Một số kỹ thuật khác

v 2.0 - 04/2013

Nội dung

1. Chỉ mục
2. Nạp chồng toán tử
3. Chuyển đổi kiểu

Chỉ mục

Chỉ mục

- Sử dụng thuộc tính chỉ mục

```
public class PeopleCollection
{
   private ArrayList arrPeople = new ArrayList();
   ...
   public Person this[int index]
   {
      get { return (Person)arrPeople[index]; }
      set { arrPeople.Insert(index, value); }
   }
}
```

```
static void Main()
{
   PeopleCollection myPeople = new PeopleCollection();
   //thêm các đối tượng với cú pháp chỉ mục
   myPeople[0] = new Person("An", 40);
   myPeople[1] = new Person("Binh", 35);
   //lấy đối tượng sử dụng chỉ mục
   for (int i = 0; i < myPeople.Count; i++)
      Console.WriteLine("Name: {0}", myPeople[i]);
}
```

Chỉ mục với giá trị chuỗi

```
public class PeopleCollection
{
   private Dictionary<string, Person> listPeople = new
                                    Dictionary<string, Person>();
   ...
   public Person this[string name]
   {
      get { return (Person)listPeople[name]; }
      set { listPeople[name] = value; }
   }
}
```

Nạp chồng chỉ mục

```
public class PeopleCollection
{
   private Dictionary<string, Person> listPeople = new
                                    Dictionary<string, Person>();
   ...
   public Person this[string name]
   {
      get { return (Person)listPeople[name]; }
      set { listPeople[name] = value; }
   }

   public Person this[int index]
   {
      get { return (listPeople.Values.ToList<Person>())[index]; }
   }
}
```

Những vấn đề khác

- Chỉ mục cho đa chiều

```
public class SomeContainer
{
   private int[,] my2DArray = new int[10, 10];
   ...
   public int this[int row, int column]
   {
      ...
   }
}
```

- Định nghĩa chỉ mục trong giao diện

```
public interface IStringContainer
{
   string this[int index] { get; set; }
}
```

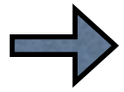
Nạp chồng toán tử

Nạp chồng toán tử

- Là khả năng thực thi các toán tử với các lớp tự định nghĩa giống với các kiểu dữ liệu cơ sở

```
Point p1 = new Point(10, 20);
Point p2 = new Point(10, 40);
Point p3;

p3 = p2.add(p1);
```



```
Point p1 = new Point(10, 20);
Point p2 = new Point(10, 40);
Point p3;

p3 = p2 + p1;
p3 = p2 + 10;
```

- Là một dạng của nạp chồng hàm
  - Là các *hàm tĩnh* có dạng `operator@` (trong đó, `@` là một toán tử có thể nạp chồng)
- Lưu ý khi nạp chồng toán tử
  - Không thay đổi ý nghĩa của bất kỳ toán tử nào
  - Không thay đổi vị trí hay tính kết hợp của toán tử
  - Không thay đổi số lượng đối số
  - Không tạo ra toán tử mới (ví dụ, **)

Các toán tử có thể nạp chồng

`+ - ! ~ ++ -- true false`	Toán tử một ngôi
`+ - * / % & | ^ << >>`	Toán tử hai ngôi
`== != < > <= >=`	Toán tử so sánh
`[ ]`	Chỉ mục Không cần nạp chồng vì đã có kỹ thuật chỉ mục
`( )`	Chuyển đổi kiểu Phần sau sẽ nói
`+= -= *= /= %= &= |= ^= <<= >>=`	Phép gán kết hợp Không cần nạp chồng

Lớp Point

```
public class Point
{
    public int X { get; set; }
    public int Y { get; set; }

    public Point (int xPos, int yPos)
    {
        X = xPos;
        Y = yPos;
    }

    public override string ToString()
    {
        return string.Format(“[{0}, {1}]”, this.X, this.Y);
    }
}
```

Nạp chồng toán tử hai ngôi

```
public class Point
{
    ...
    public static Point operator +(Point p1, Point p2)
    {
        return new Point(p1.X + p2.X, P1.Y + p2.Y);
    }

    public static Point operator - (Point p1, Point p2)
    {
        return new Point(p1.X - p2.X, P1.Y - p2.Y);
    }

    public static Point operator + (Point p1, int change)
    {
        return new Point(p1.X + change, P1.Y + change);
    }

    public static Point operator + (int change, Point p1)
    {
        return new Point(p1.X + change, P1.Y + change);
    }
}
```

Nạp chồng toán tử một ngôi

```
public class Point
{
    ...
    public static Point operator ++ (Point p1)
    {
        return new Point(p1.X + 1, P1.Y + 1);
    }

    public static Point operator -- (Point p1)
    {
        return new Point(p1.X - 1, P1.Y - 1);
    }
}
```

Với toán tử tăng/giảm, chỉ cần định nghĩa một hàm cho cả tiền tố và hậu tố

Nạp chồng toán tử bằng

```
public class Point
{
    ...
    //nạp chồng hai hàm Equals và GetHashCode
    public override bool Equals(object o)
    {
        return o.ToString() == this.ToString();
    }
    public override int GetHashCode()
    {
        return this.ToString().GetHashCode();
    }
    //nạp chồng toán tử bằng sử dụng hàm Equals
    public static bool operator == (Point p1, Point p2)
    {
        return p1.Equals(p2);
    }

    public static bool operator != (Point p1, Point p2)
    {
        return !p1.Equals(p2);
    }
}
```

Khi nạp chồng toán tử bằng, phải nạp chồng cả toán tử khác

Nạp chồng toán tử so sánh

```
public class Point : IComparable
{
   ...
   //cài đặt giao diện IComparable
   public int CompareTo(object o)
   {
      if (o is Point) {
         Point p = (Point)o;
         if (this.X > p.X && this.Y > p.Y) return 1;
         if (this.X < p.X && this.Y < p.Y) return -1;
         else return 0;
      } else throw new ArgumentException();
   }
   //nạp chồng toán tử so sánh sử dụng hàm CompareTo
   public static bool operator < (Point p1, Point p2)
   {
      return (p1.CompareTo(p2) < 0);
   }

   public static bool operator > (Point p1, Point p2)
   {
      return (p1.CompareTo(p2) > 0);
   }
}
```

Khi nạp chồng toán tử lớn hơn, phải nạp chồng cả toán tử nhỏ hơn
Tương tự cho lớn hơn hoặc bằng và nhỏ hơn hoặc bằng

Chuyển đổi kiểu

Nhắc lại - Chuyển đổi kiểu

- Chuyển đổi kiểu số học

```
int a = 123;
long b = a; // ngầm định
int c = (int)b; // tường minh
```

- Chuyển đổi kiểu giữa kiểu cơ sở và kiểu phái sinh

```
class Base {}
class Derived : Base {}
class Program {
   static void Main() {
      Base myBase;
      myBase = new Derived(); // ngầm định

      Derived myDerived = (Derived)myBase; // tường minh
   }
}
```

- Cấu tử chuyển đổi kiểu

```
class Rectangle {...}
class Square {
   ...
   public Square (Rectangle r) {
      this.Size = r.Width;
   }
}
```

Lớp Rectangle

```
public class Rectangle
{
    public int Width { get; set; }
    public int Height { get; set; }

    public Rectangle(int w, int h)
    {
        Width = w;
        Height = h;
    }
    public Rectangle() {}

    public void Draw ()
    {
        Console.WriteLine(“Rectangle”);
    }

    public override string ToString ()
    {
        return string.Format(“[Width={0}; Height={1}]”,
                             Width, Height);
    }
}
```

Lớp Square

```
public class Square
{
    public int Size { get; set; }

    public Square(int s)
    {
        Size = s;
    }
    public Square () {}

    public void Draw ()
    {
        Console.WriteLine(“Square”);
    }

    public override string ToString ()
    {
        return string.Format(“[Size={0}]”, Size);
    }
}
```

Từ khoá explicit

```
public class Square
{
    ...
    public static explicit operator Square (Rectangle r)
    {
        Square s = new Square();
        s.Size = r.Height;
        return s;
    }
}
```

```
static void DrawSquare(Square sq)
{
    Console.WriteLine(sq.ToString());
    sq.Draw();
}

static void Main()
{
    Rectangle rect = new Rectangle (10,20);
    DrawSquare((Square)rect);
}
```

Từ khoá explicit

```
public class Square
{
   ...
   public static explicit operator Square (int size)
   {
      Square s = new Square();
      s.Size = size;
      return s;
   }
   public static explicit operator int (Square s)
   {
      return s.Size;
   }
}
```

```
static void Main()
{
   Square sq2 = (Square)90;
   int size = (int)sq2;
}
```

Từ khoá implicit

```
public class Rectangle
{
    ...
    public static implicit operator Rectangle (Square s)
    {
        Rectangle r = new Rectangle();
        r.Width = s.Size;
        r.Height = s.Size;
        return r;
    }
}
```

```
static void Main()
{
    Square sq3 = new Square();
    sq3.Size = 83;
    Rectangle rect = sq3;
}
```

Cảm ơn sự chú ý

Câu hỏi ?