УДК 004.622

Использование инструментария управления Windows (WMI) для получения системной информации с применением техник объектно-ориентированного программирования

Королёнок Александр Юрьевич
Приамурский государственный университет им. Шолом-Алейхема
Студент

Аннотация
В данной статье рассмотрен способ взаимодействия с Windows Management Instrumentation в удобной, объектно-ориентированной форме, что позволяет достичь безопасности типов и гибкости расширения функционала. Для реализации метода применялись LINQPad – интерактивная программная утилита для разработки приложений и объектно-ориентированные техники программирования – Factory Method, OCP, LSP. Данное решение позволяет использовать объекты WMI в более удобной для разработчика форме, существенно снижая вероятность возникновения ошибок, и предоставляет простой способ расширения необходимого функционала.
Ключевые слова: Windows Management Instrumentation, паттерны проектирования, фабричный метод, принцип открытости-закрытости, принцип подстановки Лисков, объектно-ориентированное программирование, рефлексия типов, информация о системе.

Using Windows Management Instrumentation (WMI) for getting system information with applying object-oriented programming techniques

Korolenok Alexandr Yurievich
Sholom-Aleichem Priamursky State University
Student

Abstract
This article consider how to interact with Windows Management Instrumentation in a convenient, object-oriented manner to achieve type safety and extensibility. To implement the method, LINQPad is used - an interactive software utility for developing applications and object-oriented programming techniques - Factory Method, OCP, LSP. This solution allows you to use WMI objects in a more developer-friendly way, significantly reducing the likelihood of errors and provides an easy way to extend the functionality you need.
Keywords: Windows Management Instrumentation, design patterns, factory method, open-closed principle, Liskov substitution principle, object-oriented programming, reflection, system information.

Научный руководитель:
Глаголев Владимир Александрович
Приамурский государственный университет имени Шолом-Алейхема
к.г.н., доцент, доцент кафедры информационных систем, математики и правовой информатики

1. Введение

1.1. Актуальность исследования

При реализации информационных систем бывает необходимо получить разного рода системную информацию, такую, например, как серийный номер жесткого магнитного диска, архитектуру операционной системы, характеристики оперативной памяти, или информацию об установленном программном обеспечении.

Инструментарий управления Windows предоставляет средства для получения практически любой информации о программном/аппаратном обеспечении системы. Сценариев[1] использования предоставляемой WMI информации бесчисленное множество: управление файловой системой, журналами операций, системным реестром, процессами и службами на удаленных компьютерах, проведения инвентаризации[4] установленного оборудования и программного обеспечения на рабочих станциях в локальной сети, сопоставления имеющегося и требуемого оборудования и/или программного обеспечения[3]. При этом стоит учитывать, что WMI, являясь высокоуровневым средством ОС Windows, может вызывать некоторые проблемы производительности[2].

Вместе с тем, сам способ использования инструментария, с точки зрения .NET разработчика имеет ряд недостатков.

В данной статье был описан метод использования WMI в более удобной для разработчика форме, который сможет гарантировать безопасность типов и гибкость расширения функционала путем применения объектно-ориентированных техник[5-8]

1.2. Цель исследования

Целью исследования является создание приложения, позволяющего получить любую системную информацию, доступную в WMI в виде готовых классов на языке C#, с обеспечением безопасности типов и возможности расширения функционала.

2. Методы исследования

Инструментом для достижения цели станет LINQPad – интерактивная программная утилита, для разработки приложений .NET Framework.

Компания Microsoft, на своем сайте предоставляет в распоряжение разработчика набор поставщиков информации Win32 в виде описания классов на языке C++ для получения системной информации в ОС Windows. Эти классы представляют из себя наборы данных соответствующих

объектов, по названию полей которых мы можем получить информацию средствами WMI.

В программе на языке C# для этого нужно создать объект `ManagementObjectSearcher` при создании которого в качестве аргумента конструктору необходимо передать запрос содержащий название интересующего нас класса поставщика Win32 и вызовом метода Get() получить объект `ManagementObject`, который и будет содержать интересующую информацию. К примеру, для получения информации о всех имеющихся дисковых накопителях можно написать следующий код (рис. 1)

```
public class Program
{
    static void Main()
    {
        var mos = new ManagementObjectSearcher($"SELECT * FROM Win32_DiskDrive");
        foreach (ManagementObject mo in mos.Get())
        {
            mo.Dump();
        }
    }
}
```

Рисунок 1 – пример использования объектов WMI

Существенный минус такого подхода заключается в том, что объекты типа `ManagementObject`, в силу своего общего назначения предоставляют информацию о полученных свойствах через индексатор типа `String`, выдавая информацию типа `Object`. Следовательно, нет строгого контроля типов и всегда существует вероятность опечатки, которая не будет воспринята как ошибка на этапе компиляции, а необходимое свойство не будет получено, и лишь по сообщению «Not found» исключения ManagementException можно будет сделать предположение о вероятной ошибке. Иными словами, требуется данной системе требуется гибкость в использовании любых объектов, предоставляемых WMI, строгость контроля типов и вместе с тем удобство использования получаемой информации.

С целью обеспечения строгости соблюдения контроля типов и удобства использования информации преобразуем в класс языка C# предоставляемый компанией Microsoft описание класса на языке C++ Win32_LogicalDisk (данное описание доступно по адресу: https://docs.microsoft.com/ru-ru/windows/win32/cimwin32prov/win32-logicaldisk).

Исходный код класса на языке C++, для объекта WMI «LogicalDrive» (рис. 2)

синтаксис Копировать

```
[Dynamic, Provider("CIMWin32"), SupportsUpdate, UUID("{8502C4B7-5FBB-11D2-AAC1-006008C78BC7}"), AMENDMENT]
class Win32_LogicalDisk : CIM_LogicalDisk
{
  uint16   Access;
  uint16   Availability;
  uint64   BlockSize;
  string   Caption;
  boolean  Compressed;
  uint32   ConfigManagerErrorCode;
  boolean  ConfigManagerUserConfig;
  string   CreationClassName;
  string   Description;
  string   DeviceID;
  uint32   DriveType;
  boolean  ErrorCleared;
  string   ErrorDescription;
  string   ErrorMethodology;
  string   FileSystem;
  uint64   FreeSpace;
  datetime InstallDate;
  uint32   LastErrorCode;
  uint32   MaximumComponentLength;
  uint32   MediaType;
  string   Name;
  uint64   NumberOfBlocks;
  string   PNPDeviceID;
  uint16   PowerManagementCapabilities[];
  boolean  PowerManagementSupported;
  string   ProviderName;
  string   Purpose;
  boolean  QuotasDisabled;
  boolean  QuotasIncomplete;
  boolean  QuotasRebuilding;
  string   Size;
  string   Status;
  uint16   StatusInfo;
  boolean  SupportsDiskQuotas;
  boolean  SupportsFileBasedCompression;
  string   SystemCreationClassName;
  string   SystemName;
  boolean  VolumeDirty;
  string   VolumeName;
  string   VolumeSerialNumber;
};
```

Рисунок 2 – C++ класс объекта WMI «Дисковый накопитель»

Преобразование данного класса на языке C++ в класс на языке C# сводится к замене названий типов данных на соответствующие типы (C#), замене имен полей на свойства, реализованные через лямбда-операторы, возвращающие явно приведенные к соответствующему типу свойства значения индексатора объекта ManagementObject (mo), который будет инициализироваться через конструктор.

Преобразованный класс Win32_LogicalDisk на языке C# будет выглядеть следующим образом (рис. 3)

```
public class Win32_LogicalDisk
{
    ManagementObject mo { get; set; }
    public Win32_LogicalDisk(ManagementObject mo) { this.mo = mo; }
    public ushort Access ⇒ (ushort)mo["Access"];
    public ushort Availability ⇒ (ushort)mo["Availability"];
    public ulong BlockSize ⇒ (ulong)mo["BlockSize"];
    public string Caption ⇒ mo["Caption"].ToString();
    public bool Compressed ⇒ (bool)mo["Compressed"];
    public uint ConfigManagerErrorCode ⇒ (uint)mo["ConfigManagerErrorCode"];
    public bool ConfigManagerUserConfig ⇒ (bool)mo["ConfigManagerUserConfig"];
    public string CreationClassName ⇒ mo["CreationClassName"].ToString();
    public string Description ⇒ mo["Description"].ToString();
    public string DeviceID ⇒ mo["DeviceID"].ToString();
    public uint DriveType ⇒ (uint)mo["DriveType"];
    public bool ErrorCleared ⇒ (bool)mo["ErrorCleared"];
    public string ErrorDescription ⇒ mo["ErrorDescription"].ToString();
    public string ErrorMethodology ⇒ mo["ErrorMethodology"].ToString();
    public string FileSystem ⇒ mo["FileSystem"].ToString();
    public ulong FreeSpace ⇒ (ulong)mo["FreeSpace"];
    public object InstallDate ⇒ mo["InstallDate"]; //DateTime
    public uint LastErrorCode ⇒ (uint)mo["LastErrorCode"];
    public uint MaximumComponentLength ⇒ (uint)mo["MaximumComponentLength"];
    public uint MediaType ⇒ (uint)mo["MediaType"];
    public string Name ⇒ mo["Name"].ToString();
    public ulong NumberOfBlocks ⇒ (ulong)mo["NumberOfBlocks"];
    public string PNPDeviceID ⇒ mo["PNPDeviceID"].ToString();
    public ushort[] PowerManagementCapabilities ⇒ (ushort[])mo["PowerManagementCapabilities"];
    public bool PowerManagementSupported ⇒ (bool)mo["PowerManagementSupported"];
    public string ProviderName ⇒ mo["ProviderName"].ToString();
    public string Purpose ⇒ mo["Purpose"].ToString();
    public bool QuotasDisabled ⇒ (bool)mo["QuotasDisabled"];
    public bool QuotasIncomplete ⇒ (bool)mo["QuotasIncomplete"];
    public bool QuotasRebuilding ⇒ (bool)mo["QuotasRebuilding"];
    public string Size ⇒ mo["Size"].ToString();
    public string Status ⇒ mo["Status"].ToString();
    public ushort StatusInfo ⇒ (ushort)mo["StatusInfo"];
    public bool SupportsDiskQuotas ⇒ (bool)mo["SupportsDiskQuotas"];
    public bool SupportsFileBasedCompression ⇒ (bool)mo["SupportsFileBasedCompression"];
    public string SystemCreationClassName ⇒ mo["SystemCreationClassName"].ToString();
    public string SystemName ⇒ mo["SystemName"].ToString();
    public bool VolumeDirty ⇒ (bool)mo["VolumeDirty"];
    public string VolumeName ⇒ mo["VolumeName"].ToString();
    public string VolumeSerialNumber ⇒ mo["VolumeSerialNumber"].ToString();
}
```

Рисунок 3 – C# класс Win32_LogicalDisk

Как видно из кода класса C# имя класса полностью соответствует классу WMI, а по имени свойства можно получить значение соответствующего типа данных.

Но всякий раз вручную преобразовывать класс на языке C++ в соответствующий класс C#, при всей родственной схожести языков, было бы как минимум не рационально. Поэтому для автоматизации процесса подобного преобразования напишем код получения по URL-адресу описания класса на сайте Microsoft.com в готовый класс на языке C#.

Для достижения гибкости в использовании любых объектов создадим класс для получения готовых классов на языке C# из соответствующих классов Win32.

Сначала определим общий интерфейс классов, которые будут использовать возможности объекта ManagementObject, с возможностью полиморфной подстановки для обеспечения в дальнейшем гибкости

приложения, в соответствии с принципом LSP (Liskov Substitution Principle)[5][8] (рис. 4).

```
public abstract class Win32_Base
{
    protected ManagementObject mo { get; set; }
    public Win32_Base(ManagementObject mo)
    {
        this.mo = mo;
    }
}
```

Рисунок 4 – абстрактный базовый класс Win32_Base

Далее алгоритм преобразования сводится к трем основным шагам:

1) Получить контент по указанному URL (рис. 5)

```
private string GetContentFromURL(string url)
{
    using (var client = new WebClient())
    {
        client.Encoding = Encoding.UTF8;
        var content = client.DownloadString(url);
        return content;
    }
}
```

Рисунок 5 – Получение содержимого HTML страницы

2) Выделение из полученного контента интересующего класса C++ (рис. 6)

```
private string GetCppClass(string content)
{
    content = content.Substring(content.IndexOf("<code class=\"lang-syntax\""));
    content = content.Substring(0, content.IndexOf("};") + 2);
    return content;
}
```

Рисунок 6 – выделение из контента класса на языке C++

3) Преобразование полей класса C++ в свойства класса C# (рис. 7).

```
private string GetCSharpClass(string cppClass)
{
    cppClass = cppClass.Substring(cppClass.IndexOf("\nclass") + 1);
    cppClass = cppClass.Substring(0, cppClass.IndexOf("};") - 1)
    .Replace("boolean", "bool").Replace("uint8", "byte")
    .Replace("uint16", "ushort").Replace("uint32", "uint")
    .Replace("uint64", "ulong").Replace("int8", "byte")
    .Replace("int16", "short").Replace("int32", "int")
    .Replace("int64", "long").Replace("sint16", "short")
    .Replace("sint32", "int").Replace("sint64", "long")
    .Replace("sint", "int").Replace("datetime", "DateTime");

    var className = cppClass.Split(':')[0].Replace("class", "").Trim();
    var fields = cppClass.Substring(cppClass.IndexOf("\n{") + 2).Replace("\r\n};", "").Trim();
    var body = "public class " + className +
    ": Win32_Base\r\n{\r\n\tpublic " + className + "(ManagementObject mo) : base(mo) { }\r\nproperties}";
    var properties = new StringBuilder();

    foreach (var field in fields.Replace(";", "").Split('\n')
        .Select(x => {
            var str = x.Trim().Split(new[] { ' ' }, StringSplitOptions.RemoveEmptyEntries);
            return new { type = str[0].Trim(), value = str[1].Trim() };
        }).ToList()
    )
    {
        if (field.value.Contains("[]"))
        {
            var pname = field.value.Replace("[]", "");
            properties.Append($"\tpublic {field.type}[] {pname} => ({field.type}[])mo[\"{pname}\"];\r\n");
            continue;
        }
        if (field.type == "string")
        {
            properties.Append($"\tpublic string {field.value} => mo[\"{field.value}\"].ToString();\r\n");
            continue;
        }
        else if (field.type == "DateTime")
        { properties.Append($"\tpublic object {field.value} => mo[\"{field.value}\"]; //DateTime\r\n"); }
        else
        { properties.Append($"\tpublic {field.type} {field.value} => ({field.type}) mo[\"{field.value}\"];\r\n"); }
    }
    body = body.Replace("properties", properties.ToString().Replace("\t}", "}"));
    return body;
}
```

Рисунок 7 – преобразование класса C++ в класс C#

Таким образом, заключив три вышеуказанным метода в класс `WMIClassConverter` получим необходимый преобразователь (рис. 8).

```
public class WMIClassConverter
{
    private string GetContentFromURL(string url)...

    private string GetCppClass(string content)...

    private string GetCSharpClass(string cppClass)...

    public string GetCSharpClassFromUrl(string url) =>
    GetCSharpClass(GetCppClass(GetContentFromURL(url)));
}
```

Рисунок 8 – Класс WMIClassConverter

Теперь имея возможность получать в свое распоряжение объекты WMI в виде классов C# необходимо обеспечить возможность их использования расширяя функциональность имеющегося кода без необходимости вносить в него изменения, следуя принципу OCP (Open Closed Principle)[5-8]. Для этого, реализуем паттерн проектирования «Factory Method» [5-7], создав обобщенный класс SystemParameters, закрытый типом Win32_Base, который

будет иметь метод GetInfo(), возвращающий коллекцию объектов Win32_Base. Для отсечения необходимости вносить информацию о каждом необходимом классе, используем технику рефлексии типов из пространства имен System.Reflection, таким образом получая по имени класса одновременно и искомый класс WMI и тип инстанцируемого и возвращаемого динамически объекта подкласса Win32_Base (рис. 9).

```
public class SystemParameters<T> where T : Win32_Base
{
    public IEnumerable<Win32_Base> GetInfo()
    {
        foreach (ManagementObject mo in new ManagementObjectSearcher($"SELECT * FROM {typeof(T).Name}").Get())
        {
            var construct = typeof(T).GetConstructor(new[] { mo.GetType() });
            var instance = construct.Invoke(new[] { mo });
            yield return (Win32_Base)instance;
        }
    }
}
```

Рисунок 9 – Обобщенный класс SystemParameter<T>

Использование кода сводится к получению коллекции IEnumerable<Win32_Base> объектов

Например, для получения сведений об установленном программном обеспечении можно использовать следующий код (рис. 10)

```
public class SystemParameters<T> where T : Win32_Base
{
    public IEnumerable<Win32_Base> GetInfo()
    {
        foreach (ManagementObject mo in new ManagementObjectSearcher($"SELECT * FROM {typeof(T).Name}").Get())
        {
            var construct = typeof(T).GetConstructor(new[] { mo.GetType() });
            var instance = construct.Invoke(new[] { mo });
            yield return (Win32_Base)instance;
        }
    }
}

public abstract class Win32_Base
{
    protected ManagementObject mo { get; set; }
    public Win32_Base(ManagementObject mo)
    {
        this.mo = mo;
    }
}

public class Program
{
    public static void Main(string[] args)
    {
        new SystemParameters<Win32_Product>().GetInfo().Dump();
    }
}
public class Win32_Product : Win32_Base
{
    public Win32_Product(ManagementObject mo) : base(mo) { }
    public string Caption => mo["Name"].ToString();
    public string LocalPackage => mo["LocalPackage"].ToString();
    public string Version => mo["Version"].ToString();
```

Рисунок 10 – Использование программы для получения системной информации

Результат работы программы (рис. 11)

Caption	LocalPackage	Version
CorelDRAW Graphics Suite X7 - Draw (x64)	c:\WINDOWS\Installer\9e1ae.msi	17.0
vs_devenvmsi	C:\WINDOWS\Installer\11c137.msi	16.0.28329
Microsoft .NET Core Host FX Resolver - 2.0.6 (x64)	C:\WINDOWS\Installer\1d0f9b6.msi	16.24.26212
SQL Server 2017 Database Engine Shared	C:\WINDOWS\Installer\10483c9.msi	14.0.1000.169
Microsoft Visual Studio Tools for Applications 2012 x86 Hosting Support	C:\WINDOWS\Installer\9e262.msi	11.0.51108
Microsoft Visual Studio 2015 Shell (Minimum) Interop Assemblies	C:\WINDOWS\Installer\1a44231.msi	14.0.23107
Java SE Development Kit 8 Update 112	C:\WINDOWS\Installer\590680.msi	8.0.1120.15
Java SE Development Kit 8 Update 112 (64-bit)	C:\WINDOWS\Installer\590689.msi	8.0.1120.15
SQL Server Management Studio	C:\WINDOWS\Installer\814c90.msi	15.0.18206.0
Microsoft .NET Runtime - 5.0.1 (x86)	C:\WINDOWS\Installer\34a1c52.msi	40.4.29525
Microsoft .NET Core 3.1 Templates 5.0.101 (x64)	C:\WINDOWS\Installer\34a1cca.msi	12.18.61801
InputMapper	C:\WINDOWS\Installer\7a9ae21.msi	1.6.10.19991
CorelDRAW Graphics Suite X7 - PHOTO-PAINT (x64)	c:\WINDOWS\Installer\9e1c0.msi	17.0
Языковой пакет для окна справки (Майкрософт) 2.3 — RUS	C:\WINDOWS\Installer\814bd7.msi	2.3.27412
icecap_collection_neutral	C:\WINDOWS\Installer\34a1dc4.msi	16.8.30509
Microsoft .NET Framework 4.8 SDK	C:\WINDOWS\Installer\c3c83d0.msi	4.8.03928
Microsoft Visual Studio Tools for Applications 2012 x86 Hosting Support...	C:\WINDOWS\Installer\9e2a1.msi	11.0.51108
ClickOnce Bootstrapper Package for Microsoft .NET Framework	C:\WINDOWS\Installer\199ed8b.msi	4.8.04119
Update for Windows 10 for x64-based Systems (KB4023057)	C:\WINDOWS\Installer\2318d2d0.msi	2.57.0.0
Microsoft Visual C++ 2015 x64 Debug Runtime - 14.0.23026	C:\WINDOWS\Installer\1a4401a.msi	14.0.23026
Microsoft System CLR Types for SQL Server 2014	C:\WINDOWS\Installer\1a441ff.msi	12.0.2402.11
Microsoft Visual Studio Tools for Applications 2012 x64 Hosting Support...	C:\WINDOWS\Installer\9e2e0.msi	11.0.51108
Microsoft .NET Host FX Resolver - 5.0.1 (x64)	C:\WINDOWS\Installer\34a1ce2.msi	40.4.29525

Рисунок 11 – Результат работы программы

3. Выводы

В данной статье был описан метод получения системной информации, предоставляемой провайдерами инструментария управления Windows. При проектировании ИС, в случае необходимости получения подобного рода информации можно использовать приведенный в статье код, как отдельную библиотеку, так как примененные техники объектно-ориентированного программирования позволяют использовать объекты WMI в более удобной для разработчика форме, обеспечивая безопасность типов и гибкость расширения функционала.

Библиографический список

1. Попов А.В., Шишкин Е.А. Администрирование WINDOWS с помощью WMI и WMIC. СПб.: БХВ-Петербург, 2004. 732 с.
2. Уилер Ш. Удобный способ просмотра запросов WMI // WINDOWS IT PRO/ RE. 2010. №11. С. 46-49. URL: https://www.elibrary.ru/item.asp?id=16220099 (Дата обращения: 20.12.2020).
3. Руссинович М., Соломон Д. Внутреннее устройство Microsoft Windows. СПб.: Питер, 2013. 800 с.
4. Администрирование с помощью WMI // Sysengineering URL:

http://sysengineering.ru/notes/administrirovanie-s-pomoschyu-wmi (дата обращения: 20.12.2020).
5. Шевчук А., Охрименко Д., Касьянов А. Design Patterns via C# Приемы объектно-ориентированного проектирования. Киев: ITVDN, 2015. 288 с.
6. Nesteruk D. Design Patterns in Modern C++. NY: APRESS, 2018. 314 с. URL: https://www.elibrary.ru/item.asp?id=35750959&selid=35750960
7. Gamma E., Helm R., Johnson R., Vissides J., Vissides J. и др. Design Patterns: Elements of Reusable Object-Oriented Software. NY: Addison-Wesley Professional, 1994. 416 с.
8. SOLID Design Principles Explained: The Single Responsibility Principle URL: https://stackify.com/solid-design-principles/ (дата обращения: 20.12.2020).