Ковариантность и контравариантность применяется, главным образом, для рационального разрешения

особых ситуаций, возникающих в связи с применением обобщенных интерфейсов

и делегатов, определенных в среде .NET Framework. И поэтому некоторые интерфейсы

и делегаты, определенные в библиотеке, были обновлены, чтобы использовать

ковариантность и контравариантность параметров типа. Разумеется, преимуществами

ковариантности и контравариантности можно также воспользоваться в интерфейсах и

делегатах, создаваемых собственными силами. В этом разделе механизмы ковариантности

и контравариантности параметров типа поясняются на конкретных примерах.

**Применение ковариантности в обобщенном интерфейсе out**

Применительно к обобщенному интерфейсу ковариантность служит средством,

разрешающим методу возвращать тип, производный от класса, указанного в параметре

типа. В прошлом возвращаемый тип должен был в точности соответствовать

параметру типа в силу строгой проверки обобщений на соответствие типов. Ковариантность

смягчает это строгое правило таким образом, чтобы обеспечить типовую

безопасность. Параметр ковариантного типа объявляется с помощью ключевого слова

out, которое предваряет имя этого параметра.

Для того чтобы стали понятнее последствия применения ковариантности, обратимся

к конкретному примеру. Ниже приведен очень простой интерфейс IMyCoVarGenIF,

в котором применяется ковариантность.

// В этом обобщенном интерфейсе поддерживается ковариантность.

public interface IMyCoVarGenIF<out Т>

{

Т GetObject();

}

Обратите особое внимание на то, как объявляется параметр обобщенного типа Т.

***Его имени предшествует ключевое слово out. В данном контексте ключевое слово out***

***обозначает, что обобщенный тип Т является ковариантным. А раз он ковариантный, то***

***метод GetObject() может возвращать ссылку на обобщенный тип Т или же ссылку на***

***любой класс, производный от типа Т***.

Несмотря на свою ковариантность по отношению к обобщенному типу Т, интерфейс

IMyCoVarGenIF реализуется аналогично любому другому обобщенному интерфейсу.

Ниже приведен пример реализации этого интерфейса в классе MyClass.

// Реализовать интерфейс IMyCoVarGenIF.

class MyClass<T> : IMyCoVarGenIF<T>

{

T obj;

public MyClass(T v) { obj = v; }

public T GetObject() { return obj; }

}

Обратите внимание на то, что ключевое слово out не указывается еще раз в выражении,

объявляющем реализацию данного интерфейса в классе MyClass. Это не только

не нужно, но и вредно, поскольку всякая попытка еще раз указать ключевое слово out

будет расцениваться компилятором как ошибка.

А теперь рассмотрим следующую простую реализацию иерархии классов.

// Создать простую иерархию классов.

class Alpha

{

string name;

public Alpha(string n) { name = n; }

public string GetName() { return name; }

// ...

}

class Beta : Alpha

{

public Beta(string n) : base(n) { }

// ...

}

Как видите, класс Beta является производным от класса Alpha. С учетом всего изложенного выше, следующая последовательность операций будет считаться вполне допустимой.

// Создать ссылку из интерфейса IMyCoVarGenIF на объект типа MyClass<Alpha>.

// Это вполне допустимо как при наличии ковариантности, так и без нее.

IMyCoVarGenIF<Alpha> AlphaRef =

new MyClass<Alpha>(new Alpha("Alpha #1"));

Console.WriteLine("Имя объекта, на который ссылается переменная AlphaRef: " +

AlphaRef.GetObject().GetName());

// А теперь создать объект MyClass<Beta> и присвоить его переменной AlphaRef.

// \*\*\* Эта строка кода вполне допустима благодаря ковариантности. \*\*\*

AlphaRef = new MyClass<Beta>(new Beta("Beta #1"));

Console.WriteLine("Имя объекта, на который теперь ссылается " +

"переменная AlphaRef: " + AlphaRef.GetObject().GetName());

Прежде всего, переменной AlphaRef типа IMyCoVarGenIF<Alpha> в этом фрагменте

кода присваивается ссылка на объект типа MyClass<Alpha>. Это вполне допустимая

операция, поскольку в классе MyClass реализуется интерфейс IMyCoVarGenIF,

причем и в том, и в другом в качестве аргумента типа указывается Alpha. Далее имя

объекта выводится на экран при вызове метода GetName() для объекта, возвращаемого

методом GetObject(). И эта операция вполне допустима, поскольку Alpha — это

и тип, возвращаемый методом GetName(), и обобщенный тип Т. После этого переменной

AlphaRef присваивается ссылка на экземпляр объекта типа MyClass<Beta>,

что также допустимо, потому что класс Beta является производным от класса Alpha,

а обобщенный тип Т — ковариантным в интерфейсе IMyCoVarGenIF. Если бы любое

из этих условий не выполнялось, данная операция оказалась бы недопустимой.

Ради большей наглядности примера вся рассмотренная выше последовательность

операций собрана ниже в единую программу.

(***glava18\_22***)

//thid generalizated interface supports covariance

public interface IMyCoVarGenIF<out T>

{

T GetObject();

}

//realize interface IMyCoVarGenIF

class MyClass<T> : IMyCoVarGenIF<T>

{

T obj;

public MyClass(T v) { obj = v; }

public T GetObject() { return obj; }

}

//simple ierarchy of classes

class Alpha

{

string name;

public Alpha(string n) { name = n; }

public string GetName() { return name; }

}

class Beta : Alpha

{

public Beta(string n) : base(n) { }

}

class VarianceDemo

{

static void Main()

{

//create link from interface IMyCoVarGenIf on object type MyClass<Alpha>

//This is ok with covariance, and without it

IMyCoVarGenIF<Alpha> AlphaRef = new MyClass<Alpha>(new Alpha("Alpha #1"));

Console.WriteLine("Name of object, AlphaRef links to: "

+ AlphaRef.GetObject().GetName());

//now create object MyClass<Beta> and assign in to

//AlphaRef

//this string of code is available but for Covariance

AlphaRef = new MyClass<Beta>(new Beta("Beta #1"));

Console.WriteLine("Name of object, AlphaRef links to: "

+ AlphaRef.GetObject().GetName());

}

}

Следует особо подчеркнуть, что переменной AlphaRef можно присвоить ссылку

на объект типа MyClass<Beta> благодаря только тому, что обобщенный тип Т указан

как ковариантный в интерфейсе IMyCoVarGenIF. Для того чтобы убедиться в этом,

удалите ключевое слово out из объявления параметра обобщенного типа Т в интерфейсе

IMyCoVarGenIF и попытайтесь скомпилировать данную программу еще раз.

Компиляция завершится неудачно, поскольку строгая проверка на соответствие типов

не разрешит теперь подобное присваивание.

Один обобщенный интерфейс может вполне наследовать от другого. Иными словами,

обобщенный интерфейс с параметром ковариантного типа можно расширить,

как показано ниже.

public interface IMyCoVarGenIF2<out Т> : IMyCoVarGenIF<T>

{

// ...

}

Обратите внимание на то, что ключевое слово out указано только в объявлении расширенного

интерфейса. Указывать его в объявлении базового интерфейса не только не

нужно, но и не допустимо. И последнее замечание: обобщенный тип Т допускается не

указывать как ковариантный в объявлении интерфейса IMyCoVarGenIF2. Но при этом

исключается ковариантность, которую может обеспечить расширенный интерфейс

IMyCoVarGetIF. Разумеется, возможность сделать интерфейс IMyCoVarGenIF2 инвариантным

может потребоваться в некоторых случаях его применения.

На применение ковариантности накладываются некоторые ограничения. Ковариантность

параметра типа может распространяться только на тип, возвращаемый методом.

Следовательно, ключевое слово out нельзя применять в параметре типа, служащем

для объявления параметра метода. Ковариантность оказывается пригодной

только для ссылочных типов. Ковариантный тип нельзя использовать в качестве ограничения

в интерфейсном методе. Так, следующий интерфейс считается недопустимым.

public interface IMyCoVarGenIF2<out Т>

{

void M<V>() where V : T; // Ошибка, ковариантный тип T нельзя

// использовать как ограничение

}

**Применение контравариантности в обобщенном интерфейсе in**

Применительно к обобщенному интерфейсу контравариантность служит средством,

разрешающим методу использовать аргумент, тип которого относится к базовому

классу, указанному в соответствующем параметре типа. Параметр контравариантного

типа объявляется с помощью ключевого слова ***in***, которое предваряет имя

этого параметра.

Для того чтобы стали понятнее последствия применения ковариантности,

вновь обратимся к конкретному примеру. Ниже приведен обобщенный интерфейс

IMyContraVarGenIF контравариантного типа. В нем указывается контравариантный

параметр обобщенного типа Т, который используется в объявлении метода Show()

// Это обобщенный интерфейс, поддерживающий контравариантность.

public interface IMyContraVarGenIF<in Т>

{

void Show(T obj);

}

Как видите, обобщенный тип Т указывается в данном интерфейсе как контравариантный

с помощью ключевого слова in, предшествующего имени его параметра.

Обратите также внимание на то, что Т является параметром типа для аргумента obj

в методе Show().

Далее интерфейс IMyContraVarGenIF реализуется в классе MyClass, как показано

ниже.

// Реализовать интерфейс IMyContraVarGenIF.

class MyClass<T> : IMyContraVarGenIF<T>

{

public void Show(T x) { Console.WriteLine(x); }

}

В данном случае метод Show() просто выводит на экран строковое представление

переменной х, получаемое в результате неявного обращения к методу ToString() из

метода WriteLine().

После этого объявляется иерархия классов, как показано ниже.

// Создать простую иерархию классов.

class Alpha

{

public override string ToString()

{

return "Это объект класса Alpha.";

}

// ...

}

class Beta : Alpha

{

public override string ToString()

{

return "Это объект класса Beta.";

}

// ...

}

Ради большей наглядности классы Alpha и Beta несколько отличаются от аналогичных

классов из предыдущего примера применения ковариантности. Обратите также

внимание на то, что метод ToString() переопределяется таким образом, чтобы

возвращать тип объекта.

С учетом всего изложенного выше, следующая последовательность операций будет

считаться вполне допустимой.

// Создать ссылку из интерфейса IMyContraVarGenIF<Alpha>

// на объект типа MyClass<Alpha>.

// Это вполне допустимо как при наличии контравариантности, так и без нее.

IMyContraVarGenIF<Alpha> AlphaRef = new MyClass<Alpha>();

// Создать ссылку из интерфейса IMyContraVarGenIF<beta>

// на объект типа MyClass<Beta>.

// И это вполне допустимо как при наличии контравариантности, так и без нее.

IMyContraVarGenIF<Beta> BetaRef = new MyClass<Beta>();

// Создать ссылку из интерфейса IMyContraVarGenIF<beta>

// на объект типа MyClass<Alpha>.

// \*\*\* Это вполне допустимо благодаря контравариантности. \*\*\*

IMyContraVarGenIF<Beta> BetaRef2 = new MyClass<Alpha>();

// Этот вызов допустим как при наличии контравариантности, так и без нее.

BetaRef.Show(new Beta());

// Присвоить переменную AlphaRef переменной BetaRef.

// \*\*\* Это вполне допустимо благодаря контравариантности. \*\*\*

BetaRef = AlphaRef;

BetaRef.Show(new Beta());

Прежде всего, обратите внимание на создание двух переменных ссылочного типа

IMyContraVarGenIF, которым присваиваются ссылки на объекты класса MyClass, где

параметры типа совпадают с аналогичными параметрами в интерфейсных ссылках.

В первом случае используется параметр типа Alpha, а во втором — параметр типа

Beta. Эти объявления не требуют контравариантности и допустимы в любом случае.

Далее создается переменная ссылочного типа IMyContraVarGenIF<Beta>, но на

этот раз ей присваивается ссылка на объект класса MyClass<Alpha>. Эта операция

вполне допустима, поскольку обобщенный тип Т объявлен как контравариантный.

Как и следовало ожидать, следующая строка, в которой вызывается метод BetaRef.

Show() с аргументом Beta, является вполне допустимой. Ведь Beta — это обобщенный

тип Т в классе MyClass<Beta> и в то же время аргумент в методе Show().

В следующей строке переменная AlphaRef присваивается переменной BetaRef.

Эта операция вполне допустима лишь в силу контравариантности. В данном случае

переменная относится к типу MyClass<Beta>, а переменная AlphaRef — к типу

MyClass<Alpha>. Но поскольку Alpha является базовым классом для класса Beta, то

такое преобразование типов оказывается допустимым благодаря контравариантности.

Для того чтобы убедиться в необходимости контравариантности в рассматриваемом

здесь примере, попробуйте удалить ключевое слово in из объявления обобщенного

типа Т в интерфейсе IMyContraVarGenIF, а затем попытайтесь скомпилировать приведенный

выше код еще раз. В результате появятся ошибки компиляции.

Ради большей наглядности примера вся рассмотренная выше последовательность

операций собрана ниже в единую программу.

(***glava18\_23***)

//this is generalized interface, supports contrvariance

public interface IMyContraVarGenIf<in T>

{

void Show(T obj);

}

//realize interface IMyContraVarGenIf

class MyClass<T> : IMyContraVarGenIf<T>

{

public void Show(T x) { Console.WriteLine(x); }

}

//create simple ierarchy of classes

class Alpha

{

public override string ToString()

{

return "This is Alpha object.";

}

}

class Beta : Alpha

{

public override string ToString()

{

return "This is Beta object.";

}

}

class VarianceDemo

{

static void Main()

{

//create link from interface IMyContraVarGenIF<Alpha>

//on object type MyClass<Alpha>

//this is Ok with contrvariance and without

IMyContraVarGenIf<Alpha> AlphaRef = new MyClass<Alpha>();

//create link from interface IMyContraVarGenIF<Beta>

//on object type MyClass<Alpha>

//this is OK with contrvariance and without

IMyContraVarGenIf<Beta> BetaRef = new MyClass<Beta>();

//create link from interface IMyContraVarGenIF<Beta>

//on object type MyClass<Alpha>

//this is OK ONLY with contrvariance

IMyContraVarGenIf<Beta> BetaRef2 = new MyClass<Alpha>();

//this call is OK with contravriance and without it

BetaRef.Show(new Beta());

//assign AlphaRef to BetaRef

//this is available only bcs of contrvariance

BetaRef = AlphaRef;

BetaRef.Show(new Beta());

}

}

Контравариантный интерфейс может быть расширен аналогично описанному

выше расширению ковариантного интерфейса. Для достижения контравариантного

характера расширенного интерфейса в его объявлении должен быть указан такой же

параметр обобщенного типа, как и у базового интерфейса, но с ключевым словом in,

как показано ниже.

public interface IMyContraVarGenIf<in T>

Следует иметь в виду, что указывать ключевое слово in в объявлении базового

интерфейса не только не нужно, но и не допустимо. Более того, сам расширенный

интерфейс IMyContraVarGenIF2 не обязательно должен быть контравариантным.

Иными словами, обобщенный тип Т в интерфейсе IMyContraVarGenIF2 не требуется

модифицировать ключевым словом in. Разумеется, все преимущества, которые

сулит контравариантность в интерфейсе IMyContraVarGen, при этом будут утрачены

в интерфейсе IMyContraVarGenIF2.

Контравариантность оказывается пригодной только для ссылочных типов, а параметр

контравариантного типа можно применять только к аргументам методов. Следовательно,

ключевое слово in нельзя указывать в параметре типа, используемом в

качестве возвращаемого типа.

**Вариантные делегаты**

Как пояснялось в главе 15, ковариантность и контравариантность поддерживается

в необобщенных делегатах в отношении типов, возвращаемых методами, и типов, указываемых

при объявлении параметров. Начиная с версии C# 4.0, возможности ковариантности

и контравариантности были распространены и на обобщенные делегаты.

Подобные возможности действуют таким же образом, как было описано выше в отношении

обобщенных интерфейсов.

Ниже приведен пример контравариантного делегата.

// Объявить делегат, контравариантный по отношению к обобщенному типу Т.

delegate bool SomeOp<in Т>(Т obj);

Этому делегату можно присвоить метод с параметром обобщенного типа Т или же

класс, производный от типа Т.

А вот пример ковариантного делегата.

// Объявить делегат, ковариантный по отношению к обобщенному типу Т.

delegate Т AnotherOp<out Т, V>(V obj);

Этому делегату можно присвоить метод, возвращающий обобщенный тип Т, или

же класс, производный от типа Т. В данном случае V оказывается просто параметром

инвариантного типа.

В следующем примере программы демонстрируется применение обоих разновидностей

вариантных делегатов на практике.

(***glava18\_24***)

//defina delegate contrvariance to generazed type T

delegate bool SomeOp<in T>(T obj);

//define delegate, covariance to generaized to type T

delegate T AnotherOp<out T, V>(V obj);

class Alpha

{

public int Val { get; set; }

public Alpha(int v) { Val = v; }

}

class Beta : Alpha

{

public Beta(int v) : base(v) { }

}

class GenDelegateVarianceDemo

{

//return true, if value of obj.Val is Even

static bool IsEven(Alpha obj)

{

if ((obj.Val % 2) == 0) return true;

return false;

}

static Beta ChangeIt(Alpha obj)

{

return new Beta(obj.Val + 2);

}

static void Main()

{

Alpha objA = new Alpha(4);

Beta objB = new Beta(9);

//demonstrate contrvariance, define delegate

//SomeOp<Alpha> and assing method

SomeOp<Alpha> checkIt = IsEven;

//define delegate SomeOp<Beta>

SomeOp<Beta> checkIt2;

//now assign delegate SomeOp<Alpha> to SomeOp<Beta>

//it is allowed only but for contrvariance

checkIt2 = checkIt;

//call method from delegate

Console.WriteLine(checkIt2(objB));

//demonstrate covariance

//define two delegates type AnotherOp

//return type is Beta, and parameter is Alpha

//for delegate modifyIt assigned to method ChangeIt

AnotherOp<Beta, Alpha> modifyIt = ChangeIt;

//return type is Alpha

//parameter type Alpha

AnotherOp<Alpha, Alpha> modifyIt2;

//now assign delegat modifyIt to delegate modifyIt2

//only but for covariance

modifyIt2 = modifyIt;

//call method and output results on screen

objA = modifyIt2(objA);

Console.WriteLine(objA.Val);

}

}

Каждая операция достаточно подробно поясняется в комментариях к данной программе.

Следует особо подчеркнуть, для успешной компиляции программы в объявлении

обоих типов делегатов SomeOp and AnotherOp должны быть непременно указаны

ключевые слова in и out соответственно. Без этих модификаторов компиляция

программы будет выполнена с ошибками из-за отсутствия неявных преобразований

типов в означенных строках кода.