SqlCommand对象的方法：

1. ExecuteScalar方法：返回查询所返回的结果集中第一行的第一列，忽略其他列或行.
2. ExecuteNonQuery方法：返回受command命令影响的行数。
3. ExecuteReader方法：返回一个SQLDataReader对象的实例。

SqlDataReader对象：

SqlReader对象是数据读取器对象，提供一个只读向前的游标。

Read方法：SqlCommand对象的ExecuteReader方法会返回一个SqlDataReader对象，可用过Read方法逐条读取SqlDataReader对象。

SqlDataAdapter对象：

DataAdapter对象是一个数据适配器对象，是DataSet与数据源之间的桥梁。

Fill方法：用数据填充Dataset 对象。

Fillschema方法：添加 [DataTable](https://msdn.microsoft.com/zh-cn/library/system.data.datatable(v=vs.110).aspx) 到 [DataSet](https://msdn.microsoft.com/zh-cn/library/system.data.dataset(v=vs.110).aspx) 和配置架构以匹配数据源中的架构。

Update方法：方便一张数据表进行大量更改。

Dataset对象：

Dataset对象就像一个存放在内存中的小型数据库。

Merge方法： 将指定Dataset与当前Dataset合并。

Copy方法： 返回一个具有该DataSet相同的结构和数据的Dataset。

GridView:

DataSource属性：用于获取或设置DataGridView控件所显示数据的数据源。

修改数据：通过Datatable的ImportRow方法将更改后的数据复制到一个DataTable中，然后通过DataAdapter对象的Update方法将DataTable中的数据更新到数据库中。

ImportRow方法:用于将DataRow复制到DataTable中。

LINQ

Var关键字：用于创建一个局部隐形变量。

Lambda表达式：

Lambda运算符：“=>”,运算符左边是输入参数，右边包含表达式或语句块。

LINQ to SQL

添加一个表中的数据到gridview控件：

①创建一个项目，并添加一个“Linq to SQL”类

②打开localdb,把需要查询表映射到设计视图中

③程序会自动生成一个context类

④添加一个webform 并添加控件

⑤编译代码：利用context类和连接字符串连接localdb,利用LINQ查询语句查询数据，然后把数据绑定到Gridview控件。

向数据库中添加数据：

使用LINQ向数据库中添加数据时，需要用到InsertOnSubmit方法和SubmitChanges方法。

删除数据：

使用LINQ删除数据库中的数据时，需要用到DeleteAllOnSubnit方法和SubmitChanges方法。

LINQ to Objects

LINQ to Objects能够直接使用LINQ查询IEnumerable或IEnumerable<T>集合，而不需要使用LINQ提供程序或API，可以说，使用LINQ能查询任何可枚举的集合。

LINQ to DataSet

AsEnumerable方法：可以将DataTable对象转换为EnumerableRowCollection<DataRow>对象。

CopyToDataTable方法：用来将IEnumerable<T>对象中的数值赋值到DataTable对象中。

AsDataView方法：用来创建并返回支持LINQ的DataView对象。

Take方法：用来从序列的开头返回指定数量的连续元素。

Sum方法：用来计算数值序列之和。

程序调试

断点操作

开始执行（启动调试命令F5）

中断执行

单步执行和逐过程执行（逐语句F11，逐过程F10）

运行到指定位置（在指定代码行上右击，选择“运行到光标处”命令）

异常处理

异常处理语句

Try…catch语句

Try

{

被监管的代码

}

Catch（异常类名 异常变量名）

{

异常处理

}

Throw语句

Throw语句用于主动引发一个异常，使用throw语句可以在特定情况下自动抛出异常。（throw ExObject）

Try…catch…finally语句

无论程序是否产生异常，最后都会执行finally语句中的程序代码。

Try

{

被监管的代码

}

Catch（异常类名 异常变量名）

{

异常处理

}

finally

{

程序代码

}

抽象类

Abstract是定义抽象类的关键字，抽象类是用来继承的，抽象类被继承后需要实现其中的所有方法，一个类中只要有一个抽象方法此类就被标记为抽象类。

抽象方法

抽象方法就是在声明时，加上abstract关键字，抽象方法没有实体。

接口

声明接口时使用interface关键字。

接口具有以下特征：

1. 接口类似于抽象基类：继承接口的任何非抽象类型必须实现接口的所有成员。
2. 不能直接实例化接口
3. 接口可以包含属性、事件、索引器和方法
4. 接口不包含方法的实现
5. 类、结构和接口自身可从多个接口继承

显示接口的实现：

如果一个类继承两个接口，并且这两个接口中包含两个具有相同签名的成员，可以使用接口名称和一个句点命名该类的成员来实现。显式接口成员属于接口的成员，而不是类的成员，因此，不能使用类对象直接访问，只能通过接口对象来访问。

密闭类

使用sealed关键字声明密闭类，密闭类不能被继承。

密闭方法

并不是所有方法都可以声明为密闭方法，密闭方法只能用于对基类的虚方法进行实现,所以常和

Override关键字同时使用。

迭代器

实现继承[IEnumerable](http://luacloud.com/tag/ienumerator" \t "_blank" \o "查看 IEnumerator 中的全部文章)接口的迭代器的类，需要实现GetEnumerator()方法.

迭代器代码使用 [yield](http://luacloud.com/tag/yield) return 语句依次返回每个元素。 [yield](http://luacloud.com/tag/yield) break 将终止迭代。

分部类

定义分布类时需要使用partial关键字，分布类的每个部分都必须包含一个partial关键字，并且其声明必须与其他部分位于同一命名空间。

在设置分布类时，各个部分必须具有相同的访问性。

泛型

泛型的类型参数T可以看作时一个占位符，它不是一个类型，它仅代表了某种可能的类型。在定义泛型时T出现的位置可以在使用时用任何类型来代替。

约束：

在实例化泛型时也可以使用约束对类型参数的类型种类施加限制，约束是使用where上下文关键字指定的。

泛型方法：

泛型方法是在声明中包括了类型参数T的方法。

文件及数据流技术

System.IO命名控间

System.IO命名控间包含允许在数据流和文件上进行同步和异步读取及写入的类型。这里需要注意文件和流的差异，文件是一些具有永久储存及特定顺序的字节组成的一个有序的、具有名称的集合，相反，流提供一种向后备存储写入字节和从后备存储读取字节的方式。

File类

File类支持对文件的基本操作，它包括用于创建、复制、删除、移动、和打开文件的静态方法，并协助创建FileStream对象。

Directory类

Directory类公开了用于创建、移动、枚举、删除目录和子目录的静态方法。

FileInfo类

FileInfo类和File类之间许多方法调用都是相同的，但是FileInfo类没有静态方法，该类的方法仅可用于实例化的对象。如果在对象上进行单一方法的调用，则可以使用静态File类，如果要在文件上执行几种操作，则实例化fileinfo对象使用其方法比较好。

DirectoryInfo类

类似于File类和FileInfo类。

流

在.NET Framework中，流由Stream类来表示，该类构成了所有其他流的抽象类。不能直接创建Stream类的实例，但是必须使用它实现中的一个类。

文件流

要使用Filestream类操作文件就要先实例化一个Filestream对象，Filestream类的构造函数具有许多不同的重载形式，其中包括一个最重要的参数，即Filemode枚举。

Append：打开现有文件并查找到文件尾，或创建新文件。

Create:指定操作系统应创建新文件。如果文件存在，则它将被改写。

CreateNew:指定操作系统应创建新文件，如果文件存在，则引发异常。

Open:指定操作系统应打开现有文件，如果文件不存在，则引发异常。

OpenOrCreate:指定操作系统应打开文件，否则，应创建新文件。

Truncate:指定操作系统应打开现有文件，文件一旦打开，就将被截断为零字节大小。

文本文件的写入和读取

StreamWriter类

StreamWriter类是专门用来处理文本文件的类，可以方便地向文本文件写入字符串。同时也负责重要的转换和处理向FileStream对象写入工作。

StreamReader类

StreamReader类是专门用来读取文本文件的类，StreamReader可以从底层Stream对象创建StreamReader对象的实例，而且也能指定编码规范参数。

二进制文件的写入和读取

BinaryWriter类

BinaryWriter类以二进制形式将基元类型写入流，并支持用特定的编码写入字符串。

BinaryReader类

BinaryReader类用特定的编码将基元数据类型读作二进制值。

**网络编程技术**

**System.Net命名空间**

**System.Net命名空间为当前网络上使用的多种协议提供了简单的编程接口，而它所包含的WebRequest和WebResponse类形成了所谓的可插接式协议的基础，可插接式协议是网络服务的一种实现，它能够使用户开发出使用Internet资源的应用程序，而不考虑各种不同协议的具体细节。**

**1.Dns类**

**它是一个静态类，它从Internet域名系统（DNS）检索关于特定主机的信息。**

**2.IPAddress类**

**它包含计算机在IP网络上的地址，它主要用来提供网际协议（IP）地址。**

**3.IPEndPoint类**

**它包含应用程序连接到主机上的服务所需的主机和本地或远程接口信息。**

**4.WebClient类**

**它提供向url标识的任何本地、Intranet或Internet资源发送数据以及从这些资源接受数据的公共方法。**

**5.WebRequest类和WebResponse类**

**WebRequest类是 .NET Framework的请求/响应模型的抽象基类，用于访问Internet数据。**

**WebResponse类也是抽象基类，应用程序可以使用WebResponse类的实例以协议不可知的方式参与请求和响应事务，而从WebResponse类派生的协议类携带请求的详细信息。客户端应用程序不直接创建WebResponse对象，而是通过对WebRequest实例调用GetResponse方法来进行创建。**

**System.Net.Sockets命名空间**

**System.Net.Sockets命名空间主要提供制作Sockets网络应用程序的相关类。**

**1.Socket类**

**Socket类为网络通信提供了一套丰富的方法和属性，它主要用于管理连接，实现Berkeley通信端套接字接口。同时，它还定义了绑定、连接网络端口及传输数据所需的各种方法，提供处理端口连接传输等细节所需要的功能。**

**2．TcpClient类和TcpListener类**

**TcpClient类用于在同步阻止模式下通过网络连接、发送和接收流数据。**

**TcpListener类用于在阻止同步模式下侦听和接受传入的连接请求。**

**3.UdpClient类**

**UpdClient类用于在阻止同步模式下发送和接收无连接UDP数据。**

**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其中MailMessage类用来表示邮件内容，Attachment类用来创建邮件附件，SmtpClient类用来将电子邮件传输到指定用于邮件传送的SMTP主机。**

**POP3协议**

**POP协议用于电子邮件的接收，通过POP协议，客户机登录到服务器后，可对自己的邮件进行删除，或是下载到本地。**

注册表技术

Registry类和RegistryKey类

Registry类不能被实例化，它的作用只是实例化RegistryKey类。

RegistryKey实例表示一个注册项，这个类的方法可以浏览子建、创建新建、读取或修改建中的值。也就是说，该类可以完成对注册表项的所有操作。

OpenSubKey：检索指定子项。

GetSubKeyName:检索包含所有子项名称的字符串数组。

GetValueName：检索包含与此项关联的所有值名称的字符串数组。

CreateSubKey:创建或打开子项。

SetValue:设置或修改注册表项中的名称/值对的值。

DeleteSubKey:删除不包含任何子建的子建。

DeleteSubKeyTree：彻底删除子建目录。

DeleteValue:删除指定键值。

线程的使用

创建线程：Thread thread=new Thread(new ThreadStart(….));

线程的挂起和恢复：Suspend方法挂起；Resume恢复

线程休眠：Sleep方法

中止线程：Abort方法用来中止线程。Join方法用来阻止调用线程，直到某个线程终止为止。

线程的优先级：可以设置线程的Priority属性设置其优先级。

线程同步：使用lock方法锁定当前线程以实现同步。

通过Monitor 类实现同步。（不能实例化，直接调用） 排他锁

使用Mutex类实现实现线程同步。（可以跨进程同步） 互斥体