
第九章
面向对象

目录

一、面向对象简介

二、继承

三、类的共享成员

四、拆箱装箱

目录

五、泛型

六、集合

七、委托

八、事件

一、面向对象简介

面向对象编程的三大特征

对象有属性(记录对象的数据)和行为
对象隐藏了它的实现细节，对外提供接口

封装

继承他的父类，节省代码

继承

父类引用指向子类对象

多态

一、面向对象简介

面向对象的优点：

易维护，易扩展，易开发

掌握面向对象编程的方法：

一切皆对象

二、继承

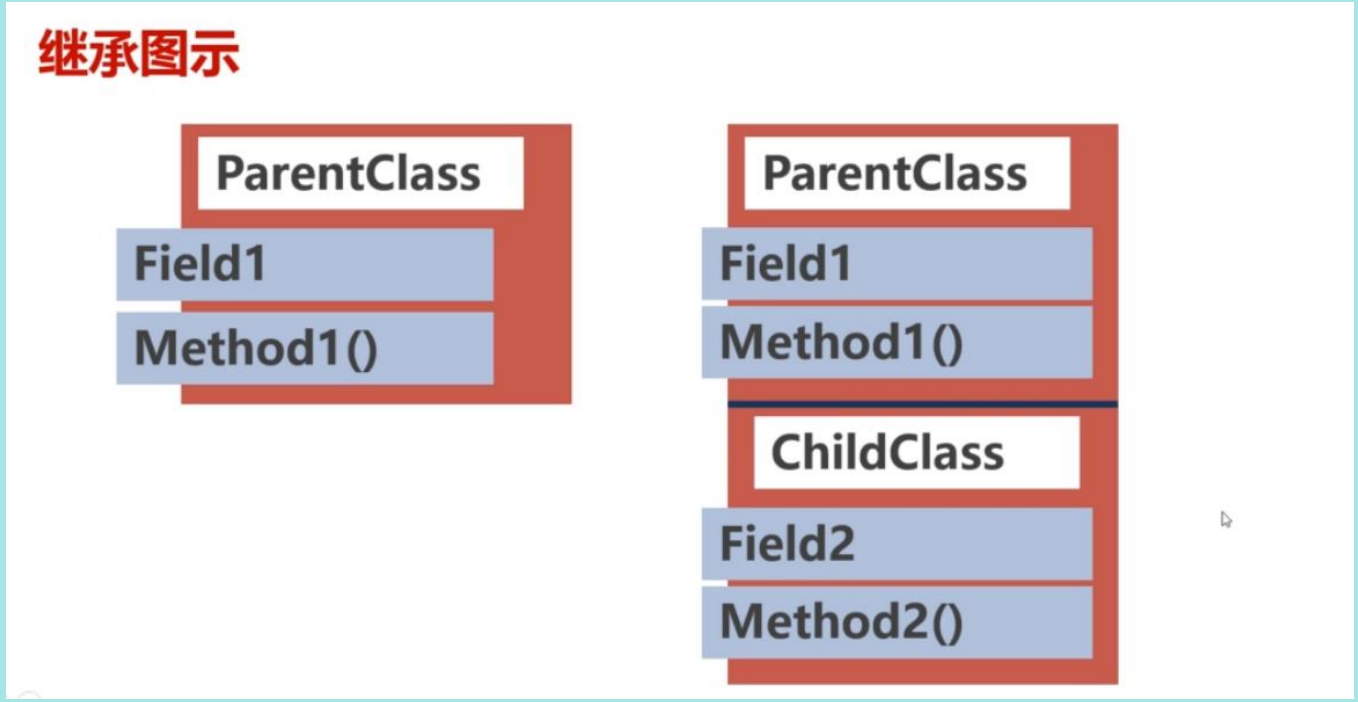
为什么需要继承：

我们不想写那么多重复的代码，就像我们不愿意白手起家，这有点像子孙继承父辈的财产。假如，我们有个宠物商店，我们有各种各样的宠物，他们有共性也有各自的特点。

二、继承

实现继承：

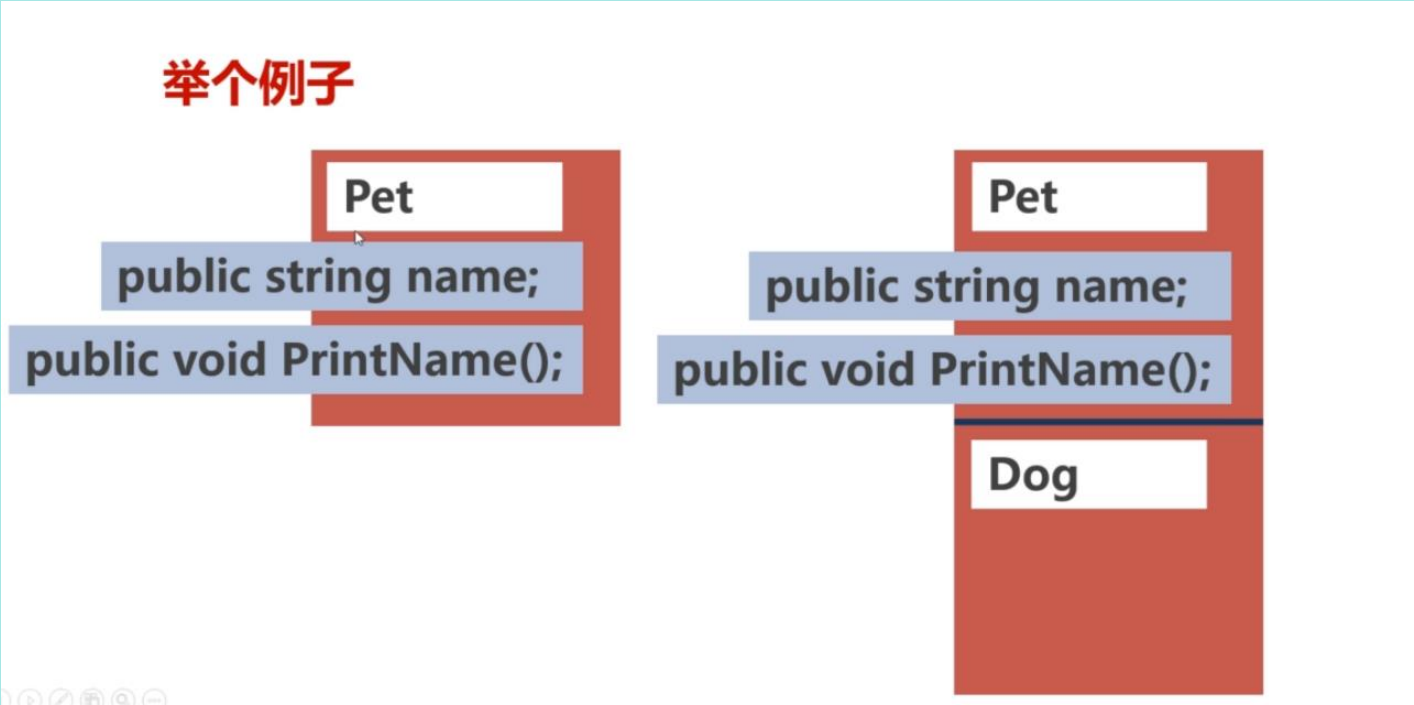
当一个类派生于另一个基类型，它拥有该基础类型的所有成员字段和函数。



二、继承

语法：

```
Class ChildClass : Inherits ParentClass
    ....
End Class
```



二、继承

```
Public Class Pet
    Public Name As String
    Public Sub PrintName()
        Console.WriteLine("宠物的名字是: " & name)
    End Sub
End Class
Public Class Dog : Inherits Pet

End Class

Sub Main()
    Dim dog As Dog = New Dog()
    dog.Name = "小明"
    dog.PrintName()
End Sub
```

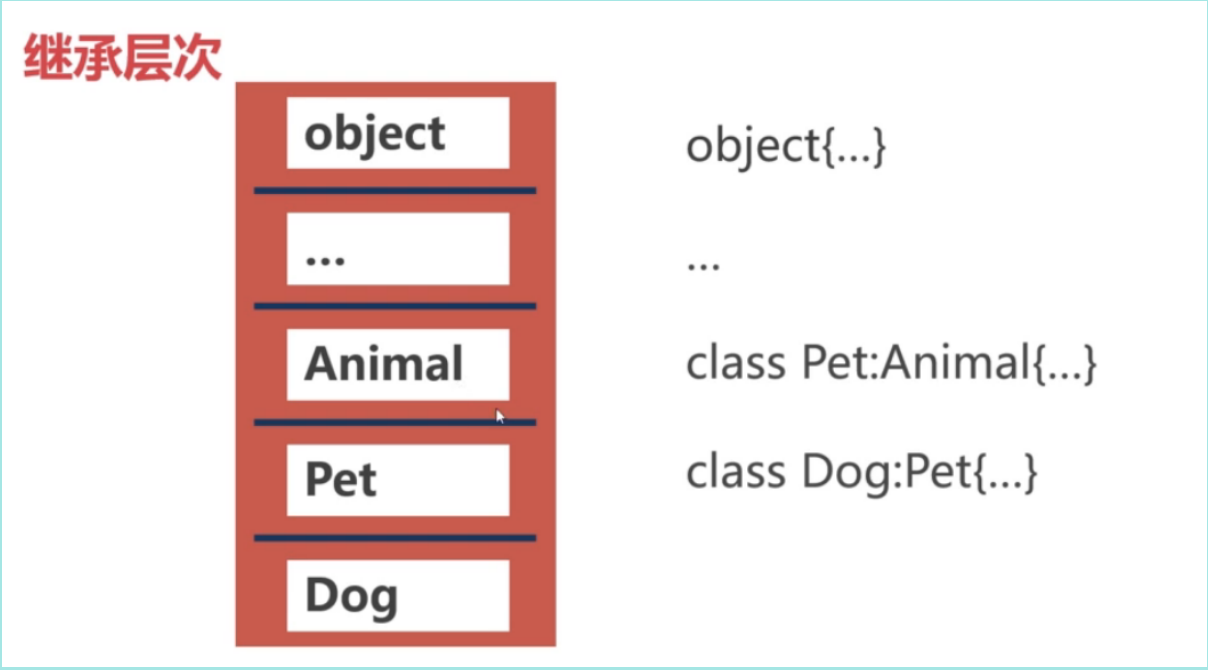
二、继承

特殊的基类：

Object是所有类的共同基类，它是唯一的非派生类，是继承层次结构的基础，对于其他类，父类和子类的概念都是相对的

规则：

继承只有单继承，也就是只能继承一个父类，当然该父类还可以继承自一个祖父类，直到Object类。



2.1隐藏方法

我们不能删除基类中的任何成员，但是可以用与基类成员名称相同的成员来屏蔽基类成员

语法细节

屏蔽数据成员：在派生类中声明名称和类型相同的成员。

屏蔽成员函数：在派生类中声明新的带有相同函数签名（函数名，参数类型和个数相同但不包括返回类型）的成员。

让编译器知道：可以添加Shadows关键字，否则会有警告。

```
Public Class Dog : Inherits Pet
    Public Shadows Sub PrintName()
        Console.WriteLine("pet name is: " & Name)
    End Sub

End Class
```

2.2多态

一个设计原则：

面向对象编程中，都遵循一个原则：依赖倒置原则。换句话就是程序设计要依赖于抽象（Pet）类，而不依赖于具体类（Dog）

基类的引用：

派生类的对象包括基类部分和派生类部分，所以，我们可以通过一个基类类型的引用指向派生类。通过指向派生类的基类引用，我们仅仅能访问派生类中的基类部分

2.2多态

统一提高效率：

有时我们需要一个容器（比如数组）保存所有的基类（Pet），基类描述了共同的属性和行为，比如宠物都有年龄，名字，都可以发出声音，活动，需要喂食

子类具有差异：

但是基类又不能涵盖所有的情况和变化，统一的行为方法往往在基类和派生类中有所区别。虽然所有的宠物都能发出声音，但是发出的声音各不相同，有的甚至不发出声音。

2.2多态

多态：

通过指向派生类的基类引用，调用函数，会根据引用所指向派生类的实际类型，调用派生类中的同名重写函数，便是多态

2.2多态

```
Public Class Pet
    Public Name As String
    Public Sub PrintName()
        Console.WriteLine("宠物的名字是：" & name)
    End Sub
    Public Overridable Sub Speak()
        Console.WriteLine("宠物在叫")
    End Sub
End Class
Public Class Dog : Inherits Pet
    Public Shadows Sub PrintName()
        Console.WriteLine("pet name is：" & Name)
    End Sub
    Public Overrides Sub Speak()
        Console.WriteLine("wangwang")
    End Sub
End Class
Public Class Cat : Inherits Pet
    Public Overrides Sub Speak()
        Console.WriteLine("miaomiao")
    End Sub
End Class
```

2.3构造函数

派生类对象中，有一部分是基类部分，在执行派生类的构造函数体之前，将会隐式或显式的调用基类构造函数

调用顺序


```
class MyDerived:MyBase{
1  int field1 =5;
3  MyDerived(){}  ...
}
class MyBase{
2  MyBase{}       ...
}
```

2.3构造函数

派生类对象中，有一部分是基类部分，在执行派生类的构造函数体之前，将会隐式或显式的调用基类构造函数

```
Public Class Pet
    Protected _name As String
    Public Sub New(name)
        _name = name
    End Sub
    Public Sub PrintName()
        Console.WriteLine("宠物的名字是：" & _name)
    End Sub
    Public Overridable Sub Speak()
        Console.WriteLine("宠物在叫")
    End Sub
End Class
Public Class Dog : Inherits Pet
    Public Sub New(name)
        MyBase.New(name)
    End Sub
    Public Shadows Sub PrintName()
        Console.WriteLine("pet name is：" & _name)
    End Sub
    Public Overrides Sub Speak()
        Console.WriteLine("wangwang")
    End Sub
End Class
Public Class Cat : Inherits Pet
    Public Sub New(name)
        MyBase.New(name)
    End Sub
    Public Overrides Sub Speak()
        Console.WriteLine("miaomiao")
    End Sub
End Class
```

2.4抽象方法抽象类

抽象类： MustInherit 修饰
抽象方法：MustOverride修饰
抽象成员：
　　指明只能由子类重写的方法（Sub，Function,Property）
　　必须用MustOverride修饰符标记
　　不能有实现代码块
抽象类：
　　抽象类的存在只有一个目的，就是被继承
　　抽象类不能实例化，用MustInherit修饰
　　抽象类可以包含抽象成员和普通成员，以及他们的任意组合
　　抽象类抽象成员在派生类中需要用overides关键字实现

2.4抽象方法抽象类

```
Public MustInherit Class Pet
    Protected _name As String
    Public Sub New(name)
        _name = name
    End Sub
    Public Sub PrintName()
        Console.WriteLine("宠物的名字是：" & _name)
    End Sub
    Public MustOverride Sub Speak()
End Class
Public Class Dog : Inherits Pet
    Public Sub New(name)
        MyBase.New(name)
    End Sub
    Public Shadows Sub PrintName()
        Console.WriteLine("pet name is：" & _name)
    End Sub
    Public Overrides Sub Speak()
        Console.WriteLine("wangwang")
    End Sub
End Class
Public Class Cat : Inherits Pet
    Public Sub New(name)
        MyBase.New(name)
    End Sub
    Public Overrides Sub Speak()
        Console.WriteLine("miaomiao")
    End Sub
End Class
```

2.5不可继承的类和方法

为什么需要不可继承：

有些类不希望其他人通过继承来修改（如String类）

有些方法不希望其他人重写该方法

如果一个基类方法不希望子类对其进行重写，就可以不声明Overridable。

如果是某个派生类方法不希望子类对其重写。同时是Overrides重写，就可以使用NotOverridable来禁止子类再次重写

2.6接口

接口就是指定一组函数成员，而不实现他们的引用类型

```
Public Interface IChatchMice
    Sub ChatchMice()
End Interface
```

方法默认就是Public但是不能加任何修饰符,接口只能用来被实现

```
Public Class Cat
    Inherits Pet
    Implements IChatchMice
    Public Sub New(name)
        MyBase.New(name)
    End Sub
    Public Overrides Sub Speak()
        Console.WriteLine("miaomiao")
    End Sub
    Sub ChatchMice() Implements IChatchMice.ChatchMice
        Console.WriteLine("抓老鼠")
    End Sub
End Class
```

接口必须有实现

2.6接口

接口也是一种引用类型，一个类可以实现多个接口

```
Dim cat As Pet = New Cat("小咪")
cat.Speak()
Dim ic As IChatchMice = cat
ic.ChatchMice()
```

```
Public Class Cat
    Inherits Pet
    Implements IChatchMice, IClimbTree
    Public Sub New(name)
        MyBase.New(name)
    End Sub
    Public Overrides Sub Speak()
        Console.WriteLine("miaomiao")
    End Sub
    Sub ChatchMice() Implements IChatchMice.ChatchMice
        Console.WriteLine("ChatchMice")
    End Sub
    Sub ClimbTree() Implements IClimbTree.ClimbTree
        Console.WriteLine("ChatchMice")
    End Sub
End Class
Public Interface IChatchMice
    Sub ChatchMice()
End Interface
Public Interface IClimbTree
    Sub ClimbTree()
End Interface
```

2.7结构和类

不同点

结构是值类型（在栈中），类是引用类型（在堆中）

结构不支持继承，类支持继承

结构不能定义默认构造函数，编译器会定义

适用场合

结构：由于分配内存快，作用域结束即被删除，不需要垃圾回收，用于小型数据结构。但传递过程中会复制，应该使用ref提高效率

类：用于其他的需要继承体系的场合

```
Public Structure Fish
    Dim weight As Integer
    Dim size As Integer
    Dim type As Integer
End Structure
```

三、类的共享成员

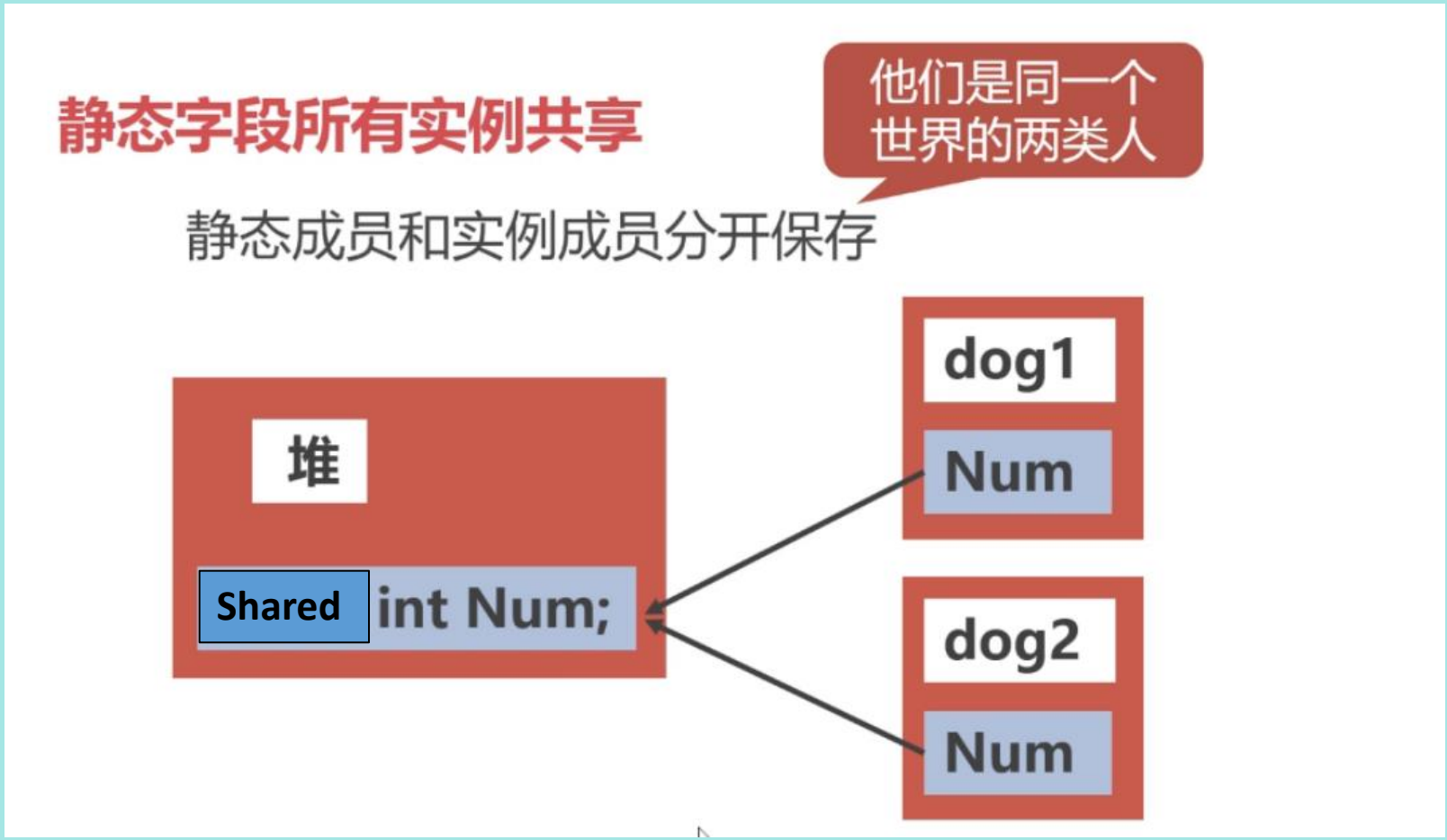
我们可以使用Shared关键字将类成员定义为静态
当我们将一个类的成员声明为Shared时，意味着无论创建多少个对象，该成员只有一个副本

```
Public Class Dog
        Public Shared Num As Integer
    End Class
```

静态字段意味着什么

静态成员将被类的所有实例共享，所有实例都访问同一个内存位置

三、类的共享成员


静态字段所有实例共享
他们是同一个
世界的两类人
静态成员和实例成员分开保存
堆
Shared
int Num;
dog1
Num
dog2
Num

三、类的共享成员

静态成员如何被访问

静态成员将直接通过类名访问

Dog.Num +=1

静态成员的生成周期

独立于任何实例，没有实例也可以访问。其初始化语句在任何静态成员使用之前调用

静态函数成员

静态函数也独立于任何实例，没有实例也可以调用

静态函数不能访问实例成员，仅能访问其他静态成员

```
Public Class Dog
    Public Shared Num As Integer
    Public Shared Sub PrintNum()
        Console.WriteLine("Num=" & Num)
    End Sub
End Class
```

四、拆箱装箱

什么是装箱

装箱：根据值类型的值，在堆上创建一个完整的引用类型对象。并返回对象的引用。是一种隐式转换

为什么要装箱

有时候需要将值类型转化为引用类型来进行统一的操作和统一的存储。

装箱实例

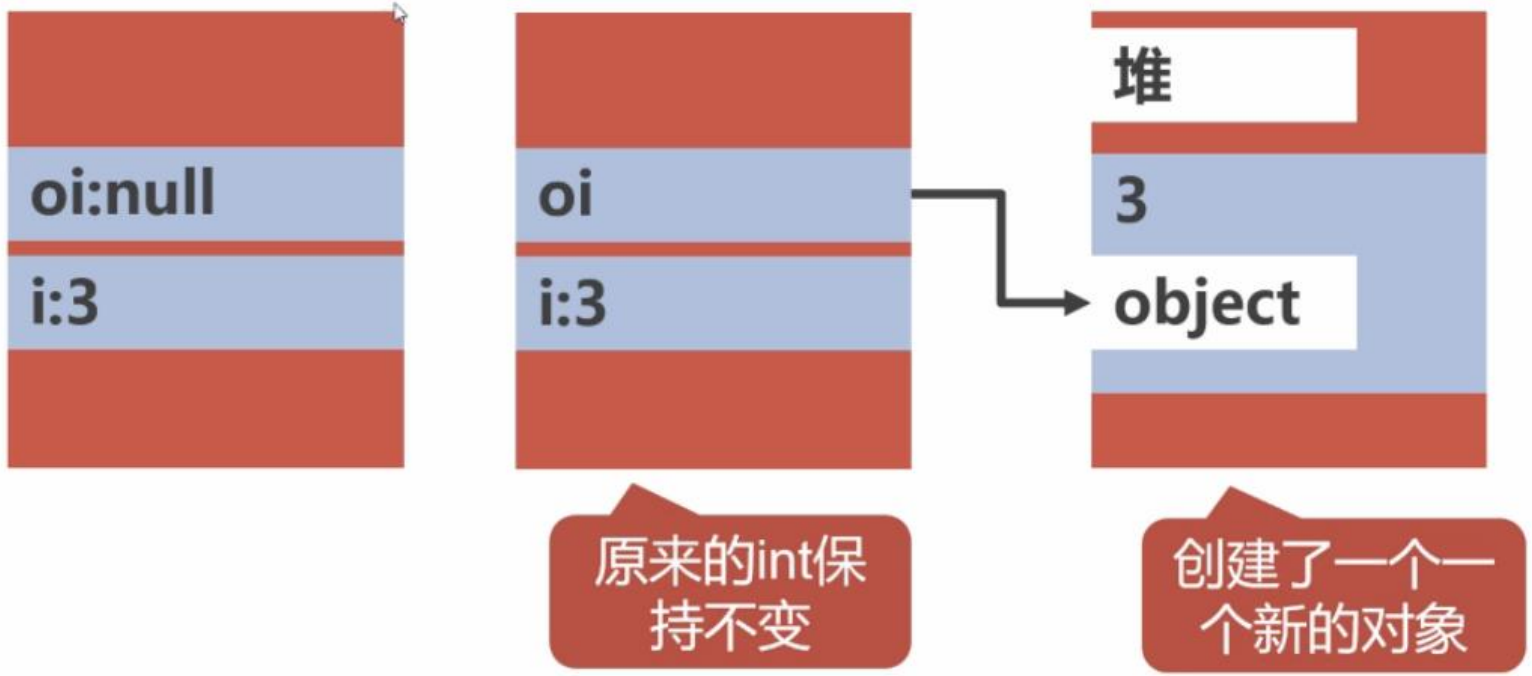
```
Dim i As Integer = 3
Dim oi As Object
oi = I
```

装箱的本质

装箱的本质就是在堆上创建了引用类型的副本，新创建的引用类型和原来的值类型相互独立

四、拆箱装箱

装箱图示



四、拆箱装箱

什么是拆箱

将装箱后的对象转换回值类型的过程，是一种显示转换

```
Dim i As Integer = 3
 Dim oi As Object
 oi = i
 Dim j As Integer = CInt(oi)
```

五、泛型

泛型就是一个模子，装入类型的材料，可以塑造出想要的产品

```
Public Class Cage(Of T)
    Dim petsArray() As T
    Public Sub PutIn(a As T)

    End Sub
End Class
```

为什么用泛型

用基类或者公共的接口，甚至所有类的基类Object，也可以实现一个Cage类，但是类型太宽泛，需要显式转换类型，并且判断真是类型是什么

泛型类的优势

代码量更小，无论多少种笼子，我们只需要一个实现

只有需要的类型才会被实例化

易于维护，修改模板，所有的实例都在改变

五、泛型

泛型方法

泛型方法就是方法的模型，给定具体的类型，就可以实例化出一个操作该类型的具体方法。

```
Public Sub PutIn(Of T)(a As T)

End Sub
```

5.1约束

约束是控制泛型这匹烈马的缰绳，缩小泛型参数的范围
只有添加了约束，才能调用泛型参数中（比如T）的方法

```
Public Sub PutIn(Of T As Dog)(a As T)

End Sub
```

5.2泛型接口

泛型接口允许我们将接口的成员的参数和返回类型设置为泛型参数的接口

```
Public Interface ICat(Of T)
    Function Act(t As T) As T
End Interface
```

接口实现语法

```
Public Class Cat
    Implements ICat(Of MaiMeng)
    Function Act(t As MaiMeng) As MaiMeng Implements ICat(Of MaiMeng).Act
        Return New MaiMeng()
    End Function
End Class
```

5.2泛型接口

```
Public MustInherit Class DogCmd
    Public MustOverride Function GetCmd() As String
End Class
Public Class Dog

End Class
Public Class SitDogCmd
    Inherits DogCmd
    Public Overrides Function GetCmd() As String
        Return "sit"
    End Function
End Class
Public Interface IDogLearn(Of T As DogCmd)
    Sub Act(cmd As T)
End Interface
Public Class Jinmao
    Implements IDogLearn(Of SitDogCmd)

    Public Sub Act(cmd As SitDogCmd) Implements IDogLearn(Of SitDogCmd).Act
        Console.WriteLine(cmd.GetCmd())
    End Sub
End Class
Sub Main()
    Dim cmd As SitDogCmd = New SitDogCmd()
    Dim yuanbao As Jinmao = New Jinmao()
    yuanbao.Act(cmd)
End Sub
```

六、集合

集合是一种存放多个数据的容器类型，比如之前学过的数组
预定义的常用集合：
动态数组：ArrayList
列表：List
字典：Dictionary
队列：Queue
栈：Stack

他们比数组更加强大，实现了更加丰富的功能，可以提高我们的开发效率

6.1动态数组ArrayList

初始化，可以不指定大小
获取长度，使用Count
添加 Add
删除 Remove RemoveAt
访问（index）

6.1动态数组ArrayList

更好用的List(Of T)
ArrayList是类型不安全的，而且有拆箱装箱的性能问题。
于是出现了List(Of T)

```
Dim dogs As List(Of Dog) = New List(Of Dog)
dogs.Add(New Dog("小乔"))
dogs.Add(New Dog("大乔"))
dogs.Add(New Dog("鲁班"))
dogs.RemoveAt(1)
For n = 0 To dogs.Count - 1
    Console.WriteLine(dogs(n).PrintName())
Next
```

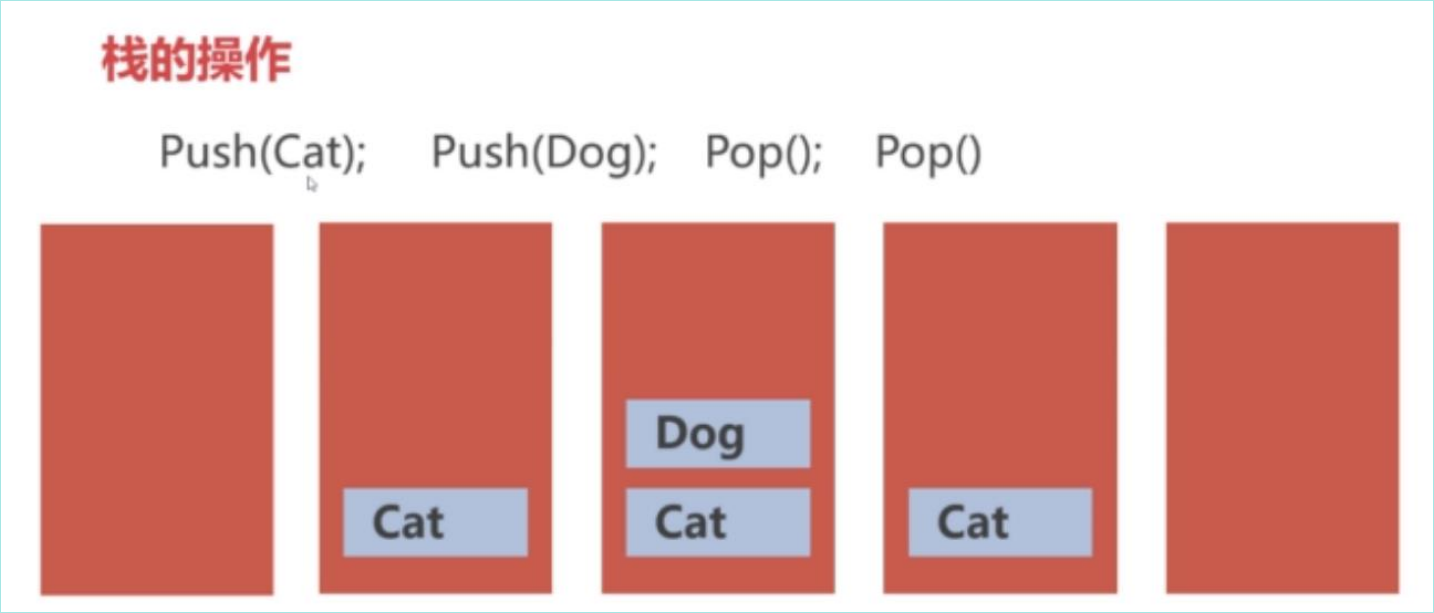
6.3字典Dictionary

字典Dictionary(Of Tkey, TValue)
字典容器存储的是一系列的键值对，每个值对应一个唯一的键。
键的意义在于，我们可以通过键相对高效的访问到值
字典操作
数量 ：Count
添加： Add（Key,Value）
删除 ： Remove
访问 ： dic(Key)

```
        Dim dic As Dictionary(Of String, Dog) = New Dictionary(Of
String, Dog)
        dic.Add("小乔", New Dog("小乔"))
        dic.Add("大乔", New Dog("大乔"))
        dic.Add("鲁班", New Dog("鲁班"))
        Console.WriteLine(dic("小乔").PrintName())
```

6.4栈

栈是先进后出，后进先出的一种容器。就好比是只有一个开口的羽毛球筒。
栈的操作：
出栈：Pop
入栈：Push
获取栈顶的元素：Peek

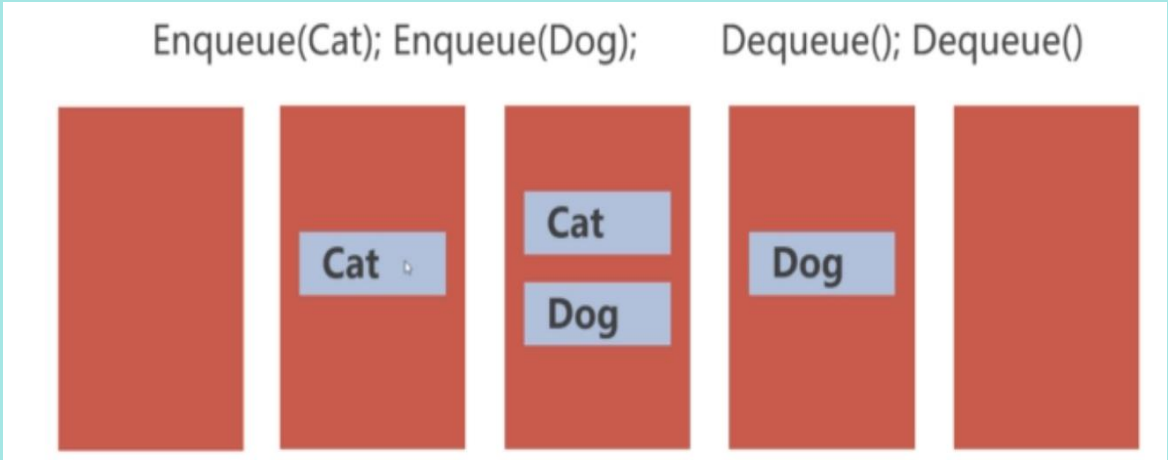


6.4栈

```
Dim stack As Stack(Of Dog) = New Stack(Of Dog)
 stack.Push(New Dog("小乔"))
 stack.Push(New Dog("大乔"))
 stack.Push(New Dog("鲁班"))
 Console.WriteLine(stack.Peek.PrintName())
 stack.Pop()
 Console.WriteLine(stack.Peek.PrintName())
```

6.5队列

队列是先进先出的容器，类似两端开口的羽毛球筒
出队：Dequeue
入队：Enqueue




```
Dim queue As Queue(Of Dog) = New Queue(Of Dog)
 queue.Enqueue(New Dog("小乔"))
 queue.Enqueue(New Dog("大乔"))
 queue.Enqueue(New Dog("鲁班"))
 queue.Dequeue()
```

七、委托

委托是持有一个或者多个方法的对象！并且该对象可以被执行，可以传递

声明委托类型

委托可以持有方法，那么它可以持有什么样的方法呢

可以声明，它是一种引用类型

```
Public Delegate Function Kanren()
```

定义委托的对象

```
Dim kan As Kanren = AddressOf hanfei2.KanKan
```

七、委托

```
Public Class Hanfei
        Public Name As String
        Sub New(name)
            Me.Name = name
        End Sub
        Public Sub KanKan(who As String)
            Console.WriteLine(Name + "kaka kan" + who)
        End Sub
    End Class
    Public Delegate Sub Kanren(who As String)
    Sub Main()
        Dim hanfei1 As Hanfei = New Hanfei("刘华强")
        Dim hanfei2 As Hanfei = New Hanfei("刘华文")

        Dim kan As Kanren = AddressOf hanfei2.KanKan
        kan.Invoke("小白")
    End Sub
```

八、事件

```
Public Class Person

    Private name As String

    Public Event walked(ByVal distance As Integer)

    Public Sub onwalk(ByVal distance As Integer)
        RaiseEvent walked(distance)
    End Sub
End Class
Public Class User
    Friend WithEvents myperson As Person
    Private Sub myperson_walked(ByVal distance As Integer) Handles myperson.walked
        Console.WriteLine("walked" & distance)
    End Sub
End Class
Sub Main()
    Dim u As User = New User
    Dim p As Person = New Person
    u.myperson = p
    p.onwalk(15)

End Sub
```