文章编号：2096-1472(2020)-08-20-05
DOI:10.19644/j.cnki.issn2096-1472.2020.08.006

基于Windows管理规范的机房信息可视化方案

严 强

(苏州市职业大学继续教育学院，江苏 苏州 215004)
✉clong320@126.com

摘 要：从机房管理实际工作出发，提出了日常机房管理中需要经常使用的系统信息和处理要求。给出了在Windows管理规范支持下的详细的同时兼容Windows 7、Windows 10和Windows Server的可视化设计方案，并运用C＃的窗体透明和嵌入桌面技术对此加以实现。本方案完成后首先在新建机房实施部署，经过一个多学期的使用，得到了上课教师和机房管理员的肯定和好评。目前正逐步在其余机房中部署。

关键词：Windows管理规范；机房管理；透明窗体；嵌入桌面；Windows服务
中图分类号：TP311 **文献标识码**：A

Computer Lab Information Visualization Project based on Windows Management Instrumentation

YAN Qiang

(*Continuing Education*, *Suzhou Vocational University*, *Suzhou* 215004, *China*)
✉clong320@126.com

Abstract: In view of system information and processing requirements that are often used in daily computer lab management, a computer lab information visualization project, which is compatible with Windows 7, Windows 10 and Windows Server under the support of Windows Management Instrumentation(WMI), is proposed and achieved by using C# form transparency and embedded desktop technology. When completed, it was first deployed in the newly-built computer labs. After a semester of trial, it was highly praised by both the lecturers and the computer lab administrators. It is now being gradually deployed in the remaining computer labs.

Keywords: WMI; computer lab management; transparent form; embedded desktop; windows service

1 引言(Introduction)

随着计算机技术和网络技术的不断发展和进步，机房管理系统技术及市场也日趋成熟和完善。从最初的纯硬件版、纯软件版到软硬件结合版、网络版，再到现在的服务器集中管理以及智慧教室。各种各样的管理系统层出不穷，但最终实现的功能都是最大程度地提高机房利用率和管理效率。

目前我院大部分机房采用的是软硬件结合的独立管理模式。管理系统将机房计算机硬盘分成若干个分区，在不同的分区中安装不同的操作系统和教学软件，满足不同专业和班级的教学及考试要求的。

根据实际管理，需要经常显示的信息包括机器号、机器名、IP地址、操作系统及类型、物理内存及可用内存、各逻辑分区的容量及剩余容量等信息，其中IP地址、可用内存以及逻辑分区剩余容量要求跟随变化。同时要求这些信息要始终显示在桌面的右上角，除文字以外其余空白处不能遮挡桌面上该位置的信息，要求显示的信息不受WIN+M键、WIN+D键、Alt+Tab键和任务栏右侧的“显示桌面”按钮控制。

2 方案确立(Project establishment)

2.1 确定显示方案

(1)需求分析要求显示文字以外的空白处不能遮挡桌面上的信息，因而确定将软件窗体设置为背景透明、窗体上的控件也设为背景透明，且为无边框窗体。

(2)要求窗体不受WIN+M键、WIN+D键、Alt+Tab键和任务栏右侧的“显示桌面”按钮控制。考虑方案有两种：

方案一：使用全局键盘钩子和鼠标钩子，在窗体的全局键盘钩子中过滤WIN+M键、WIN+D键、Alt+Tab键和Alt+F4键，鼠标钩子捕捉任务栏右侧的“显示桌面”区域并过滤。

方案二：将窗体嵌入到桌面里作为桌面的一个子窗体，因为桌面不受WIN+M键、WIN+D键、Alt+Tab键和Alt+F4键控制，所以桌面中的子窗体也不受这些键控制。

考虑到方案一要截获系统的键盘和鼠标消息，由于穷尽不了各种应用软件和考试系统环境下的测试，所以该方案存在未知风险。方案二不截获任何系统消息，不存在上述风险，安全性高于方案一，所以软件采用方案二。

2.2 数据获取方法

需求分析中提出的各种信息的获取，查阅了相关资料和网站，发现获取机器名及IP地址的方法有多种，获取和硬件相关数据，如物理硬盘、物理内存等的方法相对比较集中。结合本人掌握的编程语言，数据获取方法采用调用Win32应用程序编程接口[1−3](Application Programming Interfaces，API)和Windows管理规范[4,5](Windows Management Instrumentation，WMI)混合使用方法。

3 方案设计(Project design)

3.1 总体设计

总体结构设计如图1所示。软件由操作系统识别、Windows 10处理模块、Windows 7处理模块、Windows Server处理模块和WMI本机应用程序编程接口模块组成。Windows Server处理模块显示的信息和格式与Windows 10和Windows 7不同，且该模块设计为通用各种Windows Server操作系统。以上各模块通过WMI本机应用程序编程接口模块调用WMI核心结构获取数据。

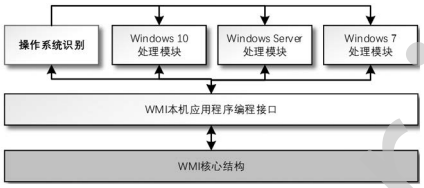

图1 系统总体结构图

Fig.1 System overall structure diagram

3.2 详细设计

(1)Windows 10模块

模块功能设计如图2所示。在Windows 10下获取并显示的数据有机器编号、机器名、IP地址、操作系统和类型、物理内存和可使用内存、物理硬盘数量及容量、逻辑分区及容量和分区剩余容量。刷新定时器用于监视IP地址、可使用内存以及逻辑分区剩余容量的实时变化。

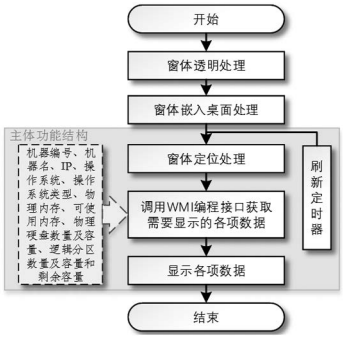

图2 Windows 10 处理模块

Fig.2 Windows 10 processing module

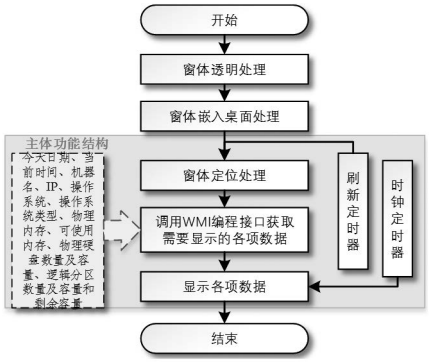

图3 Windows Server 处理模块

Fig.3 Windows Server processing module

(2)Windows Server模块

模块功能设计如图3所示。本模块用于作为服务器使用的Windows Server系统(目前使用的是Windows Server 2012 R2)，有别于作为学生机使用的Windows 7、Windows 10系统。主体功能结构中的“时钟定时器”用于同步显示日期和时间。

(3)Windows 7模块

模块功能设计如图4所示。

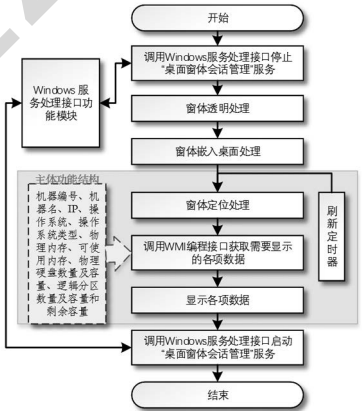

图4 Windows 7 处理模块

Fig.4 Windows 7 processing module

在Windows 7操作系统中，经过透明处理后的窗体嵌入桌面前，须要停止Windows的“Desktop Window Manager Session Manager”服务。因而在本模块中增加一个“Windows服务处理接口功能”子模块，该子模块的功能是提供停止、启动、复位已经安装的某项Windows服务的接口功能，即通过该子模块去停止、启动和复位一个指定的Windows服务。同时该子模块也提供查询某项Windows服务是否已经安装。本软件结束运行前重新启动该服务。模块的主体功能结构与处理和Windows 10的相同。

4 方案实现(Solution realization)

4.1 使用WMI获取相关数据

(1)WMI编程接口

WMI是Windows Management Instrumentation缩写，又称Windows管理规范。WMI的结构体系如图5所示。

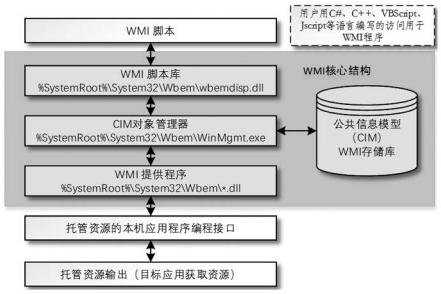

图5 WMI 结构体系

Fig.5 WMI structure system

由图5可见WMI的核心结构由四部分组成：①WMI脚本库，它的位置是%SystemRoot%\System32\Wbem\wbemdisp.dll。②公共信息模型对象管理器(Common Information Model Object Manager-CIMOM)，是描述Windows操作系统构成单元的对象数据库。它的位置是%SystemRoot%\System32\Wbem\WinMgmt.exe。③公共信息模型(CIM)和WMI存储库。④WMI提供程序，它是一组DLL文件。位置是%SystemRoot%\System32\Wbem*.dll。

一般②③④被称为CIM库。所以通常将WMI看成有CIM库和WMI脚本对象两部分组成。

(2)WMI使用方法及步骤

WMI使用一般分为四步：①创建WMI对象脚本库的指针实例。②调用实例方法，连接到CIM库，同时指明要访问资源的逻辑地址。③获取托管资源，即WMI托管资源(类)的实例集合。④遍历实例集合，获取相关属性值的数据。

(3)C#使用WMI方法

在VS2015的解决方案管理器[6-8]的项目中添加引用System.Management，并添加名字空间using System.Management。

ManagementObject或ManagementClass：是公共信息模型(CIM)管理对象或类。管理类是一个WMI类，如Win32_PhysicalMemory和Win32_LogicalDisk，前者是物理内存后者是逻辑分区。

ManagementObjectSearcher：查询检索指定的对象的集合。也可以用来枚举由ManagementObject或ManagementClass获取的对象集合。

下面是获取逻辑分区及分区容量实现代码：

```
#region获取逻辑分区及分区容量
/// <summary>
/// 获取逻辑分区及分区容量
/// </summary>
/// <returns></returns>
public List<HardDiskPartition> GetDiskListInfo()
        {
List<HardDiskPartition> hdp_list = null;
// 定义一个逻辑分区结构列表
Try   //指定分区的容量信息
            {
// WQL 查询类 SQL查询 WMI类关键字“Win32_LogicalDisk”
SelectQuery selectQuery = new SelectQuery("select * from Win32_LogicalDisk");
// WMI查询管理对象集合，用上述查询结果实例化
ManagementObjectSearcher searcher = new ManagementObjectSearcher(selectQuery);
// 定义 WMI 类对象集合，并获取查询结果(即：获取所有逻辑分区集合)
ManagementObjectCollection diskcollection = searcher.Get();
if (diskcollection != null && diskcollection.Count > 0)   // 逻辑分区数量大于0
                {
hdp_list = new List<HardDiskPartition>();
// 给逻辑分区结构列表分配空间
HardDiskPartition harddisk = null;
// 定义一个逻辑分区结构列表临时变量
foreach (ManagementObject disk in diskcollection)
// 变量获取到的逻辑分区集合
                    {
int nType = Convert.ToInt32(disk["DriveType"]);
// 获取硬盘类型标志值
if (nType != Convert.ToInt32(DriveType.Fixed)) continue;// 硬盘标志为”Fixed”(固定硬盘)
                        else
                        {
harddisk = new HardDiskPartition();
// 给临时变量分配空间
// 获取剩余空间并转成GB(保留两位小数)
harddisk.FreeSpace = ToGB(Convert.ToDouble(disk["FreeSpace"]), 1024.0, 2);
// 获取分区总空间并转成GB(保留两位小数)
harddisk.SumSpace = ToGB(Convert.ToDouble(disk["Size"]), 1024.0, 2);
// 获取逻辑分区名称(如：C:、D:、E:)
harddisk.PartitionName = disk["DeviceID"].ToString();
hdp_list.Add(harddisk);            // 加入列表
harddisk.SetSumFreeSpace(Convert.ToDouble(disk["Size"]), Convert.ToDouble(disk["FreeSpace"]));
                        }
                    }
                }
            }
            catch (Exception) {  }
            return hdp_list;
        }
        #endregion
```

4.2 实现窗体嵌入桌面

4.2.1 识别窗体

Windows操作系统有一套自成的窗体管理机制，实现对窗体结构、窗体类型、窗体状态、窗体间关系、窗体消息等进行管理。

在Windows操作系统运行过程中，用户每打开一个窗体，操作系统都会给这个窗体设定一个唯一的标识符号用于管理，以此区分其他窗体。这个唯一的标识符号称之为句柄(Handle)，是一个唯一的数值。程序员可用通过Microsoft提供的Win32应用程序编程接口(Application Programming Interfaces，API)和窗体类名来获取已经打开的窗体句柄，通过对句柄的操作来完成由句柄对应的窗体的操作。

4.2.2 相关API函数

本设计中主要相关的API函数：

FindWindow：根据窗体类名和窗体标题寻找窗体列表中第一个符合指定条件的顶级窗体，获取该窗体的句柄。

FindWindowEx：根据父窗体句柄和子窗体类名在窗体列表中寻找与指定条件相符的第一个子窗口。

GetParent：根据子窗体句柄获取其父窗体句柄。

SetParent：给窗体重新指定一个父窗体。本软件就是使用此函数实现将窗体嵌入桌面的。

4.2.3 桌面结构

要能够实现将窗体用SetParent函数嵌入桌面，首先就要能获取到桌面的句柄。在Windows中，桌面也是作为窗体来管理的，可以通过FindWindow函数获取到桌面句柄。

(1)桌面组成

要使用FindWindow函数获取到桌面的句柄，就要知道桌面的窗体标题和它的类名。通过查阅资料，得到了桌面的窗体标题和类名分别是“Program Manager”和“Progman”。这样使用FindWindow函数可以轻松获取到桌面窗体Progman的句柄。

为了更多的了解Windows桌面组成，以及Windows 7、Windows 10、Windows Server的桌面结构是否相同。使用EnumWindows和EnumChildWindows等API函数做了一个工具程序，用它枚举出系统中的所有窗体，并根据窗体之间的父子关系整理并保存到树形列表控件TreeView中，因此桌面的组成结构一目了然，分别在Windows 7、Windows 10和Windows Server 2012中运行，得到如图6所示的桌面窗体结构。

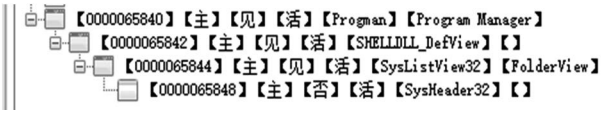

图6 Windows桌面窗体结构

Fig.6 Windows desktop form structure

从图6所示的Windows桌面结构来看，桌面的顶层父窗体是Progman，窗体标题是Program Manager，窗体可见且透明；第一层子窗体是SHELLDLL_DefView，没有窗体标题，窗体可见且透明；第二层子窗体是SysListView32(是包含桌面图标的窗体，该窗体实际上是ListView列表控件)，窗体标题是FolderView，窗体可见且透明；第三层子窗体是SysHeader32，没有窗体标题，窗体不可见。由此可见并非是网上和一些资料上说的三层结构，而是四层结构，且在Windows 7、Windows 10和Windows Server 2012 R2中具有相同的结构。

(2)选择嵌入的桌面窗体

根据图6的桌面结构，从前到后的窗体排列顺序是SysHeader32、SysListView32、SHELLDLL_DefView、Progman。可见，除不可见窗体SysHeader32外，其余三个窗体均可作为本软件窗体的父窗体。实际编程中选择了Progman窗体作为本软件窗体嵌入的父窗体，因为该窗体是最后一层窗体，嵌入后不会影响和干扰软件窗体所在位置的上层窗体的操作。

(3)窗体透明处理

C＃的窗体控件中有两个属性用于控制窗体透明的，Opacity和TransparencyKey。前者用透明度来控制窗体透明，取值是0%—100%，窗体中的控件随之一同透明。后者是窗体根据设定的关键颜色透明，窗体中的控件不随之透明。

this.Opacity = 0;窗体完全透明，窗体中所有控件均透明，不可见。

this.Opacity = 1;窗体完全不透明(常规状态)。

this.TransparencyKey = color.Red;窗体和窗体中的控件的红色部分被透明掉。

this.TransparencyKey = this.BackColor;窗体按照背景颜色透明。本设计使用此方式设置窗体透明，同时还须将窗体中的控件的背景颜色设置为Color.Transparent。

(4)透明后窗体嵌入桌面

窗体透明并嵌入桌面，只需要在窗体加载事件的合理位置执行下面两条命令即可。

this.TransparencyKey = this.BackColor;

SetParent(this.Handle，hDesktopWnd);

可是在实际的测试运行中并非如此。在Windows Server 2012 R2系统下窗体嵌入正常。

在Windows 10系统中，执行这两条命令后，窗体可以嵌入桌面，但窗体不能完全透明，窗体区域呈无规律的花屏状态。经过多次调试，最终按如下处理后可以正常透明嵌入桌面。

①设计状态将窗体的BackColor属性设为透明颜色；

②窗体的Form_Load事件中执行this.TransparencyKey = this.BackColor;

③窗体的Form_Resize事件中执行SetParent(this.Handle，hDesktopWnd);

在Windows 7操作系统中，执行这两条命令问题比较大，每次运行后软件窗体消失不见。按照在Windows 10中的方法处理后依然如故。使用断点单步调试运行，得到关键点的数据和变量状态全部正常。后来注解掉this.TransparencyKey = this.BackColor;，再次运行，在桌面的右上角可以看到非透明状态的程序窗体。取消注解后再次运行，窗体再次消失。可见问题就出在这条命令上。

(5)问题的解决

问题既然出现了，那么就要去找解决问题的方法。上网一查，发现有许多帖子都显示遇到同样的问题，也有些帖子给出一些解决方法，但是在Windows 7下都解决不了窗体透明嵌入桌面的问题。也有一些帖子认为这是Windows 7系统或Visual Studio本身问题，问题看似无解了。

经过多天毫无目标头绪的修改，问题仍然没有得到解决。近期机房有一门课程需要在Windows 10系统中安装VMware虚拟机，并要求虚拟一个Windows 7系统。安装完成后，抱着试试看的想法将软件拷贝到虚拟机的Windows 7系统中，双击运行，眼前一亮，桌面右上角透明窗体显示出现了。看来并非如网上帖子所说那样无法实现。于是仔细对照虚拟机的Windows 7系统和实体Windows 7系统的各项设置和服务，对比后发现Desktop Window Manager Session Manager(桌面窗体管理对话管理)服务是影响窗体透明嵌入桌面的根本原因。

解决方法是：软件在Windows 7系统中运行时，在窗体嵌入桌面前，通过程序停止Desktop Window Manager Session Manager服务，软件结束运行前再启动该服务。

4.3 VS2015中C#操控Windows服务接口的部分代码

```
#region 停止一项服务
/// <summary>
/// 停止一项服务
/// </summary>
/// <param name="servicename">服务名称</param>
public void StopService(String servicename)
        {
            try
            {
sController = new ServiceController(servicename);
            // 服务控制变量指向对应的服务
if (sController.Status != ServiceControllerStatus.Stopped)        // 判别此服务项是否是停止状态
                {
sController.Stop();
                // 停止服务
sController.WaitForStatus(ServiceControllerStatus.Stopped);        // 等待服务停止
sController.Close();
                // 切断控制变量与服务的联系
                }
            }
            catch { }
        }
#endregion
#region 启动一项服务
/// <summary>
/// 启动一项服务
/// </summary>
/// <param name="servicename">服务名称</param>
public void StartService(String servicename)
        {
            try
            {
sController = new ServiceController(servicename);
// 服务控制变量指向对应的服务
if (sController.Status == ServiceControllerStatus.Stopped)// 判别此服务项是否是已运行状态
                {
sController.Start();
                // 启动服务
sController.WaitForStatus(ServiceControllerStatus.Running);        // 等待服务启动完成
sController.Close();
                // 切断控制变量与服务的联系
                }
            }
            catch { }
        }
#endregion
```

5 结论(Conclusion)

基于WMI的机房信息可视化方案在本校机房管理中应用后收到了一定效果。管理人员和任课教师通过它能够了解并掌握每个分区系统信息，快速判别本分区是否满足教学或考试要求。因此该方案在一定程度上帮助了机房管理员提高服务质量和管理效率。

参考资料(References)

[1] Windows API参考手册[DB/OL].http://www.office-cn.net/book/api/214.html,2017-8-24.

[2] 李泉.现代API通往架构师之门[M].清华大学出版社,2018.

[3] 范文庆.Windows API开发详解——函数、接口、编程实例[M].人民邮电出版社,2011.

[4] WMI简介[DB/OL]. https://docs.microsoft.com/zh-cn/windows-hardware/drivers/kernel/introduction-to-wmi,2017-6-16.

[5] Marcin Policht.智慧东方工作室,译.WMI技术指南[M].北京:机械工业出版社,2002.

[6] Nagel C.,Evjen B..李铭,译.C#高级编程(第7版)[M].北京:清华大学出版社,2010.

[7] 明日科技.C#项目开发实战入门(全彩版)[M].吉林:吉林大学出版社,2017.

[8] Bruce Johnson.张卫华,裴洪文,译.Visual Studio 2015高级编程(第6版)[M].北京:清华大学出版社,2016.

作者简介：

严 强(1962-)，男，本科，工程师.研究领域：工业控制，数据通信.