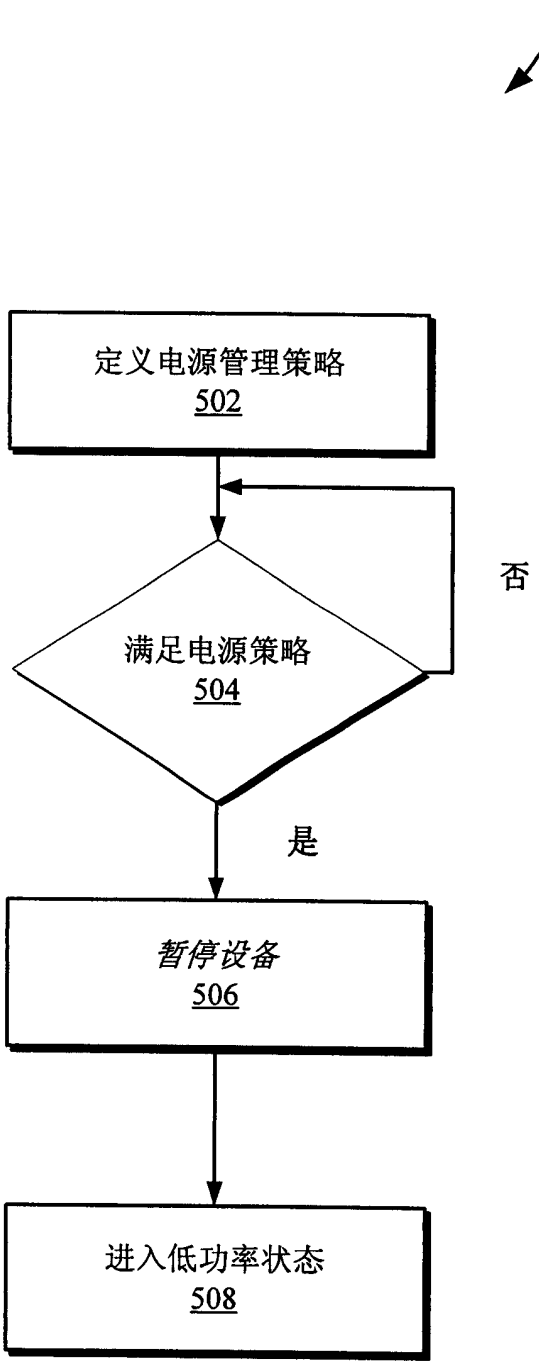

500
普通驱动器
电源管理
定义电源管理策略
502
满足电源策略
504
否
是
暂停设备
506
进入低功率状态
508

图 5

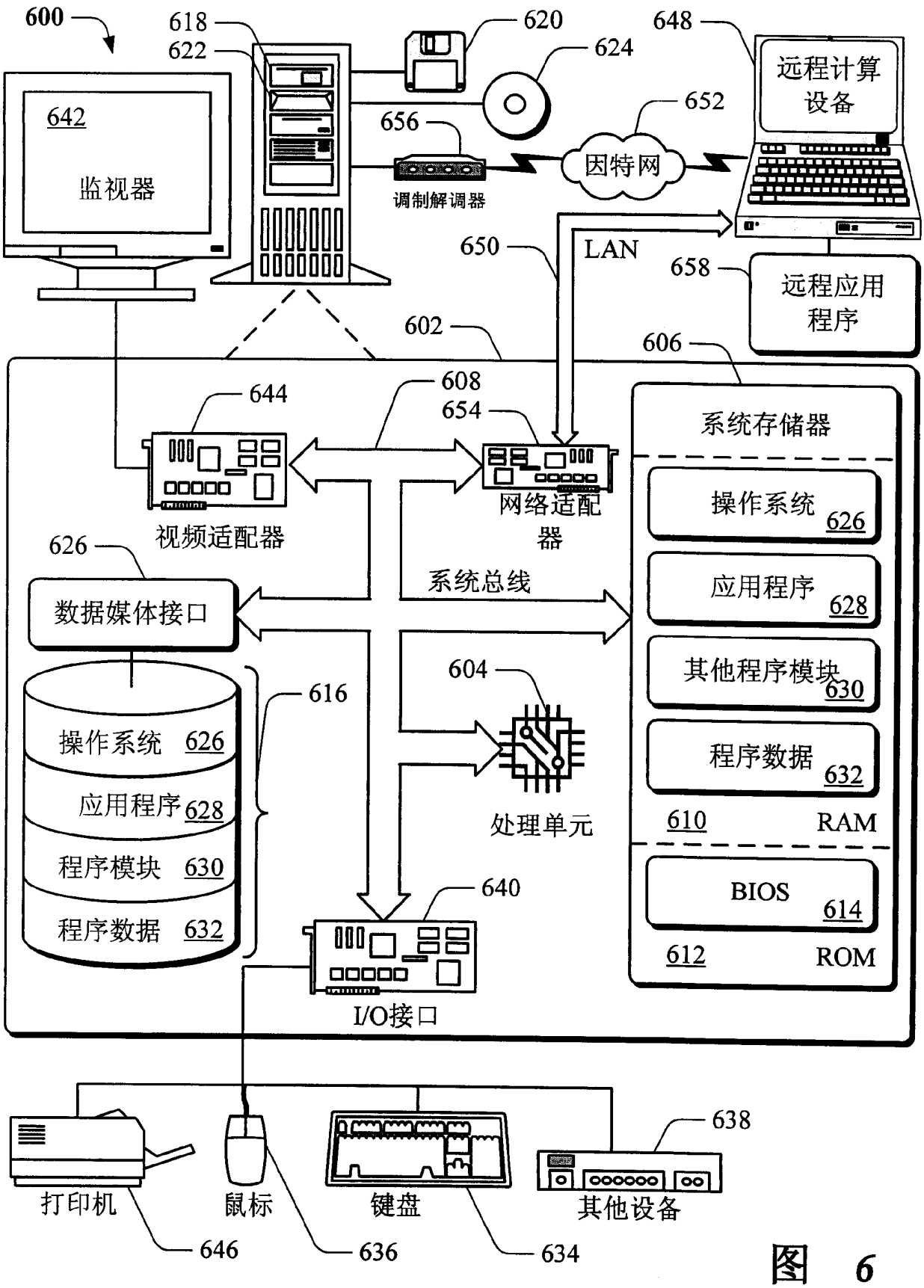

600
618
622
620
624
648
远程计算设备
642
监视器
656
652
因特网
调制解调器
650
LAN
658
远程应用程序
602
606
644
608
654
系统存储器
视频适配器
网络适配器
操作系统 626
626
数据媒体接口
系统总线
应用程序 628
其他程序模块 630
604
616
操作系统 626
应用程序 628
程序模块 630
程序数据 632
处理单元
程序数据 632
610 RAM
640
BIOS 614
612 ROM
I/O接口
638
打印机
鼠标
键盘
其他设备
646
636
634

图 6