在C#中，委托是一个能够引用方法的对象，与C/C++中的函数指针类似。

创建委托，实际创建的是一个存储方法引用的对象。

委托调用的方法不是在编译时确定的，而是在运行时确定的。

委托隐含sealed属性，不能被继承，也不可以指定为abstract，virtual.

所有的委托都隐式地派生于System.Delegate类。

委托可调用与声明的签名（参数列表）和返回类型一致的任何方法，既可以是实例方法也可以是static方法。

static方法的委托

例：程序delegate\_test1

// Copyright 2016.刘珅珅

// author：刘珅珅

// 委托

using *System*;

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

using *System*.*Linq*;

using *System*.*Text*;

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

namespace delegate\_test1

{

delegate string StringMethod(string str);

class DelegateTest

{

static string ReplaceSpaces(string s)

{

*Console*.*WriteLine*("Replacing spaces with hyphens.");

return s.*Replace*(' ', '-');

}

static string RemoveSpaces(string s)

{

string temp = "";

*Console*.*WriteLine*("Removing spaces.");

for (int i = 0; i < s.*Length*; ++i)

{

if (s[i] != ' ')

{

temp += s[i];

}

}

return temp;

}

static string Reverse(string s)

{

string temp = "";

*Console*.*WriteLine*("Reversing string.");

for (int j = 0, i = s.*Length* - 1; i >= 0; --i, ++j )

{

temp += s[i];

}

return temp;

}

static void Main(string[] args)

{

string str;

StringMethod str\_method = new StringMethod(ReplaceSpaces);

str = str\_method("This is a test.");

*Console*.*WriteLine*("Replace result: " + str);

*Console*.*WriteLine*();

str\_method = new StringMethod(RemoveSpaces);

str = str\_method("This is a test.");

*Console*.*WriteLine*("Remove result: " + str);

*Console*.*WriteLine*();

str\_method = new StringMethod(Reverse);

str = str\_method("This is a test.");

*Console*.*WriteLine*("Reverse result: " + str);

}

}

}

输出结果：

Replacing spaces with hyphens.

Replace result: This-is-a-test.

Removing spaces.

Remove result: Thisisatest.

Reversing string.

Reverse result: .tset a si sihT

委托的方法组转换：在为委托指定方法时，可以只指定方法名称，而不需要使用new关键字。

实例方法的委托：

例：程序delegate\_test2

// Copyright 2016.刘珅珅

// author：刘珅珅

// 委托：实例方法

using *System*;

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

using *System*.*Linq*;

using *System*.*Text*;

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

namespace delegate\_test2

{

delegate string StringMethod(string str);

class StringOps

{

public string ReplaceSpaces(string s)

{

*Console*.*WriteLine*("Replacing spaces with hyphens.");

return s.*Replace*(' ', '-');

}

public string RemoveSpaces(string s)

{

string temp = "";

*Console*.*WriteLine*("Removing spaces.");

for (int i = 0; i < s.*Length*; ++i)

{

if (s[i] != ' ')

{

temp += s[i];

}

}

return temp;

}

public string Reverse(string s)

{

string temp = "";

*Console*.*WriteLine*("Reversing string.");

for (int j = 0, i = s.*Length* - 1; i >= 0; --i, ++j)

{

temp += s[i];

}

return temp;

}

}

class DelegateTest

{

static void Main(string[] args)

{

StringOps obj = new StringOps();

string str;

// 委托的方法组转换

StringMethod str\_method = obj.ReplaceSpaces;

str = str\_method("This is a test.");

*Console*.*WriteLine*("Replace result: " + str);

*Console*.*WriteLine*();

str\_method = obj.RemoveSpaces;

str = str\_method("This is a test.");

*Console*.*WriteLine*("Remove result: " + str);

*Console*.*WriteLine*();

str\_method = obj.Reverse;

str = str\_method("This is a test.");

*Console*.*WriteLine*("Reverse result: " + str);

}

}

}

输出结果：

Replacing spaces with hyphens.

Replace result: This-is-a-test.

Removing spaces.

Remove result: Thisisatest.

Reversing string.

Reverse result: .tset a si sihT

多播委托：允许创建方法的调用链表，调用多播委托，所有的方法均会执行。

实例化一个委托，使用“+”或“+=”添加方法，使用“－”或“－=”移除方法。如果委托有返回值，则将列表中最后一个方法的返回值用做整个委托的返回值。

例：程序delegate\_test3

// Copyright 2016.刘珅珅

// author：刘珅珅

// 多播委托

using *System*;

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

using *System*.*Linq*;

using *System*.*Text*;

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

namespace delegate\_test3

{

delegate void StringMethod(ref string s);

class StringOps

{

public void ReplaceSpaces(ref string s)

{

*Console*.*WriteLine*("Replacing spaces with hyphens.");

s = s.*Replace*(' ', '-');

}

public void RemoveSpaces(ref string s)

{

string temp = "";

*Console*.*WriteLine*("Removing spaces.");

for (int i = 0; i < s.*Length*; ++i)

{

if (s[i] != ' ')

{

temp += s[i];

}

}

s = temp;

}

public void Reverse(ref string s)

{

string temp = "";

*Console*.*WriteLine*("Reversing string.");

for (int j = 0, i = s.*Length* - 1; i >= 0; --i, ++j)

{

temp += s[i];

}

s = temp;

}

}

class DelegateTest

{

static void Main(string[] args)

{

string str = "This is a test.";

StringOps obj = new StringOps();

StringMethod str\_method;

StringMethod replace = obj.ReplaceSpaces;

StringMethod remove = obj.RemoveSpaces;

StringMethod reverse = obj.Reverse;

// 多播委托

str\_method = replace;

str\_method += reverse;

// 先执行Replace，再执行Reverse

str\_method(ref str);

*Console*.*WriteLine*("Resulting string: " + str);

*Console*.*WriteLine*();

// 移除Replace

str\_method -= replace;

// 增加Remove

str\_method += remove;

str = "This is a test.";

// 执行Reverse，再执行Remove

str\_method(ref str);

*Console*.*WriteLine*("Resulting string: " + str);

*Console*.*WriteLine*();

}

}

}

输出结果：

Replacing spaces with hyphens.

Reversing string.

Resulting string: .tset-a-si-sihT

Reversing string.

Removing spaces.

Resulting string: .tsetasisihT

协变和逆变：增加委托的灵活性。

一般情况下，传递给委托的方法必须具有相同的参数类型和返回类型，通过协变和逆变可以增加灵活性。

协变：只要方法的返回类型派生于委托声明的返回类型所对应的类，就可以赋给委托。

逆变：只有方法的类型是委托声明中指定参数类型的基类，该方法就可以赋给委托。

例：程序delegate\_test4

// Copyright 2016.刘珅珅

// author：刘珅珅

// 协变与逆变

using *System*;

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

using *System*.*Linq*;

using *System*.*Text*;

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

namespace delegate\_test4

{

class Base

{

public int val;

}

class Derived : Base { }

delegate Base ChangeIt(Derived obj);

class CoContraVariance

{

public Base IncrA(Base obj)

{

Base temp = new Base();

temp.val = obj.val + 1;

return temp;

}

public Derived IncrB(Derived obj)

{

Derived temp = new Derived();

temp.val = obj.val + 1;

return temp;

}

}

class DelegateTest

{

static void Main(string[] args)

{

Derived d = new Derived();

CoContraVariance obj = new CoContraVariance();

// 逆变：方法参数类型是委托参数类型的基类

ChangeIt delegate\_it = obj.IncrA;

Base b = delegate\_it(d);

*Console*.*WriteLine*("Base: " + b.val);

// 协变：方法返回类型是委托返回类型的派生类

delegate\_it = obj.IncrB;

d = (Derived)delegate\_it(d);

*Console*.*WriteLine*("Derived: " + d.val);

}

}

}