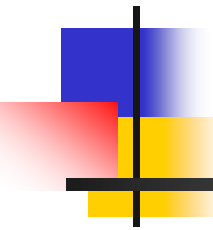


.NET Framework类库

.NET框架类库

- .NET框架类库概观
- 输入和输出：System.IO
- 序列化：System.Runtime.Serialization
- 支持XML：System.Xml
- 反射：System.Reflection
- 互操作性：System.Runtime.InteropServices
- Windows GUIS：System.Windows.Forms
- 远程安装Windows Forms应用程序:ClickOnce
- 发送电子邮件： System.Net.Mail

.NET框架类库概观

- .NET Framework 包括类、接口和值类型。它符合 CLS，可在任何符合CLS的编程语言中使用。
- .NET Framework 类是建立 .NET Framework 应用程序、组件和控件的基础。 .NET Framework 包括的类型执行下列功能：
  - 表示基础数据类型和异常。
  - 封装数据结构。
  - 执行 I/O。
  - 访问关于加载类型的信息。
  - 调用 .NET Framework 安全检查。
  - 提供数据访问、多客户端 GUI 和服务器控制的客户端 GUI。

.NET框架类库概观

- 命名空间

  .NET Framework 类库由命名空间组成。每个命名空间都包含可在程序中使用的类型：类、结构、枚举、委托和接口。

  所有 Microsoft 提供的命名空间都是以名称 System 或 Microsoft 开头的。

.NET框架类库概观

- 命名约定

  .NET Framework 类型使用点语法命名方案，该方案隐含了层次结构体系。

  此技术将相关类型分为不同的命名空间组，以便可以更容易地搜索和引用它们。全名的第一部分（最右边的点之前的内容）是命名空间名。全名的最后一部分是类型名。例如，System.Collections.ArrayList 表示 ArrayList 类型，该类型属于 System.Collections 命名空间。System.Collections 中的类型可用于操作对象集合。

.NET框架类库概观 - 命名空间

- Microsoft命名空间

  以Microsoft开头的命名空间，一般不是为所有应用程序服务的，而是在某特定场合下使用。如，下面是一些Microsoft开头的命名空间：

  - CSharp：包含支持用 C# 语言进行编译和代码生成的类。
  - JScript：包含支持用 JScript 语言进行编译和代码生成的类。
  - VisualBasic：包含支持用 VB.NET 语言进行编译和代码生成的类。

.NET Framework类库概观

- System命名空间

System 命名空间是 .NET Framework 中基本类型的根命名空间。包含

1. 由所有应用程序使用的基础数据类型的类。如：用于定义常用值和引用数据类型、事件和事件处理程序、接口、属性和处理异常的基础类和基类等。
2. 除基础数据类型外，**System** 命名空间还包含 100 多个类，这些类提供支持下列操作的服务：数据类型转换，方法参数操作，数学计算，远程和本地程序调用，应用程序环境管理以及对托管和非托管应用程序的监管等。
3. **System** 命名空间还包含许多二级命名空间。

.NET Framework类库概观

- System命名空间的一些类型
  - CLR通用类型系统
  - Console：表示控制台应用程序的标准输入流、输出流和错误流。
  - Math：为三角函数、对数函数和其他通用数学函数提供常数和静态方法。
  - Environment：提供有关当前环境和平台的信息以及操作它们的方法。
  - GC：控制系统垃圾回收器。
  - Random：表示伪随机数生成器。
  - 其他。如：ObsoleteAttribute(标记不再使用的程序元素)，OperatingSystem(表示有关操作系统的信息)等。

  可参考MSDN的“System 命名空间”

.NET框架类库概观
– System次命名空间

System 命名空间还包含许多次级命名空间。下面列出了常用的按功能分类的次级命名空间：

- 编程基础命名空间：
  主要用于集合、字符编码、文件 I/O 和线程处理的 .NET 命名空间。
- 数据命名空间
  主要用于操纵和访问数据的命名空间，这些数据来自传统的数据源和基于标准的 XML。
- Web 命名空间
  主要用于 ASP.NET Web 应用程序和 XML Web services 的 .NET 命名空间。
- Windows 应用程序命名空间
  主要用于创建功能丰富的 Windows 应用程序和带有 Windows 窗体的图形。
- 组件模型命名空间
  主要用于实现组件和控件在运行时和在设计时的行为。
- 框架服务命名空间
  主要是与各种后端服务器资源交互的命名空间。

.NET框架类库概观
– System次命名空间

- 安全性命名空间

  主要提供 .NET Framework 安全系统的基础结构，并在 Web 应用程序中实现 ASP.NET 安全性的命名空间。
- 网络命名空间

  主要是为网络中出现的多种协议提供一种简单编程界面。
- 配置命名空间

  主要在 .NET Framework 配置设置下工作并处理配置文件中的错误，同时允许您为组件编写自定义的安装程序。
- 全球化和本地化命名空间

  这些命名空间设计来开发全球可用的应用程序。
- 反射命名空间

  它们为已加载的类型、方法和字段提供一个可管理的视图，并能够动态地创建和调用各种类型。

.NET框架类库概观 – System次命名空间

- 编程基础命名空间

  .NET Framework 中的基本编程命名空间包括：

  - System.Collections：包含定义各种对象集合（如列表、队列、位数组、哈希表和字典）的接口和类。
  - System.Collections.Generic： 包含定义泛型集合的接口和类；
  - System.IO：包含的类型用于支持数据流和文件的同步和异步读写。
  - System.Text：包含表示 ASCII、Unicode、UTF-7 和 UTF-8 字符编码的类；用于来回转换字符块和字节块的抽象基类；以及在不创建 String 中间实例的情况下操作和格式化 String 对象的帮助器类。
  - System.Threading — 提供启用多线程编程的类和接口。此命名空间包括一个管理线程组的 ThreadPool 类，一个启用在指定时间后要调用的委托的 Timer 类，以及一个同步互斥线程的 Mutex 类。System.Threading 还为线程调度和等待通知提供了相应的类。

.NET框架类库概观
– System次命名空间

- 数据命名空间

  .NET Framework 中的数据和 XML 命名空间包括：

  - System.Data：由构成 ADO.NET 结构的类组成，该结构是托管应用程序的主要数据访问方法。ADO.NET 结构使您可以生成可用于有效管理来自多个数据源的数据的组件。ADO.NET 还提供对分布式应用程序中的数据进行请求、更新和协调的工具。
  - System.Data.Common：包含由 .NET Framework 数据提供程序共享的类。数据提供程序描述一个类的集合，这些类用于在托管空间中访问数据源，例如数据库。
  - System.Xml：根据标准来支持 XML 处理的类。
  - System.Data.OleDb：构成兼容数据源的 OLE DB .NET Framework 数据提供程序的类。这些类使您能连接到 OLE DB 数据源、针对数据源执行命令并读取结果。
  - System.Data.SqlClient：构成 SQL Server .NET Framework 数据提供程序的类，该提供程序允许您连接到 SQL Server 7.0、执行命令并读取结果。System.Data.SqlClient 命名空间与 System.Data.OleDb 命名空间类似，但为访问 SQL Server 7.0 和更高版本进行了优化。

.NET框架类库概观 – System次命名空间

- 数据命名空间（续）
  - System.Data.Sql：支持特定于 SQL Server 的功能的类。
  - System.Data.SqlTypes：提供一些类，它们在 SQL Server 内部用于本机数据类型。这些类提供了其他数据类型的更安全、更快速的替代方式。
  - Microsoft.SqlServer.Server：专用于 Microsoft .NET Framework 公共语言运行库 (CLR) 与 Microsoft SQL Server 和 SQL Server 数据库引擎进程执行环境的集成的类、接口和枚举。
  - System.Data.Odbc：构成 ODBC .NET Framework 数据提供程序的类。使用这些类可以在托管空间中访问 ODBC 数据源。
  - System.Data.OracleClient：构成 Orcale.NET Framework 数据提供程序的类。使用这些类可以在托管空间中访问 Oracle 数据源。
  - System.Transactions：允许您编写自己的事务性应用程序和资源管理器的类。具体来说，您可以创建事务并和一个或多个参与者参与事务（本地或分布式）。

.NET框架类库概观 – System次命名空间

- Web 命名空间

  在 .NET Framework 中，有关 ASP.NET Web 应用程序和 XML Web services 的命名空间包括：

  - System.Web：包含启用浏览器/服务器通信的类和接口。这些命名空间类用于管理到客户端的 HTTP 输出 (HttpResponse)，和读取 HTTP 请求 (HttpRequest)。附加的类则提供了一些功能，用于服务器端的实用程序以及进程、cookie 管理、文件传输、异常信息和输出缓存控制。
  - System.Web.UI：包含创建 Web 窗体页的类，包括 Page 类和用于创建 Web 用户界面的其他标准类。
  - System.Web.UI.HtmlControls：包含用于 HTML 特定控件的类，这些控件可以添加到 Web 窗体中以创建 Web 用户界面。

.NET框架类库概观 – System次命名空间

- Web 命名空间（续）
  - System.Web.UI.WebControls：包含创建 ASP.NET Web 服务器控件的类。当添加到 Web 窗体时，这些控件将呈现浏览器特定的 HTML 和脚本，用以创建与设备无关的 Web 用户界面。
  - System.Web.Mobile：包含生成 ASP.NET 移动 Web 应用程序所需的核心功能，包括身份验证和错误处理。
  - System.Web.UI.MobileControls：包含一组 ASP.NET 服务器控件，这些控件可以针对不同的移动设备呈现应用程序。
  - System.Web.Services：包含使您能够生成和使用 XML Web services 的类，这些服务是驻留在 Web 服务器中的可编程实体，并通过标准 Internet 协议公开。

.NET框架类库概观
– System次命名空间

- Windows 应用程序命名空间

  在 .NET Framework 中，用于创建丰富 Windows 应用程序和图形的命名空间包括：

  - System.Windows.Forms：它包含的类可创建基于 Windows 的应用程序，这些应用程序将充分利用 Microsoft Windows 操作系统中提供的丰富用户界面功能。在此命名空间中，您将找到可添加到窗体中创建用户界面的 Form 类和许多其他控件。
  - System.Drawing：它使您能够访问 GDI+ 的基本图形功能。System.Drawing.Drawing2D、System.Drawing.Imaging 和 System.Drawing.Text 命名空间中提供了更高级的功能。
  - System.ServiceProcess：它提供的类用于安装和运行没有用户界面且长期运行的可执行文件。

.NET框架类库概观 – System次命名空间

- 组件模型命名空间

  在 .NET Framework 中，用于创建您自己的组件和控件的命名空间包括：

  - System.ComponentModel：它提供的类用于实现组件和控件的运行时和设计时行为。此命名空间包括用于实现属性、使用类型转换器、绑定到数据源，以及授权组件的基类和接口。此命名空间中的类分为下列类别：
    - 核心组件类：Component 和 Container 类以及 IContainer 和 IComponent 接口。
    - 组件授权：License、LicenseManager、LicenseProvider 和 LicenseProviderAttribute 类。
    - 特性：Attribute 类。
    - 说明符和持久性：TypeDescriptor、EventDescriptor 和 PropertyDescriptor 类。
    - 类型转换器：TypeConverter 类。
  - System.CodeDOM：它所包含的类可用于表示源代码文档的元素和结构

.NET框架类库概观
– System次命名空间

- 框架服务命名空间

  在 .NET Framework 中，用于与各种后端服务器资源进行交互的命名空间包括：

  - System.Diagnostics：它所包括的类用于调试应用程序和跟踪代码的执行情况。允许您启动系统进程、读取和写入事件日志以及使用性能计数器监视系统性能。
  - System.DirectoryServices：它包含的类可便于从托管代码中访问 Active Directory。此命名空间中的类可以与任何 Active Directory 服务提供程序一起使用。当前的提供程序包括：Internet 信息服务 (IIS)、轻量目录服务协议 (LDAP)、Novell NetWare 目录服务 (NDS) 和 Windows NT。
  - System.Deployment.Application：提供用于将自定义升级行为编程到 ClickOnce 部署概述应用程序中的类。这些类允许您的部署提供更新是否可用的信息，提供安装更新，并根据需要下载大型文件和程序集。
  - System.IO — 它提供的类包括 FileSystemWatcher，此类侦听文件系统更改通知并在目录或目录中的文件出现更改时引发事件。

.NET框架类库概观 – System次命名空间

- 框架服务命名空间（续）
  - System.Media：包含用于播放声音文件和访问系统提供的声音的类。
  - System.Management：它提供的类用于管理一些信息和事件，它们关系到系统、设备和 Windows Management Instrumentation (WMI) 基础结构所使用的应用程序。
  - System.Messaging：它提供的类用于连接到网络上的消息队列，向队列发送消息，从队列接收或查看（读取而不移除）消息。
  - System.ServiceProcess：它所提供的类用于安装和运行服务。服务是长期运行的可执行文件，它们不通过用户界面来运行。服务可以安装在一个系统帐户下运行，此帐户将使服务能够在计算机重新启动时启动。如果服务从 ServiceBase 类中的处理导出其实现，它们就可以定义开始、停止、暂停和继续命令的特定行为以及在关闭系统时所采取的行为。
  - System.Timers：提供基于服务器的计时器组件，用以按指定的间隔引发事件。
  - Microsoft.Win32 -- 提供处理操作系统引发的事件和操作系统注册表的类。

.NET框架类库概观
– System次命名空间

- 安全性命名空间

  .NET Framework 中的安全性命名空间包括：

  System.Security：它提供公共语言运行库安全性系统的基础结构，其中包括权限的基类。

  System.Net.Security：提供用于主机间安全通信的网络流。

  System.Web.Security：它包含的类用于在 Web 应用程序中实现 ASP.NET 安全性。

.NET框架类库概观 – System次命名空间

- 网络命名空间

  .NET Framework 中的网络命名空间包括：

  - System.Net：它包含的类可为当前网络上的多种协议提供简单的编程接口。 WebRequest 和 WebResponse 类形成了“可插入协议”的基础，利用这种网络服务的实现，您可以开发在使用 Internet 资源时不用考虑所用协议的具体细节的应用程序。
  - System.Net.Cache：定义类型和枚举，这些类型和枚举用于为使用 WebRequest 和 HttpWebRequest 类获取的资源定义缓存策略。
  - System.Net.Configuration：应用程序用来以编程方式访问和更新 System.Net 命名空间的配置设置的类。
  - System.Net.Mail：用于将电子邮件发送到简单邮件传输协议 (SMTP) 服务器进行传送的类。

.NET框架类库概观 – System次命名空间

- 网络命名空间（续）
  - System.Net.Mime：包含用于表示多用途 Internet 邮件交换 (MIME) 标头的类型。这些类型与 System.Net.Mail 命名空间中的类型一起使用，用于在使用 SmtpClient 类发送电子邮件时指定 Content-Type、Content-Disposition 和 Content-transfer-Encoding 标头。
  - System.Net.NetworkInformation：提供对网络流量数据、网络地址信息和本地计算机的地址更改通知的访问。该命名空间还包含实现 Ping 实用工具的类。您可以使用 Ping 和相关的类检查是否可通过网络访问某台计算机。
  - System.Net.Sockets：为需要严格控制网络访问的开发人员提供 Windows 套接字 (Winsock) 接口的托管实现。

.NET框架类库概观 – System次命名空间

- 配置命名空间

  .NET Framework 中的配置命名空间包括：

  - System.Configuration：包含用于以编程方式访问 .NET Framework 配置设置并处理配置文件中错误的类。
  - System.Configuration.Assemblies：包含用于配置程序集的类。
  - System.Configuration.Install：提供用于为自己的组件编写自定义安装程序的类。
  - System.Configuration.Provider：包含由服务器和客户端应用程序共享以支持可插接式模型轻松添加或移除功能的基类。

.NET框架类库概观 – System次命名空间

- 全球化和本地化命名空间

  在 .NET Framework 中，用于将应用程序全球化和本地化的命名空间包括：

  - System.Globalization — 包含的类定义与区域性相关的信息，其中包括语言、国家/地区、所使用的日历、日期格式的模式、货币与数字以及字符串的排序顺序。
  - System.Resources — 提供一些类和接口，它们使开发人员得以创建、存储并管理应用程序中使用的各种区域性特定的资源。
  - System.Resources.Tools -- 包含 StronglyTypedResourceBuilder 类，该类提供对强类型资源的支持。这个编译时功能通过创建包含一组静态只读 (get) 属性的类封装对资源的访问，从而使得使用资源变得更加容易。
  - System.Text — 包含表示 ASCII、Unicode、UTF-7 和 UTF-8 字符编码的类。

.NET框架类库概观 – System次命名空间

- 反射命名空间

  .NET Framework 中的反射命名空间包括：

  - System.Reflection — 包含的类和接口用于访问已加载的类型及其成员。
  - System.Reflection.Emit — 包含的类允许您发出元数据和 Microsoft 中间语言 (MSIL) 并可选择在磁盘上生成 PE 文件。这些类的主要客户端是脚本引擎和编译器。

.NET框架类库概观 -- 用法摘要

- 当您在 Visual Studio 中创建 Visual Basic 或 Visual C# 项目时，已经引用了最常用的基类 DLL（程序集）。但是，如果您需要使用尚未引用的 DLL 中的类型，则需向此 DLL 添加引用。
  1. 确定提供所需功能的类的位置。
  2. 在类型的文档概述中，记下该类型的程序集和命名空间的名称。
  3. 查看是否已经在项目中引用程序集。打开“解决方案资源管理器”，在“引用”节点下查看。
  4. 如果没有看到程序集引用，请右击“引用”节点并选择“添加引用”。
  5. 当添加程序集引用后，即可访问程序集中的类型。

  如不想使用完全限定名，则可以使用 Imports 语句（在 VB 中）或使用 using 关键字（在 Visual C# 中）。

.NET框架类库 - System.IO

- 输入和输出：System.IO

  System.IO 命名空间包含允许在数据流和文件上进行同步和异步读写的类型。

- 文件和流的差异

  文件是一些具有永久存储及特定顺序的字节组成的一个有序的、具有名称的集合。因此，对于文件，人们常会想到目录路径、磁盘存储、文件和目录名等方面。相反，流提供一种向后备存储器写入字节和从后备存储器读取字节的方式，后备存储器可以为多种存储媒介之一。正如除磁盘外存在多种后备存储器一样，除文件流之外也存在多种流。例如，还存在网络流、内存流和磁带流等。

基本的文件 I/O

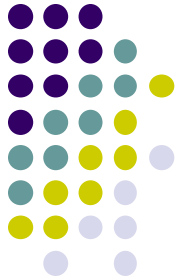
- 抽象基类 Stream 支持读取和写入字节 。它集成了异步支持。
- 用于文件 I/O 的类
  - Directory（静态）、DirectoryInfo（实例）：提供通过目录和子目录进行创建、移动和枚举的方法。
  - File、FileInfo：提供用于创建、复制、删除、移动和打开文件的方法，并协助创建 FileStream。
  - FileStream 以同步方式打开文件，但它也支持异步操作。
  - FileSystemInfo 是 FileInfo 和 DirectoryInfo 的抽象基类。
  - Path 提供以跨平台的方式处理目录字符串的方法和属性。

.NET框架类库 - System.IO

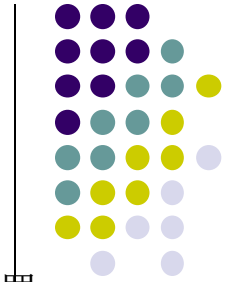
- 用于从流读取和写入流的类
  - BinaryReader 和 BinaryWriter ：从 Streams 读取或向 Streams 写入编码的字符串和基元数据类型。
  - StreamReader、StreamWriter：通过使用特定的编码(默认为 UTF-8)将字符转换为字节，从/向Streams 中读取/写入字符。

    （例：VbExam2： Directory 、 File IO 、Binary IO、 StreamIO、 String IO ）

  - StringReader、 StringWriter：从 Strings 中读取/写入字符。输出可以是 String 或以任何编码表示的 Stream。
  - TextReader：是 StreamReader 和 StringReader 的抽象基类。抽象 Stream 类的实现用于字节输入和输出，而 TextReader 的实现用于 Unicode 字符输出。
  - TextWriter：是 StreamWriter 和 StringWriter 的抽象基类。抽象 Stream 类的实现用于字节输入和输出，而 TextWriter 的实现用于 Unicode 字符输出。



```
// 读配置文件举例
// 配置文件格式：
//   参数 = 值
// 如：Server = 211.64.253.143
// 调用形式 ：  strServer = ReadPara(“Server”)
// 函数返回 “211.64.253.143”给变量strServer
public static string ReadPara(string pstrL)
{   string strExe =
    System.Reflection.Assembly.GetExecutingAssembly().Location;
    string strExePath =
    strExe.Substring(0,strExe.LastIndexOf("\\")).Trim();
    //System.IO.Directory.SetCurrentDirectory(strExePath);
    string strConfigFile=strExePath + "\\Ffh.ini";
    if (System.IO.File.Exists(strConfigFile)==false )
    {      MessageBox.Show("配置文件" +
                    strConfigFile + "未找到。","错误提示",
                    MessageBoxButtons.OK,
                    MessageBoxIcon.Error);
           return "";
    }
```



```
try
{     System.IO.FileInfo fileT=new System.IO.FileInfo(strConfigFile);
      System.IO.StreamReader streamR=fileT.OpenText();
      string strT,strL;
      int intPos;
      while (streamR.Peek()>-1)    //Peek：返回下一个字符，不会更改 streamR的当前位置。
      {    strT=streamR.ReadLine();
         if (strT.Trim().Substring(0,1)!="“”)       //如果第一个字符不是’(注解行)
         {      intPos=strT.IndexOf("=",0);         //第一个”=“
                strL=strT.Substring(0, intPos);
                if (strL.Trim().ToUpper()==pstrL.Trim().ToUpper())
                {  if (strT.IndexOf(“”,1)>0)          //如果行后还有注解
                     return  strT.Substring(intPos+1,strT.IndexOf("",1) - intPos -1 ).Trim();
                   else
                     return  strT.Substring(intPos+1,strT.Length - intPos -1 ).Trim();
                }
         }
      }
      return "";
}
catch
{     return "";
}

}
```


返回

序列化：System.Runtime.Serialization

- **System.Runtime.Serialization** 命名空间包含可用于将对象序列化和反序列化的类。
  - 序列化是将对象或对象图形转换为线性字节序列，以存储或传输到另一个位置的过程。
  - 反序列化是接受存储的信息并利用它重新创建对象的过程。
- 为什么您想要使用序列化？有两个最重要的原因：
  - 一是将对象的状态保持在存储媒体中，以便可以在以后重新创建精确的副本；
  - 另一原因是通过值将对象从一个应用程序域发送到另一个应用程序域中。　　如，序列化可用于在 ASP.NET 中保存会话状态并将对象复制到 Windows 窗体的剪贴板中。远程处理还可以使用序列化通过值将对象从一个应用程序域传递到另一个应用程序域中。

序列化：System.Runtime.Serialization

.NET Framework 提供两种序列化技术：

- 二进制序列化保持类型保真度，这对于在应用程序的不同调用之间保留对象的状态很有用。例如，通过将对象序列化到剪贴板，可在不同的应用程序之间共享对象。您可以将对象序列化到流、磁盘、内存和网络等等。远程处理使用序列化“通过值”在计算机或应用程序域之间传递对象。
- XML 序列化仅序列化公共属性和字段，且不保留类型保真度。当您希望提供或使用数据而不限制使用该数据的应用程序时，这很有用。由于 XML 是一个开放式标准，因此，对于通过 Web 共享数据，它是一个有吸引力的选择。SOAP 同样是一个开放式标准，这使它也成为一个颇具吸引力的选择。

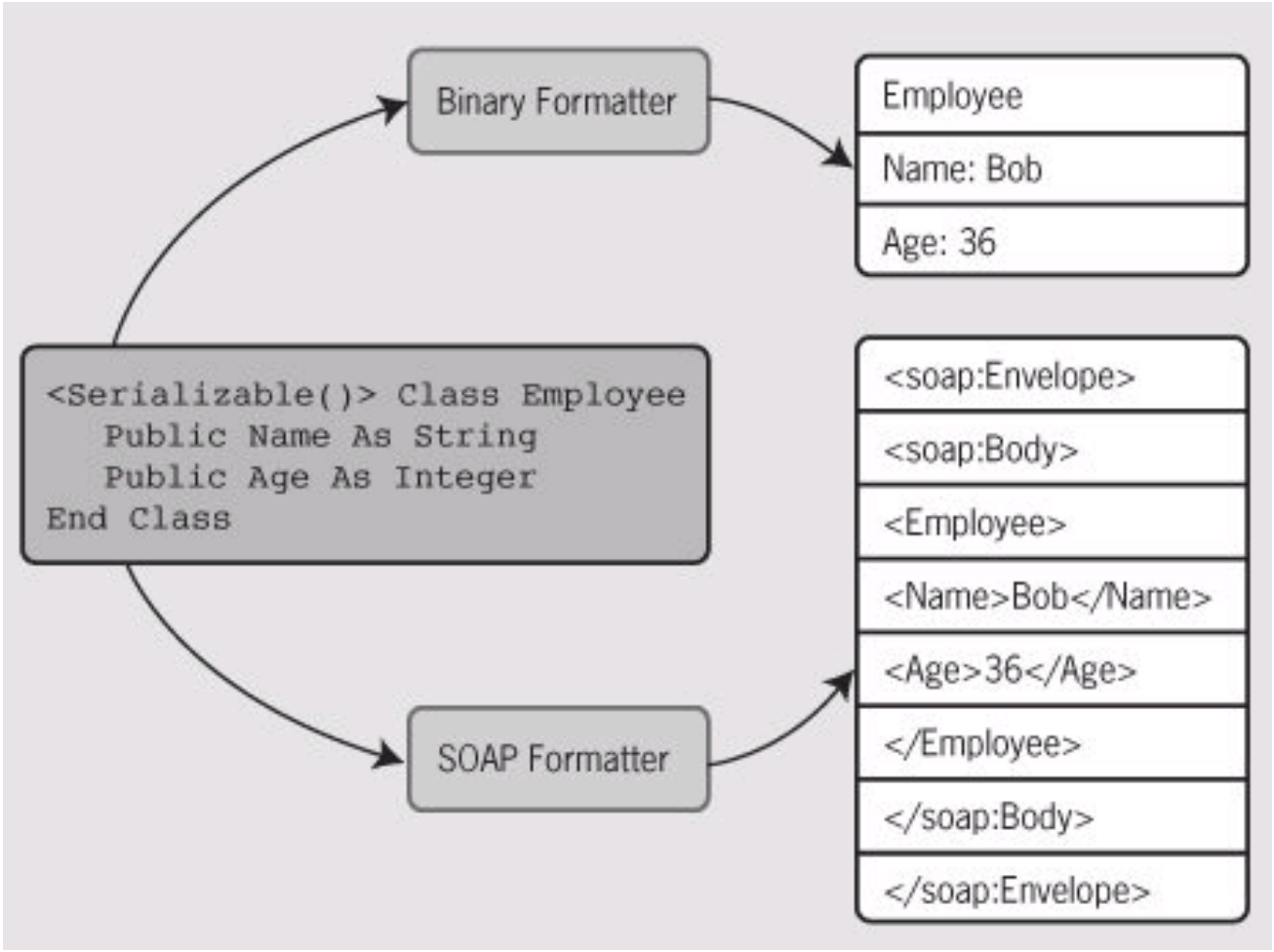
序列化：System.Runtime.Serialization

- 令一个类可序列化的最简单的方式是按如下所示使用 SerializableAttribute 特性标记它。

```
[Serializable]
public class MyObject {
  public int n1 = 0;
  public int n2 = 0;
  public String str = null;
}
```

- 然后用格式器(Formatter)使“对象序列化”
  - 二进制格式器：BinaryFormatter
  - SOAP格式器（需引用Soap.dll）：SoapFormatter
- 序列化的结果放在stream中。

序列化：System.Runtime.Serialization



序列化：System.Runtime.Serialization

如：

```
MyObject obj = new MyObject();
obj.n1 = 1;
obj.n2 = 24;
obj.str = "Some String";
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin",
   FileMode.Create,FileAccess.Write, FileShare.None);
formatter.Serialize(stream, obj);
stream.Close();
```

序列化：System.Runtime.Serialization

- 将对象还原回其以前的状态十分简单。首先，创建用于读取的流和格式化程序，然后指示格式化程序反序列化该对象。如：

```
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin",
    FileMode.Open,FileAccess.Read, FileShare.Read);
MyObject obj = (MyObject) formatter.Deserialize(stream);
stream.Close();
// Here's the proof.
Console.WriteLine("n1: {0}", obj.n1);
Console.WriteLine("n2: {0}", obj.n2);
Console.WriteLine("str: {0}", obj.str);
```

（例：VbExam2-“序列化”）

序列化：System.Runtime.Serialization

- 并非每个类型都可序列化,如未标Serializable的类
- 可序列化类型的某些成员,只要在成员前加Serializable特性
- 被序列化的类也可指明某些成员不可序列化,只要在成员前加NonSerializable特性
- 序列化过程可自定制

返回

支持XML：System.Xml

- XML技术家族
  （http://www.w3.org/XML/
  http://www.w3.org/TR/xmlbase/）

  可扩展标记语言 (Extensible Markup Language, XML)，它是标准通用标记语言 (SGML) 的子集，非常适合 Web 传输。

  XML 提供统一的方法来描述和交换独立于应用程序或供应商的结构化数据。

  XML提供了一种使用标记（文字由‘<’和‘>’括起）描述文档结构的机制。

  SGML 是 IBM 创造的一个用于出版业的文档格式标准，后来被 ISO 采纳作为国际标准（ISO 8879）。

支持XML：System.Xml

- 一个XML文档包含一个或多个元素，彼此间以标签分界。
  <test>……/<test>
  标记区分大小写，起始标记与结束标记大小写必须一致。
  多个起始、结束标记之间不能发生顺序错误。
  可选择性地忽略元素的结束标签。
- 元素可包含一个或多个属性，属性值可有引号，引号可以是双引号也可以是单引号。一般采用双引号。
- XML文档可包含注释
  <!-- 注释 -->
- 如

```
<book id="1">
        <title>XML 高级编程</title>
        <isbn>7-111-07315-0</isbn>
        <author>Didier Martin 等</author>
        <pages>944</pages>
    </book>
```

XML技术家族： XML信息集

- XML信息集 (XML infoset，XML Information Set)

  XML 信息集是对 XML 文档中数据的抽象说明，它可视为 XML 文档的主要数据模型。

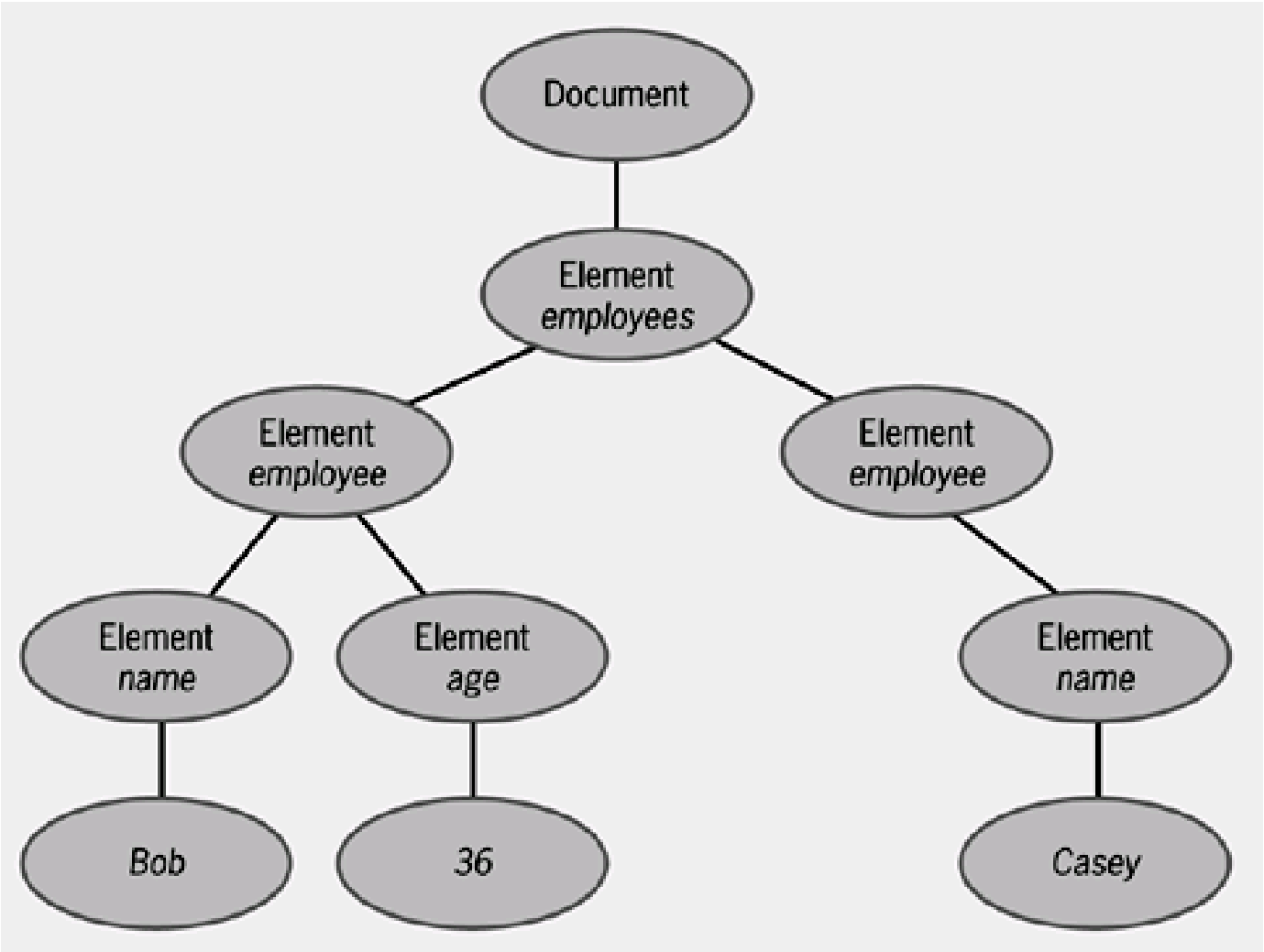
  XML 信息集是 XML 文档的树状层次表示。一个 XML 文档的信息集包含许多信息项，这些信息项是 XML 文档组件的抽象表示，其中包括表示文档、文档的元素、属性、处理指令、注释、字符、表示法、命名空间、未分析的实体、未扩展的实体引用和文档类型声明的信息项。

XML技术家族： XML信息集

如：

```
<employees>
   <employee>
      <name>Bob</name>
      <age>36</age>
   </employee>
   <employee>
      <name>Casey</name>
   </employee>
</employees>
```

XML技术家族： XML信息集



XML技术家族： XPath

- Xpath：是一个获取XML文档中节点元素的技术。它允许你用很少的代码就能获取指定的路径下的节点值。如

  /authors/author[@period="classical"]

  含义是：在根节点为“authors”的XML文档中，查询period属性值为classical的author元素（element）。

  再如，有一个XML文档：

XML技术家族： XSLT 转换

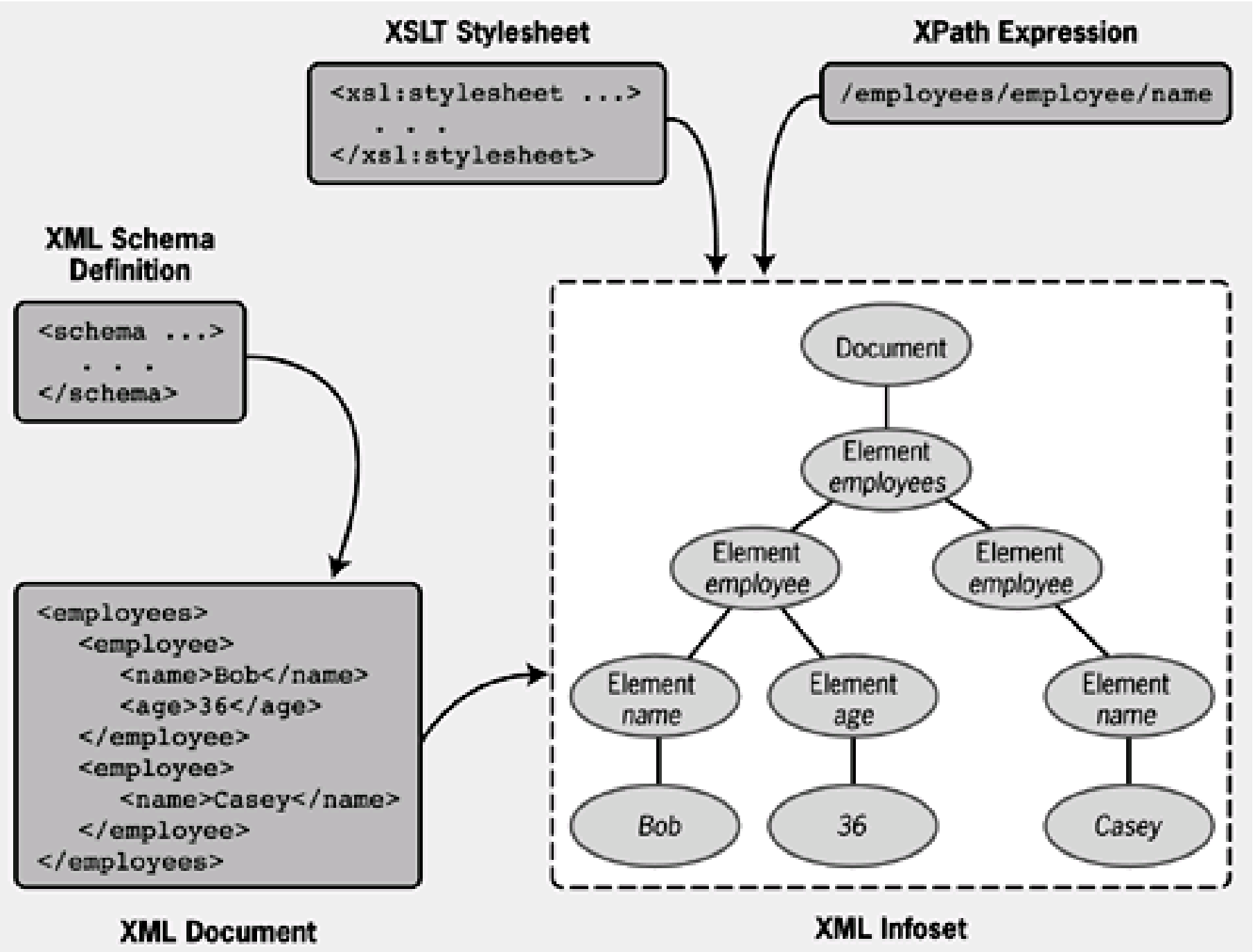
- 可扩展样式表语言转换 (XSLT) 的目的是将源 XML 文档的内容转换为另一个格式或结构不同的文档。例如，将 XML 转换为在 Web 站点上使用的 HTML 或转换为只包含应用程序所需字段的文档。

- .NET Framework 中，**System.Xml.Xsl** 命名空间中的 **XslTransform** 类是实现此规范功能的 XSLT 处理器。

XML技术家族： XML架构

- XML 架构是用于定义和验证 XML 数据的内容和结构的文档。
- XML 架构通过 XML 架构定义 (XSD) 语言定义和描述某些 XML 数据类型。XML 架构元素（元素、属性、类型和组）用于定义某些 XML 数据类型的有效结构、有效数据内容和关系。XML 架构还可为属性和元素提供默认值。

例如，在买方和卖方之间发送以 XML 表示的订单之前，可以用 XML 架构对其进行验证。该验证校验数据的所有元素（各片段）均存在，都按预期顺序排列，并且均为正确的数据类型。这确保订单收件人在收到它时能够正确解释数据。


XSLT Stylesheet
<xsl:stylesheet ...>
. . .
</xsl:stylesheet>
XPath Expression
/employees/employee/name
XML Schema Definition
<schema ...>
. . .
</schema>
<employees>
<employee>
<name>Bob</name>
<age>36</age>
</employee>
<employee>
<name>Casey</name>
</employee>
</employees>
Document
Element employees
Element employee
Element employee
Element name
Element age
Element name
Bob
36
Casey
XML Document
XML Infoset

支持XML：System.Xml

- 处理XML文档中的信息
  - DOM

    文档对象模型 (DOM) 是 XML 文档在内存中的表示形式。DOM 使您能以编程方式读取、操作和修改 XML 文档。但处理大型文件时其性能下降的非常厉害。这个问题是由DOM的树结构所造成的，这种结构占用的内存较多，而且DOM必须在解析文件之前把整个文档装入内存,适合对XML的随机访问
  - SAX:SAX(Simple API for XML)

    SAX是事件驱动型的XML解析方式。它顺序读取XML文件，不需要一次全部装载整个文件。当遇到像文件开头，文档结束，或者标签开头与标签结束时，它会触发一个事件，用户通过在其回调事件中写入处理代码来处理XML文件，适合对XML的顺序访问。SAX无法处理XPath。

粗看之下，DOM和SAX各自的优缺点刚好形成互补。DOM使用内存保存对象结构；而SAX则基于事件并且不使用内存来存储任何数据。因此，DOM比较适合文档较小而数据访问模式复杂的情况，相反情况下，则使用SAX。

支持XML：System.Xml

- STAX:Streaming API for XML (STAX)

  类似于SAX的事件，但与只能被动实现回调函数的不能进行暂停和跳过等操控的Sax不同，STAX是主动用遍历器函数循环去读取XML的，控制权完全在处理程序手中。

  一般说来,STAX适合于对已知结构(Schema)的XML文件进行读写操作。

支持XML：System.Xml

- .NET Framework 中的 XML 类提供全面、集成的类集，使您得以使用 XML 文档和数据。
  - 内存中的 XML 处理： .NET Framework 提供三个用于在内存中处理 XML 数据的选项： LINQ to XML、XPathNavigator 类和 XmlDocument 类。
  - 基于流的 XML 处理： XmlWriter 和 XmlReader 类提供一种非缓存的只进 XML 数据处理方式。

支持XML： LINQ to XML

- LINQ to XML 提供利用 .NET Framework Language-Integrated Query (LINQ) 的内存中 XML 编程 API。 它相当于经过改进的、重新设计的文档对象模型 (DOM) XML 编程接口。LINQ to XML 使用最新的 .NET Framework 语言功能。
- 它将 XML 文档置于内存中，这一点很像文档对象模型 (DOM)。然后可以查询和修改 XML 文档，修改之后，可以将其另存为文件，也可以将其序列化然后通过网络发送。

- （复习）所有 LINQ 查询操作都由以下三个不同的操作组成：
  - 获取数据源：（ XElement ffh = XElement.Load(“name.xml”);）
  - 创建查询：Ienumerable<?> ffhs = from lt in ffh.Descendants(“xx") select (?)lt;
  - 执行查询。foreach语句

支持XML： LINQ to XML

- 例：从purchaseOrder.xml文件中获取采购单每个项元素的部件号属性值（XMLExample/LINQ to XML）

```
XElement ffh = XElement.Load("purchaseOrder.xml");
IEnumerable<string> partNos =
    from FfhItem in ffh.Descendants("Item")
     select (string)FfhItem.Attribute("PartNumber");
foreach (var vT in partNos)
{    Console.WriteLine(vT.ToString()); }
```

ffh.Descendants(XName) :按文档顺序返回此文档或元素的子代元素Xname的集合。

支持XML： LINQ to XML

- LINQ to XML 提供了改进的 XML 编程接口。 通过 LINQ to XML，对 XML 编程时，您可以实现任何预期的操作，包括：
  - 从文件或流加载 XML。
  - 将 XML 序列化为文件或流。
  - 使用函数构造从头开始创建 XML。
  - 使用类似 XPath 的轴查询 XML。
  - 使用 Add、Remove、ReplaceWith 和 SetValue 等方法对内存 XML 树进行操作。
  - 使用 XSD 验证 XML 树。
  - 使用这些功能的组合，可将 XML 树从一种形状转换为另一种形状。

  （如：下面为创建 XML 树的示例代码）

支持XML： LINQ to XML

```
XElement treeExam = new XElement("通讯录",
                    new XElement("名单",
                    new XElement("姓名", "老朱"),
                    new XElement("电话", "3492000",
                        new XAttribute("类型", "家庭")),
                    new XElement("电话", "37201111",
                        new XAttribute("类型", "单位")),
                    new XElement("地址",
                    new XElement("街道", "徽州大道 1121"),
                    new XElement("城市", "合肥"),
                    new XElement("区", "包河"),
                    new XElement("邮编", "230051")
                    )       )    );
```

XElement 类是 LINQ to XML 中的基础类之一。 它表示一个 XML 元素。 可以使用该类创建元素；更改元素内容；添加、更改或删除子元素；向元素中添加属性；或以文本格式序列化元素内容。

支持XML： XPathNavigator

- XPathNavigator 类:

  为定位和编辑 XML 数据提供游标模型。可读取 XML 数据、 选择、计算和匹配 XML 数据、浏览节点，提取 XML 数据，以及访问强类型 XML 数据，插入、修改和移除节点和值以及验证架构。

支持XML： XmlDocument

- XmlDocument类：表示 XML 文档，此类实现 DOM API,允许对该文档的导航和编辑。包括的一些成员有：
  - 方法
    - Load：加载指定的 XML 数据。
    - Save将 XML 文档保存到指定的位置。
    - InsertAfter将指定的节点紧接着插入指定的引用节点之后。
    - InsertBefore将指定的节点紧接着插入指定的引用节点之前。
    - SelectNodes：选择匹配 XPath 表达式的节点列表。

支持XML： XmlDocument

- 属性
  - HasChildNodes：指示节点是否有任何子节点 。
  - LastChild：获取节点的最后一个子级。
  - FirstChild：获取节点的第一个子级。
  - ParentNode：获取该节点（对于可以具有父级的节点）的父级。

(例：更改Booksort.xml中所有 Austen 写的书的价格为15.95。 XMLExample-“XmlDocument”)

VbExam2-“XmlDocument”的部分代码

```
Dim doc As XmlDocument = New XmlDocument
doc.Load("booksort.xml")
Dim book As XmlNode
Dim nodeList As XmlNodeList
Dim root As XmlNode = doc.DocumentElement
nodeList = root.SelectNodes("descendant::book[author/last-
 name='Austen']")
'Change the price on the books.
For Each book In nodeList
    book.LastChild.InnerText = "15.95"
Next
'Create a new node.
Dim elem As XmlElement = doc.CreateElement("price")
elem.InnerText = "123.05"
'Add the node to the document.
root.InsertAfter(elem, root.FirstChild)
Console.WriteLine("Display the modified XML document....")
doc.Save(Console.Out)
```

返回

支持XML：XmlReader

- **XmlReader** 类提供对 XML 数据进行快速、非缓存、只进访问的读取器。

  XmlReader 实例使用 Create 方法创建。 XmlReaderSettings类指定在 Create 方法创建的 XmlReader 对象上支持的一组功能。XmlReaderSettings设定的属性包括：

  - ValidationType ： 指示 **XmlReader** 在读取时是否使用文档类型定义 (DTD) 或架构定义语言 (XSD) 架构强制进行验证。
  - CheckCharacters：指示是否进行非法字符或无效的XML名字检查。
  - IgnoreComments：指示是否忽略注释。
  - IgnoreWhitespace：指示是否忽略无关紧要的空白。
- XmlTextReader 类是 XmlReader 的实现，为 XML 文本提供分析器。

  （例：XMLExample-“XmlReader”）

支持XML： XmlWriter

- XmlWriter 是定义用于编写 XML 的接口的抽象基类。XmlWriter 提供只进、只读、不缓存的 XML 流生成方法。

  XmlWriter 实例使用 Create 方法创建。XmlWriterSettings 类用于指定要在新的 XmlWriter 对象上启用的功能集。XmlWriterSettings 属性包括：

  - Encoding：要使用的文本编码（UTF8,ASCII,Unicode等）。
  - Indent ：指示是否缩进元素。
  - IndentChars：缩进时要使用的字符串。当 Indent 属性设置为 **true** 时使用此设置。
- 使用 XmlTextWriter 编写 XML

  (例：XMLExample-“Xml Writer”)

反射：System.Reflection

- 在运行时检查一个类型的元数据的操作称为“反射”。Object定义了方法GetType用于实现反射功能。 GetType返回当前实例的确切运行时类型。
- System.Reflection 命名空间提供了一些类支持反射功能。另外System.Reflection 命名空间还提供了加载类型、方法和字段的托管视图以及动态创建和调用类型的功能。

  可用Ildasm查看一个程序集的元数据

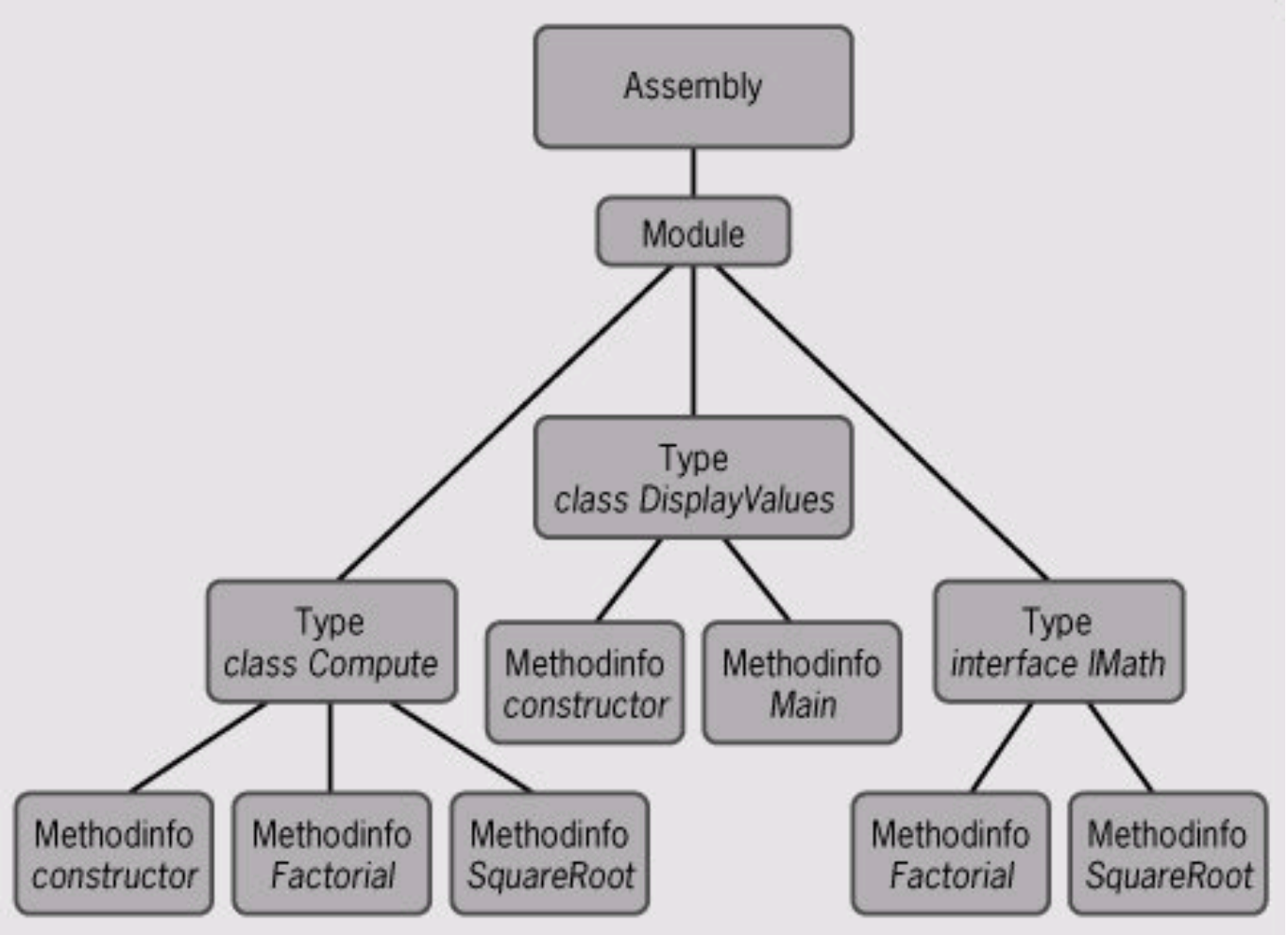
反射：System.Reflection

- 为能以编程方式访问元数据，System.Reflection为程序集的元数据内的每一种信息定义了一个类。它们都属于MemberInfo的子类。

  MemberInfo发现成员特性并提供对成员元数据的访问。MemberInfo类是用于获取类的所有成员（构造函数、事件、字段、方法和属性）信息类的抽象基类。

- 这些类的实例被组织为层次结构，下图是前章例子的一个描述

反射： System.Reflection



反射：System.Reflection

- MemberInfo类是许多“Info”作为名字结尾类的基类，这些类被称为反射“Info”类。 MemberInfo类的子类有：
  - EventInfo：发现事件的特性并提供对事件元数据的访问。
  - FieldInfo：发现字段特性并提供对字段元数据的访问。
  - MethodBase：提供有关方法和构造函数的信息。是 MethodInfo 和 ConstructorInfo 的基类。
    - MethodInfo：发现方法的特性并提供对方法元数据的访问。
    - ConstructorInfo：发现类构造函数的特性并提供对构造函数元数据的访问。
  - PropertyInfo：发现属性的特性并提供对属性元数据的访问。
- MemberInfo类提供了对“Info”类调用的共同方法。其中一个很有用的方法是GetCustomAttributes（返回所有在该成员上定义的特性 ）

反射：System.Reflection

- Type 类表示类型声明：类类型、接口类型、数组类型、值类型和枚举类型。Type 为 System.Reflection 功能的根，也是访问元数据的主要方式。使用 Type 的成员获取关于类型声明的信息，如构造函数、方法、字段、属性和类的事件，以及在其中部署该类的模块和程序集。（注： Type 类属于System命名空间）

反射：System.Reflection

- 如，下例列出了指定类的每个成员的 **Name** 和 **DeclaringType** 属性。

```
Type MyType = Type.GetType("System.String");
MemberInfo[] Myarray = MyType.GetMembers();
     //或 GetMethods(), GetProperties()等

// Get and display the DeclaringType method.
Console.Write("{0}.", MyType.FullName);
foreach (MemberInfo MyI in Myarray)
{
    Console.Write(“\n” + MyI .Name + " declaring type - " +
                                MyI.DeclaringType);
}
```

反射：System.Reflection

- 如：下例显示命令按钮的所有属性和方法
  (例：C#-exam\反射举例)

```
using System.Reflection;
foreach (PropertyInfo property in
    typeof(Button).GetProperties())
  Console.WriteLine(property.Name);

foreach (MethodInfo method in
    typeof(Button).GetMethods())
  Console.WriteLine(method.Name);
```

反射：System.Reflection

- System.Reflection 命名空间还提供了加载类型、方法和字段的托管视图以及动态创建和调用类型的功能。下面代码实现对另外的程序集的加载和调用。

```
Form frmT;
System.Reflection.Assembly assemblyT;
assemblyT = System.Reflection.Assembly.LoadFrom(exe文件);
//assemblyT = System.Reflection.Assembly.Load(Dll文件);
frmT =(Form)assemblyT.CreateInstance(命名空间.窗体名);
frmT.ShowDialog();
```

返回

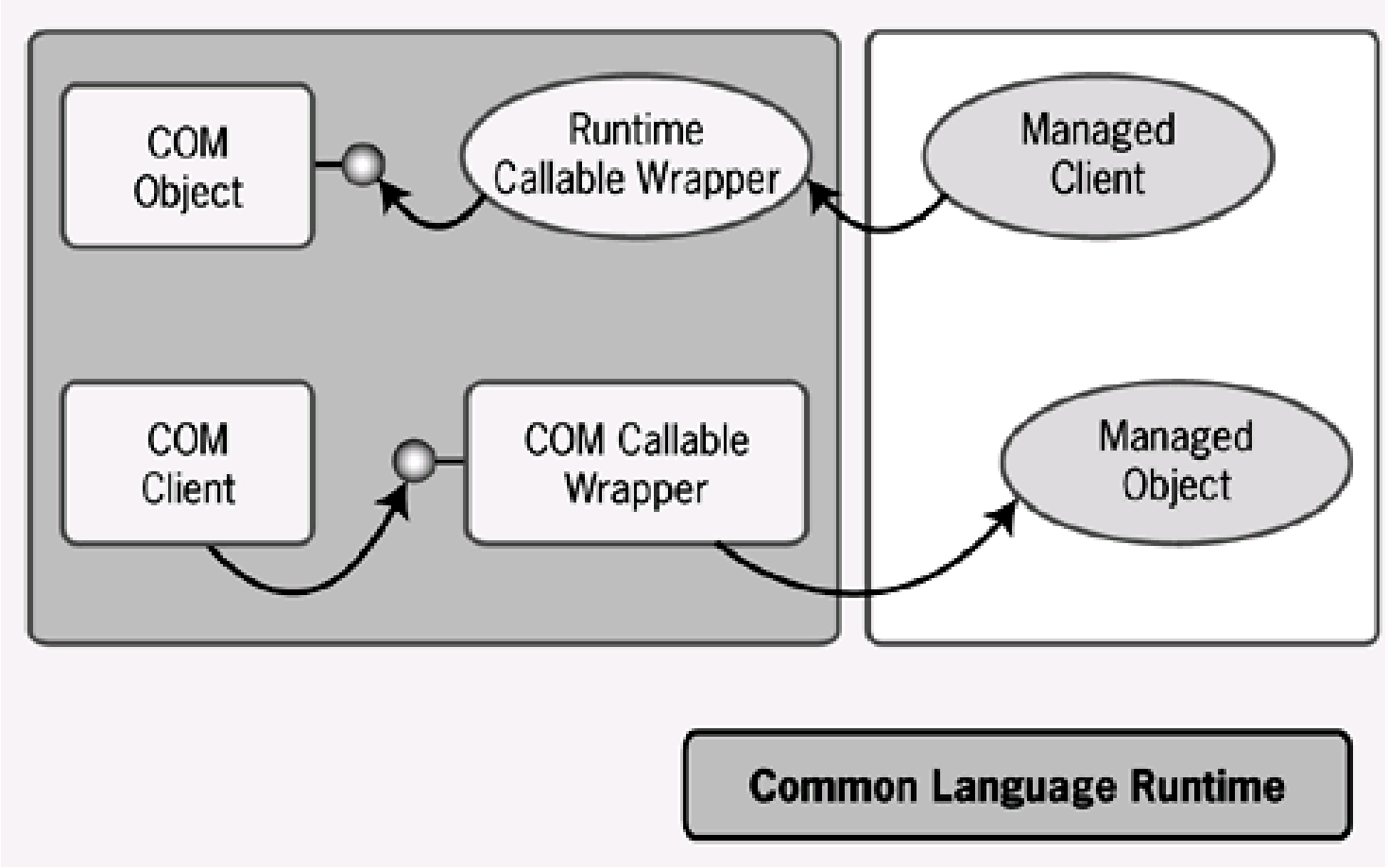
互操作性：System.Runtime.InteropServices

- 提供各种与COM 互操作和平台调用服务的功能。（与非托管代码交互操作）
  - 互操作能力，使得COM代码和托管代码可以相互访问。
  - 利用平台调用这种服务，托管代码可以调用在动态链接库 (DLL)（如 Win32 API 中的 DLL）中实现的非托管函数。

互操作性：System.Runtime.InteropServices

- .Net框架的互操作依赖二种包装器
  - 运行期可调用包装器(runtime callable wrapper, RCW)：使托管代码可访问COM对象。
  - COM可调用包装器(COM callable wrapper, CCW)：允许COM代码访问托管代码。

互操作性：System.Runtime.InteropServices



互操作性：System.Runtime.InteropServices

- 可利用工具创建包装器
- .NET Framework 提供了在执行从类型库向程序集的转换时所需的工具和 API。有几种方法可用于从类型库生成程序集，每种方法都会产生相同的结果。一般情况下都使用使用 .NET Framework SDK 提供的类型库导入程序 (Tlbimp.exe)。

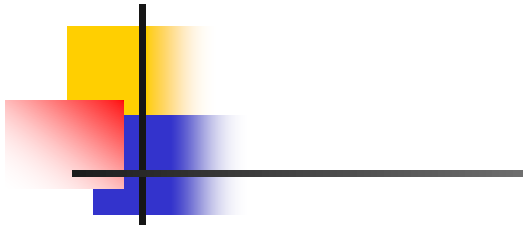
互操作性：System.Runtime.InteropServices

- 类型库导入程序(Tlbimp.exe)将 COM 类型库中的类型定义转换为公共语言运行库程序集中的等效定义。Tlbimp.exe 的输出为二进制文件（程序集），该文件中包含在原始类型库中定义的类型的运行库元数据。 语法是：

  tlbimp *tlbFile* [*options*]

  在这个命令中，*tlbfile* 是包含 COM 类型库的文件，*options* 是选项。如：下面的命令生成名为 myTest.dll 的元数据 DLL。

  tlbimp myTestCom.dll /out:myTest.dll



- 举例

互操作性：System.Runtime.InteropServices

- .NET Framework 提供了在执行从程序集到类型库的转换时所需的工具和 API。虽然有几种机制可用于生成类型库，但每种机制都会产生相同的结果。一般情况下都使用使用 .NET Framework SDK 提供的类型库导出程序 (Tlbexp.exe)。

互操作性：System.Runtime.InteropServices

- 类型库导出程序 (Tlbexp.exe)：生成一个类型库，该类型库包含程序集中定义的类型的定义。
语法：

  tlbexp *assemblyName* [*options*]

  *其中，assemblyName*为其导出类型库的程序集。如下面的命令生成一个与 myTest.dll 中的程序集同名的类型库(.tlb)。

  tlbexp myTest.dll

  再如，下面的命令生成一个名为 clipper.dll 的类型库。

  tlbexp myTest.dll /out:clipper.dll

互操作性：System.Runtime.InteropServices

- 由于COM使用注册表来决定应该为某个特定的类装载那些代码，因此被COM客户访问的程序集必须有相应的注册表入口。
- 程序集注册工具 (Regasm.exe)：读取程序集中的元数据，并将所需的项添加到注册表中。

  注册表允许 COM 客户程序以透明方式创建 .NET Framework 类。类一经注册，任何 COM 客户程序都可以使用它，就好像该类是一个 COM 类。类仅在安装程序集时注册一次。程序集中的类实例直到被实际注册时，才能从 COM 中创建。 语法：

  regasm *assemblyFile* [*options*]

互操作性：System.Runtime.InteropServices

下面的命令注册 myTest.dll 中包含的所有公共类。

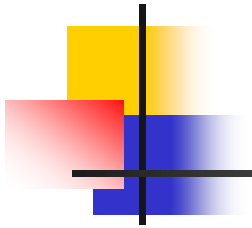
```
regasm myTest.dll
```

下面的命令生成文件 myTest.reg，该文件包含所有必要的注册表项。此命令不更新注册表。

```
regasm myTest.dll /regfile:myTest.reg
```

下面的命令注册 myTest.dll 中包含的所有公共类，并生成和注册类型库 myTest.tlb，该类型库包含 myTest.dll 中定义的所有公共类型的定义。

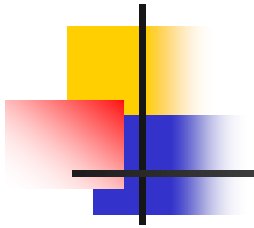
```
regasm myTest.dll /tlb:myTest.tlb
```



- 下例说明如何从 Visual Basic 6.0 应用程序使用 .NET 对象。可使用相同的技术从任何 COM 应用程序创建该对象，包括用 Visual C++ 6.0、VBScript 或 JScript 生成的应用程序。
  - 定义.NET 类TestServer

```
namespace TestServer
{  public interface Itest
    {  DateTime GetTime(); }
   public class Test : Itest
   {   DateTime ITest.GetTime()
       { return DateTime.Now; }
   }
}
```

编译：csc /target:library TestServer.cs



- 用 tlbexp.exe 从托管程序集生成一个类型库:TestServer.Lib。

  tlbexp TestServer.dll

- 用 regasm.exe 实用工具来注册程序集。可使用 /u 选项来卸载程序集。

  regasm TestServer.dll

- 然后可通过 Visual Basic 6.0 中的“项目/引用”对话框向项目添加由 tlbexp.exe 生成的类型库。一旦向项目添加了对类型库的引用，程序集中所定义的类型就能够在 Visual Basic 代码中直接引用了。

```
Dim dotNETServer As TestServer.ITest
Set dotNETServer = New TestServer.Test
Debug.Print ".NET server returned: " +
    FormatDateTime(dotNETServer.GetTime, vbGeneralDate)
```

互操作性：System.Runtime.InteropServices

- 利用平台调用这种服务，托管代码可以调用在动态链接库 (DLL)（如 Win32 API 中的 DLL）中实现的非托管函数。用特性DllImportAttribute指定。

  在 C# 中，**DllImport**特性用于识别包含导出函数的实际 DLL 的名称。必须将方法定义为静态和外部的才能应用特性。

  如：

```
public class Win32 {
   [DllImport("Kernel32.dll")]
   public static extern void GetSystemTime(ref SystemTime
   sysTime);
            ...
}
```

  一旦定义了类和方法，就可像调用任何其他方法那样调用该方法了。

- C#中，如果方法的名称与 DLL 中实际的导出名称不同，则可使用 DllImport.EntryPoint 属性将该函数映射到正确的入口点。

如：

```
using System.Runtime.InteropServices;
public class Win32 {
    ...
    [DllImport("User32.dll", EntryPoint="MessageBox", CharSet=CharSet.Auto)]
    public static extern int MsgBox(int hWnd, String text, String caption, uint
    type);
}

public class TestPInvoke {
    public static void Main() {
      ...
      String date = ...;
      Win32.MsgBox(0, date, "PInvoke Sample", 0);
    }
}
```

Windows GUIS：System.Windows.Forms

- .NET框架创建用户界面的三种方案
  - 控制台（命令行）
  - Windows Forms（窗体）
  - ASP.NET（网页）

Windows GUIS：System.Windows.Forms

- 窗体是应用程序的基本单元。System.Windows.Forms包含的类可创建基于 Windows 的应用程序，在此命名空间中，您将找到可添加到窗体中创建用户界面的窗体 (Form) 类和其他多种控件。

Windows GUIS：System.Windows.Forms

- 使用Windows Forms开发你的GUIs程序
  - Forms具有属性、方法、事件
  - Forms是容器，可以包含控件
- 控件
  - 控件也有属性、方法、事件
  - Windows Forms可以仿真ActiveX控件

Windows GUIS：System.Windows.Forms

- 此命名空间中的类可分为：
  - **Control**、**User Control** 和 **Form**。

    System.Windows.Forms 命名空间中的大多数类都从 Control 类派生而来。

    Control 类为在 Form 中显示的所有控件提供基本功能。

    Form 类表示应用程序内的窗口。这包括对话框，无模式窗口和多文档界面 (MDI) 客户端窗口及父窗口。

    若要创建由其他控件组成的自定义控件，请使用 UserControl 类。

Windows GUIS：System.Windows.Forms

- 此命名空间中的类可分为：（续）
  - **Controls**

    System.Windows.Forms 命名空间提供各种控件类，这些控件类能创建丰富的用户界面。

    某些控件用于在应用程序内进行数据输入，如 TextBox 和 ComboBox 控件。其他控件显示应用程序数据，如 Label 和 ListView。

    命名空间还提供用于调用应用程序内的命令的控件，比如 Button 和 ToolBar。

    此外，还可以使用 PropertyGrid 控件，创建自己的 Windows 窗体设计器，以显示设计器可见的控件属性。

Windows GUIS：System.Windows.Forms

- 此命名空间中的类可分为：（续）
  - **Components**。

    除控件外，**System.Windows.Forms** 命名空间还提供其他一些类，这些类不是从控件类派生的，但仍然向基于 Windows 的应用程序提供可视化功能。某些类，例如 ToolTip 和 ErrorProvider，扩展了这些功能或者向用户提供信息。还有一些类，比如 Menu、MenuItem 和 ContextMenu，向用户提供显示菜单的能力，以便从应用程序内调用命令。Help 和 HelpProvider 类可使您能够向应用程序的用户显示帮助信息。

Windows GUIS：System.Windows.Forms

- 此命名空间中的类可分为：（续）
  - **Common Dialog Boxes**。

    Windows 提供许多通用对话框，在执行诸如打开和保存文件、操作字体或文字颜色，或打印之类的任务时，使应用程序具有一致的用户界面。

    OpenFileDialog 和 SaveFileDialog 类提供显示对话框，以便允许用户浏览和输入要打开或保存的文件名。

    FontDialog 类显示一个对话框，以更改应用程序所使用的 Font 对象的元素。

    PageSetupDialog、PrintPreviewDialog 和 PrintDialog 类显示对话框，以便允许用户控制打印文档的各个方面。

    除通用对话框外，**System.Windows.Forms** 命名空间还提供 MessageBox 类，用于显示消息框，该消息框可以显示和检索用户的数据。

Windows GUIS：System.Windows.Forms

- 此命名空间中的类可分为：（续）
  - **System.Windows.Forms** 命名空间内还有许多类，它们为前面的摘要中提及的类提供支持。支持类的示例有枚举、事件参数类，以及控件和组件内的事件使用的委托。

根据常规功能列出的 Windows 窗体控件

- 文本编辑
  - TextBox：显示设计时输入的文本，它可由用户在运行时编辑或以编程方式更改。
  - RichTextBox ：使文本能够以纯文本或 RTF 格式显示。
- 文本显示（只读）
  - Label ：显示用户无法直接编辑的文本。
  - LinkLabel：将文本显示为 Web 样式的链接，并在用户单击该特殊文本时触发事件。该文本通常是到另一个窗口或 Web 站点的链接。
  - StatusBar ：通常在父窗体的底部使用有框架窗口显示该应用程序的当前状态信息。

根据常规功能列出的 Windows 窗体控件

- 从列表中选择
  - CheckedListBox：显示一个可滚动的项列表，每项旁边都有一个复选框。
  - ComboBox：显示一个下拉式项列表。
  - DomainUpDown：显示用户可用向上和向下按钮滚动的文本项列表。
  - ListBox：显示一个文本项和图形项（图标）列表。
  - ListView：在四个不同视图之一中显示项。这些视图包括纯文本视图、带有小图标的文本视图、带有大图标的文本视图和详细信息视图。
  - NumericUpDown：显示用户可用向上和向下按钮滚动的数字列表。
  - TreeView：显示一个节点对象的分层集合，这些节点对象由带有可选复选框或图标的文本组成。

根据常规功能列出的 Windows 窗体控件

- 图形显示
  - PictureBox：在一个框架中显示图形文件（如位图和图标）。
- 图形存储
  - ImageList ：用作图像的储存库。**ImageList** 控件及其包含的图像能够在应用程序之间重复使用。
- 值的设置
  - CheckBox：显示一个复选框和一个文本标签。通常用来设置选项。
  - CheckedListBox：显示一个可滚动的项列表，每项旁边都有一个复选框。
  - RadioButton：显示一个可打开或关闭的按钮。
  - TrackBar：允许用户通过沿标尺移动“缩略图”来设置标尺上的值。

根据常规功能列出的 Windows 窗体控件

- 数据的设置
  - DateTimePicker：显示一个图形日历以允许用户选择日期或时间。
  - MonthCalendar：显示一个图形日历以允许用户选择日期范围。
- 对话框
  - ColorDialog：显示允许用户设置界面元素的颜色的颜色选择器对话框。
  - FontDialog：显示允许用户设置字体及其属性的对话框。
  - OpenFileDialog：显示允许用户定位文件和选择文件的对话框。
  - PrintDialog：显示允许用户选择打印机并设置其属性的对话框。
  - PrintPreviewDialog：显示一个对话框，该对话框显示
  - PrintDocument： 对象打印时的样子。
  - SaveFileDialog：显示允许用户保存文件的对话框。

根据常规功能列出的 Windows 窗体控件

- 菜单控件
  - MainMenu：提供创建菜单的设计时界面。
  - ContextMenu：实现当用户右击对象时出现的菜单。
- 命令
  - Button：用来启动、停止或中断进程。
  - LinkLabel：将文本显示为 Web 样式的链接，并在用户单击该特殊文本时触发事件。该文本通常是到另一个窗口或 Web 站点的链接。
  - NotifyIcon：在表示正在后台运行的应用程序的任务栏的状态通知区域中显示一个图标。
  - ToolBar：包含一个按钮 (Button) 控件的集合。

根据常规功能列出的 Windows 窗体控件

- 其他
  - Panel：将一组控件分组到未标记、可滚动的框架中。
  - GroupBox：将一组控件（如单选按钮(RadioButton)）分组到带标记、不可滚动的框架中。
  - TabControl：提供一个选项卡式页面以有效地组织和访问已分组对象。

远程安装Windows Forms应用程序:ClickOnce

- ClickOnce 部署概述

  ClickOnce 是一种部署技术，它可创建自行更新的基于 Windows 的应用程序。

- ClickOnce 克服了部署中所固有的三个主要问题：

  1)更新应用程序的困难

  使用 Microsoft Windows Installer 部署，每次应用程序更新，用户都必须重新安装整个应用程序；使用 ClickOnce 部署，则可以自动提供更新。只有更改过的应用程序部分才会被下载，更新。

远程安装Windows Forms应用程序:ClickOnce

- ClickOnce 克服了部署中原有的三个主要问题：(续)

  2)对用户的计算机的影响

  使用 Windows Installer 部署时，应用程序通常依赖于共享组件，这便有可能发生版本冲突；而使用 ClickOnce 部署时，每个应用程序都是独立的，不会干扰其他应用程序。

  3)安全权限。

  Windows Installer 部署要求管理员权限并且只允许受限制的用户安装；而 ClickOnce 部署允许非管理用户安装应用程序并仅授予应用程序所需要的那些代码访问安全权限。

远程安装Windows Forms应用程序:ClickOnce

- ClickOnce 部署策略

有三种不同的策略用于部署 ClickOnce 应用程序；

1）从 Web 或网络共享安装

应用程序部署到 Web 服务器或网络文件共享。当最终用户要安装应用程序时，在网页上单击图标或是在文件共享上双击图标。然后会在最终用户的计算机上下载、安装并启动应用程序。

2）从 CD 安装

应用程序会部署到可移动媒体（如 CD-ROM 或 DVD）。

3）从 Web 或网络共享启动应用程序

与第一种策略相似，但应用程序在此策略中的行为类似于 Web 应用程序。当用户在网页上单击链接（或在文件共享上双击图标）时，应用程序被启动。

远程安装Windows Forms应用程序:ClickOnce

- ClickOnce自动更新策略

ClickOnce会定期读取其部署清单文件，以查看是否有可用的应用程序更新。如果有，则会下载并运行应用程序的新版本。为提高效率，仅下载那些已更改的文件。有三种基本策略可以使用：

1）在应用程序启动时检查更新;

2）在应用程序启动后检查更新（在后台线程中运行）或是提供进行更新的用户界面。

3）可确定应用程序检查更新的时间间隔，并且可以强制必须执行更新。

注:应用程序更新需要网络连接。如不存在网络连接，则应用程序会在不检查更新的情况下运行。

远程安装Windows Forms应用程序:ClickOnce

- ClickOnce应用程序依赖基于CLR的代码访问安全机制

  ClickOnce 应用程序在本质上是被隔离的，所以安装或运行 ClickOnce 应用程序不会干扰现有的应用程序。

  ClickOnce 应用程序是完全独立的；每个 ClickOnce 应用程序都安装到一个安全的基于每个用户、每个应用程序的缓存中，并从该缓存运行。默认情况下，ClickOnce 应用程序运行在 Internet 或 Intranet 安全区域中。如果有必要，应用程序可以请求提升的安全权限。

远程安装Windows Forms应用程序:ClickOnce

- **ClickOnce 部署的工作方式**

  核心 ClickOnce 部署结构基于两个 XML 清单文件：一个应用程序清单和一个部署清单。

  1. 应用程序清单

     描述应用程序本身，包括程序集、组成应用程序的依赖项和文件、所需的权限以及提供更新的位置。

  2. 部署清单

     描述如何部署应用程序，包括应用程序清单的位置以及客户端应运行的应用程序的版本。

远程安装Windows Forms应用程序:ClickOnce

- 发布 ClickOnce 应用程序
  - 发布到 Web （要求IIS有FrontPage扩展）
    1. 在“解决方案资源管理器”中，右击项目节点，然后选择“发布”。
    2. 在“要在何处发布应用程序?”页上，输入一个有效的URL，如：http://202.38.64.11/~shizhu/，然后单击“下一步”。
    3. 在“该应用程序可以脱机使用吗?”页中，单击适当的选项：
       (如果要从发布位置直接运行应用程序，则单击“否”，该应用程序只能联机使用”。“开始”菜单上不创建快捷方式)
    4. 单击“下一步”继续。单击“完成”以发布应用程序。

    发布状态显示在任务栏的状态通知区域中。

远程安装Windows Forms应用程序:ClickOnce

- 发布 ClickOnce 应用程序(续)
  - **发布到文件共享**
    - 在“要在何处发布应用程序?”页上，输入一个有效的文件路径如： ftp://202.38.64.11/public_html/，然后单击“下一步”。
    - 在“用户如何安装应用程序?”页中，选择用户安装应用程序的位置：
      - 如果用户从网站安装，则单击“从网站”，并输入与上一步中输入的文件路径相对应的 URL，然后单击“下一步”。（此选项通常在将 FTP 地址指定为发布位置时使用。从 FTP 的直接下载不受支持，因此需要在此处输入 URL。）
      - 如果用户从文件共享直接安装应用程序，则单击“从 UNC 路径或文件共享”，然后单击“下一步”。（此选项用于形式为 c:\deploy\myapp 或 \\server\myapp 的发布位置。）
      - 如果用户从可移动媒体安装，则单击“从 CD-ROM 或 DVD-ROM”，然后单击“下一步”。
    - 在“该应用程序可以脱机使用吗?”页中，单击适当的选项;单击“下一步”继续。单击“完成”以发布应用程序。

远程安装Windows Forms应用程序:ClickOnce

- 发布 ClickOnce 应用程序(续)
  - 发布到 CD-ROM 或 DVD-ROM
    - 在“您要在哪里发布该应用程序?”页中，输入发布应用程序的文件路径或 FTP 位置（如 d:\deploy）。然后，单击“下一步”继续。
    - 在“用户如何安装应用程序?”页中，单击“从 CD-ROM 或 DVD-ROM”，然后单击“下一步”。
    - 如果在 CD-ROM 上发布应用程序，可能会希望从网站提供更新。在“应用程序将到哪里检查更新?”页中，选择更新选项：
      - 如果应用程序将检查更新，则单击“该应用程序将从下列位置检查更新”，然后输入发布更新的位置。输入的位置可以是文件位置、网站或 FTP 服务器。
      - 如果应用程序不检查更新，则单击“该应用程序将不检查更新”。
    - 单击“下一步”继续。
    - 单击“完成”以发布应用程序。

远程安装Windows Forms应用程序:ClickOnce

- 选择 ClickOnce 更新策略
  - 在“解决方案资源管理器”中，右击应用程序项目并选择“属性”。出现“项目设计器”；
  - 单击“发布”选项卡在“项目设计器”中打开“发布”页；
  - 单击“更新”按钮，在“应用程序更新”对话框中选择。

System.Net.Mail 命名空间

- System.Net.Mail 命名空间

  System.Net.Mail 命名空间包含用于将电子邮件发送到简单邮件传输协议 (SMTP) 服务器进行传送的类。 主要有:

  MailMessage 类：表示邮件的内容。

  SmtpClient 类：将电子邮件传输到您指定用于邮件传送的 SMTP 主机。

  Attachment 类：创建邮件附件。

System.Net.Mail 命名空间

- MailMessage 类

  表示可以使用 SmtpClient 类发送的电子邮件。

  可用属性指定电子邮件的发件人、收件人和内容。

邮件部分	属性
发件人	From
收件人	To
抄送 (CC)	CC
密件抄送 (BCC)	Bcc
附件	Attachments
主题	Subject
邮件正文	Body

System.Net.Mail 命名空间

- MailMessage 类(续)

  另外，使用 AlternateViews 属性可指定一个电子邮件不同格式的副本。

  组织好电子邮件后，可以使用 Send 或 SendAsync 方法发送邮件。

System.Net.Mail 命名空间

- SmtpClient类

  允许应用程序使用简单邮件传输协议 (SMTP) 来发送电子邮件。

  下表中显示的类用于构造可以使用 **SmtpClient** 发送的电子邮件。

类	说明
Attachment	表示文件附件。此类允许您将文件、流或文本附加到电子邮件中。
MailAddress	表示发件人和收件人的电子邮件地址。
MailMessage	表示电子邮件。

System.Net.Mail 命名空间

- SmtpClient类（续）

  若要使用 **SmtpClient** 构造并发送电子邮件，必须指定以下信息：

  - 用来发送电子邮件的 SMTP 主机服务器。（Host 和 Port属性）
  - 身份验证凭据（如果 SMTP 服务器要求）。（ Credentials 属性）
  - 发件人的电子邮件地址。（接受 *from* 参数的 Send 方法和 SendAsync 方法。还有 MailMessage.From 属性）
  - 收件人的电子邮件地址。（接受 *recipient* 参数的 Send 方法和 SendAsync 方法。还有 MailMessage.To 属性）
  - 邮件内容。（接受 *body* 参数的 Send 方法和 SendAsync 方法。还有 MailMessage.Body 属性）

System.Net.Mail 命名空间

- SmtpClient类（续）
  - 若要在电子邮件中包括附件，首先使用 Attachment 类创建附件，再使用 MailMessage.Attachments 属性将附件添加到邮件中。
  - 可以使用应用程序或计算机配置文件来指定所有 **SmtpClient** 对象的默认主机、端口和凭据值。
  - 若要发送电子邮件并在等待电子邮件传输到 SMTP 服务器期间阻止其他操作（同步），使用 Send 方法。若要允许程序的主线程在传输电子邮件的过程中继续执行，使用 SendAsync（异步）方法。SendAsync 操作完成时会引发 SendCompleted 事件。若要取消异步电子邮件传输，使用 SendAsyncCancel 方法。

System.Net.Mail 命名空间

- Attachment 类

  表示电子邮件的附件。

  若要将附件添加到邮件中，请将附件添加到 MailMessage.Attachments 集合中。

例：创建和发送包括附件的电子邮件。

```
// Create a message and set up the recipients.
MailMessage message = new MailMessage(
    “test@ustc.edu”,“shizhu@ustc.edu.cn",“标题","邮件正文");
//附件
Attachment data = new Attachment(“data.xls”);
// Add the file attachment to this e-mail message.
message.Attachments.Add(data);
//Send the message.
SmtpClient client = new SmtpClient(“202.38.64.8”);
// Add credentials if the SMTP server requires them.
client.Credentials = CredentialCache.DefaultNetworkCredentials;
client.Send(message);
```

另一例（服务器要求有发送帐户的用户名和密码）

```
MailAddress from = new MailAddress(“shizhu@ustc.edu”); // 发件人邮件地址
MailAddress to = new MailAddress(“test@ustc.edu.cn”);  // 收件人邮件地址
MailMessage message = new MailMessage(from, to);
message.Subject = “test”;//设置邮件主题
message.IsBodyHtml = true;//设置邮件正文为html格式
message.Body = “This is a test string<br>Return”;//设置邮件内容
SmtpClient client = new SmtpClient(“202.38.64.8”);
//设置发送邮件身份验证方式
//注意如果发件人地址是abc@def.com，则用户名是abc而不是abc@def.com
client.Credentials = new NetworkCredential(userName, password);
client.Send(message);
```