## 背包系统

MemberwiseClone():MemberwiseClone 方法创建一个浅表副本，具体来说就是创建一个新对象，然后将当前对象的非静态字段复制到该新对象。如果字段是值类型的，则对该字段执行逐位复制。如果字段是引用类型，则复制引用但不复制引用的对象；因此，原始对象及其复本引用同一对象。

为了实现深度复制，我们就必须遍历有相互引用的对象构成的图，并需要处理其中的循环引用结构。这无疑是十分复杂的。幸好借助.Net的序列化和反序列化机制，可以十分简单的深度Clone一个对象。原理很简单，首先将对象序列化到内存流中，此时对象和对象引用的所用对象的状态都被保存到内存中。.Net的序列化机制会自动处理循环引用的情况。然后将内存流中的状态信息反序列化到一个新的对象中。这样一个对象的深度复制就完成了。在原型设计模式中CLONE技术非常关键。

下面的代码就是演示这个问题：

using System;using System.IO;using System.Runtime.Serialization.Formatters.Binary;

namespace CloneDemo

{

[Serializable]

class DemoClass

{

public int i = 0;

public int[] iArr = { 1, 2, 3 };

public DemoClass Clone1() //浅CLONE

{

return this.MemberwiseClone() as DemoClass;

}

public DemoClass Clone2() //深clone

{

MemoryStream stream = new MemoryStream();

BinaryFormatter formatter = new BinaryFormatter();

formatter.Serialize(stream, this);

stream.Position = 0;

return formatter.Deserialize(stream) as DemoClass;

}

}

class Program

{

static void Main(string[] args)

{

DemoClass a = new DemoClass();

a.i = 10;

a.iArr = new int[] { 8, 9, 10 };

DemoClass b = a.Clone1();

DemoClass c = a.Clone2();

// 更改 a 对象的iArr[0], 导致 b 对象的iArr[0] 也发生了变化 而 c不会变化

a.iArr[0] = 88;

Console.WriteLine("MemberwiseClone");

Console.WriteLine(b.i);

foreach (var item in b.iArr)

{

Console.WriteLine(item);

}

Console.WriteLine("Clone2");

Console.WriteLine(c.i);

foreach (var item in c.iArr)

{

Console.WriteLine(item);

}

Console.ReadLine();

}

}

}