C#从入门到入土

源码：http://pan.aqrose.com/f/61ab889a15d84db089fe/

1. 编译

1.1 编译过程

1. 选择编译器。
2. 将代码编译为 MSIL。 编译将你的源代码转换为 Microsoft 中间语言 (MSIL) 并生成必需的元数据。
3. 将 MSIL 编译为本机代码。 在执行时，实时 (JIT) 编译器将 MSIL 转换为本机代码。 在此编译期间，代码必须通过检查 MSIL 和元数据的验证过程以查明是否可以将代码确定为类型安全。
4. 运行代码。 公共语言运行时提供启用要发生的执行的基础结构以及执行期间可使用的服务。

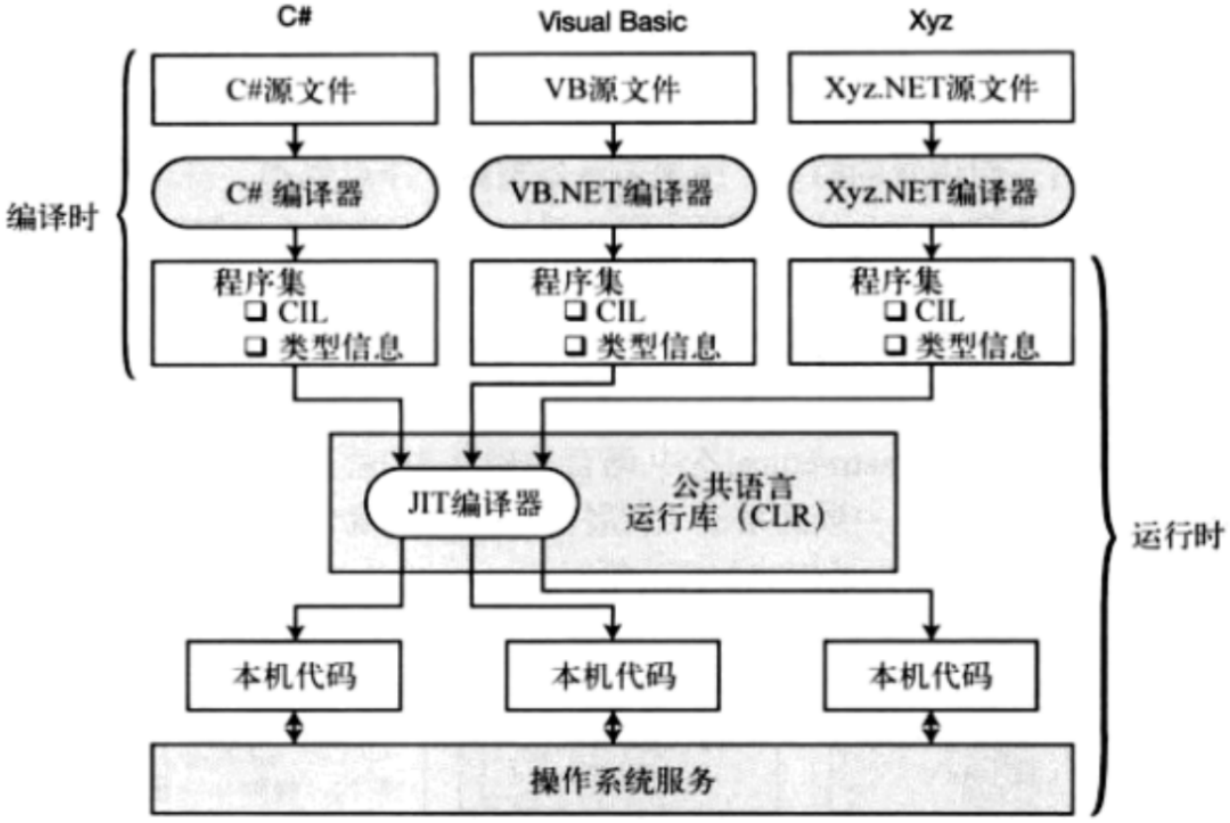
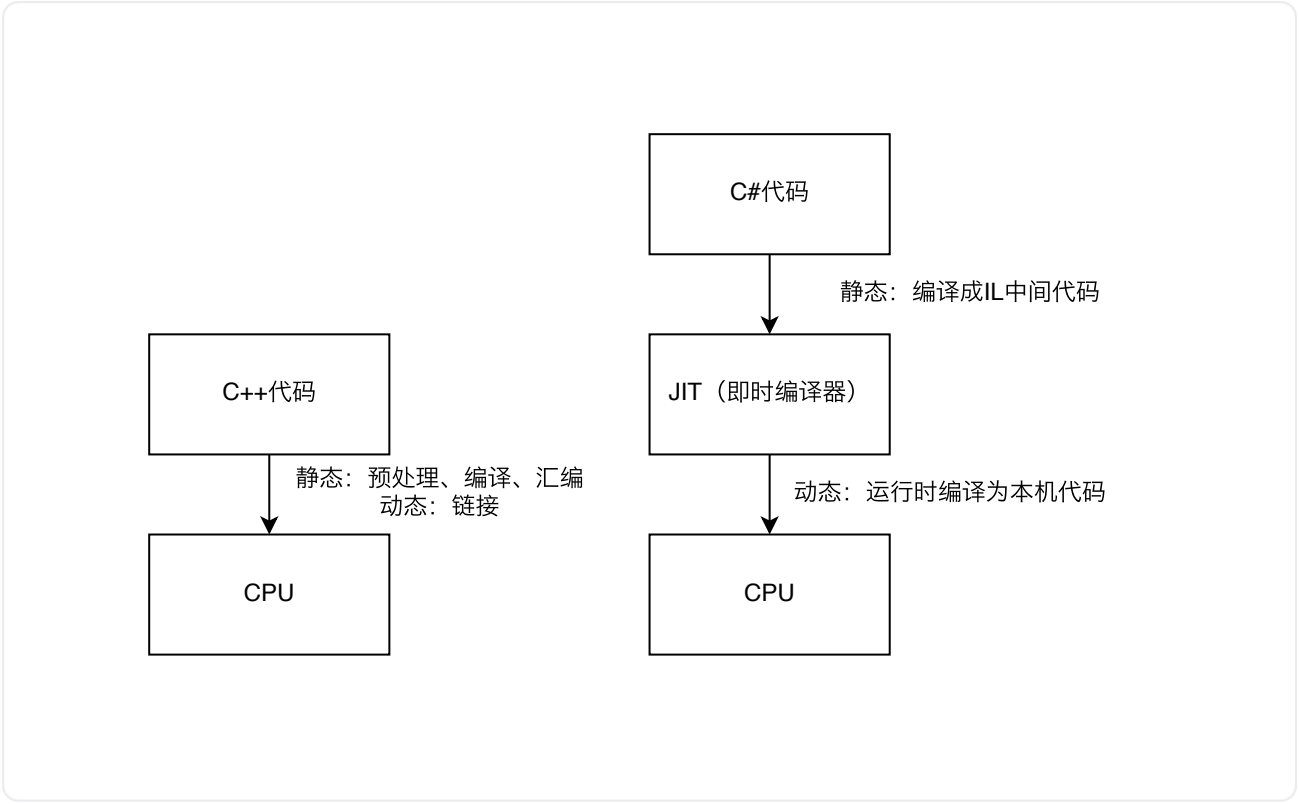


图1.1.1 托管语言编译过程



1.2 中间代码查看

中间代码可以通过ildasm.exe查看，软件详情如图1.2.1。

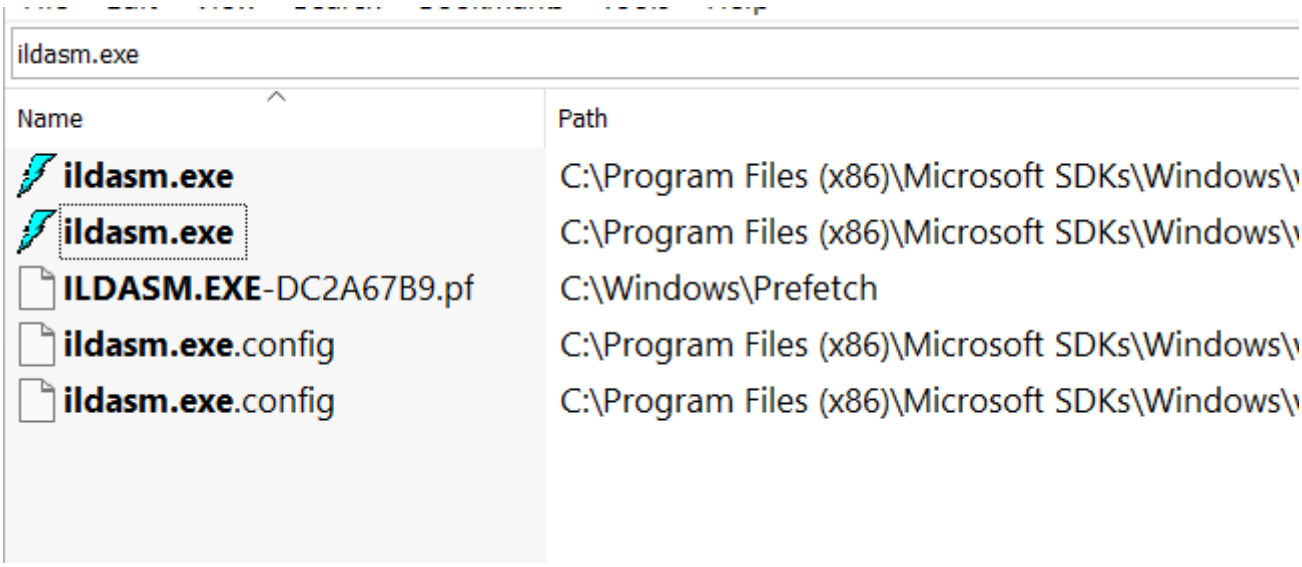


图1.2.1 ildasm.exe软件详情

1.3 C#、C++对比

	C#	C++
内存管理	C# 中的内存管理由托管运行时（CLR）负责，包括自动内存分配和垃圾回收。开发人员不需要手动管理内存，可以专注于业务逻辑的实现。	在 C++ 中，开发人员需要手动管理内存，包括内存的分配和释放。这给开发带来了更大的灵活性，但也增加了出错的可能性。
安全性	C#虚拟机提供了严格的类型安全和边界检查，可以防止许多常见的内存错误和安全漏洞。	C++允许直接访问硬件和系统资源，所以会造成段错误等崩溃。
平台相关	通过 .NET Core，C# 实现了跨平台支持，使得开发的应用程序可以在不同操作系统上运行。	由于C++直接编译为汇编语言，所以在不同平台的时候需要重新编译代码，并需要保证代码跨平台，例如：Linux与Windows API差异、编译器差异、构建环境差异等。
平台相关	C#方法被调用时，公共语言运行库CLR把具体的方法编译成适合本地计算机运行的机器码，并且将编译好的机器码缓存起来，以备下次调用使用。	C++直接编译为机器码

2. 反射

2.1 属性原理讲解

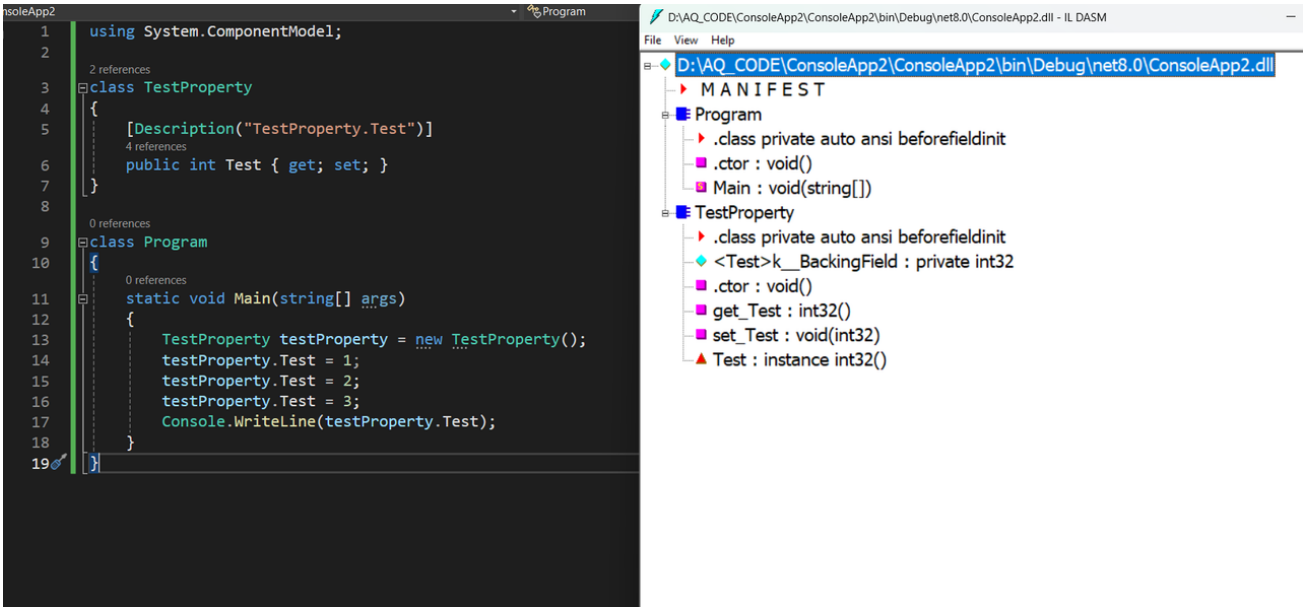


图2.1.1 属性的中间代码结构图

如图2.1.1可知。

属性在生成中间代码时，会生成对应的get和set方法，所以存在抽象属性，抽象属性生成中间代码时将get、set定义为抽象方法，可以继承并重写。

```
.property instance int32 Test()
{
  .custom instance void [System.ComponentModel.Primitives]System.ComponentModel.DescriptionAttribute::.ctor(string) =
( 01 00 11 54 65 73 74 50 72 6F 70 65 72 74 79 2E   // ...TestProperty.

54 65 73 74 00 00 )                                 // Test..
  .get instance int32 TestProperty::get_Test()
  .set instance void TestProperty::set_Test(int32)
} // end of property TestProperty::Test
```

图2.1.2 属性的中间代码详情

如图2.1.2可知。

属性[DescriptionAttribute]相当于在Test中内嵌一个类，并通过ctor（构造函数）进行初始化。

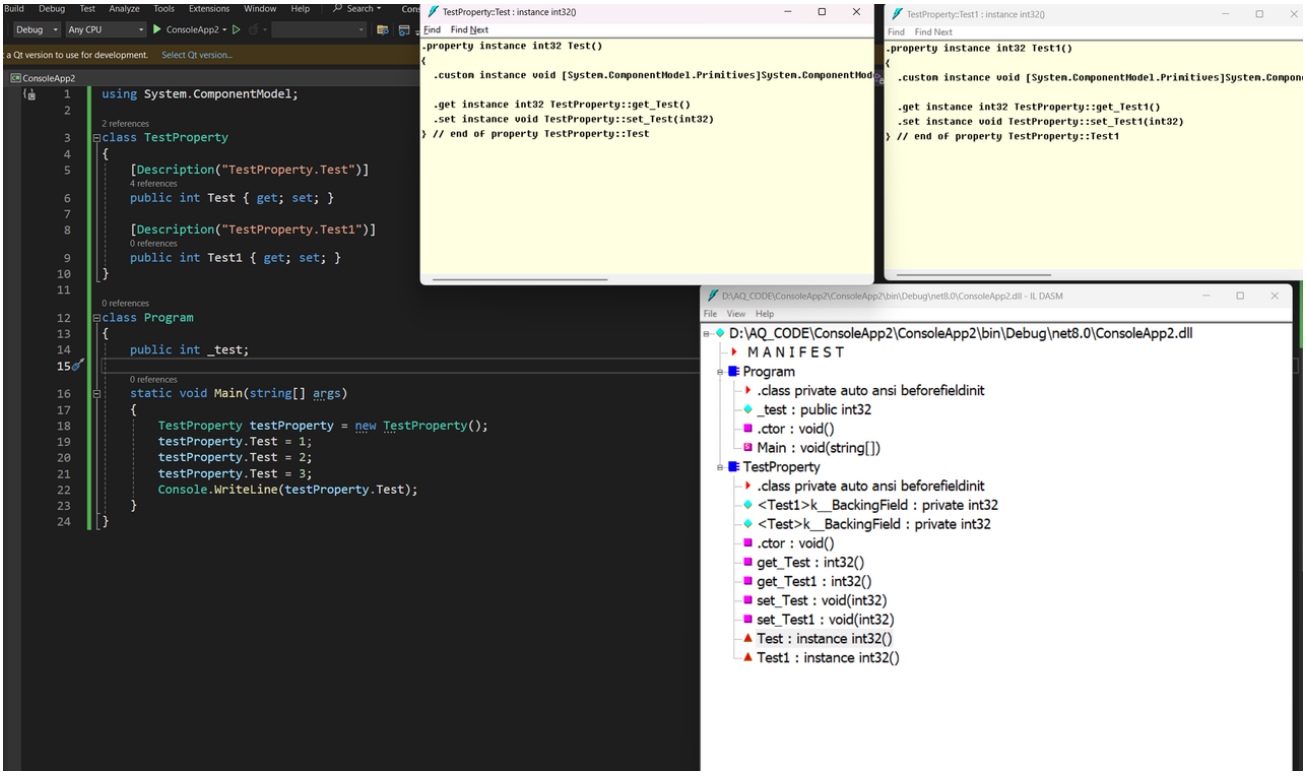


图2.1.3 不同属性对比图

如图2.1.3可知。

属性Test和属性Test1中的描述属性各一份。

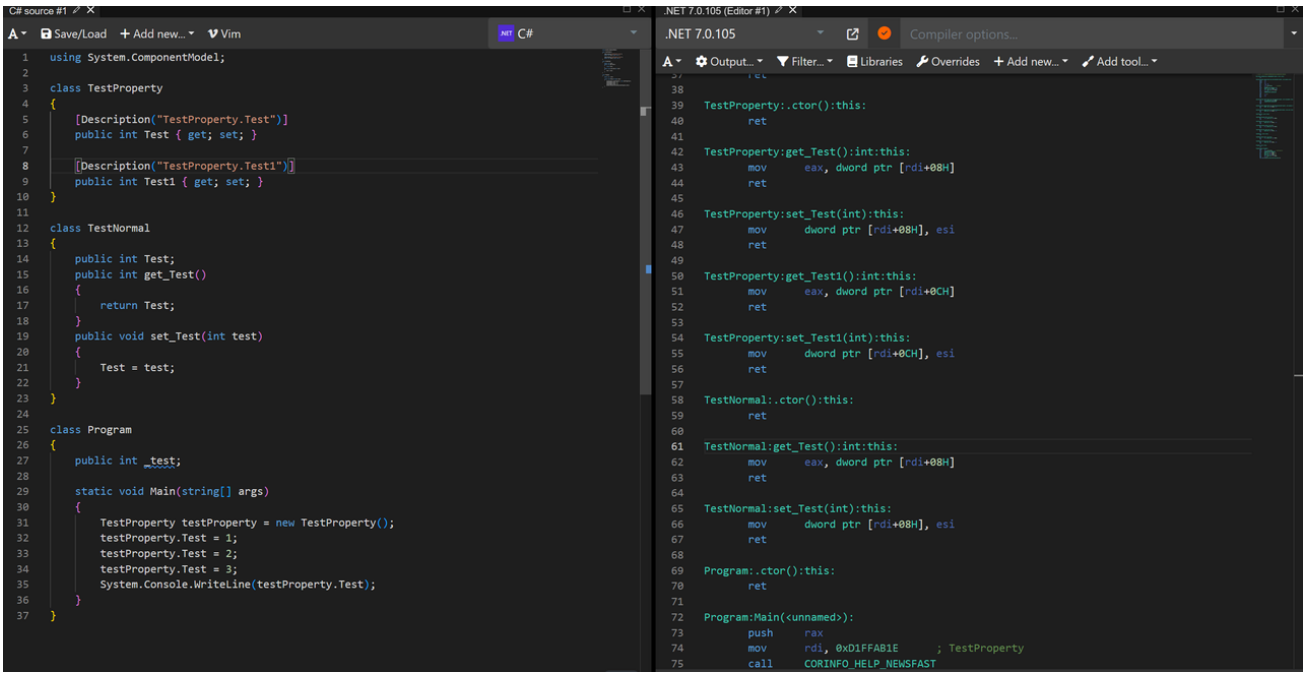


图2.1.4 属性与成员

如图2.1.4可知。

Test的内存地址是[rdi+08H]

Test1的内存地址是[rdi+0CH]

相差为4，正好是Test的sizeof，和成员变量的内存模型一致；

TestProperty与TestNormal中的test汇编源码一致，属性与成员变量一致。

2.2 属性用法

1. 通过属性对Json序列化反序列化

```
using Newtonsoft.Json; // 需要Nuget中安装此包 Newtonsoft
using System.Collections.Generic;

public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
    public List<string> Hobbies { get; set; }
}

public class Program
{
    static void Main(string[] args)
    {
        string json = "{ \"Name\": \"John\", \"Age\": 30, \"Hobbies\":
[\"Reading\", \"Swimming\"] }";
        Person person = JsonConvert.DeserializeObject<Person>(json);
    }
}
```

```
type Mysql struct {
    User     string `yaml:"user"`
    Password string `yaml:"password"`
    Host     string `yaml:"host"`
    Port     int    `yaml:"port"`
    Database string `yaml:"database"`
}
```

```
type UserAuthorize struct {
    ID         uint64    `gorm:"type:bigint;PRIMARY_KEY;COMMENT:唯一标识符;"`
    Type       string    `gorm:"type:varchar(5);NOT NULL;COMMENT:第三方平台类
型;"`
    AppId      string    `gorm:"type:varchar(150);NOT NULL;COMMENT:平台AppId"`
    OpenId     string    `gorm:"type:varchar(150);size:150;NOT NULL;COMMENT:平
台OpenId;"`
    UserName   string    `gorm:"type:varchar(255);NOT NULL;COMMENT:平台用户名;"`
    UserId     uint64    `gorm:"type:bigint;COMMENT:用户Id;"`
    UserInfo   string    `gorm:"type:json;COMMENT:平台用户信息;"`
    CreateTime time.Time `gorm:"type:datetime;NOT NULL;COMMENT:创建时间;"`
    UpdateTime time.Time `gorm:"type:datetime;NOT NULL;COMMENT:更新时间;"`
}
```

```
func (this *UserDao) SelectOneById(id int64) (*model.User, *gorm.DB) {
    user := &model.User{}
    result := mysql.GetInstance().Where(&model.User{
        ID: id,
    }).First(&user)
    return user, result
}
```

He3-开发者必备的万能工具箱

3. 委托

3.1 指向非静态函数的委托

C#代码

```
NumberChangerusing System;
namespace DelegateAppl
{
    class TestDelegate
    {
        delegate int NumberChanger(int n);
        int num = 10;
        public int AddNum(int p)
```

```
        {
            num += p;
            return num;
        }
        public int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public int getNum()
        {
            return num;
        }
        static void Main(string[] args)
        {
            TestDelegate testDelegate = new TestDelegate();
            NumberChanger nc1 = new NumberChanger(testDelegate.AddNum);
            NumberChanger nc2 = new NumberChanger(testDelegate.MultNum);
            nc1(25);
            Console.WriteLine("Value of Num: {0}", testDelegate.getNum());
            nc2(5);
            Console.WriteLine("Value of Num: {0}", testDelegate.getNum());
            Console.ReadKey();
        }
    }
}
```

IL中间代码

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  .custom instance void
System.Runtime.CompilerServices.NullableContextAttribute::.ctor(uint8) = ( 01
00 01 00 00 )
  // Code size       101 (0x65)
  .maxstack  2
  .locals init (class DelegateAppl.TestDelegate V_0,
           class DelegateAppl.TestDelegate/NumberChanger V_1,
           class DelegateAppl.TestDelegate/NumberChanger V_2)
  // 初始化三个变量 V_0 V_1 V_2
  IL_0000:  nop
  // no operation 没有任何操作，占位符
```

```
  IL_0001:  newobj     instance void DelegateAppl.TestDelegate::.ctor()
  // newobj
  IL_0006:  stloc.0
  // store locals 0: 将函数返回值存储至V_0
  IL_0007:  ldloc.0
  // load locals 0 V_0 是TestDelegate实例，将V_0 push到参数栈中
  IL_0008:  ldftn      instance int32 DelegateAppl.TestDelegate::AddNum(int32)
  // load function 将函数指针push到参数栈中
  IL_000e:  newobj     instance void
DelegateAppl.TestDelegate/NumberChanger::.ctor
                       (object, native int)
  // 调用TestDelegate/NumberChanger构造函数，栈中有V_0和AddNum函数指针
  // 注：native int 是一种平台特定的整数类型，用于表示指针或句柄的值。它的大小和符号
取决于所
  //     运行的平台（32 位或 64 位）。
  IL_0013:  stloc.1
  // store locals 1: 将函数返回值存储至V_1
  IL_0014:  ldloc.0
  // load locals 0 V_0 是TestDelegate实例，将V_0 push到参数栈中
  IL_0015:  ldftn      instance int32 DelegateAppl.TestDelegate::MultNum(int32)
  // load function 将函数指针push到参数栈中
  IL_001b:  newobj     instance void
DelegateAppl.TestDelegate/NumberChanger::.ctor
                       (object, native int)
  // 调用TestDelegate/NumberChanger构造函数，栈中有V_0和MultNum函数指针
  IL_0020:  stloc.2
  // store locals 2: 将函数返回值存储至V_2
  IL_0021:  ldloc.1
  IL_0022:  ldc.i4.s   25
  IL_0024:  callvirt   instance int32
                       DelegateAppl.TestDelegate/NumberChanger::Invoke(int32)
  IL_0029:  pop
  IL_002a:  ldstr      "Value of Num: {0}"
  IL_002f:  ldloc.0
  IL_0030:  callvirt   instance int32 DelegateAppl.TestDelegate::getNum()
  IL_0035:  box        [System.Runtime]System.Int32
  IL_003a:  call       void [System.Console]System.Console::WriteLine(string,
                                                                       object)
  IL_003f:  nop
  IL_0040:  ldloc.2
  IL_0041:  ldc.i4.5
  IL_0042:  callvirt   instance int32
                       DelegateAppl.TestDelegate/NumberChanger::Invoke(int32)
  IL_0047:  pop
  IL_0048:  ldstr      "Value of Num: {0}"
```

```
  IL_004d:  ldloc.0
  IL_004e:  callvirt   instance int32 DelegateAppl.TestDelegate::getNum()
  IL_0053:  box        [System.Runtime]System.Int32
  IL_0058:  call       void [System.Console]System.Console::WriteLine(string,
                                                                       object)
  IL_005d:  nop
  IL_005e:  call       valuetype [System.Console]System.ConsoleKeyInfo
[System.Console]System.Console::ReadKey()
  IL_0063:  pop
  IL_0064:  ret
} // end of method TestDelegate::Main
```

3.2 指向静态函数的委托

C#代码

```
using System;
delegate int NumberChanger(int n);
namespace DelegateAppl
{
    class TestDelegate
    {
        static int num = 10;
        public static int AddNum(int p)
        {
            num += p;
            return num;
        }
        public static int MultNum(int q)
        {
            num *= q;
            return num;
        }
        public static int getNum()
        {
            return num;
        }
        static void Main(string[] args)
        {
            NumberChanger nc1 = new NumberChanger(AddNum);
```

```
            NumberChanger nc2 = new NumberChanger(MultNum);
            nc1(25);
            Console.WriteLine("Value of Num: {0}", getNum());
            nc2(5);
            Console.WriteLine("Value of Num: {0}", getNum());
            Console.ReadKey();
        }
    }
}
```

IL中间代码

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  .custom instance void
System.Runtime.CompilerServices.NullableContextAttribute::.ctor(uint8) = ( 01
00 01 00 00 )
  // Code size       93 (0x5d)
  .maxstack  2
  .locals init (class NumberChanger V_0,
           class NumberChanger V_1)
  IL_0000:  nop
  IL_0001:  ldnull
  // 差异
  IL_0002:  ldftn      int32 DelegateAppl.TestDelegate::AddNum(int32)
  IL_0008:  newobj     instance void NumberChanger::.ctor(object,
                                                          native int)
  IL_000d:  stloc.0
  IL_000e:  ldnull
  // 差异
  IL_000f:  ldftn      int32 DelegateAppl.TestDelegate::MultNum(int32)
  IL_0015:  newobj     instance void NumberChanger::.ctor(object,
                                                          native int)
  IL_001a:  stloc.1
  IL_001b:  ldloc.0
  IL_001c:  ldc.i4.s   25
  IL_001e:  callvirt   instance int32 NumberChanger::Invoke(int32)
  IL_0023:  pop
  IL_0024:  ldstr      "Value of Num: {0}"
  IL_0029:  call       int32 DelegateAppl.TestDelegate::getNum()
  IL_002e:  box        [System.Runtime]System.Int32
  IL_0033:  call       void [System.Console]System.Console::WriteLine(string,
```

```
                                                                    object)
  IL_0038:  nop
  IL_0039:  ldloc.1
  IL_003a:  ldc.i4.5
  IL_003b:  callvirt   instance int32 NumberChanger::Invoke(int32)
  IL_0040:  pop
  IL_0041:  ldstr      "Value of Num: {0}"
  IL_0046:  call       int32 DelegateAppl.TestDelegate::getNum()
  IL_004b:  box        [System.Runtime]System.Int32
  IL_0050:  call       void [System.Console]System.Console::WriteLine(string,
                                                                       object)
  IL_0055:  nop
  IL_0056:  call       valuetype [System.Console]System.ConsoleKeyInfo
[System.Console]System.Console::ReadKey()
  IL_005b:  pop
  IL_005c:  ret
} // end of method TestDelegate::Main
```

3.3 C++ std::function

```cpp
#include <iostream>
#include <functional>

class MyClass
{
public:
    void PrintMessage(const std::string& message)
    {
        std::cout << "Message: " << message << std::endl;
    }
};

int main()
{
    MyClass obj;

    std::function<void(MyClass&, const std::string&)> func =
&MyClass::PrintMessage;
    func(obj, "Hello, world!");

    return 0;
}
```

3.4 委托总结

1. 委托实质上就是对函数进行封装，为了保证通用性，封装的第一个参数是object类型，第二个参数是native int（函数指针）；
2. 使用静态方法初始化委托时委托参数object为null，使用非静态方法初始化委托时委托构造object指向对应函数的类实例化；
3. C++中的std::function也可以指向所有的函数，但是C++的类没有共同基类，导致std::function初始化时需要指定具体类名；
4. 上述IL中间代码可以看出，委托有ctor构造函数，所以委托本质是一个类，因此类可以申明在哪里，委托就可以申明在哪里。

4. 动态库调用

4.1 使用方法

```
[DllImport("test.dll",
           EntryPoint = "?test@@YAHH@Z",
           CharSet = CharSet.Ansi,
           CallingConvention = CallingConvention.Cdecl)]
```

4.2 EntryPoint属性

实践：

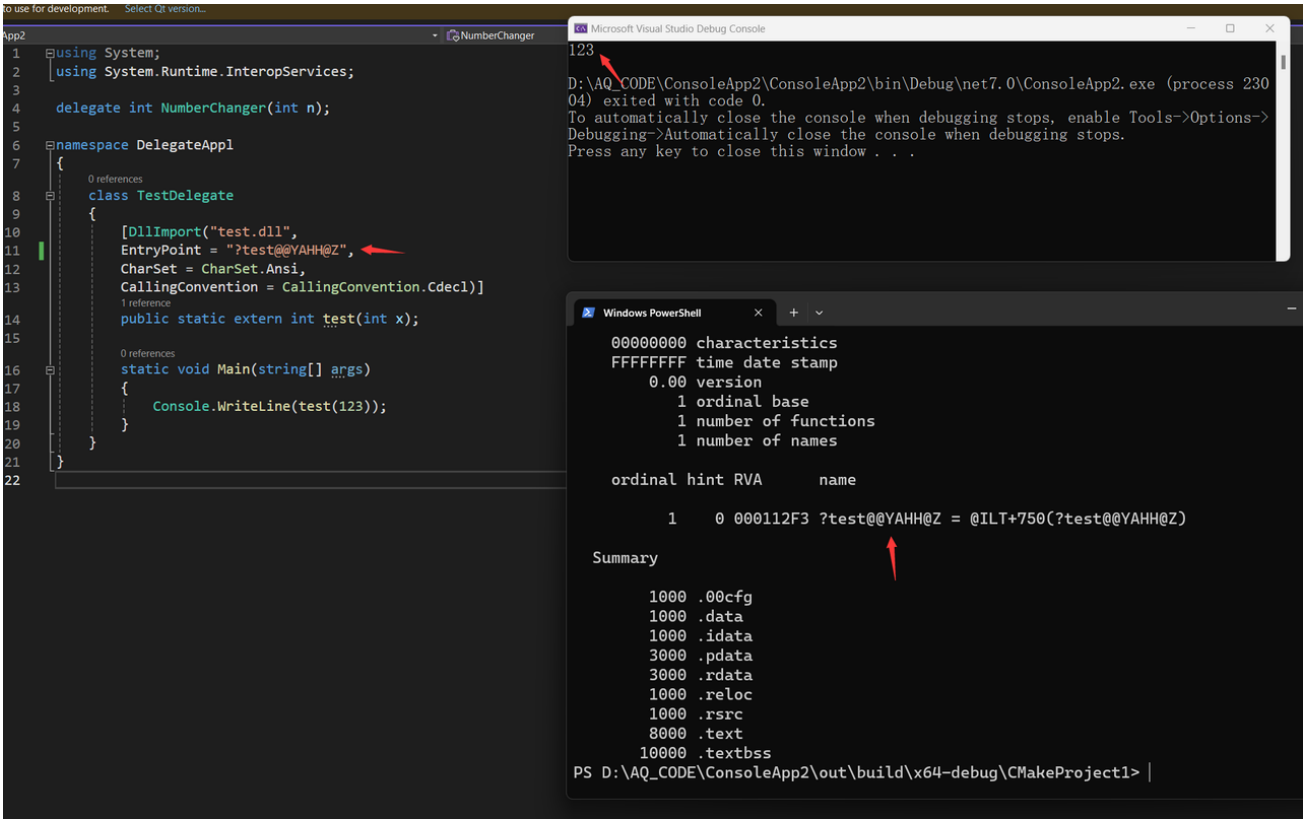


图4.1.1 通过符号调用C++接口

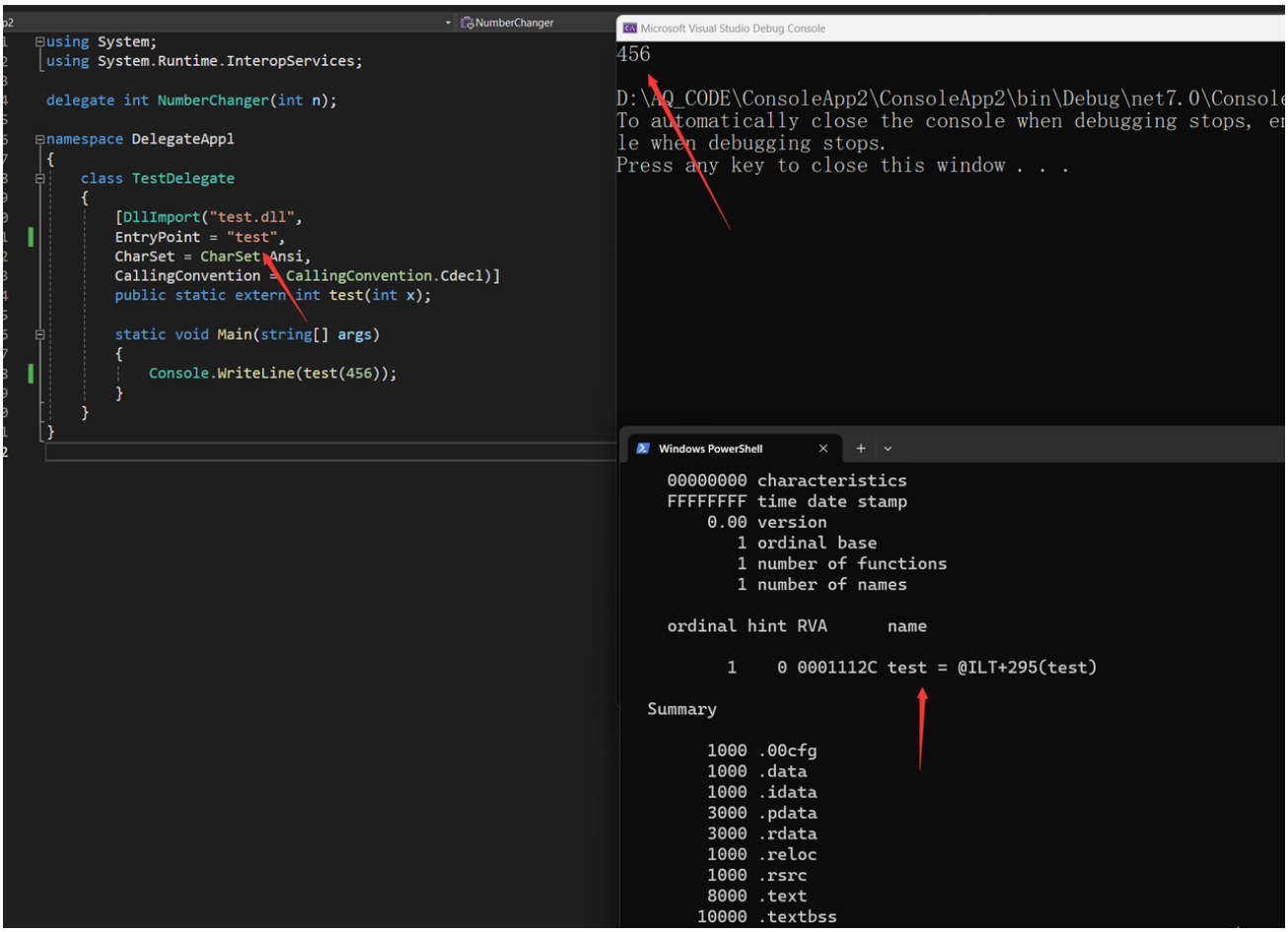


图4.1.2 通过符号调用C接口

结论：DllImport中的EntryPoint属性值的是动态库中的符号；调用C++接口通过不会考虑这种方法，因为编译器、构建系统等因素会导致C++同一份代码生成的符号不同，所以在调用C++接口时可以将C++接口使用C语言封装后进行调用，C语言的符号和函数名一致，不存在上述不一致的问题。

4.3 CharSet属性

控制名称重整以及将字符串参数封送到函数中的方式。 默认值为 CharSet.Ansi。

4.4 CallingConvention属性

CallingConvention 枚举 (System.Runtime.InteropServices) | Microsoft Learn

在C#中，调用约定（Calling Convention）是指定如何在编译时和运行时处理函数调用的一组规则。C#通过DllImport属性来与外部的非托管代码进行交互，并可以明确指定调用约定。常见的调用约定有以下几种：

1. **WinAPI调用约定**：
   - 默认情况下，DllImport使用的调用约定是StdCall。这是Windows API函数的标准调用约定，因此在使用DllImport导入Windows API函数时，通常不需要显式指定调用约定。
2. **Cdecl调用约定**：
   - 在DllImport属性中指定CallingConvention参数为CallingConvention.Cdecl，使用C语言调用约定。使用此约定时，调用者清理堆栈。
3. **StdCall调用约定**：
   - 在DllImport属性中指定CallingConvention参数为CallingConvention.StdCall，使用标准调用约定。使用此约定时，被调用者清理堆栈。
4. **FastCall调用约定**：
   - 在DllImport属性中指定CallingConvention参数为CallingConvention.FastCall，使用快速调用约定。

在DllImport属性中，可以显式指定调用约定，例如：

csharp Copy Code

```
[DllImport("MyDll.dll", CallingConvention = CallingConvention.Cdecl)]
public static extern void MyFunction();
```

通过在DllImport属性中指定调用约定，可以确保与非托管代码进行交互时采用正确的调用约定，从而保证函数调用的正确性。

图4.1.3 调用约定

4.5 符号表

实践：

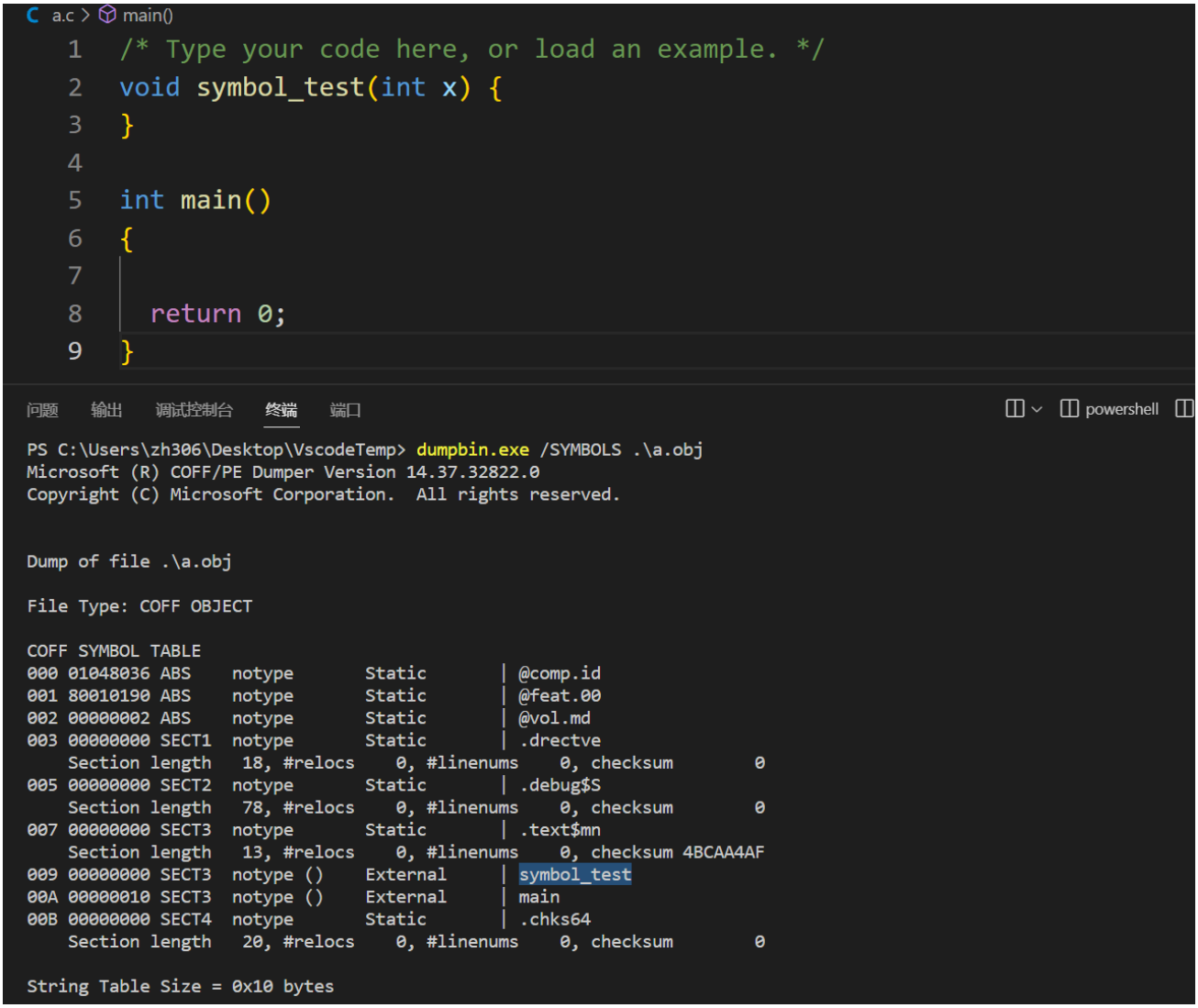

a.c > main()
1 /* Type your code here, or load an example. */
2 void symbol_test(int x) {
3 }
4
5 int main()
6 {
7
8 return 0;
9 }
问题 输出 调试控制台 终端 端口
powershell
PS C:\Users\zh306\Desktop\VscodeTemp> dumpbin.exe /SYMBOLS .\a.obj
Microsoft (R) COFF/PE Dumper Version 14.37.32822.0
Copyright (C) Microsoft Corporation. All rights reserved.
Dump of file .\a.obj
File Type: COFF OBJECT
COFF SYMBOL TABLE
000 01048036 ABS notype Static | @comp.id
001 80010190 ABS notype Static | @feat.00
002 00000002 ABS notype Static | @vol.md
003 00000000 SECT1 notype Static | .drectve
Section length 18, #relocs 0, #linenums 0, checksum 0
005 00000000 SECT2 notype Static | .debug$S
Section length 78, #relocs 0, #linenums 0, checksum 0
007 00000000 SECT3 notype Static | .text$mn
Section length 13, #relocs 0, #linenums 0, checksum 4BCAA4AF
009 00000000 SECT3 notype () External | symbol_test
00A 00000010 SECT3 notype () External | main
00B 00000000 SECT4 notype Static | .chks64
Section length 20, #relocs 0, #linenums 0, checksum 0
String Table Size = 0x10 bytes

图4.1.3 C 符号表

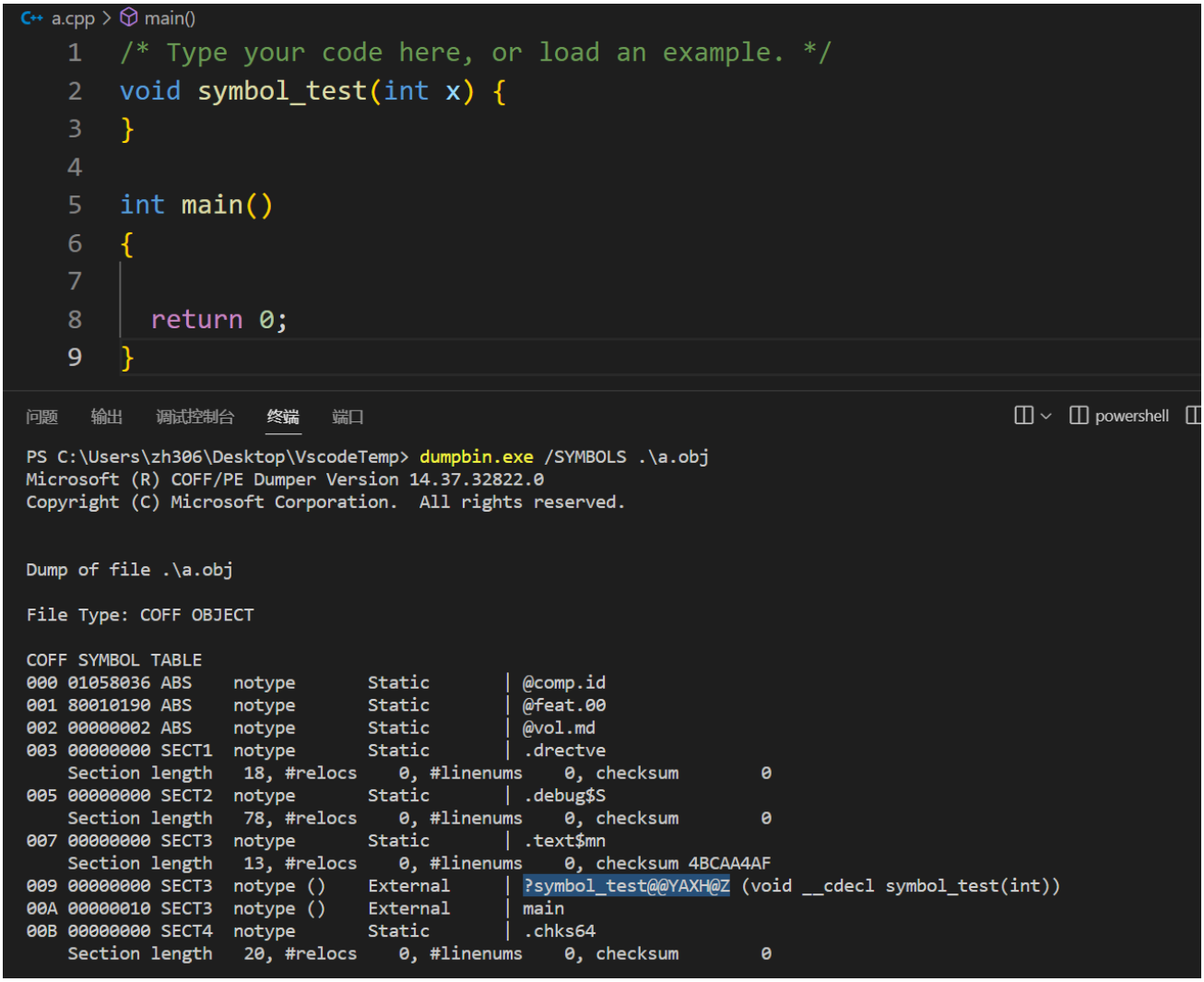

a.cpp > main()
1 /* Type your code here, or load an example. */
2 void symbol_test(int x) {
3 }
4
5 int main()
6 {
7
8 return 0;
9 }
问题 输出 调试控制台 终端 端口
powershell
PS C:\Users\zh306\Desktop\VscodeTemp> dumpbin.exe /SYMBOLS .\a.obj
Microsoft (R) COFF/PE Dumper Version 14.37.32822.0
Copyright (C) Microsoft Corporation. All rights reserved.
Dump of file .\a.obj
File Type: COFF OBJECT
COFF SYMBOL TABLE
000 01058036 ABS notype Static | @comp.id
001 80010190 ABS notype Static | @feat.00
002 00000002 ABS notype Static | @vol.md
003 00000000 SECT1 notype Static | .drectve
Section length 18, #relocs 0, #linenums 0, checksum 0
005 00000000 SECT2 notype Static | .debug$S
Section length 78, #relocs 0, #linenums 0, checksum 0
007 00000000 SECT3 notype Static | .text$mn
Section length 13, #relocs 0, #linenums 0, checksum 4BCAA4AF
009 00000000 SECT3 notype () External | ?symbol_test@@YAXH@Z (void __cdecl symbol_test(int))
00A 00000010 SECT3 notype () External | main
00B 00000000 SECT4 notype Static | .chks64
Section length 20, #relocs 0, #linenums 0, checksum 0

图4.1.4 C++ 符号表

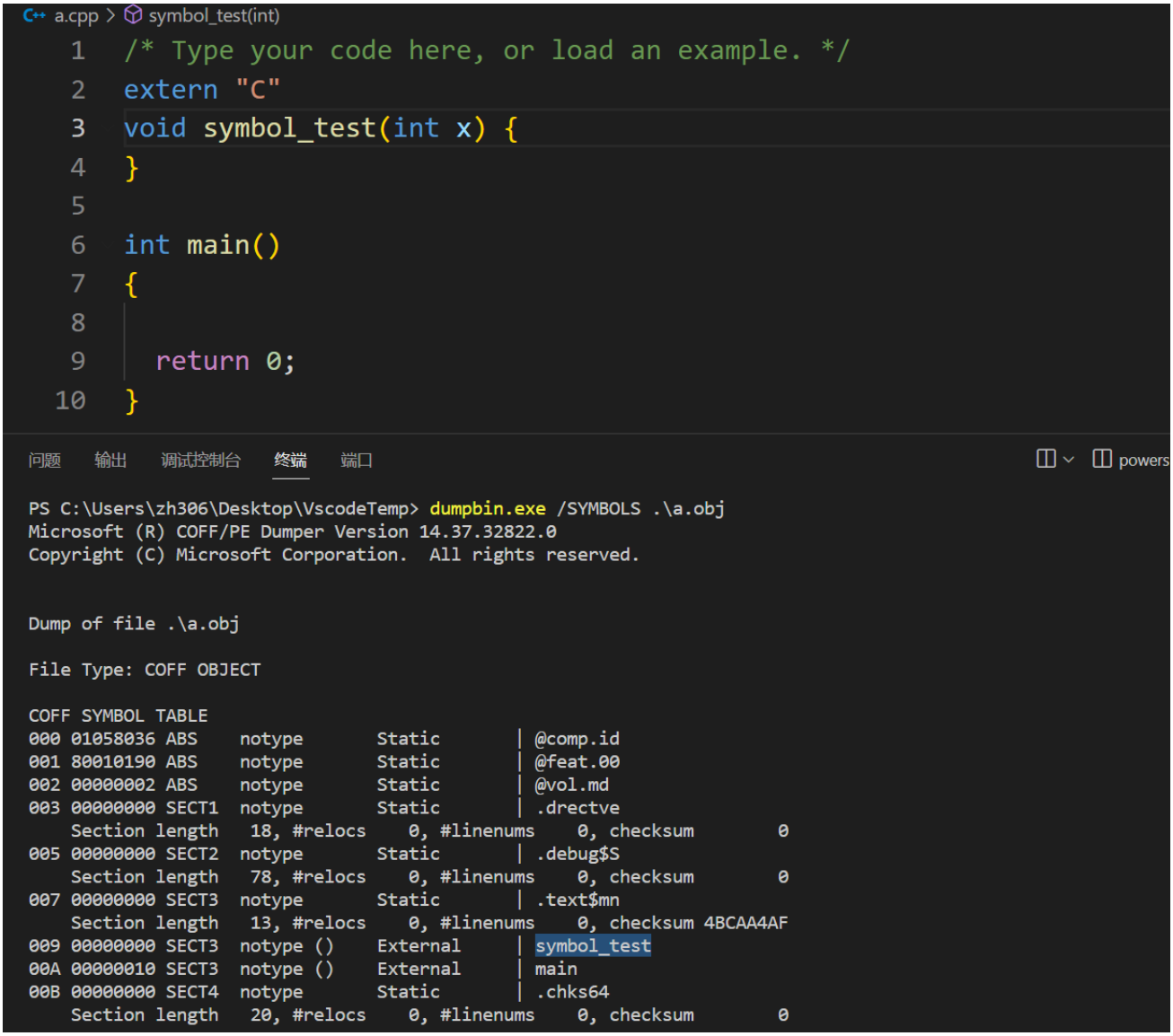


图4.1.5 C++ 中 extern "C" 符号表

结论：C++为了实现静态多态，生成的符号会待用类、参数、模板、命名空间等信息，C语言符号与函数一致，所以在调用C语言动态库时可以方便找到函数对应的符号。

4.6 浅谈SWIG

实践：

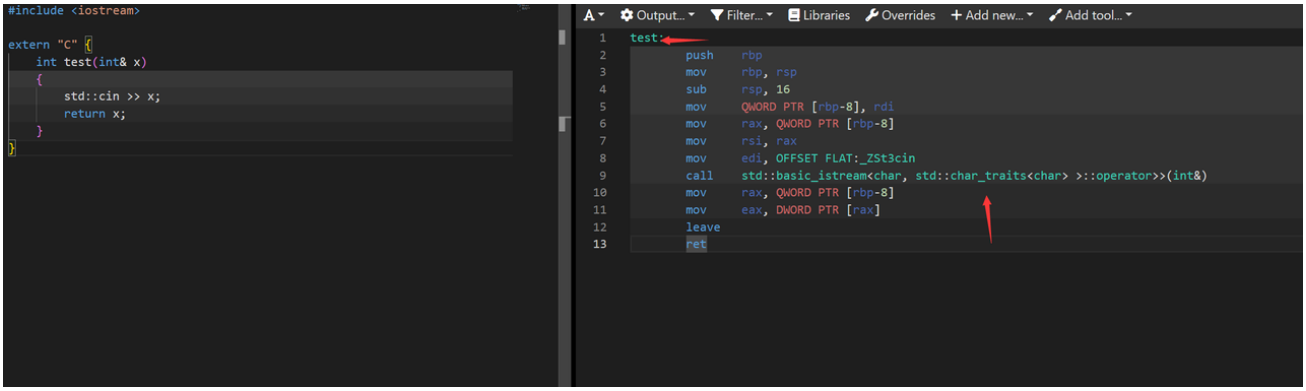


图4.1.6 extern "C"原理

结论：extern "C"内部定义的符号统一使用C语言的方式编译，但是可以在内部使用C++内容。

实践：

```
#ifdef __cplusplus
extern "C" {
#endif

SWIGEXPORT void * SWIGSTDCALL CSharp_new_AQParamPanelBaseCallBack() {
  void * jresult ;
  AQParamPanelBaseCallBack *result = 0 ;

  result = (AQParamPanelBaseCallBack *)new SwigDirector_AQParamPanelBaseCallBack();
  jresult = (void *)result;
  return jresult;
}
```

图4.1.7 SWIG包装类构造函数

```
SWIGEXPORT void SWIGSTDCALL CSharp_AQParamPanelInterface_show(void * jarg1) {
  AQParamPanelInterface *arg1 = (AQParamPanelInterface *) 0 ;

  arg1 = (AQParamPanelInterface *)jarg1;
  (arg1)->show();
}
```

图4.1.8 SWIG包装类普通函数

```
[global::System.Runtime.InteropServices.DllImport("csharp_aqparampanel", EntryPoint="CSharp_new_AQParamPanelBaseCallBack")]
public static extern global::System.IntPtr new_AQParamPanelBaseCallBack();
```

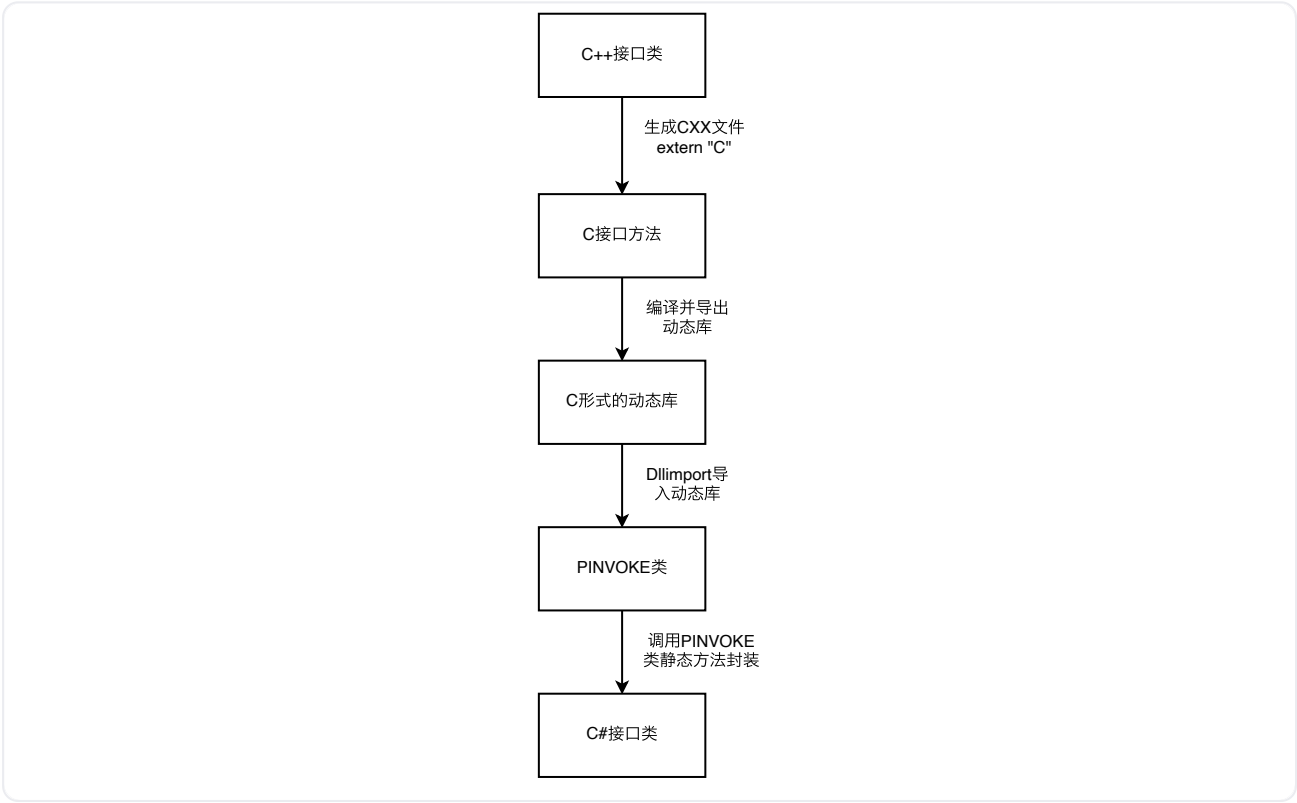
图4.1.9 SWIG C#导入格式

```
  }

  public AQParamPanelBaseCallBack() : this(csharp_aqparampanelPINVOKE.new_AQParamPanelBaseCallBack(), true) {
    SwigDirectorConnect();
  }

  public virtual void test_param_changed() {
```

图4.1.10 SWIG C#接口类

结论：

1. SWIG首先将C++代码使用extern "C"的方式对C++类进行封装并导出符号；
2. 将第一步生成的CXX文件编译进动态库中，利用Dllimport将C接口声明成静态接口；
3. 将静态接口封装为C#接口形式。



[SWIG] SWIG原理（以C#为例） swig c#-CSDN博客

5. 多线程

C# Task详解 - 五维思考 - 博客园 (cnblogs.com)

6. WPF

WPF中文网 - 从小白到大佬 (wpfsoft.com)

WPF起源

Dispatcher更新UI

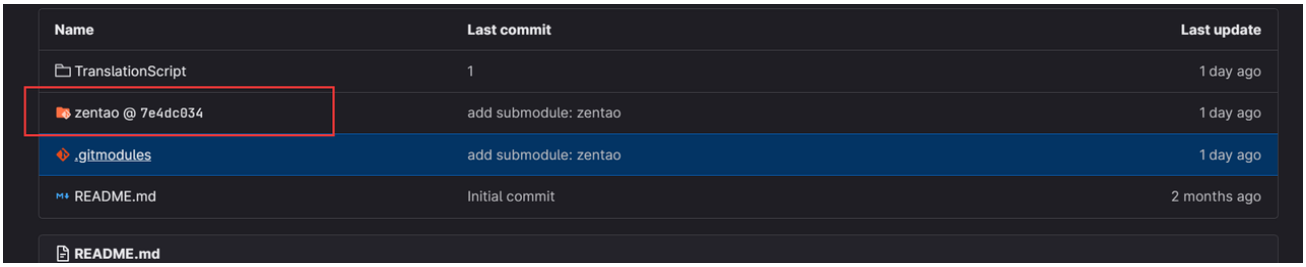
数据驱动

什么是数据绑定？ - WPF中文网 - 从小白到大佬 (wpfsoft.com)

什么是路由事件

7. Git子仓库

https://blog.csdn.net/whuzhang16/article/details/120182063




```
git submodule add [remote_url]
```

流程：

1. 父仓库创建issue并创建对应分支
2. 本地父仓库代码切换对应远程仓库
3. 当涉及子仓库修改时，对应远程子仓库创建issue，本地同步分支
4. 子仓库修改完毕提交代码
5. 父仓库提示.gitmodule文件有修改，一并提交

开源仓库：

https://github.com/DrKLO/Telegram

https://github.com/dotnetcore/SmartSql

https://github.com/qian-o/Dimension

https://github.com/telegramdesktop/tdesktop

https://github.com/shadowsocks/shadowsocks-windows

https://github.com/microsoft/PowerToys