一、队列

队列是其元素以先进先出（Firstin,Firstout,FIFO）的方式来处理的集合。先放入队列中的元素会先读取。队列使用System.Collections.Generic命名空间中的泛型类Queue<T>实现。

队列的成员

Count：Count属性返回队列中元素个数。

Enqueue：Enqueue()方法在队列一端添加一个元素。

Dequeue：Dequeue()方法在队列的头部读取和删除元素。如果在调用Dequeue()方法时，队列中不再有元素，就抛出一个InvalidOperationException类型的异常。

Peek：Peek()方法从队列的头部读取一个元素，但不删除它。

TrimExcess：TrimExcess()方法重新设置队列的容量。Dequeue()方法从队列中删除元素，但它不会重新设置队列的容量。要从队列的头部去除空元素，应使用TrimExcess()方法。

Clear：Clear()方法从队列中移除所有的元素。

ToArray：ToArray()复制队列到一个新的数组中。

实例：将员工信息列表加入到队列中并读取队列。

创建员工信息实体类：

```
/// <summary>
/// 用户信息实体类
/// </summary>
public class UserInfo
{
    public UserInfo(int userID, string userName, string sex)
    {
```

```
        this.UserID = userID;
        this.UserName = userName;
        this.Sex = sex;
    }

    public int UserID { get; set; }
    public string UserName { get; set; }
    public string Sex { get; set; }
}
```

获取用户列表方法：

```
/// <summary>
/// 获取用户列表
/// </summary>
public static List<UserInfo> GetUserList()
{
    List<UserInfo> userList = new List<UserInfo>();
    userList.Add(new UserInfo(1, "张三", "男"));
    userList.Add(new UserInfo(2, "李四", "女"));
    userList.Add(new UserInfo(3, "王五", "男"));
    return userList;
}
```

队列的使用：

```
/// <summary>
/// 队列的使用
/// </summary>
public static void QueueSample()
{
    //创建一个队列
    Queue<UserInfo> queue = new Queue<UserInfo>();

    //获取用户列表
    List<UserInfo> userList = GetUserList();
```

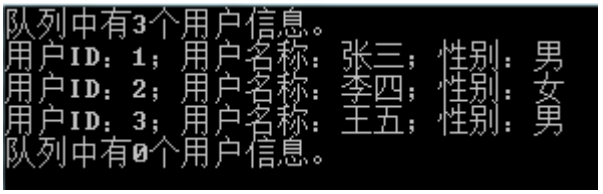
```
    //使用Enqueue()方法将用户信息加入到队列中（入列）
    foreach (var user in userList)
    {
        queue.Enqueue(user);
    }

    //使用Count属性获取队列中元素个数
    int queueCount = queue.Count;
    Console.WriteLine(String.Format("队列中有{0}个用户信息。", queueCount));
//输出：3

    //使用Dequeue()方法从队列的头部读取和删除元素（出列）
    for (int i = 0; i < queueCount; i++)
    {
        UserInfo user = queue.Dequeue();
        Console.WriteLine(String.Format("用户ID：{0}；用户名称：{1}；性别：{2}",
user.UserID, user.UserName, user.Sex));
    }

    //使用Count属性获取队列中元素个数
    queueCount = queue.Count;
    Console.WriteLine(String.Format("队列中有{0}个用户信息。", queueCount));  //
输出：0
}
```

运行结果：



二、栈

栈是与队列非常类似的另一个容器，只是要使用不同的方法访问栈。最后添加到栈中的元素会最先读取。栈是一个后进先出（Lastin,Firstout,LIFO）的容器。栈使用

System.Collections.Generic命名空间中的泛型类Stack<T>实现。

栈的成员

Count：Count属性返回栈中的元素个数。

Push：Push()方法在栈顶添加一个元素。

Pop：Pop()方法从栈顶删除一个元素，并返回该元素。如果栈是空的，就抛出一个InvalidOperationException类型的异常。

Peek：Peek()方法返回栈顶的元素，但不删除它。

Contains：Contains()方法确定某个元素是否在栈中，如果是，就返回true。

实例：将员工信息列表添加到栈中并读取栈。

```
/// <summary>
/// 栈的使用
/// </summary>
public static void StackSample()
{
    //创建一个栈
    Stack<UserInfo> stack = new Stack<UserInfo>();

    //获取用户列表
    List<UserInfo> userList = GetUserList();

    //使用Push()方法将用户信息加入到栈顶（入栈）
    foreach (var user in userList)
    {
        stack.Push(user);
    }

    //使用Count属性获取栈中元素个数
```

```
    int stackCount = stack.Count;
    Console.WriteLine(String.Format("栈中有{0}个用户信息。", stackCount));    //输出：3

    //使用Pop()方法从栈顶删除一个元素，并返回该元素（出栈）
    for (int i = 0; i < stackCount; i++)
    {
        UserInfo user = stack.Pop();
        Console.WriteLine(String.Format("用户ID：{0}；用户名称：{1}；性别：{2}", user.UserID, user.UserName, user.Sex));
    }

    //使用Count属性获取栈中元素个数
    stackCount = stack.Count;
    Console.WriteLine(String.Format("栈中有{0}个用户信息。", stackCount));    //输出：0
}
```

运行结果：