

**硕士学位论文**

**Windows扫描仪设备和应用兼容方法研究与实践**

**作者姓名： 马国舜**

**指导教师: 马越 研究员**

**中国科学院软件研究所**

**学位类别: 工学硕士**

**学科专业: 软件工程**

**研究所: 中国科学院软件研究所**

**2021年 6月**

附件2

**英文论文题目**

**By**

**Ma Guoshun**

**A Dissertation/Thesis Submitted to**

**The University of Chinese Academy of Sciences**

**In partial fulfillment of the requirement**

**For the degree of**

**Master of Software engineering**

**Institute of Software, Chinese Academy of Sciences**

**June, 2021**

**中国科学院大学**

**研究生学位论文原创性声明**

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。

作者签名：

日 期：

**中国科学院大学**

**学位论文授权使用声明**

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名： 导师签名：

日 期： 日 期：

**摘 要**

当前 Linux 应用生态匮乏，严重阻碍了国产桌面操作系统的应用和推广。Wine是一个免费并且开源的兼容层软件，它利用API转换技术让为Windows设计和开发的程序能在类 UNIX 操作系统上运行，可以用来解决 Linux 上生态匮乏的问题。

但是，Wine 对 Windows设备访问相关的软件兼容效果还不理想。设备访问相关软件同时需要 Windows系统通用驱动框架和厂商特殊驱动模块的支持，但 Windows 的设备驱动相关组件的关键部分处于内核态。Wine 作为一个 Linux 的用户态兼容框架，目前还无法对 Windows 内核态二进制代码提供完善的运行支撑，这就对 Windows 设备访问相关软件的兼容带来了极大的困难与挑战。针对该问题，本文以扫描仪为背景，分析和探索了设备相关软件的可行兼容方法，为其它类型设备及其应用软件的兼容提供了共性支撑。主要工作和贡献如下：

1. 1.针对设备软件能否兼容的问题，本文通过分析多种设备及其应用软件的工作方式，和 Windows上面对应的支持框架，总结出大多数设备及其应用软件的独特特点，为设备及其软件的兼容奠定了可行性。比如对同一类设备，一定存在Windows 与 Linux 系统共同遵循的通讯协议；最大化共性设备访问框架 + 最小化个性驱动的模式，共性设备驱动放在Windows的通用设备框架中，厂商只需要提供用户态的个性化设备驱动模块。
2. 针对 Linux 平台无法支持 Windows 扫描仪驱动运行的问题，本文设计和实现了 Wine 中的扫描仪的兼容框架，增加了 Wine 的设备支持类型。在该扫描仪支持框架中，完善了 Wine 中的 TWAIN 协议，实现了 usbscan.sys 模块，并补充了 Wine 中的设备通用框架，达到了兼容扫描仪 Windows 驱动的目的。
3. Windows 设备和应用的兼容的方法研究。在本文中，以某种型号扫描仪的兼容为背景，在补充实现设备通用框架和扫描仪框架的过程中，尝试了多种兼容方法，对比了使用不同兼容方法可以达到的兼容效果。最后对设备兼容方法进行了归纳总结，为之后其他设备的兼容提供方法支持。

**关键词：**Linux，Windows，扫描仪，驱动，兼容方法

**Abstract**

**目 录**

**图目录**

**表目录**

1. **引言**

**1.1 研究背景及意义**

在个人计算机操作系统领域，由于Windows所占的统治地位，使得用户无论是在操作系统的可选择方面还是个人信息的安全性方面都处于不利的地位，也影响了我国软件业的健康发展。想要冲破 Windows在操作系统领域的垄断，发展一种新的国产桌面操作系统显得尤为重要。Linux的出现提供了一种机遇，它卓越的性能、几乎免费的价格、完全开放源代码的特性使其成为Windows的强劲对手【引用】。在Linux的基础上发展国产桌面操作系统，可以做到真正的自主可控可定制，对国家的信息安全也具有重大意义。

但推广国产操作系统面临许多问题【引用】。其中，阻碍国产操作系统推广的一个重要原因之一，是 Linux 生态匮乏，在外置设备上表现的尤为突出。比如扫描仪、打印机之类的办公设备等，都只能在Windows平台上使用，很难在国产操作系统上使用。

Wine恰好可以解决 Linux 应用生态匮乏的问题【引用】。Wine 是一个免费并且开源的兼容层软件，它利用API转换技术，可以让Windows的程序能在Linux操作系统上运行。所以目前的解决办法，是通过Wine在国产操作系统上使用这些办公设备。

但是，Wine 对设备访问相关的软件兼容效果还不理想。设备支持软件需要 Windows系统通用驱动框架和厂商驱动模块的支持，但 Windows 的设备驱动相关组件的关键部分处于内核态。Wine 作为一个用户态的兼容框架，目前还无法对 Windows 内核态二进制代码提供完善的运行支撑。想要在 Wine 中对设备实现兼容面临着很大的挑战。同时，对目前已经兼容的设备缺少方法层面的归纳总结。当面临一款新的设备需要兼容时，无法从之前的兼容工作中发现借鉴价值来为新设备提供方法指导，导致每一款新设备的兼容都面临从头开始的局面，严重影响国产操作系统的发展。

因此，通过对扫描仪进行多种方法的兼容，借此过程归纳和总结出完整的设备兼容方法，具有重要的现实意义。不仅可以解决扫描仪设备的兼容问题，同时可以为之后其他类型的设备兼容提供可靠的方法支持。

**1.2 研究现状**

**1.2.1 Wine 对设备的支持**

经过开源社区和前人的探索，Wine 目前已经对一些设备提供了兼容，包括 Ukey，HID设备，打印机等。在兼容这些设备时，Wine 除了提供某种设备专属的模块外，也模拟 Windows 提供了一些所有设备都需要的共性支持。Wine 以Linux中的设备和驱动访问框架为基础，在兼容系统通用模拟机制的支撑下，提供即插即用管理、设备I/O框架、会话管理等兼容环境设备及驱动管理公用机制，并向上提供设备属性操作接口、设备文件访问接口等，实现跨设备类的共性支撑。

此外，Wine 因为处于用户态，Windows 内核中的许多功能无法直接实现，需要借助 Linux 提供的内核功能。比如 Windows 设备的总线驱动，Wine 额外启一个监控线程，在监控线程中利用 libudev 从 Linux 内核捕获设备信息来模拟总线驱动的行为。比如 Windows 在内核中完成的跨进程通信，Wine 通过 Wine Server 进程来协助另外两个进程完成进程间的通信。这些已有的支持框架都可以为我们兼容扫描仪提供帮助。

但是，对这些设备兼容过程中缺乏对兼容方法的总结，从哪个模块介入进行兼容更符合 Windows 的设计，在哪个层面进入 Linux 兼容工作量更小等问题都没有得到回答。扫描仪的兼容该使用怎样的方法无法从现有的设备中得到借鉴。

**1.2.1 扫描仪兼容**

扫描仪设备和应用的正常工作主要依赖两个方面的功能。应用和驱动之间的通信协议，和扫描仪驱动所依赖的系统框架。

在应用和驱动之间的通信协议方面，扫描仪驱动的协议主要可以分为两种类型，SANE 协议【引用】和 TWAIN 协议【引用】。Twain是一个主要应用在Windows下的扫描仪协议，它定义了图像设备（扫描仪，数码相机）和应用之间相互通信的协议，各个设备厂商通过支持Twain协议使得上层应用可以用同样的代码去访问不同扫描仪。TWAIN主要包括三个要素，Application ，Data Source Manager和Data Source。SANE是一个主要用在 Linux 的协议，它是“Scanner Access Now Easy”的首字母缩略词。SANE分为前端和后端，清晰的结构可以让它方便地实现应用和驱动所需的各种API，满足扫描仪硬件和应用所需的所有功能。

在驱动依赖的系统框架方面，Windows 和 Linux 有各自的扫描仪驱动支持框架。在 Windows 中，系统提供了 WIA（Windows Image Acquire），STI（Still Image）和 USBSCAN 等模块，用来完成扫描仪的驱动和设备间的通信。厂商的驱动既可以通过WIA 和 STI 来实现功能，也可以直接使用 USBSCAN 来实现功能，这些在 Wine 中都是完全没有的。而在 Linux 中，SANE 的后端既可以依赖于Linux的 libusb 和 libudev 库，也可以直接使用 Linux 的文件系统来完成和设备之间的通信。

要让扫描仪在兼容系统中正常工作，其依赖的协议和系统框架都需要兼容。现有的可以利用的资源有开源的 TWAIN 协议，开源的 SANE 协议，Wine 中对 TWAIN 协议的兼容实现。

开源的 TWAIN 协议，主要指的是 TWAIN 中的 Data Source Manager 部分是开源的，可以对协议的兼容提供参考。

开源的 SANE 协议，主要指的是提供了 SANE 的前后端通信部分是开源的。但是绝大部分的厂商不会提供 SANE 协议的扫描仪驱动，所以开源的 SANE 协议无法在兼容系统中直接使用。

Wine 中对 TWAIN 协议的兼容实现也存在诸多问题，关键路径缺失，处于无法工作的状态，需要补充完整后使用。

**1.2.3 总结**

基于以上两点的分析，扫描仪设备的兼容及兼容方法主要存在以下几点问题：

1. 兼容系统中对目前已经兼容的设备所使用的兼容方法缺乏归纳总结。目前在 Wine 中已经对部分设备完成了兼容，并提供了一些设备兼容的共性模块支持可以被扫描仪使用。但是这些设备所使用的兼容方法没有得到合理的归纳和总结，各种兼容方法的优劣也没有指出。在面对扫描仪的兼容需求时，无法从现有的设备兼容中得到方法指导，给兼容工作的展开带来了困难。

2. 兼容系统中对 Windows 的扫描框架支持能力不足。Windows 扫描仪所需的 TWAIN 协议关键路径缺失，Windows 的 USBSCAN 内核驱动模块在 Wine 中完全没有。这造成 Windows 扫描仪的应用和驱动在 Wine 中都是无法工作的。同时由于 USBSCAN 是处于内核态之中，给该模块的兼容带来了极大的困难。

**1.3 研究内容**

通过对各平台扫描仪的支持现状的分析，和目前 Wine 中的设备兼容状况的分析，发现了 Linux 平台对 Windows 驱动扫描仪支持能力不足的问题，和现有的设备兼容方法缺乏总结归纳的问题。本文以扫描仪的兼容为背景，设计实现了 Wine 中基于不同驱动的扫描仪兼容框架，并使用不同的兼容方法对扫描仪尝试进行了兼容，通过对比各种方法的兼容效果，总结归纳除了完整的设备兼容方法及各方法的优劣。研究内容将从以下三个方面展开：

**1. 基于 Linux 驱动设计和实现 Wine 的扫描仪兼容框架。**

设计和实现基于 Linux 驱动的扫描仪兼容框架，使具有 Linux 驱动的扫描仪在该框架下可以正常扫描。对于厂商提供了不同形式的 Linux 驱动，比如 SDK 或 SANE 后端驱动，该框架可以采用不同的兼容方法进行兼容，达到兼容扫描仪 Linux 驱动的目的，并为后续多种兼容方法的实践和对比做铺垫。

1. **基于 Windows 驱动设计和实现 Wine 的扫描仪兼容框架。**

设计和实现基于 Windows 驱动的扫描仪兼容框架，使具有 Windows 驱动的扫描仪在该框架下可以正常扫描。补充了 Wine 中驱动即插即用功能，使扫描仪驱动 USBSCAN 模块在设备插入后再进行加载；并在 Wine 中实现 USBSCAN 模块，使 Windows 的 Data Source 驱动模块可以对通过文件的方式对驱动进行访问；同时实现了 USB 通信模块，使 USBSCAN 可以通过USB接口和扫描仪进行交互。达到了 Windows 驱动正常供做的目的，并为后续多种兼容方法的实践和对比做铺垫。

1. **Windows 扫描仪设备和应用的兼容的方法研究。**

总结归纳出设备兼容方法，并说明每种方法的优劣，为之后设备的兼容提供方法支持。在本文中以某种型号扫描仪的兼容为背景，在设计实现扫描仪兼容框架的过程中，尝试多种设备兼容方法，对比不同方法的兼容效果，总结归纳出各兼方法的特点。

**1.4 论文组织结构**

本文首先阐述了研究的背景、意义、研究内容和研究创新性，接着对相关技术进行分析，根据分析内容，提出了基于不同驱动的扫描仪兼容框架，并阐述其设计与实现。在该框架的设计和实现过程中，实践了多种设备兼容方法，之后对这些设备兼容方法进行总结归纳，最后是总结和展望。具体章节安排如下：

第 1 章介绍本文研究背景和意义，并对当前的研究现状进行了阐述，指出了兼容系统对扫描仪设备支持能力不足，及现有的设备兼容方法缺乏归纳总结的问题，然后引出了本文的主要研究内容。

第 2 章介绍与本文研究内容相关的扫描仪技术和现有的设备兼容方法，主要包括Windows 设备驱动框架，Windows 扫描仪体系，Linux 扫描仪体系，Wine 的扫描仪框架及现有已兼容的设备所采用的兼容方法。

第 3 章描述扫描仪兼容框架的共性模块的设计和实现。主要包括 TWAIN 协议的设计实现，模拟 Data Source 的设计实现，USBSCAN 内核模块的设计实现，USB 通信模块的设计实现。

第 4 章在上一章各模块实现的基础上，使用不同的兼容方法对扫描仪进行兼容，设计不同的扫描仪兼容框架，并对比各兼容方法的优势和劣势，对兼容方法进行总结归纳。

第 5 章阐述特定扫描仪兼容支持实践与框架验证。主要是对扫描仪的 Linux驱动和 Windows 驱动进行分析，并使用 S6100 扫描仪进行扫描功能的实践。

第 6 章是对当前的研究工作的总结，并对下一步工作进行展望。

1. **相关技术概述**

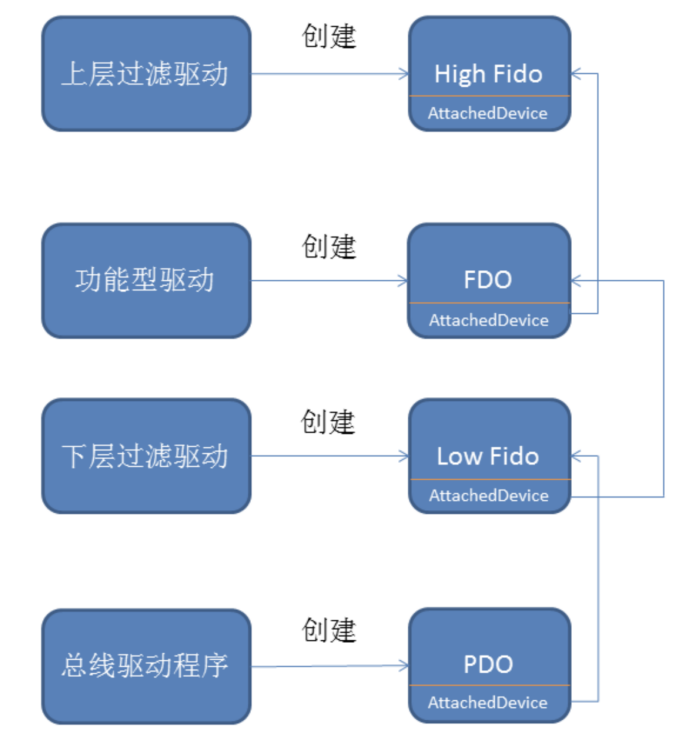
本章主要分析与本文研究内容相关的扫描仪技术及现有设备的兼容方法。首先分析 Windows 驱动模型，再分析Windows 的扫描仪技术和 Linux 的扫描仪技术，接着分析 Wine 的扫描仪框架，再介绍现有设备的兼容方式。最后对相关技术和方法进行总结，并分析对本文的参考价值。

**2.1 Windows 驱动模型**

本文的研究内容包括基于 Windows 驱动设计实现扫描仪兼容框架，而 Windows 设备驱动都是工作在 WDM 框架中的，所以对 WDM 框架的分析是必要的。

Windows 驱动模型（Windows Driver Model）是Windows 98和Windows 2000引入的设备驱动程序的框架。

WDM为设备驱动程序实现了分层的体系结构。为每个设备都构建了驱动堆栈，由整个堆栈的驱动程序为设备提供服务。堆栈中的驱动程序不需要彼此直接交互。每个驱动程序都可以通过 I/O request packets (IRP) 和上下层的驱动进行通信。堆栈中一般有两种设备对象：物理设备对象（Physical Device Object，PDO）和功能设备对象（Function Device Object，FDO）。在FDO和PDO之间，可以创建过滤驱动。位于FDO下面的，称之为下层过滤驱动；位于FDO上面的，称之为上层过滤驱动。其结构如下图所示：



**图 2.1 设备对象堆栈示意图**

**Figure 2.1 Schematic Diagram of Device Object Stack**

WDM主要包括以下几个概念：

设备对象（DEVICE\_OBJECT)：每个驱动会创建一个或者多个设备对象，用DEVICE\_OBJECT表示。每个设备对象都有一个指针指向下一个设备对象，因此形成一个设备链。设备链的第一个设备是由 DRIVER\_OBJECT 的 DeviceObject 指定。其中 NextDevice 指向下一个设备对象；AttachedDevice，指向更高一层驱动的设备对象。

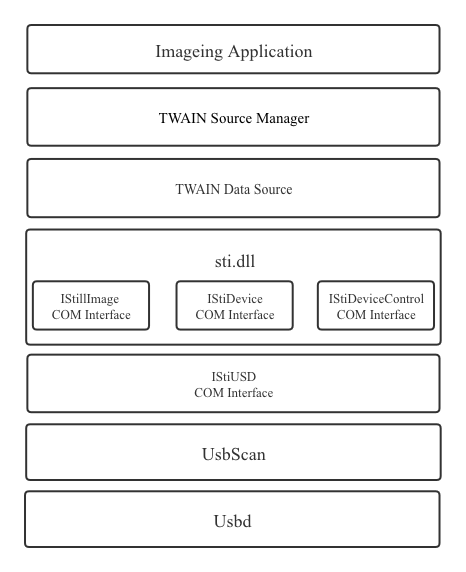
设备扩展（DEVICE\_EXTENSION)：DEVICE\_OBJECT里面记录一些通用信息，但是大多数情况下，驱动程序需要额外记录一些信息，这个时候就需要设备扩展了。设备扩展是由程序员指定内容和大小，由I/O管理器创建的，并且保存在非分页内存中。

入口程序 DriverEntry：WDM 驱动的入口函数是 DriverEntry。WDM驱动需要设置2个派遣函数：AddDevice 和 IRP\_MJ\_PNP。WDM驱动框架中，操作系统加载 PDO 以后，会调用驱动的 AddDevice 函数，然后在 AddDevice 函数里面创建 FDO，并且附加到 PDO 之上。

**2.2 Windows 扫描仪系统**

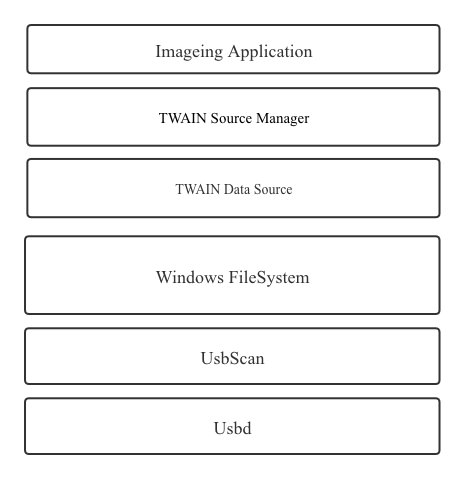
本文的研究内容，包括在 Wine 中模拟 Windows 操作系统与 Windows 扫描仪驱动的交互，因此对 Windows 扫描仪系统的分析必不可少。

在 Windows 中，提供了 USBSCAN 和 USBD 内核模块来和设备进行交互。其中 USBD 是 USB 设备的通用内核驱动，USBSCAN 是对 USBD 的封装，提供了一些扫描仪专用的功能，比如扫描仪参数读取，扫描仪控制接口等。在 USBSCAN 之上，新版的 Windows 还提供了 STI（Still Image）和 WIA（Windows Image Acquisition）模块来简化厂商驱动对设备的控制。但由于推广的问题，WIA 和 STI 尚未被大范围使用。因此 Windows 中存在两种驱动和设备交互的方式：厂商驱动通过 STI 模块访问设备和厂商驱动直接使用 USBSCAN 内核驱动访问设备。两种访问方式如下图所示：



**图2.1 通过 STI 访问内核态驱动**

**Figure 2.1 Access Kernel Mode Driver Through STI**



**图2.2 通过文件系统访问内核态驱动**

**Figure 2.2 Access Kernel Mode Driver Through File System**

从以上两种访问方式中，可以看到涉及到的模块为 TWAIN 协议，STI 模块，USBSCAN 驱动和 USBD 驱动。

**2.2.1 TWAIN 协议**

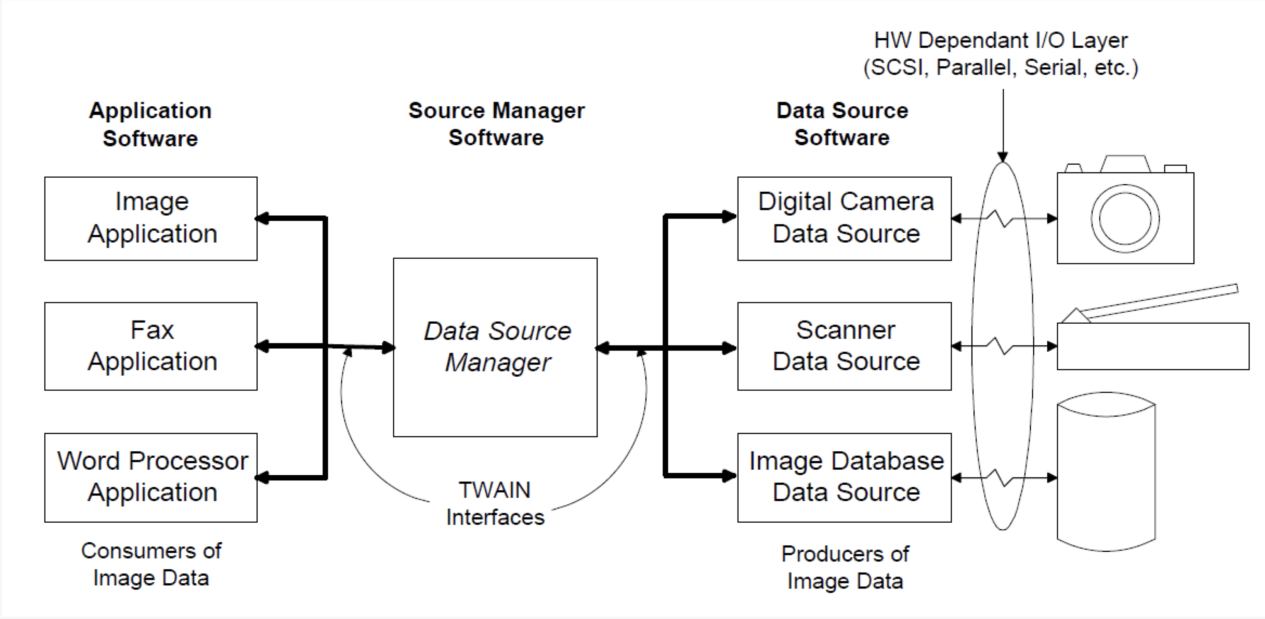
TWAIN定义了图像设备（扫描仪，数码相机）和应用之间相互通信的协议，各个设备厂商通过支持TWAIN协议使得上层应用可以用同样的代码去访问不同扫描仪。

TWAIN主要包括三个要素，结构如下图所示：

（1）Application - 使用TWAIN的图像应用，如Photo Shop。

（2）Source Manager - 用来管理数据源和应用之间的交互，这个模块由TWAIN Group来维护，并提供开源代码。

（3）Data Source - 主要用于控制图像设备，这块主要由硬件厂商开发。

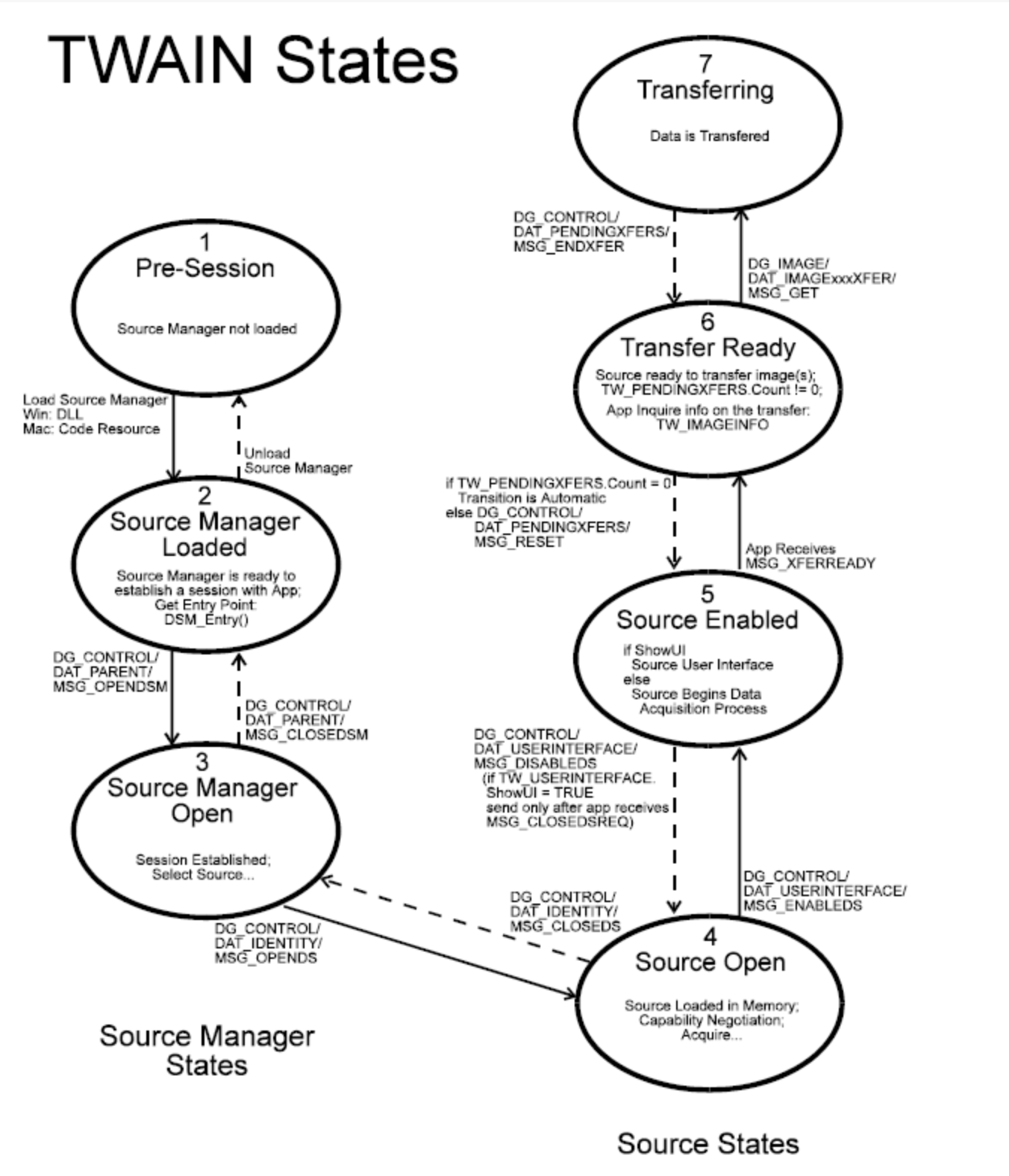


**图2.3 TWAIN 结构**

**Figure 2.3 TWAIN Structure**

三个要素之间通信通过DSM\_Entry() 和 DS\_Entry() 两个接口，Application 通过调用 Data Source Manager 的 DSM\_Entry() 接口来使用 Data Source Manager 的功能，同理 Data Source Manager 会调用 Data Source 的 DS\_Entry() 接口来使用 Data Source 的功能。

同时，在 TWAIN 协议中规定，应用 Application 操作扫描仪需要在固定的 session内完成，每个TWAIN session包括7个状态，不同的状态使用不同 Tuple（三元组）对扫描仪操作和进行状态转移，应用程序应该遵循状态对source进行操作。TWAIN 规定的状态转移如下图所示：



**图2.4 TWAIN 状态转移图**

**Figure 2.4 TWAIN State Transition Diagram**

**2.2.2 扫描仪驱动**

扫描仪驱动通常是由扫描仪厂商提供的，由于其中包括该型号扫描仪所能够识别的各种命令的二进制流，所以是兼容扫描仪时必不可少的一部分。但由于该部分无法通过开源渠道获得源码，给兼容带来了极大的困难，因此需要仔细分析驱动的组成，找出其中必须要兼容的部分。

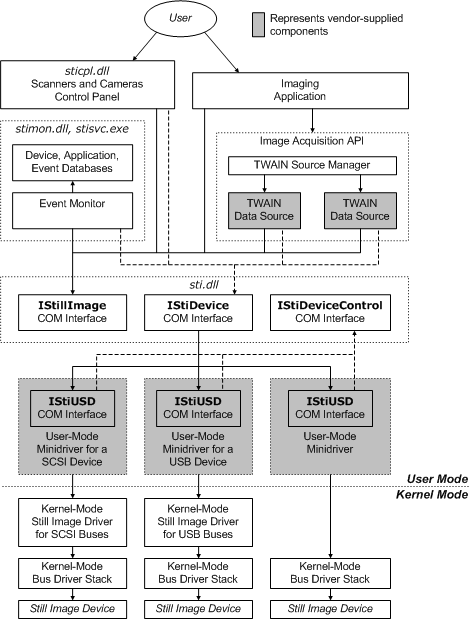
从上一节介绍的 TWAIN 协议中可以看出，厂商提供的驱动通常以 TWAIN Data Source 的形式出现，该部分起到了承上启下的作用，是兼容过程中必不可少的。Data Source 是一个后缀为 DS 的 DLL 文件，它对外暴露 DS\_Entry()接口供 Data Source Manager 调用。在该 Data Source 中，通过调用 STI 接口或直接调用文件系统的函数来完成和底层框架交互的功能。

同时，在 Data Source 中通常会对下层传上来的数据进行大量的图像处理，来满足应用的请求，因此 Data Source 会依赖大量的图像处理 DLL。这些 DLL 如果兼容失败会造成从下层 USBSCAN 传上来的图片像素、大小错误等问题，无法正确保存，所以这些 DLL 也必须要兼容。

**2.2.3 STI（Still Image）模块**

为了简化厂商实现扫描仪驱动的复杂度，微软基于内核态的 USBSCAN 驱动，封装了 STI 这样一个 COM模块来供厂商的驱动进行调用，并提供了可视化的配置面板。此外，在 STI 的COM接口中提供了 IStillImage::CreateDevice、IStillImage::GetDeviceList 之类的设备管理接口，避免了用户直接调用内核模块带来的复杂度。

STI 的结构如下图所示：



**图2.5 STI 架构**

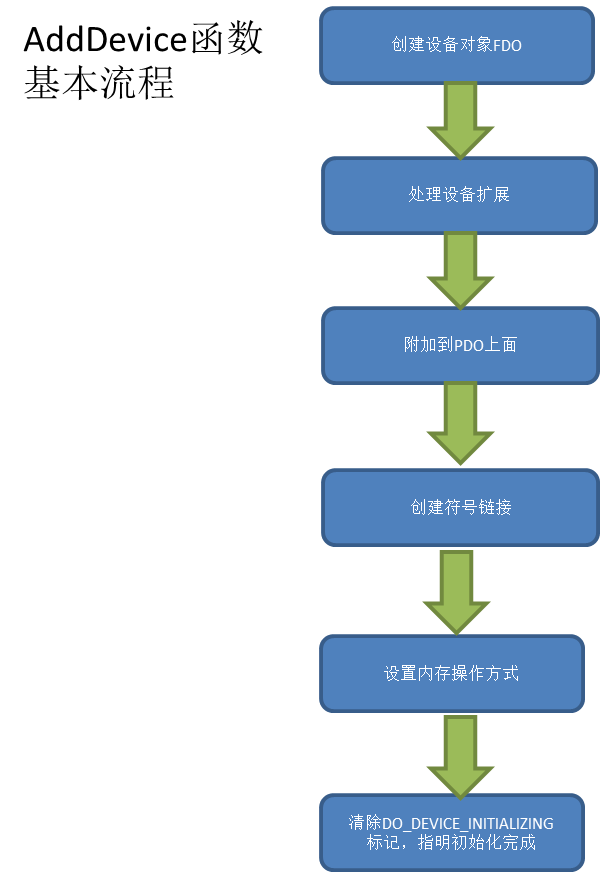
**Figure 2.5 STI Structure**

但由于商业化中的推广的问题，STI 并没有在设备厂商中得到大范围使用。厂商的驱动更多的是直接使用 USBSCAN 进行设备交互的。

**2.2.4 USBSCAN 内核态驱动**

Windows 提供了 USBSCAN 内核态驱动，供厂商的驱动进行调用来和设备进行交互。该驱动程序支持设备的即插即用，并提供用于添加，删除，启动，停止和读写注册表项的服务。此外，该驱动程序支持电源管理，为设备提供了挂起和恢复操作。

USBSCAN 作为内核态驱动，也是 WDM 框架的一部分，必须要实现 WDM 规定的函数接口。从之前对 WDM 的分析中可以得知，函数接口中最重要的是 AddDevice 函数。 AddDevice函数有2个参数：DriverObject和PDO。DriverObject就是当前驱动程序的一个实例，PDO是物理设备对象（由总线驱动创建）。根据 MSDN 的说明，AddDevice函数主要工作就是创建一个功能设备对象FDO，然后附加在传递进来的PDO上面。AddDevice 函数的基本流程如下图所示：



**图 2.2 AddDevice 函数基本流程**

**Figure 2.2 Basic Flow of AddDevice Function**

同时，WDM支持PNP（即插即用），所有WDM驱动都需要设置PNP IRP的派遣函数，在入口函数DriverEntry里面设置的。Pnp派遣函数有2个参数：fdo和Irp。Fdo就是由AddDevice函数创建的那个功能设备对象，Irp是I/O管理器传进来的一个包（I/O Request Packet）。

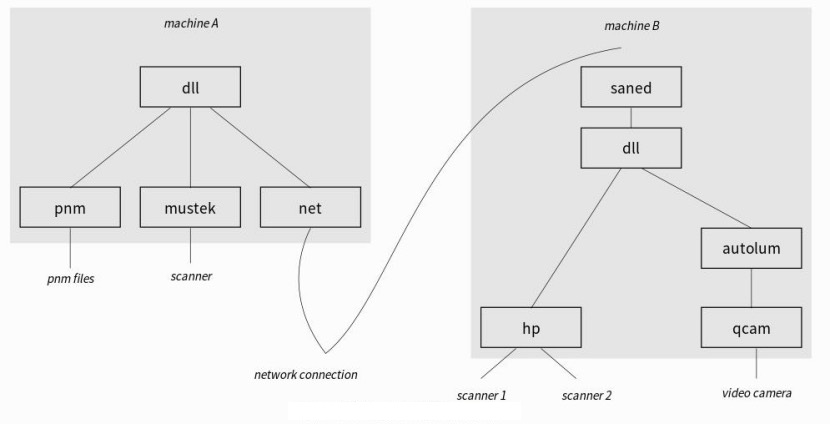
厂商的驱动程序可以通过调用 CreateFile()，ReadFile()，WriteFile() 和DeviceIoControl() 等操作来访问该内核驱动程序。其中 ReadFile() 和 WriteFile() 用于块数据传输。 具体来说，使用 ReadFile() 来从驱动中获得图像数据，并使用 WriteFile() 将命令作为数据流发送到接受的设备。

**2.3 Linux 扫描仪系统**

因为本文要在 Wine 中设计实现 Windows 的扫描仪兼容框架，所以对Linux 中的扫描仪系统的分析是必不可少的。

**2.3.1 SANE 协议**

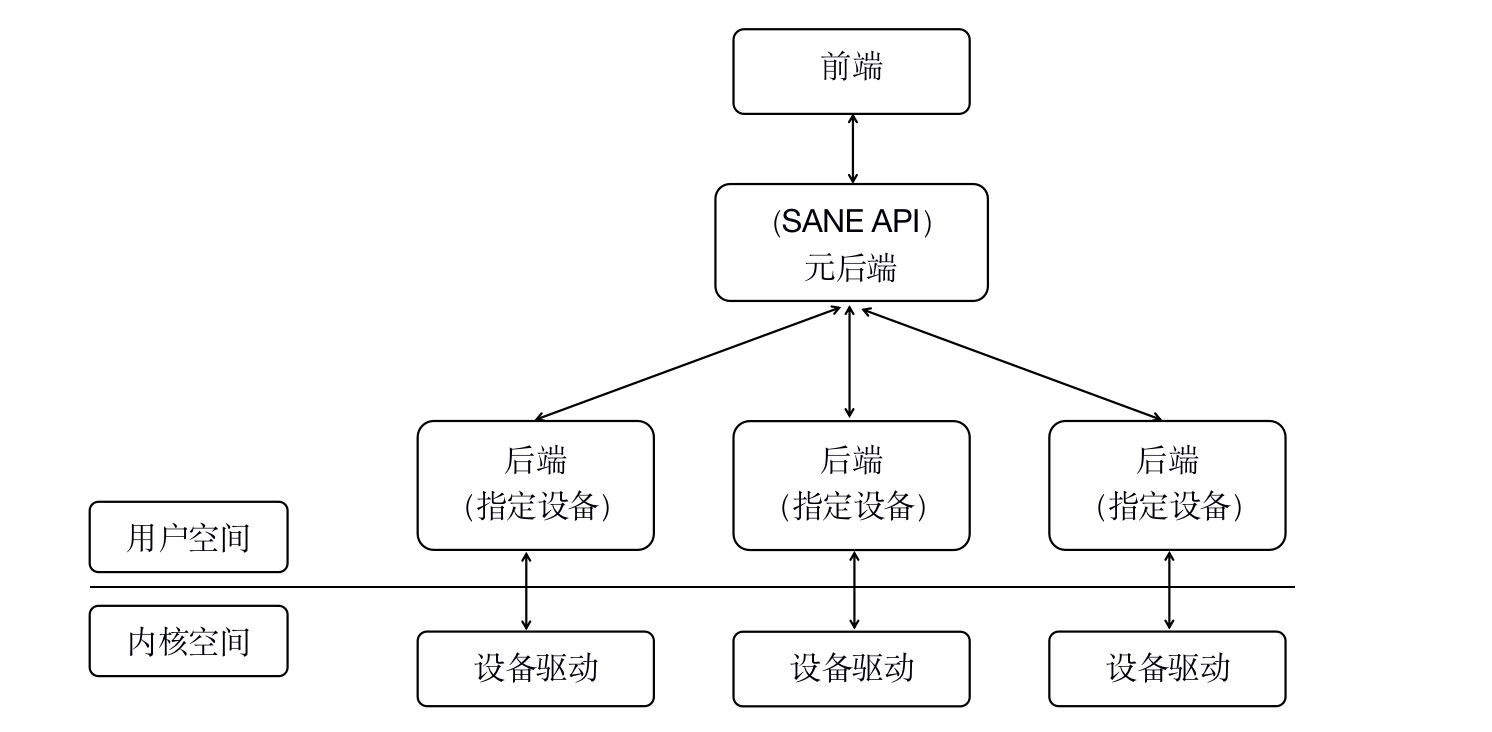
SANE是一个主要用在 Linux 的协议，它是“Scanner Access Now Easy”的首字母缩略词。SANE分为前端和后端，清晰的结构可以让它方便地实现应用和驱动所需的各种API，满足扫描仪硬件和应用所需的所有功能。前后端分离的架构如下图所示：



**图2.6 SANE 分层**

**Figure 2.6 SANE Hierarchy**

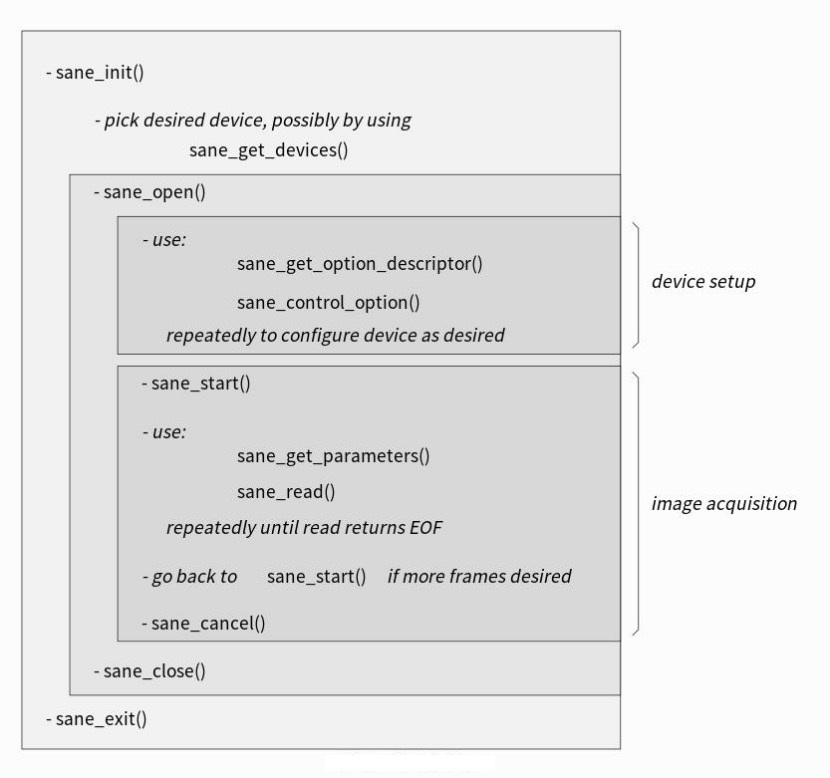
这样分离的好处是可以轻松进行网络扫描，而无需在前端或后端进行任何特殊处理。在装有扫描仪的主机上，只需要 Demon Process运行并处理网络请求。 在客户端计算机上，应用程序（驱动前端）连接到远程主机来获取扫描仪选项，并执行预览和扫描。SANE 的元 API 层负责请求的转发。它根据配置文件，把前端对 SANE API 的请求，转发到配置文件中指定的 SANE 后端去，达到控制目标设备的目的。元后端的转发过程如下图所示：



**图 2.7 SANE 请求转发**

**Figure 2.7 SANE Request Forwarding**

和 TWAIN 的状态转移类似，SANE 协议中规定了代码的执行流程，要求应用必须按照一定的顺序进行调用。如下图所示：



**图 2.7 SANE 代码流**

**Figure 2.7 SANE Code Flow**

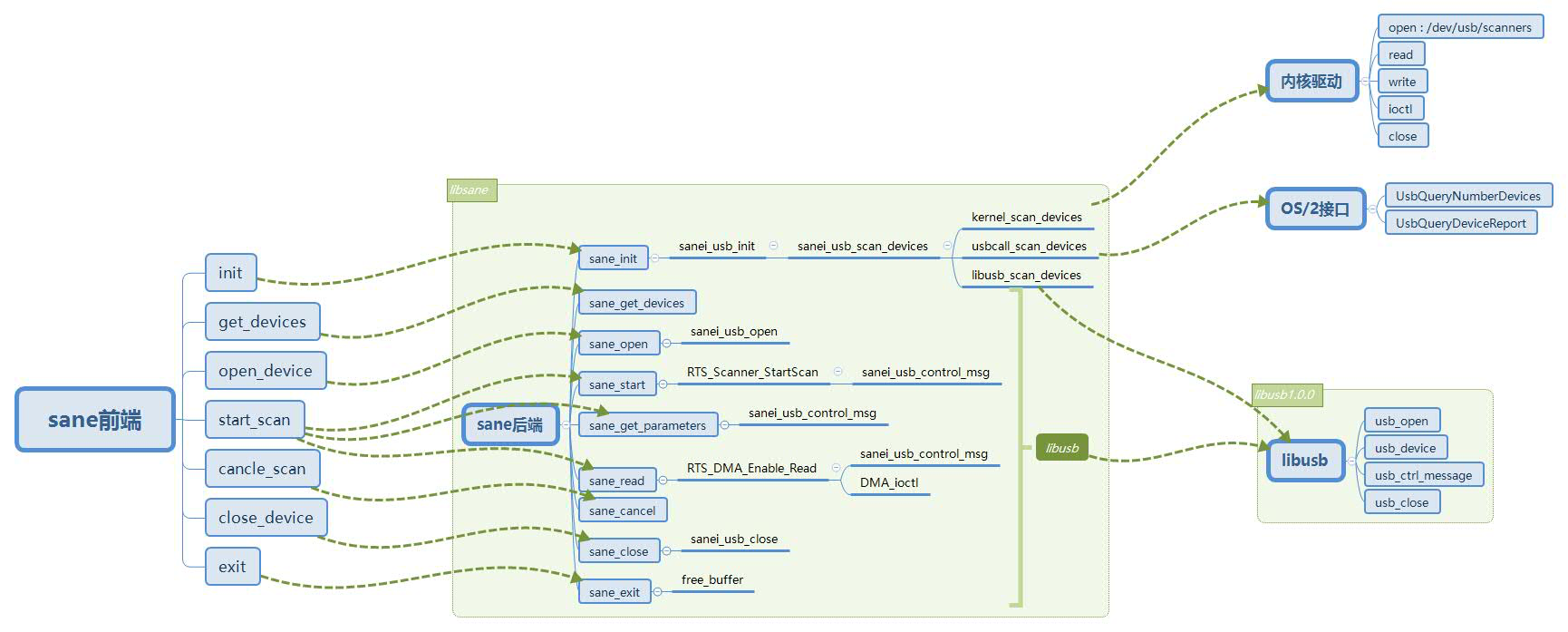
函数sane\_init() 和sane\_exit() 分别进行初始化和退出 SANE 后端。所有其他调用必须在初始化之后，退出后端之前执行。中间的操作分为两类，设备控制函数和图片扫描函数，在 SANE 前端中可以根据功能需要进行调用。

**2.3.2 Sanei 和 Libusb**

SANE 后端是扫描仪厂商提供的用户态模块，需要依赖 Linux 的库进入内核态来完成与设备的交互。

Sanei 是 SANE 官方提供的一个支持跨平台 USB 访问的库，可以被所有的 SANE 后端使用。它基于不同平台的底层库进行封装，避免 SANE 后端的代码中需要对不同平台进行兼容，而产生大量的 #ifdef 之类的代码。在 Sanei 库中使用最为广泛的是 sanei usb 函数库，在该库中对 USB 库进行了封装，提供了 sanei\_usb\_init()，sanei\_usb\_open() 等设备控制函数，还提供了sanei\_usb\_read\_bulk()，sanei\_usb\_write\_bulk() 等设备读写函数。设备控制函数的封装避免了在 SANE 后端中维护设备状态带，读写端点等来的复杂度。而设备读写函数的封装，避免了在 SANE 后端中需要处理的内存对齐，内存溢出等情况。可以看到 Sanei 为 SANE 后端的开发带来了极大的便利。

Sanei 的跨平台特性大部分是通过利用 Libusb 提供的跨平台功能来完成的。Libusb 是一个C语言库，它提供对 USB 设备的常规访问。它的目的是让开发人员简化和 USB 硬件通信的过程。Libusb是跨平台可移植的，它使用统一的与平台无关的API，可以支持Linux，macOS，Windows等平台上的USB设备；同时它也是用户态的，这样应用程序与设备进行通信时不需要获取超级权限；它支持多版本的 USB 协议，从1.0到3.1都可以支持。除了利用 Libusb 之外，Sanei 还会直接调用 OS 提供的内核驱动的接口，read()，write()，ioctl() 之类的函数。其函数对应关系如下图所示：

  
**图 2.8 Sanei Usb 实现方式**

**Figure 2.8 Sanei Usb Implementation Method**

**2.4 Wine 设备框架**

经过开源社区和前人的探索，Wine 目前已经对一些设备提供了兼容，包括 Ukey，HID设备，打印机等。在兼容这些设备时，Wine 除了提供某种设备专属的模块外，也模拟 Windows 提供了一些所有设备都需要的共性支持。Wine 以Linux中的设备和驱动访问框架为基础，在兼容系统通用模拟机制的支撑下，提供即插即用管理、设备I/O框架、会话管理等兼容环境设备及驱动管理公用机制，并向上提供设备属性操作接口、设备文件访问接口等，实现跨设备类的共性支撑。Wine 的设备框架如下图所示：

****

**图 2.4 Wine 设备框架图**

**Figure 2.4 Wine Device Frame**

由上图可见，在 Wine 中，为了能够让Windows应用程序能够感知到设备的插入、拔出事件，能够获得设备基础的属性信息，基于Linux平台的即插即用功能，模拟了 Windows 平台的即插即用机制及访问接口。当 Windows 应用程序访问 libusb 等 Windows 上的用户态通用设备访问库，并且 Linux 系统上实现了（或经过局部扩展即可实现）功能与之相对应的动态库时，Wine 会根据Windows平台动态库接口的规定，封装实现该类设备的Linux平台模拟库；当应用程序以设备文件或其它系统调用的方式与设备进行交互，并且现有Linux平台设备支持软件已经完全实现了（或经过局部扩展即可实现）交互功能所需机制，但未提供应用所需的访问接口时，Wine 会基于Linux平台上相关机制，在系统中提供用户态的模拟驱动框架以及I/O调度响应机制；当应用程序的有效执行依赖于一个闭源的只能在内核态加载的Windows内核驱动模块时，则 Wine 会在Linux平台上实现内核态的Windows驱动框架；当Windows兼容支持组件或必须要使用的Windows闭源驱动模块需要调用Windows内核态接口时，Wine 会根据其执行上下文，在Linux用户态或内核态提供Windows内核态接口的模拟实现。

**2.4.1 Wine 的会话管理**

Windows中会话管理服务是系统最重要的基础服务，会话管理机制非常强，系统中硬件设备、输入输出设备、系统服务一般都属于会话0，代表用户在系统中运行的进程属于其它会话，系统通过一套会话管理接口管理会话信息，并管理进程和设备的会话归属。因此，对于Windows应用程序的设备访问操作而言，只有实现有效的会话管理机制及会话管理接口，才有可能实现预期的设备访问。

在 Windows 中每个会话都有独立的 Session ID，标识独立的 Namespace，会话中创建的进程有 Session ID 标识，不可跨 Namespace 进行访问。在Windows中，一个会话是由代表单个用户登录会话的所有进程和系统对象组成的。其中的对象包括所有的窗口，桌面和Windows Stations。一个Session可以包含多个Windows Station，而每个Windows Station又可拥有多个桌面。

在兼容系统的本方案中，将在建立系统基本的会话管理相关数据结构的基础上，通过辨别系统服务、用户代理进程，确定进程和设备的会话归属，并依托会话管理相关的API的预期功能定义，实现会话管理相关的API接口，为后面设备访问操作奠定共性基础。通过在 Wine 程序初始化过程中添加 Session 划分，建立不同的Session目录，来确定系统中不同进程的会话归属，并完善一些现阶段必需的会话相关的API接口。目前 Wine 中的会话管理设计思路是，对比Windows多用户会话管理机制，先实现服务进程和用户进程的分离，后面逐渐完善对多用户分隔的实现。并且在Wine Server端将相关内核对象的数据结构进行重新设计，添加上有关 Session 的标识来进行 Windows Stations以及窗口的区分。

**2.4.2 Wine 即插即用机制**

Windows提供了设备即插即用功能，插入设备后自动完整设备信息注册、设备对象构建、驱动匹配和设备事件广播等功能。Wine中仅实现了部分HID的设备对象构建，多数功能没有实现，尤其是设备事件的传递。

在Linux中的即插即用管理模块是udev，Window世界的即插即用功能要生效，就必须与udev机制建立对接，然后Windows世界以此为基础建立自己的设备即插即用支持框架。对比 Windows 系统中即插即用的机制，在兼容系统中的方案中，Linux 上 Windows 即插即用技术兼容如下图所示。



**图 2.4 Wine 即插即用**

**Figure 2.4 Wine Plug Play**

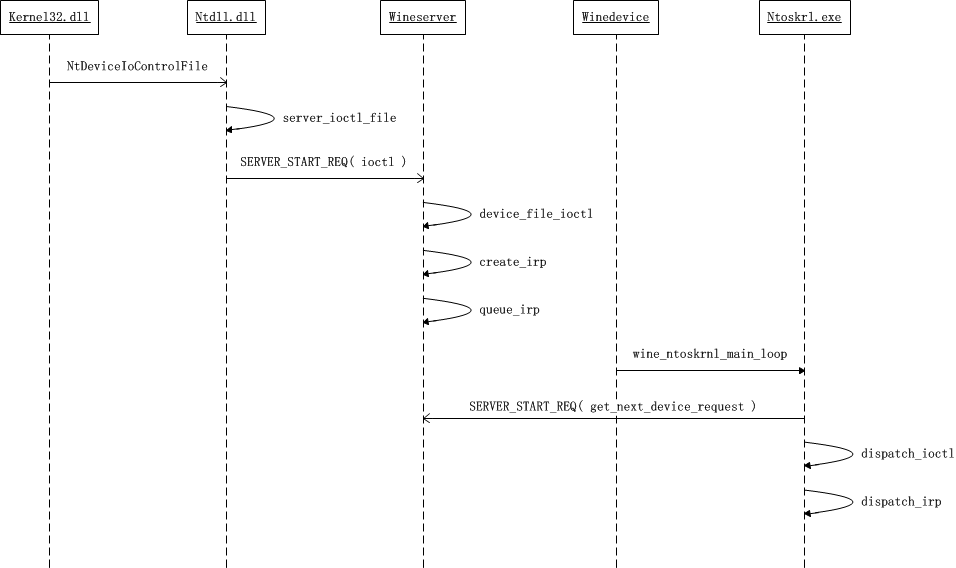
窗口和服务通过接口 RegisterDeviceNotification 向 PlugPlay 注册自己，通知 PlugPlay 需要接收某一类的设备信息模拟 Windows 总线驱动，基于 Linux的 Libudev 监听设备插拔，获取设备详情，通过服务 RPC 告诉 PlugPlay，由 PlugPlay进行记录和注册，并向已经注册的窗口或服务进行事件分发 PlugPlay 属于系统服务，所以需要实现服务管理器及其对应的 SetupAPI 来实现注册和事件传递的 RPC 过程。最终实现在 Linux 上 Windows 即插即用的兼容效果。

**2.4.3 Wine 设备 IO 管理**

计算机由各种设备组成，这些设备向外界提供输入和输出（I / O）。 典型的设备是键盘，鼠标，视频控制器，磁盘驱动器，网络端口等。设备驱动程序提供设备与操作系统之间的软件连接。I / O功能对设备驱动程序非常重要。

Windows内核模式提供 I / O管理器，用来管理应用程序与设备驱动程序之间的通信。由于设备的运行速度可能与操作系统不匹配，因此操作系统和设备驱动程序之间的通信主要是通过I / O请求数据包（IRP）完成的。 这些数据包类似于网络数据包或 Windows 消息数据包。它们从操作系统传递到特定的驱动程序，并从一个驱动程序传递到另一个驱动程序。

Wine的I/O系统组件可以划分为Wine Server、Wine Device 和 Wine Bus 三部分，其中 Wine Server 提供了I/O管理器的功能，Wine Bus在总线级别提供了设备自动识别功能，Wine Device提供了对内核驱动的加载功能，Wine Bus和Wine Device一起模拟了即插即用管理器功能。Wine 的设备 IO 过程如下图所示：



**图 2.4 Wine IO 流程**

**Figure 2.4 Wine IO Process**

由上图可见，Kernel32.dll 暴露出 DeviceIoControl 接口。若设备类型为非文件类型的设备，该接口调用 Ntdll.dll 的NtDeviceIoControlFile，该函数调用server\_ioctl\_file函数向 Wine Server发送Socket请求，将该I/O请求发送给Wine Server进行处理。Wine Server 将接受到的I/O请求封装成 IRP，并将该 IRP 加入消息队列中。Wine Device 服务启动后，会轮询的向 Wine Server 请求 IRP 进行处理，若 IRP 队列为空，进行阻塞。若Wine Server向队列里插入 IRP 时，队列为空，则发送消息唤醒正在阻塞的 Wine Device 进程。Wine Device获取Wine Server产生的 IRP 后，将 IRP 分发给相应设备的驱动程序进行处理。

**2.5 Wine 已兼容设备及兼容方法**

设备的兼容相较于单纯应用软件的兼容更为复杂，因为涉及到的模块从用户态到内核态都有分布，所以需要一个完整的方法进行指导，否则容易陷入误区导致兼容不具有通用性。

**2.5.1 Ukey**

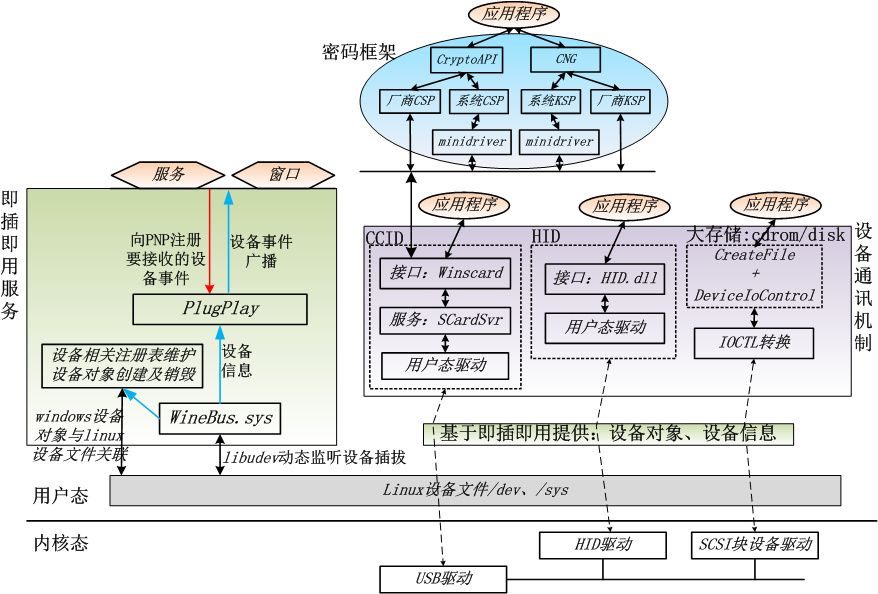
**2.5.2 打印机**

**2.5.3 HID**

**2.5.4 Ndiswrapper**

目前的兼容系统经过之前长期的探索和尝试，已经在 Wine中对多种设备成功兼容。包括CCID类 Ukey 设备，飞天诚信Ukey 设备和打印机等，这些设备并分别采用了不同的兼容方法。本文的研究内容是对扫描仪设备的兼容，所以也需要一个完整的兼容方法作为指导。

CCID 类的 Ukey 设备的兼容方法，是在 Wine中基于 Linux 上的开源 PCSC lite 开源软件实现了 Winscard 服务，提供了对无个性化驱动的 CCID 类设备的兼容。但是这种方法对于有个性化内核驱动模块的设备还是无法支持。其兼容框架如下图所示：

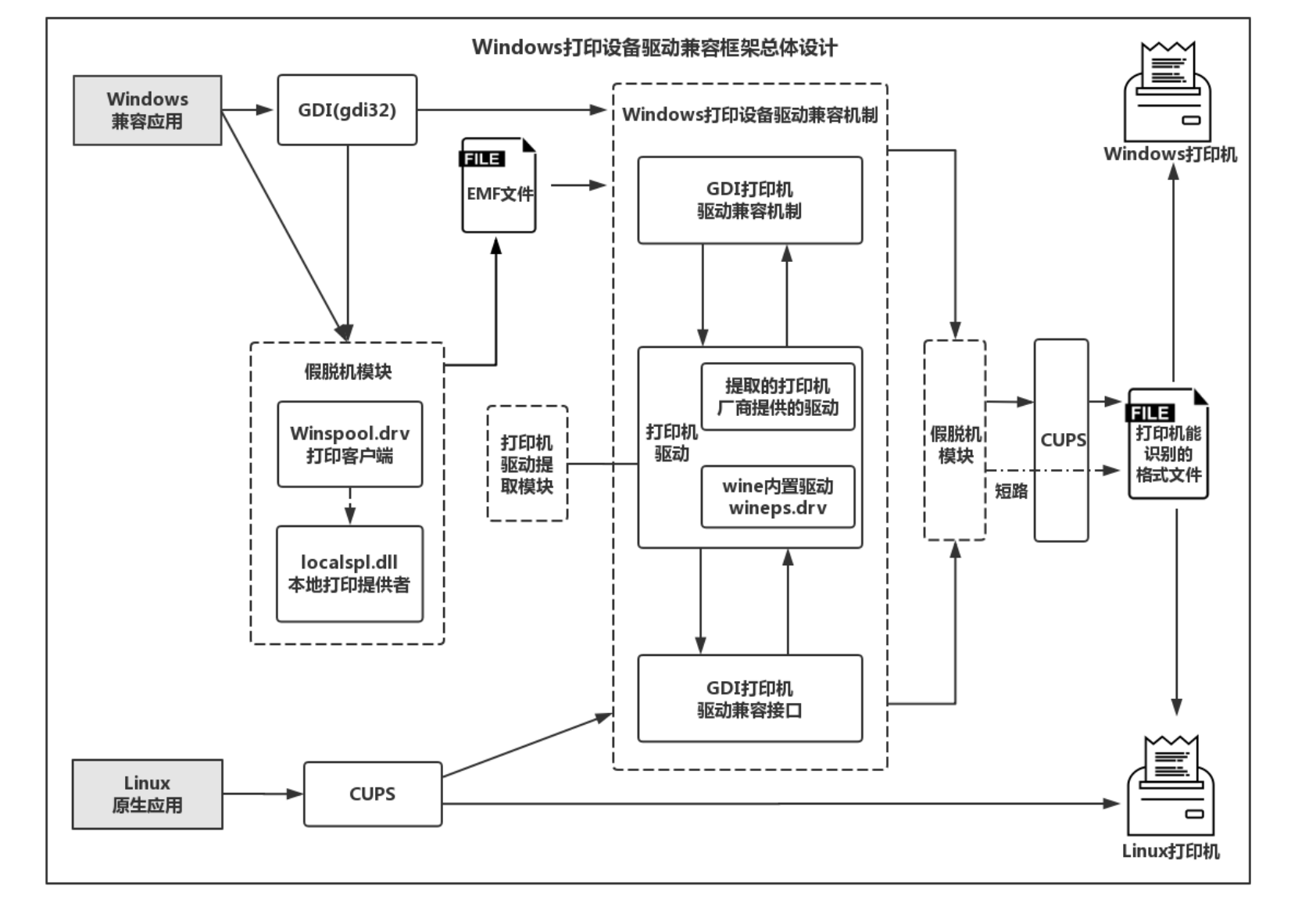


**图 2.10 CCID Ukey 兼容框架**

**Figure 2.10 CCID Ukey Compatible Framework**

飞天诚信的 Ukey 设备，属于有个性化内核驱动的设备，无法使用上一种方法进行兼容。但因为厂商恰好提供了匹配度很高的 Windows SDK 和 Linux SDK，所以通过对 Linux SDK 封装，模拟了设备应用软件所需的 Windows SDK， 实现了对驱动的兼容支持。但是这种方法强依赖于设备的 SDK，对同类设备不具有通用性。

打印机设备较为复杂，Windows 和 Linux 都为打印机设备设计了自己的工作框架，但他们的工作方式是不同的，这就给兼容工作带来了极大的困难。打印机的兼容框架见下图所示：



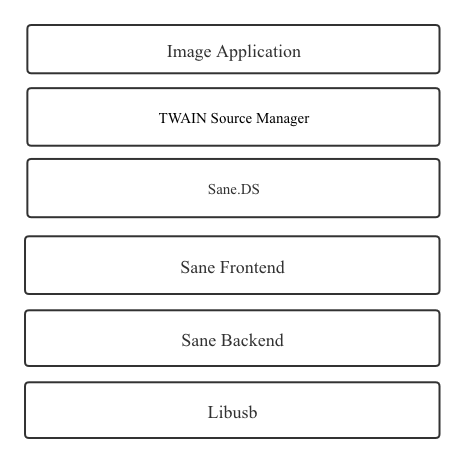
**图 2.11 打印机兼容框架**

**Figure 2.11 Printer Compatible Framework**

在上图中可以看到，兼容框架中实现了假脱机模块，打印机驱动提取模块，GDI 打印驱动兼容接口等部分。兼容框架在 Wine 中基于 CUPS 框架模拟实现了 Windows 打印机访问接口，把 Windows 的打印机数据流转换为 Linux 的数据流，实现了兼容支持。

**2.6 Wine 扫描仪框架**

Wine 中的扫描仪框架是利用了 Linux 中现有的功能，来模拟 Windows 的扫描仪框架。Wine 的扫描仪框架的层次结构如下图所示：



**图 2.9 Wine 扫描仪框架结构**

**Figure 2.9 Wine Scanner Frame Structure**

Wine 为了支持 Windows 的应用，必须要支持 TWAIN 协议。但是 TWAIN 协议所需要的 Data Source 目前 Wine 没有办法兼容。所以 Wine 采用了另一种方式，从上图中可以看出，Wine 的扫描仪框架使用 Sane 协议中的前端和后端，模拟了 TWAIN 协议中的 Data Source。将应用对 TWAIN 协议的请求转换为了对 SANE 协议的请求，所以对于有 SANE 后端的扫描仪来说，可以较好的支持。但是对于没有 SANE 后端的扫描仪，目前的兼容方式是完全不能支持的。

同时，Wine 对 TWAIN 协议中 TWAIN Data Source Manager 的兼容也存在种种缺陷，比如不支持回调，不维护设备状态等，造成大多数的 Windows 应用也是无法使用的。

**2.7 相关技术分析**

Windows 的扫描仪系统较为复杂，主要有协议，厂商驱动，扫描仪驱动，USB 驱动组成。清晰的模块划分让极大减小了厂商驱动的开发负担，让厂商只需要专注于扫描逻辑的开发，而不需要关注底层的 USB 通信逻辑，因此 Windows 的扫描仪数量远远多于 Linux 扫描仪的数量，且扫描功能更为丰富。而这也为在 Linux 兼容扫描仪带来了困难，不仅要兼容厂商的驱动，还需要兼容 Windows 的扫描仪驱动和 USB 驱动。

Linux 的扫描仪系统相对 Windows 而言比较简单，仅提供了 Sanei 库对 USB 的简单封装，所以厂商驱动不仅要处理自身扫描仪的逻辑，还需要处理大量的 USB 通信的逻辑，这给厂商的驱动开发带来了很大的复杂度。因此对同一款扫描仪，Linux 驱动支持的功能往往比 Windows 驱动要少很多。

Wine 的扫描仪系统是为了兼容 Windows 的扫描仪系统而设计开发的。所以 Wine 中也实现了 TWAIN 协议，但 Wine 的实现残缺不全，不能满足应用的需求。同时因为 Wine 中完全没有 USBSCAN，厂商的 Windows 驱动是无法使用的。但是Wine 用了另一种方法尝试兼容扫描仪。它没有使用厂商提供的扫描仪驱动 Data Source，而是用 SANE 协议模拟出了 SANE.DS 作为 TWAIN 的 Data Source。这种兼容方式对于支持了 SANE 协议的扫描仪来说是可以正常工作的，但对于仅支持 TWAIN 协议的扫描仪是不行的。同时，模拟的驱动相比于厂商的驱动往往会有许多功能的缺失，造成扫描仪厂商的应用无法正常工作。

通过对 Wine 中已经兼容的设备及设备兼容方法的分析，可以看出目前的设备兼容方法不全面，并缺乏归纳总结。导致无法从目前已兼容设备中找出适用于扫描仪的设备兼容方法，为扫描仪的兼容提供方法性指导。但是通过对 Windows 驱动模型和 Wine 设备框架的对比分析，可以看出 Wine 的设备框架已经部分模拟了 Windows 的驱动模型，并提供了会话管理，即插即用机制管理和设备 I/O 管理等功能，虽然目前还没有完整的模拟 Windows 的 WDM 框架，但是对我们归纳总结设备兼容方法已经提供了充分的帮助。

**2.8 本章小结**

本章主要分析了各个平台上设备支持框架和扫描仪相关的研究，及当前已兼容的设备和设备兼容方法。

首先分析了 Windows 扫描仪系统和 Linux 扫描仪系统。在 Windows 扫描仪系统中，首先分析了整个 Windows 扫描仪框架，然后分别分析了 TWAIN 协议，STI 框架，扫描仪个性化驱动和 USBSCAN 内核态驱动，这些内容都与后面的兼容设计密切相关。Linux 扫描仪系统首先分析了 Linux 的扫描仪框架， 然后分析了 SANE 协议，Sanei 和 Libusb。这些都是后面兼容框架中可以借鉴的内容。

然后分析了 Wine 的扫描仪框架。Wine 对于使用 SANE 协议的扫描仪驱动有一定程度的支持，但支持十分有限，功能残缺。对于 Windows 上使用 TWAIN 协议的驱动完全无法支持。

接着分析了已经兼容的设备和设备兼容方法。包括 HID，CCID Ukey，飞天诚信 UKey 和打印机 Ndiswrapper 等。这些设备的兼容过程对本文设计扫描仪的兼容方法具有重要的借鉴意义。但是这些设备的兼容方法缺乏总结归纳，无法从中看出各种方法的优缺点，很难对扫描仪的兼容提供方法上的指导。

之后对比分析了 Windows 的 WDM 驱动模型和 Wine 中的设备框架，从中可以看出 Wine 的设备框架已经部分模拟了 Windows 的 WDM 驱动模型，可以为我们实践探索、归纳总结设备兼容方法提供充分的帮助。

最后对上文的相关技术进行总结和分析，提出本文的研究价值。

1. **扫描仪兼容框架关键组件的设计和实现**

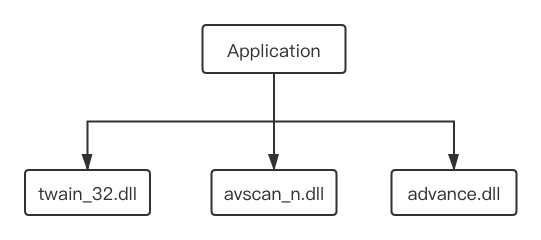
为了实践探索、归纳总结出更多的设备兼容方法，本章中将根据上一张中对相关技术及问题的分析，设计和实现扫描仪兼容框架中必须依赖的关键组件。首先是分析出扫描仪兼容框架必须依赖的关键组件，之后对每个关键组件，描述其设计和实现方式。下一章在这些关键组件的基础上，探索和实践多种设备兼容方法，并对这些方法进行总结归纳。

**3.1 关键组件分析**

设备兼容时，厂商提供了两个兼容目标：应用和驱动。通过对这两个兼容目标的依赖进行层层分析，就可以得出扫描仪兼容框架中的关键组件。在本文中，以 S6100 型号的扫描仪为例，进行分析。

**3.1.1 扫描仪应用的兼容依赖分析**

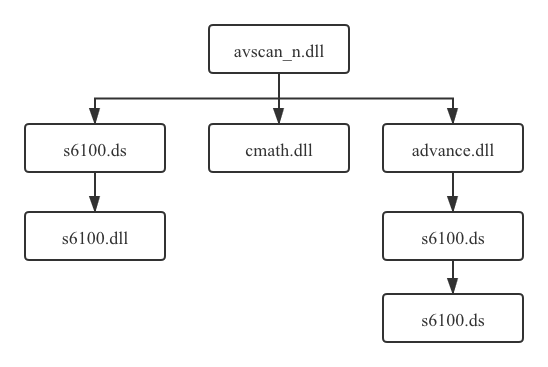
应用的依赖主要指的是应用对系统 DLL 和厂商 DLL 的依赖关系。通过使用 API Monitor 和 Dependence 分析两款工具对 Windows 应用完整的启动和扫描流程进行监控，可以从中得出应用加载的 DLL，进而得到依赖关系。对 S6100 扫描仪来说，其应用的依赖关系如下图所示，该应用会依赖 Windows 系统的 twain\_32.dll 和厂商的 avscan\_n.dll，advance.dll。



**图 3.1 应用直接依赖 DLL**

**Figure 3.1 DLL directly dependent**

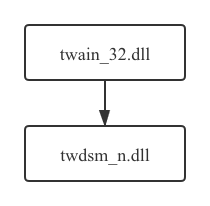
进一步对 avscan\_n.dll 和 advance.dll 的依赖关系进行分析，可以得到如下图所示的依赖关系，经过再 Wine 中进行实验，可以得到图像处理相关的数学库 cmath.dll 等可以正常兼容。所以排除掉图像处理相关的 DLL 后，该部分主要的依赖对象是 S6100.ds 和 S6100.dll。

****

**图 3.2 间接依赖 DLL**

**Figure 3.2 Indirectly Dependent DLL**

再对 twain\_32.dll 的依赖关系进行分析，可以看出其依赖关系比较简单，只依赖了 twdsm\_n.dll，如下图所示：



**图 3.3 Twain 依赖关系**

**Figure 3.3 Twain Dependencies**

通过对 TWAIN 官网资料的查阅，可以得知 twain\_32.dll 和 twdsm\_n.dll 都属于 TWAIN 协议规范的内容，不需要拆分为两个 DLL 来分别进行兼容，按照 TWAIN 协议整体进行兼容即可。

通过以上的分析，可以得出扫描仪应用的所依赖的关键组件分别是 TWAIN 协议，S6100.dll 和 S6100.ds。进一步通过对 S6100.ds 中的内建函数的功能分析，可以大致得出 S6100.ds 中主要是暴露 SANE 协议 DS\_Entry 入口函数，而没有其他作用。其功能实现主要依赖于 S6100.dll，同样也是 S6100.dll 负责和下层系统进行通信的，所以 S6100.dll 和 S6100.ds 也可以合并为一个整体来看待，即扫描仪驱动 Data Source。

**3.1.2 扫描仪驱动的兼容依赖分析**

由应用分析可知，扫描仪驱动 Data Source 不会依赖额外的厂商组件。所以扫描仪驱动的兼容需求分析主要指的是，驱动对系统模块的依赖分析。

同样使用 API Monitor 进行对完整的扫描过程进行函数调用的分析，从其部分的函数调用中可以看到 CreateFile 函数的调用，且目标文件名是 USBSCAN。如下图所示：

再结合第二章中对 Windows 扫描仪系统的分析，Windows 提供了两种对扫描仪的访问方式，这里的访问特征恰好符合第二种方式，即通过文件系统对内核态驱动的访问。

由此进行合理推断，扫描仪驱动依赖于 Windows 的内核模块 USBSCAN。

**3.1.3 依赖的关键组件总结**

通过对应用的兼容依赖分析，和对驱动的兼容依赖分析，可知必须要兼容的模块分为三个部分：TWAIN 协议，扫描仪 Data Source 驱动，Windows 的 USBSCAN 内核模块。

下一节中将分别阐述这三个部分的设计和实现方案。

**3.2 TWAIN 协议兼容的设计和实现**

TWAIN 协议的兼容主要指的是 TWAIN Data Source Manager 的兼容。其兼容目标，是为了在符合 TWAIN 协议标准的基础上，尽可能的满足应用的需求。

**3.2.1 TWAIN 协议的分析和设计**

**3.2.2 TWAIN 协议的实现**

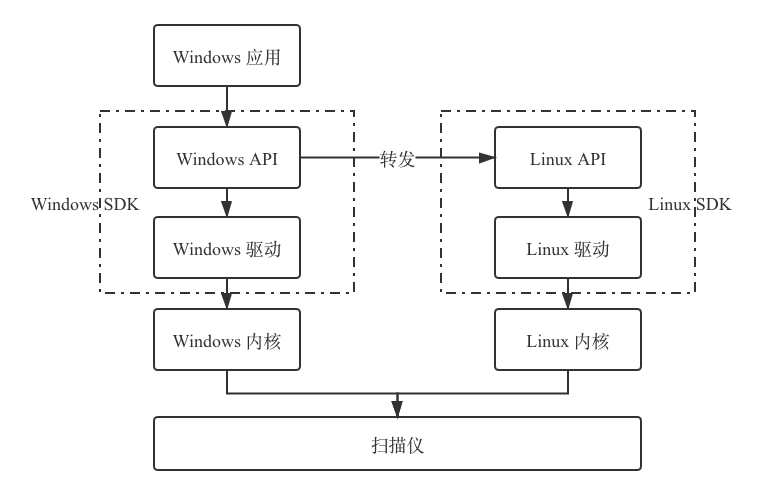
**3.3 Data Source 驱动兼容的设计和实现**

Data Source 的兼容目标很清晰，是为了暴露出 DS\_Entry 接口接受上层 TWAIN 协议的调用，并完成 TWAIN 协议标准中规定的相应的操作。为了达到这一个兼容目标，我们有多条路可走。

以 S6100 扫描仪为例。该型号扫描仪厂商同时提供了 Windows SDK，Linux SDK，Windows 驱动和 Linux 驱动四种驱动形式，因此我们就可以把每种驱动形式封装为 Data Source 的形式，来完成 Data Source 的兼容。

**3.3.1 Linux SDK 模拟 Windows SDK**

在兼容时，把对 Windows SDK 的调用转发到 Linux SDK 即可满足兼容的需求。结构设计如下图所示：

****

**3.3.2 Linux 驱动模拟 Data Source**

**3.3.3 SANE 驱动模拟 Data Source**

**3.4 USBSCAN 内核模块兼容的设计和实现**

1. **扫描仪兼容方法及兼容框架设计**

本章将根据上一章中对相关技术及问题的分析，提出一种新的设备兼容方法，并根据该方法设计 Wine 中的扫描仪的兼容框架。首先是总结现有的兼容方法，使用各个兼容方法分别设计出扫描仪的兼容框架，并根据这些不同的兼容框架分析各方法的优点及缺点。之后根据分析结果提出一个全新的设备兼容方法及对应的扫描仪兼容框架。

**3.1 现有设备兼容方法**

从之前的分析中可以看出，目前的设备兼容方法整体上可以分为两类：Linux SDK模拟 Windows 驱动，和 Linux 驱动模拟 Windows 驱动。在扫描仪场景下，Windows 驱动即指的是 TWAIN 中的 Data Source。

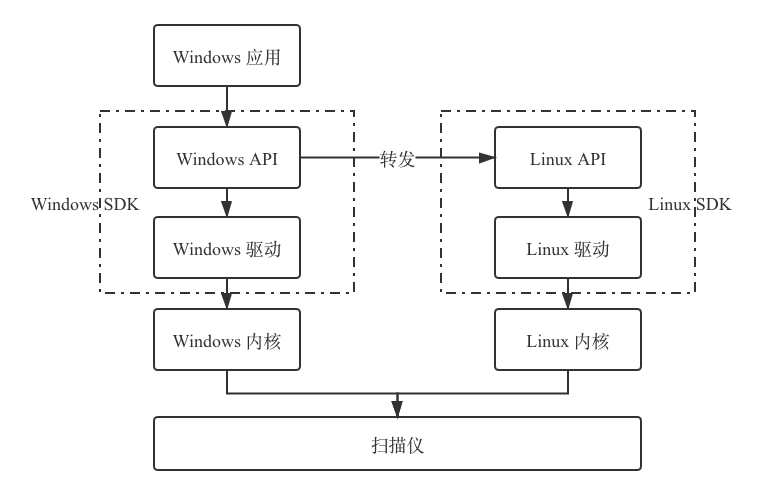
**3.2 现有各方法的兼容框架设计**

本节中将根据上一节中介绍的两类兼容方法，分别设计兼容框架对扫描仪进行兼容，并对比分析该两类兼容方法的优劣。

**3.2.1 Linux SDK 模拟 Windows SDK**

通常，设备厂商提供的是 Windows 应用和 Windows SDK，Windows 应用基于 SDK 开发，这就要求兼容的时候应用和 SDK 同时兼容。但是有些厂商会额外提供 Linux SDK，给我们的兼容带来极大便利。

以 S6100 扫描仪为例，厂商提供了 Windows 应用，Windows SDK，Linux SDK。在兼容时，把对 Windows SDK 的调用转发到 Linux SDK 即可满足兼容的需求。结构设计如下图所示：

****

**图 3.1 SDK 转发兼容方案设计**

**Figure 3.1 Design of SDK Simulation Compatible**

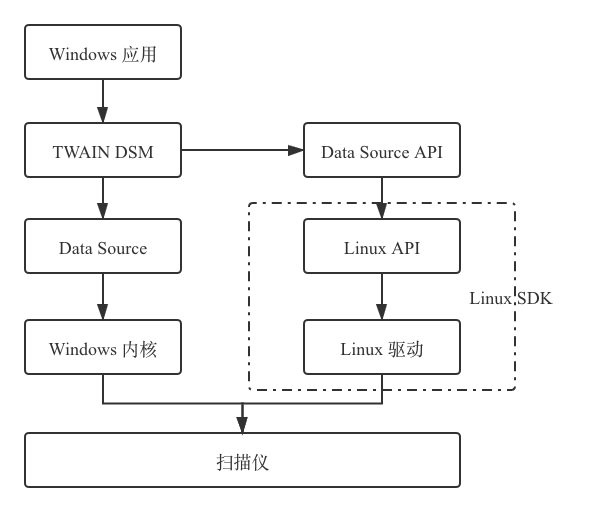
从上图中可以看出，该兼容方案是把应用对 Windows SDK 的 API 的请求，在兼容系统中将其转发到 Linux SDK 的 API 中，直接摆脱了对 Windows 驱动和 Windows 内核的依赖。使用该方法进行兼容可以达到应用对底层驱动的变化完全无感知，应用的使用方式、操作流程等都不需要改变，可以达到最稳定、理想的兼容效果。

但同时，该方法因为强依赖于 SDK 也会有很大的局限性。首先，厂商提供的 SDK 往往是以一款型号的设备专用的，即使是同一个厂商的另一个型号的设备也无法使用。以方德扫描仪为例，该厂商对 S6100 扫描仪和 KS8130 扫描仪分别提供了各自的 SDK，使用 SDK 转发的兼容方式就需要对两种型号的扫描仪 SDK 分别进行模拟，带来了额外的工作量，不具备通用性。此外，由之前的分析可知，因为 Linux 对扫描仪支持的匮乏，厂商提供的 Linux SDK 的功能相较于 Windows SDK 也比较少，使用 SDK 转发的方法虽然可以成功兼容应用，但是应用中的很多功能无法满足。

**3.2.2 Linux SDK 模拟 Data Source**

对于扫描仪，大多数 Windows 应用都是基于 TWAIN 协议的，所以只要扫描仪的 Data Source 可以用，所有 Windows 应用都是可以使用这款扫描仪的。这对我们兼容也带来了一定的便利，当厂商的应用有一些独特的依赖没办法兼容时，可以使用另一款已经容易兼容的应用来使用扫描仪。

SDK 模拟 Data Source 的结构设计如下图所示：



**图 3.2 SDK 模拟 DS 兼容方案设计**

**Figure 3.2 Design of SDK Simulation DS Compatibility**

TWAIN Data Source Manager（DSM）对 Data Source（DS） 的访问是通过 dlopen() 的方式加载的，所以 DSM 和 DS 是完全隔离的，只需要用 SDK 模拟出 DS 的 API 即可被 DSM 访问。从图中可以看出，Windows 应用在访问 DSM 时，在兼容系统中把 DSM 对 Windows DS 的请求直接转发到 Linux 中模拟的 DS 中，而该 DS 的下层功能是基于 Linux 的 SDK 实现的。因为 DS 的功能是完全由 DS\_Entry() 中的三元组确定的，具有清晰统一的标准，所以用 SDK 模拟 DS 时也是有统一的标准可以参考，可以满足应用的需求。

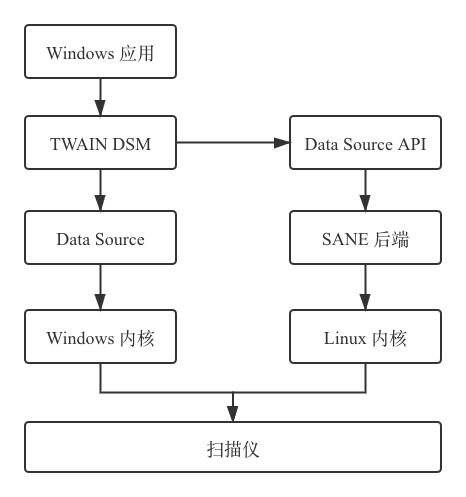
虽然该方法在一定程度上解决了 Windows 难以兼容的问题，但该方法也是强依赖于 SDK 的，也会面临和上一小节中相同的问题和局限性。

**3.2.3 Linux SANE 驱动模拟 Data Source**

有些设备厂商在提供了 SDK 之外，还会额外提供 Windows 驱动和 Linux 驱动来方便用户使用。基于驱动进行兼容会让该兼容方式的通用性大大提高。

对扫描仪来说，Windows 驱动指的是 TWAN Data Source（DS），Linux 驱动指的是 SANE 后端。因为 TWAIN 协议和 SANE 协议都对驱动的接口做了明确的规定，所以每款扫描仪的驱动接口都是一致的，基于驱动进行兼容可以极大提高通用性。

SANE 驱动模拟 Data Source 的结构如下图所示：

****

**图 3.3 SANE 后端模拟 DS兼容方案设计**

**Figure 3.3 Design of SANE Back-end Simulation DS Compatibility**

从上图中可以看出，该兼容方法和上一小节中的兼容方法类似，Windows 应用在访问 DSM 时，在兼容系统中把对 Windows DS 的请求转发到我们模拟的 DS 中。不同之处是，DS 不再是使用 SDK 模拟的，而是使用厂商提供的 SANE 后端进行模拟的。

因为 SANE 后端的接口是由 SANE 协议规定的，所以所有扫描仪的 SANE 后端都有这相同的接口，这让我们兼容设计的通用性得到了极大的提升，这种兼容方式可以让所有具有 SANE 后端的扫描仪被一次性兼容，极大减小了兼容的工作量。

但同时，该兼容方法对扫描仪来说也存在一些问题。从之前的分析可以得出，厂商提供的扫描仪 Linux 驱动相比 Windows 驱动功能较为匮乏，因此通过该方式兼容的 SANE 后端并不能实现 Data Source 所需要的所有功能，造成应用的部分功能无法使用。

**3.3 兼容内核模块的设备兼容方法**

从前面的的分析可以看出，对扫描仪来说，现有的兼容方法所面临的主要问题，是 Linux 驱动相比 Windows 驱动本身有诸多功能缺陷，而这些缺失的功能往往和设备命令强相关，是我们没办法去模拟的，只能使用厂商提供的 Windows 驱动。这就造成现有的使用 Linux 驱动模拟 Windows 驱动这条路不可行，必须要采取可以直接兼容 Windows 驱动的兼容方法。

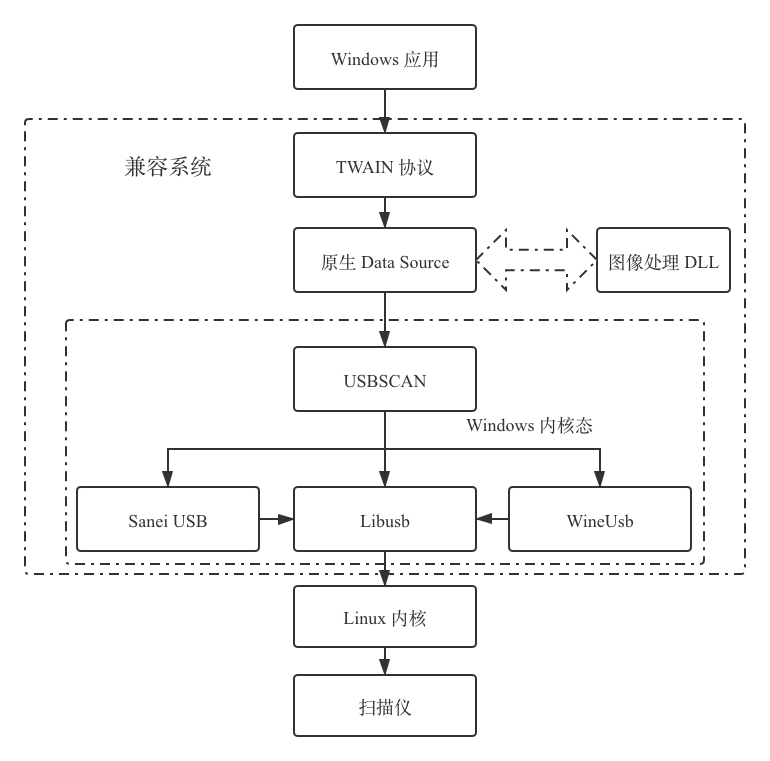
因为 Windows 的驱动都会依赖 Windows 的内核模块，所以兼容 Windows 的内核模块成为必不可少的一部分。因此，在本文中提出一种新的设备兼容方法，兼容内核模块的设备兼容方法。该方法通过兼容 Windows 内核模块达到兼容 Windows 原生驱动的目的。

**3.4 兼容内核模块的扫描仪兼容框架设计**

本节中将使用兼容内核模块的设备兼容方法对扫描仪兼容框架进行设计，并给出框架中各个模块的详细设计和设计思路，最后对该方法和现有方法进行对比，并对该方法的优劣进行总结。

**3.4.1 总体兼容方案**

为了使 Windows 驱动能够在兼容系统中成功运行，本文需要处理的部分包括 Windows 应用的兼容，应用所使用的扫描协议的兼容，Windows 内核模块的兼容，Windows 设备通信的兼容。在借鉴第二章中介绍的 Windows 扫描框架，Linux 扫描框架和 Wine 扫描框架的基础上，设计了兼容内核模块的扫描仪兼容框架，总体架构如下图所示：

****

**图 3.4 内核模块兼容框架设计**

**Figure 3.4 Design of Kernel Module Compatible Framework**

从上图中可以看出，整个框架在兼容系统中把 Windows 应用的请求转交给 Linux 内核来进行处理。框架主要包括以下几个部分：

1. TWAIN 协议。该扫描协议在 Wine 中重新实现，而没有直接采用开源的实现，因为开源的实现中会引入复杂的依赖问题，其中多数依赖目前在 Wine 中尚未支持，无法直接使用。同时重新实现 TWAIN 协议可以对其中的功能进行精简和优化，只保留需要的功能，并可以在协议中增加对设备状态的缓存来减少对设备的请求，提高响应速度。

2. 原生 Data Source。该部分指的是从扫描仪厂商提供 Windows 的驱动中提取出来的 Data Source 模块，该模块是以 DLL 的形式出现的，所以可以被 Wine 直接加载和调用。该兼容框架需要做的只是从 Windows 驱动中提取出来即可。

3. USBSCAN 内核模块。该模块指的是对 Windows 内核中 USBSCAN 模块的模拟实现。USBSCAN 接受 Data Source 对的请求，并负责把请求加工处理后转发给下层的 USB 模块。该模块负责处理 Windows 设备管理，包括写注册表，创建软连接，加载驱动等，而不负责 USB 通信等逻辑。

4. Sanei USB、Wineusb 模块。USBSCAN 的部分功能依赖这两个模块实现。其中 Sanei USB 是开源 SANE 协议提供的底层通信库，主要负责维护设备列表和调用 libusb 进行底层通信。设备列表中包括每个设备对应的输入、输出和控制端口，并基于维护的这些信息提供了 init，open，close，read，write 等功能的封装，在这些功能的封装中，进行了大量的类型枚举，buffer 对齐，错误处理等操作，极大避免了上层维护设备信息带来的复杂性，方便了实现。Wineusb 是 Wine 5.0 提供的基于 libusb 的 Windows 内核驱动模拟模块，借助该模块实现设备的发现和监控，减轻

**3.4.2 Windows 兼容应用扫描方案**

**3.4.3 设备通信方案**

**3.4.3 USBSCAN 模块设计**

**3.4.4 USB 通信模块设计**

**3.5 本章小结**

1. **扫描仪兼容框架的实现**

**4.1 现有各方法的兼容框架实现**

**4.1.1 Linux SDK 模拟 Windows SDK**

**4.1.2 Linux SDK 模拟 Data Source**

**4.1.3 Linux SANE 驱动模拟 Data Source**

**4.2 新方法的兼容框架实现**