您的潜力，我们的动力

Microsoft
微软（中国）有限公司

C#锐利体验2.0：泛型编程

李建忠
www.lijianzhong.com

上海祝成科技 高级讲师

msdn

MSDN Webcasts

Agenda

- C#泛型及机制
- 泛型类型
- 泛型方法
- 泛型约束
- 讲座总结
- Q&A

C#泛型演示

```
class Stack<T> {
    private T[] store;
    private int size;
    public Stack()   {
        store = new T[10]; size = 0;
    }
    public void Push(T x) {
        store[size++] = x; }
    public T Pop()  {
        return store[--size];
    }
}
```

C#泛型简介

```
Stack<int> x = new Stack<int>();
x.Push(17);
```

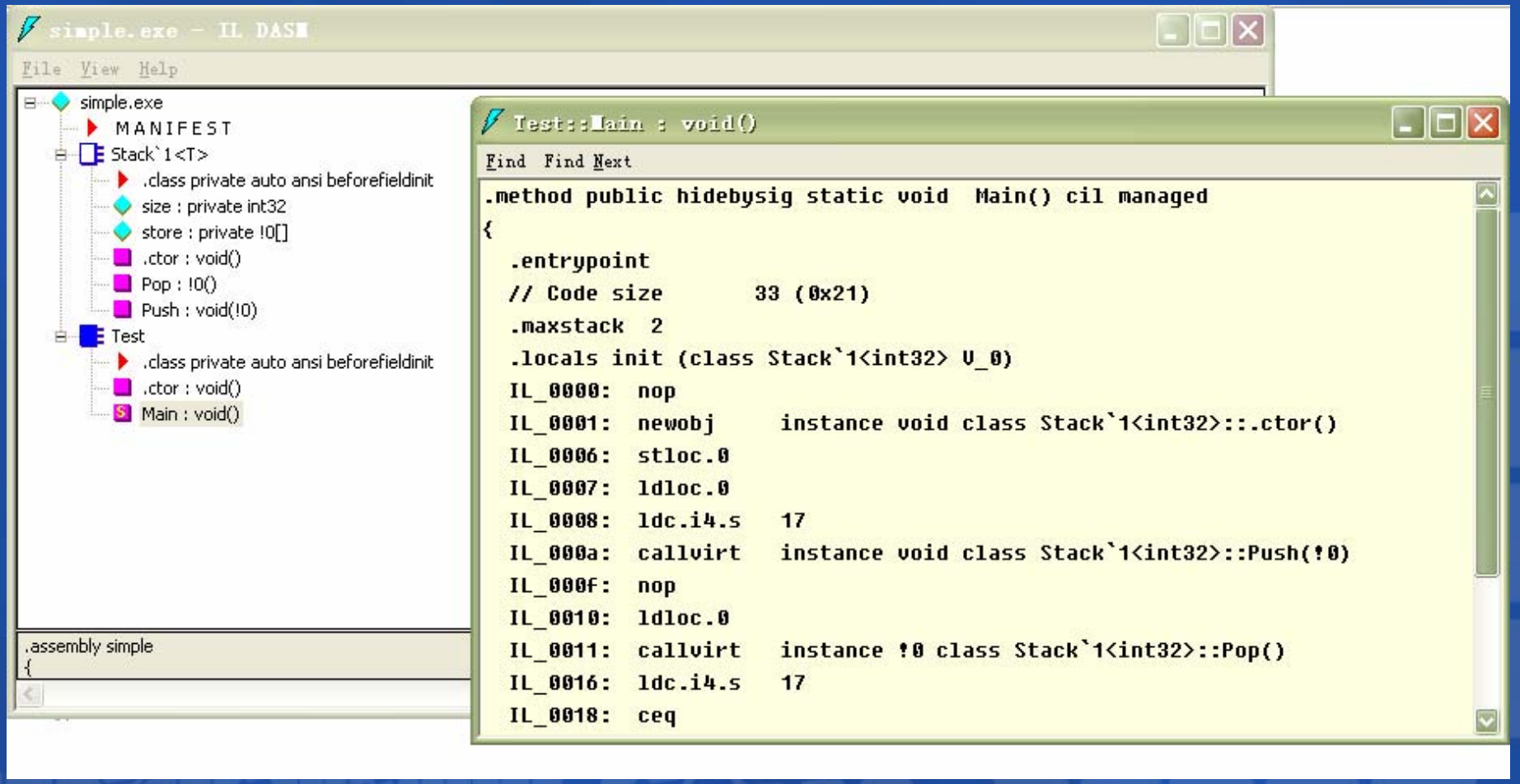
- 所谓泛型，即通过参数化类型来实现在同一份代码上操作多种数据类型。泛型编程是一种编程范式，它利用“参数化类型”将类型抽象化，从而实现更为灵活的复用。
- C#泛型赋予了代码更强的类型安全，更好的复用，更高的效率，更清晰的约束。

C#泛型机制简介

- C#泛型能力由CLR在运行时支持，区别于C++的编译时模板机制，和Java的编译时“搽拭法”。这使得泛型能力可以在各个支持CLR的语言之间进行无缝的互操作。

- C#泛型代码在被编译为IL代码和元数据时，采用特殊的占位符来表示泛型类型，并用专有的IL指令支持泛型操作。而真正的泛型实例化工作以“on-demand” 的方式，发生在JIT编译时。

泛型IL代码与元数据



C#泛型编译机制

- 第一轮编译时，编译器只为Stack<T>类型产生“泛型版”的IL代码与元数据——并不进行泛型类型的实例化，T在中间只充当占位符
- JIT编译时，当JIT编译器第一次遇到Stack<int>时，将用int替换“泛型版”IL代码与元数据中的T——进行泛型类型的实例化。
- CLR为所有类型参数为“引用类型”的泛型类型产生同一份代码；但如果类型参数为“值类型”，对每一个不同的“值类型”，CLR将为其产生一份独立的代码

C#泛型的几个特点

- 如果实例化泛型类型的参数相同，那么JIT编译器会重复使用该类型，因此C#的动态泛型能力避免了C++静态模板可能导致的代码膨胀的问题。

- C#泛型类型携带有丰富的元数据，因此C#的泛型类型可以应用于强大的反射技术。

- C#的泛型采用“基类, 接口, 构造器, 值类型/引用类型”的约束方式来实现对类型参数的 “显式约束”，提高了类型安全的同时，也丧失了C++模板基于“签名”的隐式约束所具有的高灵活性。

Agenda

- C#泛型及机制
- 泛型类型
- 泛型方法
- 泛型约束
- 讲座总结
- Q&A

C#泛型类与结构

```
class C<U, V> {}   //合法
class D: C<string,int>{}  //合法
class E<U, V>: C<U, V> {}  //合法
class F<U, V>: C<string, int> {}  //合法
class G : C<U, V> { }  //非法
```

C#除可单独声明泛型类型（包括类与结构）外，也可在基类中包含泛型类型的声明。但基类如果是泛型类，它的类型参数要么已实例化，要么来源于子类（同样是泛型类型）声明的类型参数。

泛型类型的成员

```
class C<V>{
    public V f1;   //声明字段
    public D<V> f2;   //作为其他泛型类型的参数
    public C(V x) {
        this.f1 = x;
    }
}
```

泛型类型的成员可以使用泛型类型声明中的类型参数。但类型参数如果没有任何约束，则只能在该类型上使用从System.Object继承的公有成员。

泛型接口

```
interface IList<T> {
    T[] GetElements();
}
interface IDictionary<K,V> {
    void Add(K key, V value);
}
// 泛型接口的类型参数要么已实例化，
// 要么来源于实现类声明的类型参数
class List<T> : IList<T>, IDictionary<int, T> {
    public T[] GetElements() { return null; }
    public void Add(int index, T value) { }
}
```

泛型委托

```
delegate bool Predicate<T>(T value);
class X {
    static bool F(int i) {...}
    static bool G(string s) {...}
    static void Main() {
        Predicate<string> p2 = G;
        Predicate<int> p1 = new Predicate<int>(F);
    }
}
```

泛型委托支持在委托返回值和参数上应用参数类型，这些参数类型同样可以附带合法的约束。

Agenda

- C# 泛型及机制
- 泛型类型
- 泛型方法
- 泛型约束
- 讲座总结
- Q&A

泛型方法简介

- C#泛型机制只支持“在方法声明上包含类型参数”——即泛型方法
- C#泛型机制不支持在除方法外的其他成员（包括属性、事件、索引器、构造器、析构器）的声明上包含类型参数，但这些成员本身可以包含在泛型类型中，并使用泛型类型的类型参数
- 泛型方法既可以包含在泛型类型中，也可以包含在非泛型类型中

泛型方法的声明与调用

```
public class Finder {
   // 泛型方法的声明
   public static int Find<T>  ( T[] items,  T item) {
        for(int i=0;i<items.Length;i++){
              if (items[i].Equals(item))  { return i; }
        }
        return -1;
   }
}
// 泛型方法的调用
int i=Finder.Find<int> ( new int[]{1,3,4,5,6,8,9}, 6);
```

泛型方法的重载

```
class MyClass {

    void F1<T>(T[] a, int i);       // 不可以构成重载方法
    void F1<U>(U[] a, int i);

    void F2<T>(int x);    //可以构成重载方法
    void F2(int x);

    void F3<T>(T t) where T : A; //不可以构成重载方法
    void F3<T>(T t) where T : B;
}
```

泛型方法的重写

```
abstract class Base
{
    public abstract T F<T,U>(T t, U u) where U: T;
    public abstract T G<T>(T t) where T: IComparable;
}
class Derived: Base{
    //合法的重写，约束被默认继承
    public override X F<X,Y>(X x, Y y)  {  }

    //非法的重写，指定任何约束都是多余的
    public override T G<T>(T t) where T: IComparable {}
}
```

Agenda

- C#泛型及机制
- 泛型类型
- 泛型方法
- 泛型约束
- 讲座总结
- Q&A

泛型约束简介

- C#泛型要求对“所有泛型类型或泛型方法的类型参数”的任何假定，都要基于“显式的约束”，以维护C#所要求的类型安全。
- “显式约束”由where子句表达，可以指定“基类约束”，“接口约束”，“构造器约束”，“值类型/引用类型约束”共四种约束。
- “显式约束”并非必须，如果没有指定“显式约束”，泛型类型参数将只能访问System.Object类型中的公有方法。

基类约束

```
class A  {    public void F1() {…} }
class B  {  public void F2() {…} }

class C<S,T>
    where S: A   // S继承自A
    where T: B   // T继承自B
{
    // 可以在类型为S的变量上调用F1，
    // 可以在类型为T的变量上调用F2
    ….
}
```

接口约束

```
interface IPrintable  {    void Print();  }
interface IComparable<T>  { int CompareTo(T v);}
interface IKeyProvider<T>  { T GetKey(); }

class Dictionary<K,V>
    where K: IComparable<K>
    where V: IPrintable, IKeyProvider<K>
{
    // 可以在类型为K的变量上调用CompareTo，
    // 可以在类型为V的变量上调用Print和GetKey
    ....
}
```

构造器约束

```
class A {     public A() { }  }
class B {     public B(int i) { }  }

class C<T>
   where T : new()
{
   //可以在其中使用T t=new T();
   ....
}
C<A> c=new C<A>();   //可以，A有无参构造器
C<B> c=new C<B>();   //错误，B没有无参构造器
```

值类型/引用类型约束

```
public struct A { … }
public class B { … }

class C<T>
   where T : struct
{
   // T在这里面是一个值类型
   …
}
C<A> c=new C<A>();  //可以，A是一个值类型
C<B> c=new C<B>();  //错误，B是一个引用类型
```

Agenda

- C#泛型及机制
- 泛型类型
- 泛型方法
- 泛型约束
- 讲座总结
- Q&A

讲座总结

- C#的泛型能力由CLR在运行时支持，它既不同于C++在编译时所支持的静态模板，也不同于Java在编译器层面使用“搽拭法”支持的简单的泛型。
- C#的泛型支持包括类、结构、接口、委托共四种泛型类型，以及方法成员。
- C#的泛型采用“基类, 接口, 构造器, 值类型/引用类型”的约束方式来实现对类型参数的 “显式约束”，它不支持C++模板那样的基于签名的隐式约束。

Agenda

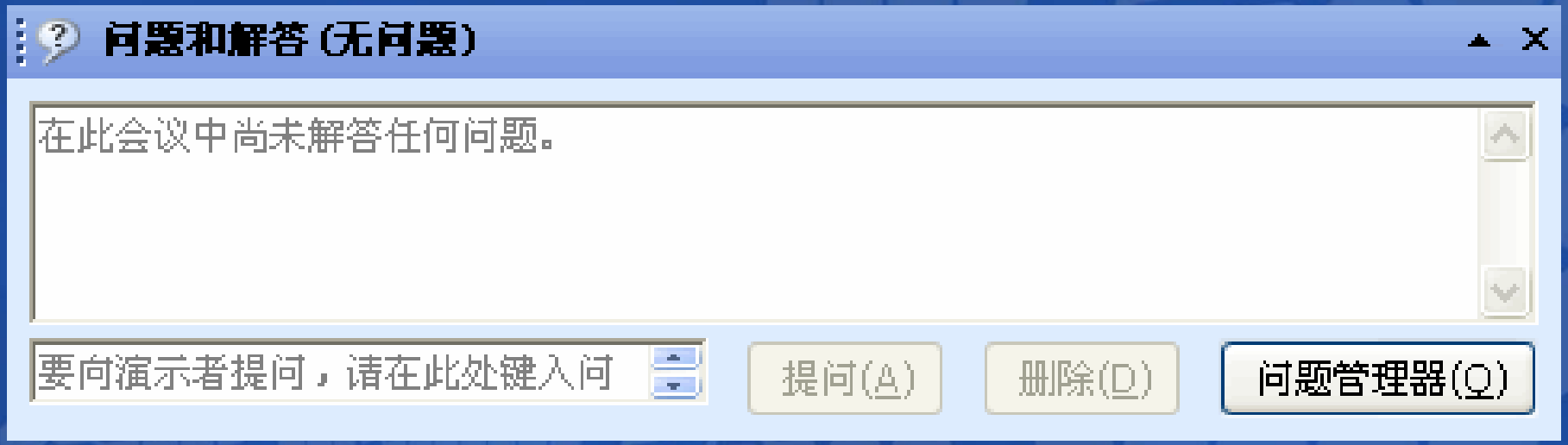
- C#泛型及机制
- 泛型类型
- 泛型方法
- 泛型约束
- 讲座总结
- Q&A

您的潜力，我们的动力

Microsoft
微软(中国)有限公司

Q&A

如需提出问题，请单击“提问”按钮并在随后显示的浮动面板中输入问题内容。一旦完成问题输入后，请单击“提问”按钮。




您的潜力，我们的动力
Microsoft
微软（中国）有限公司
Microsoft
msdn
MSDN Webcasts