C#程序设计及应用

唐大仕

dstang2000@263.net

北京大学

第14章 深入理解C#语言

唐大仕
dstang2000@263.net
http://www.dstang.com

内容提要

- 类型与转换
- 变量及其传递
- 多态与虚方法调用
- 动态类型确定
- 对象构造与析构
- csharp语言新特性

1 类型与转换

Now Loading...

数据类型

- C#的数据类型两大类

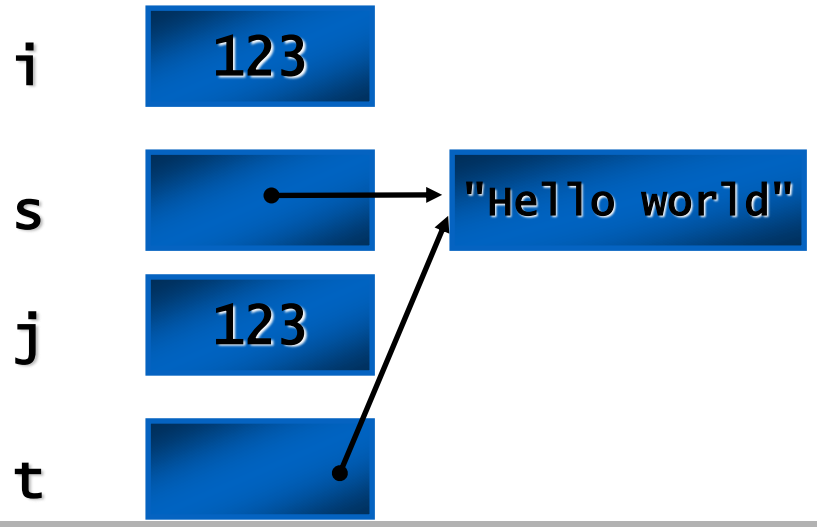
- 值类型（Value Type）
  - 简单类型（Simple Type）
  - 结构类型（Struct Type）
  - 枚举类型（Enum Type）
- 引用类型引用类型（Reference Type）
  - 类类型（Class Type）
  - 数组类型（Array Type）
  - 指代类型（Delegate）

Reference and Value Types

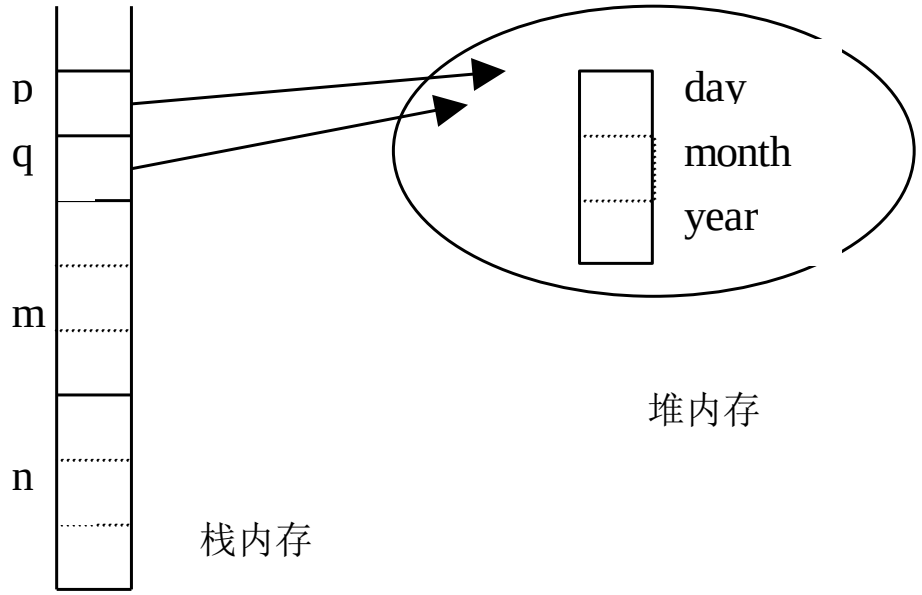
	Reference引用型 (Class)	Value 值 (Struct)
变量中的内容	引用	实际值
分配	Heap 堆	Inline 栈
默认值	Null	Zeroed
= 的含义	复制引用	复制值

```
int i = 123;
string s = "Hello world";


int j = i;
string t = s;
```



示例



值类型的转换

- 数字类型之间可以互相转化
  - 隐式转换　　float f = 3;
  - 显式转换　　int i = (int)3.14;
  - 整型提升　　sbyte a=1,b=2; a+b为int型
- 枚举类型与数字类型之间可以显式转换（强制类型转换）
- 结构类型之间不能转换

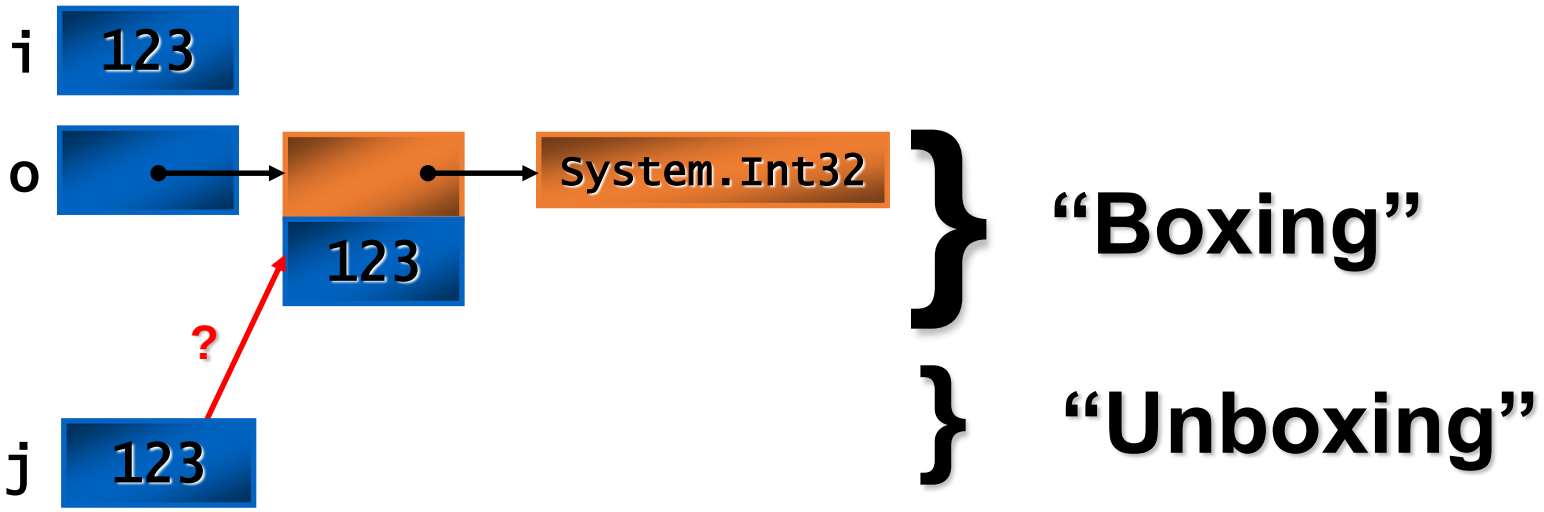
引用类型转换

- 显式或隐式转换（都要求有继承关系）
  - Person p = new Student(); //隐式转换
  - Student s = (Student) p; // 显式转换
    - 可能转成功，也可能异常
  - IRunnable r = p; //隐式转换
- as 运算符
  - Student s = p as Student;
    - 可能成功，也可能为null

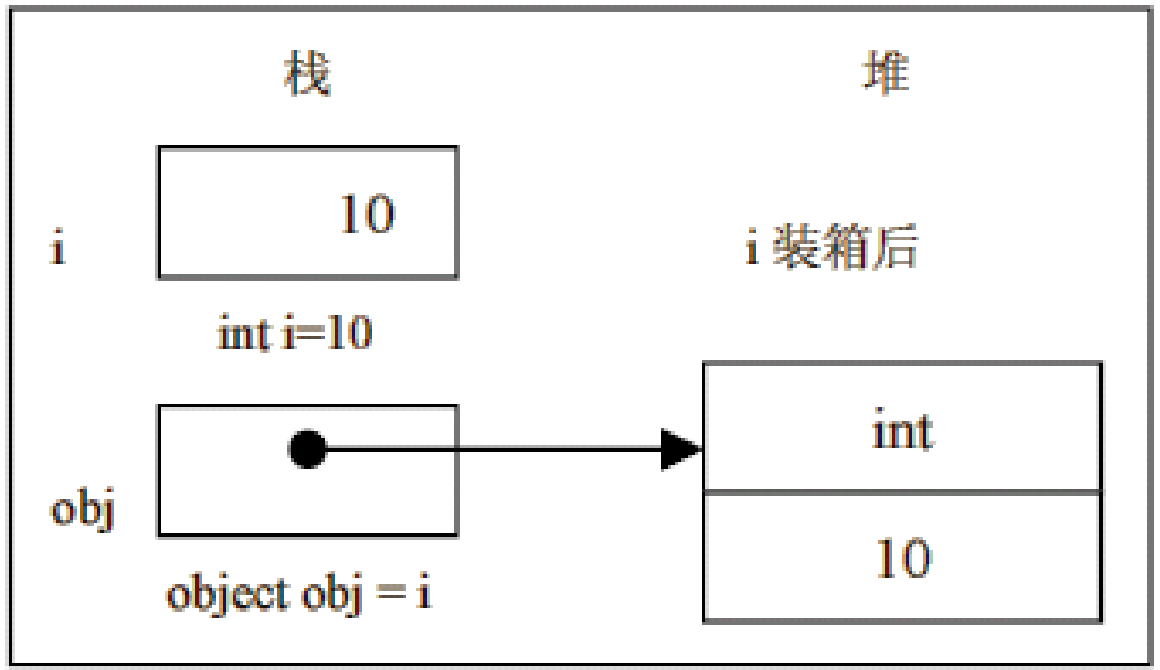
AsObject.cs

引用类型与值类型的转换

```
int i = 123;
object o = i;
int j = (int) o;
```



Boxing and unboxing



Unboxing必须显式转换

BoxingTest.cs

```
int total = 35;
string s = String.Format(
   "Your total was {0} on {1}", total, date);

Hashtable t = new Hashtable();
t.Add(0, "zero");
t.Add(1, "one");

DateTime d = DateTime.Now;
String s = d.ToString();      // no box

int[] a = new int[2];
a[0] = 33;                    // no box

ArrayList a = new ArrayList();
a.Add(33);                    // box here
```

2 变量及其传递

Now Loading...

域变量及局部变量

- 类static变量
  - 相当于类中的“全局变量”
- 域变量
  - 相当于对象中的变量
- 局部变量
  - 在函数体及其{}中，在栈中分配，自动消失
- 域变量自动有初值，局部变量则不

LocalVarAndMemberVar.cs

按值传送的参数

- 按值传送
  - 要注意值类型与引用类型
  - TransByValue.cs　TransByValueStructClass.cs

ref参数及out参数

- ref参数在传之间必先赋值
- out参数在函数中必须赋值后才能返回
- 表达式及对象的属性不能作ref及out参数
  - TransByRef.cs
  - RefSwap.cs
  - TransByOut.cs
  - RefColorRGB.cs

params参数

- 数组参数（相当于VB的可变参数）
- 如 string.的 Split(params char[]);
- 参数必须放在最后
- 调用时，可用数组，也可用多个参数
  - 定义：
    - double Multi(params double [] nums)
  - 调用
    - Multi();
    - Multi(27);
    - Multi(3.14, 0.9, 0.9);
    - Multi(1,2,3,4,5);
    - Multi(new double [] {1,2,3,4,5});

ParamsTest.cs

默认参数

- 在参变量中使用
- 如：
  - void ZoomIn(Point p, double k = 1){ ... ... }
  - 调用时可以这样
    - ZoomIn(p, 1.5)
    - ZoomIn(p) 相当于 ZoomIn(p, 1)

3 多态与虚方法调用

Now Loading...

多态(Polymorphism)

- 在面向对象的系统中，多态性是一个非常重要的概念，它允许客户对一个对象进行操作，由对象来完成一系列的动作，具体实现哪个动作、如何实现由系统负责解释。
- 在C#中，多态性的定义是，同一操作作用于不同的类的实例，不同的类将进行不同的解释，最后产生不同的执行结果。C#支持两种类型的多态性。

两种类型的多态性

- 编译时的多态性
  - 编译时的多态是通过重载来实现的。对于非虚的成员来说，系统在编译时，根据传递的参数、返回的类型等信息决定实现何种操作。
- 运行时的多态性
  - 运行时的多态性是指直到系统运行时，才根据实际情况决定实现何种操作。C#中，运行时的多态性通过虚成员实现。

- 编译时的多态性提供了运行速度快的特点，而运行时的多态性则带来了高度灵活和抽象的特点。

上溯造型与虚方法调用

```
static void doStuff( Shape s )      {
        s.draw();
}
Shape  c = new Circle();
Shape  r = new Rectangle();
Shape  s = new Square();
doStuff(c);
doStuff(r);
doStuff(s);
```

VirtualInvokeShape.cs

虚方法的语法规范

- 必须有virtual或abstract或override所修饰。
- 虚方法不能省略访问控制符，不能是private的，不能是static的，因为它们应该可以被子类所覆盖。
- 子类中要覆盖父类的虚方法，必须用override。否则认为是新（new）的一个方法，并隐藏了父类的方法，不会实行虚方法调用。
- 覆盖和被覆盖的方法必须有相同的可访问性和相同的返回类型。

虚方法与非虚方法的区别

- 虚方法调用的方法是由对象实例的类型所决定
- 非虚方法调用的方法是由所声明的对象变量来决定的。

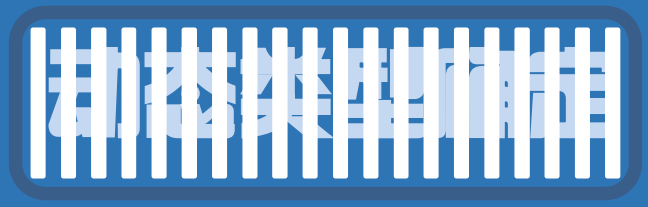
VirtualAndNoneVirtual.cs

最可派生的方法

- 当多次派生，并有virtual与非virtual方法时

VirtualComplex.cs

4 动态类型确定



Now Loading...

is运算符

- 用于判断运行时对象的类型
- 格式：对象 is 类型
  - 不是null
  - 可转换，而不异常

== 与 !=

- 值类型相等
- 引用类型的相等
  - 是判断是否是同一对象
  - 除非重载了==及!=操作
  - 对于string等类型，已经进行了重载
  - 对于两个boxing的对象，==总是false

得到类型信息

- typeof运算符
  - typeof(System.Console)
- 对象.GetType()
- Type.GetType(string 类名)

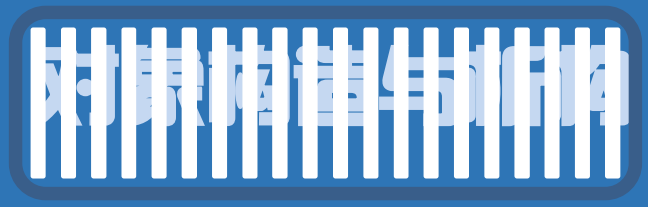
TypeGetType.cs TypeGetMembers.cs

反射

- 反射Reflection

ReflectionTest.cs

5 对象构造与析构



Now Loading...

构造方法

- 构造方法的初始化部分，使用this及base
  - 调用本类或父类的构造方法
  - 不用this且不用base，则会自动认为base()

```
class A{
        A(int a){}
}
class B : A{
        B(String s ){}  //编译不能通过
}
```

ConstructCallThisAndBase.cs

域的初始化

- 域的初始化中不能引用this
- 在base被调用之前不能引用 this

```
class B : A{
        int x = 1;
        int M(){ return 1; }
        int y = x+M(); //Error
        B():base(x){}  //Error
}
```

构造方法的执行过程

- 执行过程
  - 若有this(…)，转向之
  - 执行域的初始化
  - 转到base
  - 执行方法体
- 应避免在构造方法中调用任何虚方法

ConstructorExecution.cs

静态构造方法

- 静态构造方法总是在该类的所有静态域初始化之后执行；
- 静态构造方法总是在该类被使用（如访问静态域、生成实例）之前完成；
- 静态构造方法最多被执行一次。
- 静态构造方法的执行顺序的不确定性，所以在使用构造方法时应谨慎
- 应尽量避免在静态初始化或静态域中出现循环引用的情况。
  - static int a=b+1; static int b=a+1;

析构方法

- 析构方法
  - ~类名(){}
- 析构方法会自动调用父类的析构方法
- 注：
  - 编译器生成的构造方法名字为ctor()
  - 生成的析构方法名字为Finalize()
- 不能显式地调用析构方法

自动垃圾回收

- 垃圾回收 (garbage collection)
- 自动回收
- “强制”回收 System.GC.Collect();

```
String a,b;
a = String.Copy("hello world");
b = String.Copy("game over");
Console.WriteLine(a+b+"ok");
a = null;
a = b;
```

显式资源管理

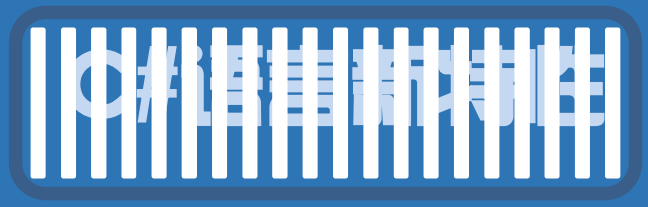
- 实现IDisposable接口
- 其中有方法：
  - void Dispose();

使用using语句

```
R r1 = new R();
try {
  r1.F();
}
finally {
  if (r1 != null)

  ((IDisposable)r1).Dispose();
}
```

```
using (  R  r1 = new R() ){
        r1.F();
}
```



Now Loading...

csharp语言新特性

- C#2.0 引入泛型
- C#3.0 引入Lambda及Linq
- C#4.0 引入动态特性 dynamic
- C#5.0 引入并行及异步 async/await及Task
- C#6.0 改进编译，属性初始化

泛型(Generic) C#2.0

- List<Book> books=new List<Book>();
- Book book = books[0];

- //以前要用强制类型转换
- ArrayList books = new ArrayList();
- Book book = (Book) books[0];

常用的泛型的delegate

- Func<T, TResult>
  - 多个参数Func<T1, T2, T3, T4, TResult>
- Action<T>
  - 多个参数Action<T1, T2, T3, T4>
- Predicate<T>
- Comparison<T>
- Converter<TInput,TOutput>
- EventHandler<TEventArgs>

匿名方法 C#2.0

- delegate(参数){ 方法体}
- 可以当一个委托
  - new Thread(
    - new ThreadStart(delegate(){ .....}));
- 可以被隐式转换为一个兼容的委托类型
  - new Thread(delegate(){ .....});

Lambda表达式 C#3.0

- 常见的形式
  - from c in customers
  - where c.Age>10
  - orderby c.Name
  - select new {c.Name, c.Phone}
- 相当于
  - customers
  - .Where(c=> c.Age>10)
  - .OrderBy(c => c.Name).
  - .Select(c => new { c.Name, c.Phone })
- 更多Linq示例：
  - https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

Linq C# 3.0

- 常见的形式
  - from c in customers
  - where c.Age>10
  - orderby c.Name
  - select new {c.Name, c.Phone}
- 相当于
  - customers
  - .Where(c=> c.Age>10)
  - .OrderBy(c => c.Name).
  - .Select(c => new { c.Name, c.Phone })
- 更多Linq示例：
  - https://code.msdn.microsoft.com/101-LINQ-Samples-3fb9811b

扩展方法 C#3.0

- 用于扩展一个类，给对象加上方法
  - public static class T{
    - public static bool IsLonger(this string str, int n)
    - {
    - return str!=null && str.Length>n;
    - }
  - }

  - 使用 string s = “abc”;
  - Console.Write(s.IsLonger(2));

动态语言特性 C#4.0

- dynamic
- 可以动态表示任意对象，在编译时不进行语法检查
  - //ExpandoObject：表示一个对象，该对象包含可在运行时动态添加和移除的成员。
  - dynamic expando=new ExpandoObject();
  - expando.Name = "DebugLZQ";
  - expando.Speak = new Action(()=>Console.Writee("My name is {0}",expando.Name));
  - expando.Speak();
- 主要用于与COM组件的交互
  - dynamic cell = new Cell();
  - cell.Formula = “=SUM(A1:A3)” ;

异步编程 C#5.0

- async及await
  - Task<double> FacAsync(int n) {
  - return Task<double>.Run(()=>{return s; });
  - }
  - async void Test() {
  - double result = await FacAsync(10);
  - Console.WriteLine(result); }
- 参见多线程及异步编程
  - 更多示例 https://code.msdn.microsoft.com/C-50-AsyncAwait-Demo-Code-334679a5

并行编程 C#5.0

- Parallel
  - Parallel.Invoke(.....) ;
  - Parallel.For(0, 10, i =>{.....}) ;
  - Parallel.ForEach(list,
  - (double x, ParallelLoopState state) =>{.....});
- PLinq
  - from n in dic.Values.AsParallel()
  - where n.Age > 20 && n.Age < 25
  - select n

初始化更方便 C#6.0

- class T{
  - public string Name{get;} = “Mr.”;
  - public int Age => 20;
  - public int Add(int a, int b) => a+b;
  - public void Fun(string s) => Console.Write(s);

小结

- 类型与转换
  - as boxing
- 变量及其传递
  - 按值 out ref params
- 多态与虚方法调用
  - abstract, virtual override
- 动态类型确定
  - typeof is == != 反射
- 对象构造与析构
  - 构造顺序 IDisposable using

问题与讨论

dstang2000@263.net
http://www.dstang.com