### Collections 类、泛型类和 Timing 类概述

##### 群集的定义

###### 定义：

1. 群集是一种结构化的数据类型。它存储数据，并且提供数据的添加、删除、更新操作，以及对群集的不同属性值的设置与返回操作。

###### 分类：

1. 线性:
   1. 线性群集是一张元素列表，表中的元素顺次相连
   2. 线性群集中的元素通常由位置来决定次序（例如，第一个元素、第二个元素、第三个元素，依次类推）。
   3. 在计算机世界中（当然这也是真实世界）则把数组设计成线性群集。
2. 非线性
   1. 非线性群集所包含的元素在群集内没有位置次序之分。
   2. 组织结构图就像用架子垒好的台球一样式一个非线性群集的实例
   3. 在计算机世界中树、堆、图和集都是非线性群集。样是一个非线性群集的实例。

###### 描述：

1. 无论是线性的还是非线性的群集都拥有一套定义好的属性和操作的集合。
2. 属性用来描述群集，而操作就是群集能执行的内容。
3. 群集Count 就是群集属性的一个实例。它保存着群集中数据项的数量。这里把群集的操作称为方法，它包括Add（即向群集添加新元素），Insert（即在群集指定的索引位置添加新元素）、Remove（即从群集中移除指定元素）、Clear（即从群集中移除所有元素）、Contains（即确定指定元素是否是群集的成员）、以及IndexOf（即确定指定元素在群集中的索引位置）。

##### 群集的描述

线性的群集可能是直接存取群集，也可能是顺序存取群集。而非线性的群集既可以是层次群集，也可以是组群集。

直接存取常见实例：数组、字符串、结构

顺序存取常见实例：堆栈、队列、散列表、字典

层次存取常见实例：树（二叉树）、堆

组存取常见实例：集合、图、网络

###### 直接存取群集

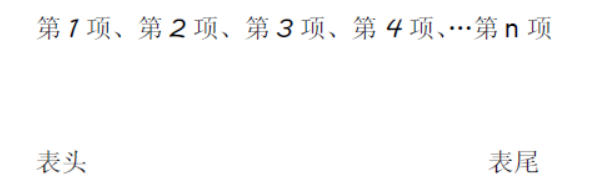
1. 直接存取群集最常见的实例就是数组。
2. 这里把数组定义为具有相同数据类型的元素的群集，而且所有数组元素如同图说明的那样可以通过整数型索引直接进行存取访问。



1. 数组可以是静态的，这样当声明数组的时候便于针对程序的长度来固定指定元素的数量。
2. 组也可以是动态的，通过ReDim 或者ReDim Preserve 语句就可以增加数组元素的数量。
3. 在C#语言中，数组不只是内置的数据类型，它还是一种类。
4. 我们可以用数组来存储一个线性的群集。
5. 字符串是直接存取群集的另外一种类型。
6. 字符串是字符的群集。和存取数组元素的方式一样，也可以基于字符的索引对其进行存取。在C#语言中，字符串也是作为类对象来实现的。
7. C#字符串是不可变的。这意味着一旦对字符串进行了初始化，就不能再改变它了。当要修改字符串的时候，不是改变原始的字符串，而是创建一个字符串的副本。在某些情况下这种行为可能会导致性能下降
8. 结构（在其他编程语言中也被称为记录）是最后一种直接存取的群集类型。结构是一种复合数据类型。它所包含的数据可能拥有许多不同的数据类型。
9. 例如，一名雇员记录就是由雇员的姓名（字符串）、薪水（整数）、工号（字符串或整数），以及其他属性组成的。由于把这些数据的值分别存储在分散的变量内是很容易变混淆的，所以编程语言采用结构来存储此类数据。
10. C#语言的结构所增加的强大能力就是为执行存储在数据上的操作定义了方法。尽管不能从结构继承或推导出一种新的类型，但是这种做法使得结构在某些地方很像一个类。

###### 顺序存取群集

1. 顺序存取群集是把群集元素按顺序存储的表。这里也把此类群集称为线性表。
2. 线性表在创建时没有大小限制，这就意味着它们可以动态地扩展和收缩
3. 不能对线性表中数据项进行直接存取访问，而要通过数据项的位置对其进行存取。线性表的第一个元素在表头的位置，而最后一个元素在表尾的位置。



1. 由于不能直接存取线性表的元素，为了访问某个元素就需要遍历线性表直到到达要找元素的位置为止。
2. 线性表的实现通常允许两种遍历表的方法：一种是单向从前往后遍历，而另一种则是双向遍历，即从前向后和从后先前遍历。
3. 线性表的某些类型限制访问数据元素。这类线性表有堆栈和队列。堆栈是一种只允许在表头（或顶端）存取数据的表。在表的顶端放置数据项，而且也只能从表的顶端移出数据项。正是基于这种原因，堆栈也被称为后进先出结构。这里把向堆栈添加数据项的操作称为入栈，而把从堆栈移出数据项的操作称为出栈。
4. 堆栈是非常常见的一种数据结构，特别是在计算机系统编程中尤为普遍。在堆栈的众多应用中，它常用于算术表达式的计算和平衡符号
5. 队列是一种只允许在表尾进行数据项添加和移出操作的表。它也被称为是先进先出结构。这里把向队列添加数据项称为EnQueue，而把从队列移出数据项称为DeQueue。
6. 通用的索引群集，这类群集的第一种就是散列表，然后是字典
7. 散列表存储了一组与关键字相关联的数据值。在散列表中有一个被称为散列函数的特殊函数。此函数会取走一个数据值，并且把此数据值（称为关键字）转换成用来取回数据的整数索引。然后此索引会用来存取访问与关键字相关联的数据记录。
8. 字典也被称为联合，它是由一系列键值对构成的。此结构与词典类似，词典中的词是关键字，而词的定义则是与关键字相关联的值。关键字就是与其相关联的值内的索引。虽然索引不需要就是整数，但是由于上述这种索引方案，所以还是常把字典称为联合数组。

###### 层次群集

1. 非线性群集分为两大主要类型：层次群集和组群集。层次群集是一组划分了层次的数据项集合。位于某一层的数据项可能会有位于下一较低层上的后继数据项。
2. 树是一种常见的层次群集。树群集看上去像是一棵倒立的树，其中一个数据项作为根，而其他数据值则作为叶子挂在根的下面。树的元素被称为节点，而且在特定节点下面的元素被称为是此节点的孩子
3. 二叉树是树群集的一种特殊类型，树中每个节点最多只有两个孩子。。二叉树可以变成二叉查找树，这样做可以极大地提高查找大量数据的效率。实现的方法是依据从根到要存储数据的节点的路径为最短路径的方式来放置节点。
4. 堆这样组织就是为了便于把最小数据值始终放置在根节点上。在删除时会移除根节点。此外，堆的插入和删除操作总是会导致堆的重组，因为只有这样才能把最小值放在根节点上。我们经常会用堆来排序，这被称为是堆排序。通过反复删除根节点以及重组堆的方式就可以对存储在堆内的数据元素进行排序。

###### 组群集

1. 数据项为无序的非线性群集被称为组。集合、图和网络是组群集的三种主要类型。
2. 集合是一种无序数据值的群集，并且集合中每一个数据值都是唯一的。当然，就像整数一样，班级中学生的列表就是一个集合的实例。在集合上执行的操作包括联合和交叉
3. 图是由节点集合以及与节点相连的边集合组成的。图用来对必须访问图中每个节点的情况进行建模，而且有些时候还要按照特定顺序进行访问。这样做的目的是为了找到“遍历”图的最有效的方法。
4. 网络是图的一种特殊类型。网络的每一条边都被赋予了权。权同使用某边从一个节点移动到另一个节点所花费的代价相关

##### CollectionBase 类

###### 用 ArrayLists 实现 Collection 类

###### 定义 Collection 类

###### 实现 Collection 类

##### 泛型编程

##### 时间测试

###### 一个简单化的时间测试

###### 用于.NET 环境的时间测试

###### Timing Test 类

### 数组和 ArrayLists

##### 数组基本概念

###### 数组的声明和初始化

###### 数组元素的设置和存取访问

###### 取回数组元数据的方法和属性

###### 多维数组

###### 参数数组

###### 锯齿状数组

##### ArrayList 类

###### ArrayList 类的成员

###### 应用 ArrayList 类

### 基础排序算法

##### 排序算法

###### 数组类测试环境

###### 冒泡排序

###### 检验排序过程

###### 选择排序

###### 插入排序

##### 基础排序算法的时间比较

### 基础查找算法

##### 顺序查找算法

###### 查找最小值和最大值

###### 自组织数据加快顺序查找速度

##### 二叉查找算法

##### 递归二叉查找算法

### 堆栈和队列

##### 堆栈、堆栈的实现以及 STACK 类

###### 堆栈的操作

###### Stack 类的实现

##### STACK 类

###### Stack 构造器方法

###### 主要的堆栈操作

###### Peek 方法

###### Clear 方法

###### Contains 方法

###### CopyTo 方法和 ToArray 方法

###### Stack 类的实例：十进制向多种进制的转换

##### 队列、QUEUE 类以及 QUEUE 类的实现

###### 队列的操作

###### Queue 的实现

###### Queue 类：实例应用

###### 用队列存储数据

###### 源自 Queue 类的优先队列

### BitArray 类

##### 激发的问题

##### 位和位操作

###### 二进制数制系统

###### 处理二进制数：按位运算符和位移运算符

##### 按位运算符的应用

###### 位移运算符

##### 整数转换成二进制形式的应用程序

##### 位移的示例应用程序

##### BITARRAY 类

###### 使用 BitArray 类

###### 更多 BitArray 类的方法和属性

##### 用 BITARRAY 来编写埃拉托斯特尼筛法

##### BITARRAY 与数组在埃拉托斯特尼筛法上的比较

### 字符串、String 类和 StringBuilder 类

##### STRING 类的应用

###### 创建 String 对象

###### 常用 String 类的方法们

###### Split 方法和 Join 方法

###### 比较字符串的方法

###### 处理字符串的方法

##### STRINGBUILDER 类

###### 构造 StringBuilder 对象

###### 获取并且设置关于 StringBuilder 对象的信息

###### 修改 StringBuilder 对象

##### STRING 类与 STRINGBUILDER 的性能比较

### 模式匹配和文本处理

##### 正则表达式概述

###### 概述：使用正则表达式

##### 数量词

##### 使用字符类

##### 用断言修改正则表达式

##### 使用分组构造

###### 匿名组

###### 命名组

###### 零宽度正向预搜索断言和零宽度反向预搜索断言

##### CAPTURESCOLLECTION 类

##### 正则表达式的选项

### 构建字典：DictionaryBase 类和 SortedList 类

##### DICTIONARYBASE 类

###### DictionaryBase 类的基础方法和属性

###### 其他的 DictionaryBase 方法

##### 通用的 KEYVALUEPAIR 类

##### SORTEDLIST 类

###### 使用 SortedList 类

### 散列和 Hashtable 类

##### 散列概述

##### 选择散列函数

##### 查找散列表中数据

##### 解决冲突

###### 桶式散列法

###### 开放定址法

###### 双重散列法

##### HASHTABLE 类

###### 实例化 Hashtable 对象并且给其添加数据

###### 从散列表中分别取回关键字和数值

###### 取回基于关键字的数值

###### Hashtable 类的实用方法

##### HASHTABLE 的应用程序：计算机术语表

### 链表

##### 数组存在的问题

##### 链表的定义

##### 面向对象链表的设计

###### Node 类

###### LinkedList 类

##### 链表设计的改进方案

###### 双向链表

###### 循环链表

##### 使用 ITERATOR 类

###### 新的 LinkedList 类

###### 实例化 Iterator 类

##### 通用的 LINKED LIST 类和通用的 NODE 类

###### 通用链表实例

### 二叉树和二叉查找树

##### 树的定义

##### 二叉树

###### 构造二叉查找树

###### 遍历二叉查找树

###### 在二叉查找树中查找节点和最大/最小值

###### 从 BST 中移除叶子节点

###### 删除带有一个子节点的节点

###### 12.2.6 删除带有两个子节点的节

### 集合

##### 集合的基础定义、操作及属性

###### 集合的定义

###### 集合的操作

###### 集合的属性

##### 第一个用散列表的 SET 类的实现

###### 类数据成员和构造器方法

###### Add 方法

###### Remove 方法和 Size 方法

###### Union 方法

###### Intersection 方法

###### Subset 方法

###### Difference 方法

###### 测试 CSet 实现的程序

##### CSET 类的 BITARRAY 实现

###### 使用 BitArray 实现的概述

###### BitArray 集合的实现

### 高级排序算法

##### 希尔排序算法

##### 归并排序算法

##### 堆排序算法

###### 构造堆

##### 快速排序算法

###### 快速排序算法的描述

###### 快速排序算法的代码

###### 快速排序算法的改进

### 查找的高级数据结构和算法

##### AVL 树

###### AVL 树的基本原理

###### AVL 树的实现

##### 红黑树

###### 红黑树规则

###### 红黑树的插入

###### 红黑树实现代码

##### 跳跃表

###### 跳跃表的基本原理

###### 跳跃表的实现

### 图和图的算法

##### 图的定义

##### 由图模拟真实世界系统

##### 图类

###### 顶点的表示

###### 边的表示

###### 图的构造

###### 图的第一个应用：拓扑排序

###### 拓扑排序算法

###### 拓扑排序算法的实现

##### 图的搜索

###### 深度优先搜索

###### 广度优先搜索

##### 最小生成树

###### 最小生成树算法

##### 查找最短路径

###### 加权图

###### 确定最短路径的 Dijkstra 算法

###### Dijkstra 算法的代码

### 高级算法

##### 动态规划

###### 动态规划实例：计算斐波纳契数列

###### 寻找长公共子串

##### 背包问题

##### 贪心算法

###### 贪心算法实例：找零钱问题

###### 采用哈夫曼编码的数据压缩

###### 用贪心算法解决背包问题