反射是一种允许用户获得类型信息的功能，允许学习和使用只在运行时才能知道的类型的功能。

4种关键的反射技术：

获取类型支持的方法列表：MethodInfo[] GetMethods()

例：程序reflection\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 反射

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace reflection\_test1

{

class MyClass

{

int x;

int y;

public MyClass(int i, int j)

{

x = i;

y = j;

}

public int Sum()

{

return x + y;

}

public bool IsBetween(int i)

{

if (x < i && i < y)

{

return true;

}

return false;

}

public void Set(int a, int b)

{

x = a;

y = b;

}

public void Show()

{

Console.WriteLine("x : {0}, y : {1}", x, y);

}

}

class ReflectionTest

{

static void Main(string[] args)

{

Type t = typeof(MyClass);

Console.WriteLine("Analyzing methods in " + t.Name);

Console.WriteLine();

Console.WriteLine("Methods supported: ");

MethodInfo[] method\_info = t.GetMethods();

// 被MyClass支持的方法

// 包含自身的所有方法以及继承的公有非静态方法

// 不包含构造函数和析构函数

foreach (MethodInfo m in method\_info)

{

Console.Write(" " + m.ReturnType.Name + " " + m.Name + " (");

// 方法参数

ParameterInfo[] parameter\_info = m.GetParameters();

for (int i = 0; i < parameter\_info.Length; ++i )

{

Console.Write(parameter\_info[i].ParameterType.Name + " " + parameter\_info[i].Name);

if (i + 1 < parameter\_info.Length)

{

Console.Write(", ");

}

}

//foreach (ParameterInfo p in parameter\_info)

//{

// Console.Write(p.ParameterType.Name + " " + p.Name);

// Console.Write(", ");

//}

Console.WriteLine(")");

}

}

}

}

输出结果：

Analyzing methods in MyClass

Methods supported:

Int32 Sum ()

Boolean IsBetween (Int32 i)

Void Set (Int32 a, Int32 b)

Void Show ()

String ToString ()

Boolean Equals (Object obj)

Int32 GetHashCode ()

Type GetType ()

获取指定的方法：

例：程序reflection\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 反射：获取方法

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace reflection\_test2

{

class MyClass

{

int x;

int y;

public MyClass(int i, int j)

{

x = i;

y = j;

}

public int Sum()

{

return x + y;

}

public bool IsBetween(int i)

{

if (x < i && i < y)

{

return true;

}

return false;

}

public void Set(int a, int b)

{

x = a;

y = b;

}

public void Show()

{

Console.WriteLine("x : {0}, y : {1}", x, y);

}

private void PrivateMethod()

{

Console.WriteLine("Private method.");

}

}

class ReflectionTest

{

static void Main(string[] args)

{

Type t = typeof(MyClass);

Console.WriteLine("Methods supported: ");

// 获取类自定义的公共的实例方法

// 这里使用|运算符连接更改的标记

// 不能使用&运算符

MethodInfo[] method\_info = t.GetMethods(BindingFlags.DeclaredOnly

| BindingFlags.Instance

| BindingFlags.Public);

// 被MyClass支持的方法

// 包含自身的所有方法以及继承的公有非静态方法

// 不包含构造函数和析构函数

foreach (MethodInfo m in method\_info)

{

Console.Write(" " + m.ReturnType.Name + " " + m.Name + " (");

// 方法参数

ParameterInfo[] parameter\_info = m.GetParameters();

for (int i = 0; i < parameter\_info.Length; ++i)

{

Console.Write(parameter\_info[i].ParameterType.Name + " " + parameter\_info[i].Name);

if (i + 1 < parameter\_info.Length)

{

Console.Write(", ");

}

}

//foreach (ParameterInfo p in parameter\_info)

//{

// Console.Write(p.ParameterType.Name + " " + p.Name);

// Console.Write(", ");

//}

Console.WriteLine(")");

}

}

}

}

输出结果：

Methods supported:

Int32 Sum ()

Boolean IsBetween (Int32 i)

Void Set (Int32 a, Int32 b)

Void Show ()

使用反射调用方法：

使用MethodInfo类的Invoke()方法：

object Invoke(object obj, object[] parameters)

其中，obj是一个对象引用，将调用它所指向的对象的方法，对于static方法，obj必须为null。传递给方法的参数必须在parameters中，parameters的长度必须等于参数数量。如果不需要传递参数，parameters必须为null。

例：程序reflection\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 使用反射调用方法

using System;

using System.Collections.Generic;

using System.Linq;

using System.Reflection;

using System.Text;

using System.Threading.Tasks;

namespace reflection\_test3

{

class MyClass

{

int x;

int y;

public MyClass(int i, int j)

{

x = i;

y = j;

}

public int Sum()

{

return x + y;

}

public bool IsBetween(int i)

{

if (x < i && i < y)

{

return true;

}

return false;

}

public void Set(int a, int b)

{

Console.WriteLine("Inside Set(int, int).");

x = a;

y = b;

Show();

}

// 重载Set

public void Set(double a, double b)

{

Console.WriteLine("Inside Set(double, double).");

x = (int)a;

y = (int)b;

Show();

}

public void Show()

{

Console.WriteLine("x : {0}, y : {1}", x, y);

}

}

class ReflectionTest

{

static void Main(string[] args)

{

Type t = typeof(MyClass);

MyClass reflect\_obj = new MyClass(10, 20);

int val;

Console.WriteLine("Invoking methods in " + t.Name);

Console.WriteLine();

MethodInfo[] method\_info = t.GetMethods();

foreach (MethodInfo m in method\_info)

{

ParameterInfo[] param\_info = m.GetParameters();

if (m.Name.Equals("Set", StringComparison.Ordinal)

&& param\_info[0].ParameterType == typeof(int))

{

object[] param\_args = new object[param\_info.Length];

param\_args[0] = 9;

param\_args[1] = 18;

m.Invoke(reflect\_obj, param\_args);

} else if (m.Name.Equals("Set", StringComparison.Ordinal)

&& param\_info[0].ParameterType == typeof(double))

{

object[] param\_args = new object[param\_info.Length];

param\_args[0] = 1.12;

param\_args[1] = 23.4;

m.Invoke(reflect\_obj, param\_args);

}

else if (m.Name.Equals("Sum", StringComparison.Ordinal))

{

val = (int)m.Invoke(reflect\_obj, null);

Console.WriteLine("sum is " + val);

}

}

}

}

}

输出结果为：

Invoking methods in MyClass

sum is 30

Inside Set(int, int).

x : 9, y : 18

Inside Set(double, double).

x : 1, y : 23

采用反射技术获取类的方法的主要用途是在对象是在运行时创建。

全自动类型查询:可以从程序集中获取类的声明,然后通过反射使用一个类型.

例:程序reflection\_test4

// Copyright 2016.刘珅珅

// author：刘珅珅

// 全自动类型查询

using *System*;

using *System*.*Collections*.*Generic*;

using *System*.*Linq*;

using *System*.*Reflection*;

using *System*.*Text*;

using *System*.*Threading*.*Tasks*;

namespace reflection\_test4

{

class ReflectionTest

{

static void Main(string[] args)

{

*Assembly* asm = *Assembly*.*LoadFrom*("class\_test3.exe");

*Type*[] all\_types = asm.*GetTypes*();

foreach (*Type* temp in all\_types)

{

*Console*.*WriteLine*("Found: " + temp.*Name*);

}

// 使用第一个类

*Type* t = all\_types[0];

*Console*.*WriteLine*("Using: " + t.*Name*);

// 获取构造函数

*ConstructorInfo*[] constructor\_info = t.*GetConstructors*();

// 使用第一个构造函数

*ParameterInfo*[] param\_info = constructor\_info[0].*GetParameters*();

object reflect\_obj;

if (param\_info.*Length* > 0)

{

object[] con\_args = new object[param\_info.*Length*];

// 初始化构造参数

for (int n = 0; n < param\_info.*Length*; ++n)

{

con\_args[n] = 10 + n \* 20;

}

reflect\_obj = constructor\_info[0].*Invoke*(con\_args);

}

else

{

reflect\_obj = constructor\_info[0].*Invoke*(null);

}

*Console*.*WriteLine*("\nInvoking methods on reflect\_obj");

*Console*.*WriteLine*();

// 忽略继承的方法

*MethodInfo*[] method\_info = t.*GetMethods*(*BindingFlags*.*DeclaredOnly*

| *BindingFlags*.*Instance*

| *BindingFlags*.*Public*);

foreach (*MethodInfo* m in method\_info)

{

*Console*.*WriteLine*("Calling {0} ", m.*Name*);

*ParameterInfo*[] p\_info = m.*GetParameters*();

switch (p\_info.*Length*)

{

case 0: // 无参

if (m.*ReturnType* == typeof(void))

{

m.*Invoke*(reflect\_obj, null);

}

break;

default:

break;

}

}

*Console*.*WriteLine*();

}

}

}

输出结果为：

Found: MyClass

Found: ClassTest

Using: MyClass

Invoking methods on reflect\_obj

Calling Change

Calling Show

a is 10 b is 30