инженерия программного обеспечения

Сафонов Владимир Олегович

ПРАКТИЧЕСКОЕ РУКОВОДСТВО ПО СИСТЕМЕ АСПЕКТНО-ОРИЕНТИРОВАННОГО ПРОГРАММИРОВАНИЯ ASPECT.NET. ЧАСТЬ 2

Аннотация

Описаны возможности Aspect.NET - системы аспектно-ориентированного программирования для платформы .NET. Приведены примеры аспектов, описан метаязык их спецификации, даны практические сведения для работы в системе.

Продолжение. Начало статьи см. № 3, 2008.

7. ПОДРОБНОЕ ОПИСАНИЕ МЕТАЯЗЫКА ASPECT.NET.ML

Приведем полное подробное описание синтаксиса и семантики конструкций метаязыка АОП – Aspect.NET.ML, соответствующих версии Aspect.NET 2.1. Синтаксические определения приведены в расширенной форме Бэкуса-Наура (РБНФ). Ключевые слова Aspect.NET.ML (например, **`%aspect`**) даны жирным шрифтом. Метасимволы РБНФ, например:

`[ ] * | .`

(точка обозначает конец синтаксического правила) также даны жирным шрифтом.

7.1. ОПРЕДЕЛЕНИЕ АСПЕКТА

Синтаксис (см. листинг 1).

Семантика

Определение аспекта начинается с его заголовка – ключевого слова **`%aspect`**, за которым следует имя аспекта. За заголовком следует тело аспекта, которое может быть любым определением класса на языке C#. Имя класса должно совпадать с именем аспекта.

Заголовок класса (**`class`** *AspectName*) может отсутствовать. В таком случае он актоматически генерируется конвертором Aspect.NET.ML.

Класс, реализующий определение аспекта, должен содержать определения *мо-*

Листинг 1

```
AspectDefinition =
  %aspect AspectName CSharpClassDefinitionWithModulesAndActions .
AspectName = Id .
```

дулей (после ключевого слова **%modules**) и *правил внедрения* (после ключевого слова **%rules**).

Модули аспекта (раздел *modules* не обязателен), как правило, должны быть приватными методами класса (см. приведенный выше пример).

Заметим, что такой подход отличается от подхода AspectJ. Мы не вводим в Aspect.NET новую разновидность конструкции в сам язык C#, а вместо этого предоставляем простые и наглядные аннотации к коду на C#, объясняющие пользователям и самой системе Aspect.NET, что данный код определения класса должен рассматриваться как аспект. Мы считаем, что такой подход гораздо проще, не создает коллизий аспектов и классов на уровне языка реализации, а в будущих версиях Aspect.NET он позволит нам использовать аналогичные АОП-аннотации для кода на других языках платформы .NET, например на VB.NET.

7.2. ПРАВИЛА ВНЕДРЕНИЯ АСПЕКТА

Синтаксис (см. листинг 2)
Примеры

1) (см. листинг 3)

Данное правило внедрения обеспечивает следующую функциональность: после каждого вызова каждого метода в целевой программе вставляется вызов статического метода (действия) *SayBye()*, который выводит сообщение «Bye» на консоль. Таким образом, применение подобного рода правил внедрения существенно упрощает реализацию протоколирования целевых программ.

Листинг 2

```
AspectWeavingRulesSection =
         %rules
         WeavingRule + .

WeavingRule =
        WeavingRuleCondition
        WeavingRuleAction .

WeavingRuleCondition =
        WeavingRuleConditionClause  [  '||' WeavingRuleConditionClause ] *    .

WeavingRuleConditionClause = WeavingContext JoinPointKind MethodPattern

WeavingContext =
        %before  |   %after  |  %instead  .

JoinPointKind =
        %call  .

MethodPattern = MethodNameFilter  [  MethodArgumentTypes  ]  [ Restrictions  ]  .

MethodArgumentTypes = ' (' '..' | CLRType [', ' CLRType]* ') ' .

MethodNameFilter =
      [ public | private | *] [static] [CLRType| *] MethodNameWildCard  .

Restrictions  =  Restriction  [ '&&'  Restriction  ] *  .

Restriction  =
     %args '(' arg'[' IntLiteral ']'  [  ', '  arg'[' IntLiteral ']'  ] *  ')'  |
      %args '(..)' |
     %[ '!' ]within '(' TypeNameWildCard')'   |
     %[ '!' ]withincode '('  MethodNameWildCard ')'   .

WeavingRuleAction =
    %action PublicStaticMethodDef .
```

2) (см. листинг 4)

Данное правило внедрения обеспечивает следующую функциональность. В целевом приложении, как предполагается в определении аспекта, должны быть классы, имена которых оканчиваются на *BankAccount*. В таких классах, как предполагается, должен быть статический метод *withdraw* с одним аргументом типа *float*. В соответствии с данным правилом, все такие вызовы метода *withdraw* будут заменены на вызовы нового метода *WithdrawWrapper* (действия аспекта – «обертки» для метода *withdraw*), реализующего некоторые дополнительные проверки, которые необходимо выполнить перед вызовом банковской операции снятия со счета (*withdraw*).

Ограничение *args(..)* означает, что *все* аргументы целевого метода w*ithdraw* будут захвачены и обработаны как соответствующие аргументы действия аспекта. В соответствии с данным ограничением, действие аспекта *WithdrawWrapper* также имеет ровно один аргумент типа *float*.

Функциональность *args* реализована аналогично AspectJ [13].

Примеры фильтрации методов

Возможность фильтрации методов по их сигнатурам – весьма важный и мощный инструмент. Вот несколько примеров фильтрации методов (см. листинг 5).

Семантика

Правила внедрения определяют, каким образом аспект в Aspect.NET вставляет вызовы действий аспекта перед *точкой присоединения*, после нее или вместо нее – точки (фрагмента) в коде целевой программы, который подсистема внедрения находит, в соответствии с условием правила.

В текущей версии – Aspect.NET 2.1 – реализована только разновидность *call* точек присоединения. Это означает, что подсистема внедрения может вставлять вызовы действий аспекта перед вызовом, после или вместо вызова метода в целевой программе.

В последующих версиях мы также планируем реализовать две других разновидности точек присоединения – *assign* (присваивание именованной сущности) и *use* (использование именованной сущности).

Действие аспекта должно быть *public static* методом.

Листинг 3

```
%after %call *
%action
   public static void SayBye ()
   { System.Console.WriteLine("Bye"); }
```

Листинг 4

```
%instead %call *BankAccount.withdraw(float) && args(..)
%action
        public static float WithdrawWrapper(float amount)
         {
             BankAccount acc = (BankAccount)TargetObject;

             if (isLicenceValid(TargetMemberInfo.Name))
                 return acc.withdraw(amount);

             Console.WriteLine("Withdraw operation is not allowed");
              return acc.Balance;
         }
```

Условие правила внедрения в общем случае может быть *дизъюнкцией* предложений:

c_1 || … || c_n

Такое условие удовлетворяется, если удовлетворяется любая из его компонент.

Например, условие:

```
%before %call *Set || %before %call *Get
```

означает «перед вызовом любого метода, имя которого оканчивается на *Set* или *Get*.

Условие в простейшем случае представлено шаблоном для выбора имен методов (например «*», что, как обычно, обозначает любое имя метода). Если искомый метод целевой программы принадлежит ее классу *C*, то для имени данного метода должен быть использован шаблон *C.**

При использовании условия вида *%instead* (*вместо*) необходимо указывать только *статические* методы целевой программы, так как действие аспекта также является статическим методом. Иначе подсистема внедрения выдаст сообщение об ошибке, и внедрение будет невозможно.

Условие может также содержать *ограничения*, отделяемые от шаблона для имени метода знаком *&&*. Это означает, что условие, задаваемое шаблоном имени метода, и данное ограничение рассматриваются как *конъюнкция* условий.

Ограничение вида *%arg* указывает, сколько и какие именно аргументы целевого метода будут захвачены действием аспекта. В нем конструкция *IntLiteral* специфицирует номер аргумента в целевой программе (нумерация начинается от 0) для захвата.

Например, если задано ограничение вида *%arg(args[1])*, соответствующее действие аспекта должно иметь один аргумент, тип которого идентичен типу первого (начиная от 0) аргумента метода целевой программы. При вставке вызова действия аспекта, его аргумент должен быть первым аргументом целевого метода.

Если указано ограничение вида *%arg(args[0],args[1])*, то действие аспекта должно иметь два аргумента, типы которых идентичны типам нулевого и первого аргументов целевого метода и т. д.

Ограничение вида *%arg(..)* означает, что действие аспекта должно иметь то же число аргументов, что и целевой метод.

Таким образом, ограничения вида *%arg* предоставляют полезную возможность взаимодействия аспекта с контекстом его точек присоединения.

Ограничение вида *%within(T)* означает, что поиск целевых методов будет осуществляться только внутри определения типа *T*.

Ограничение вида *%!within(T)* означает, что поиск целевых методов будет осуществляться везде, *кроме* определения типа *T*.

Ограничение вида *%withincode(M)* означает, что поиск целевых методов будет осуществляться только внутри кода методов, имена которых удовлетворяют шаблону *M*.

Ограничение вида *%!withincode(M)* означает, что поиск целевых методов будет осуществляться везде, *кроме* кода методов, имена которых удовлетворяют шаблону *M*. Например, ограничение *%!withincode(*.ctor)*

Листинг 5

```
public static float *BankAccount.withdraw(float)

float *BankAccount.withdraw(..)
  – любые аргументы

private * *BankAccount.withdraw(float)

private * *BankAccount.withdraw(float, ..)
  – первый аргумент – типа float, остальные аргументы могут быть произвольными
```

означает, что поиск целевых методов будет осуществляться везде, кроме кода конструкторов.

7.3. ЗАХВАТ ИНФОРМАЦИИ ИЗ КОНТЕКСТА ТОЧКИ ПРИСОЕДИНЕНИЯ

Aspect.NET 2.1 обеспечивает целый ряд способов коммуникации кода аспекта с контекстом его точек присоединения.

Первый из них, ограничения вида *%args*, описан в пункте 7.2. Он обеспечивает доступ к аргументам целевого метода и аналогичен возможностям AspectJ.

Второй способ коммуникации – это набор простых конструкций метаязыка Aspect.NET.ML, которые могут быть использованы в коде определения аспекта и обеспечивают доступ к имени целевого метода, номеру строчки исходного кода, где расположена точка присоединения, и т. д.

Синтаксис (см. листинг 6)

Семантика

Простые конструкции Aspect.NET.ML, перечисленные выше, могут использоваться в определении аспекта на Aspect.NET.ML везде, где может быть использовано *первичное выражение* языка C#, за исключением левых частей присваиваний, то есть указанные сущности *не* могут быть модифицированы аспектом. Они предоставляют базовые способы коммуникации аспекта с его точками присоединения.

%TargetObject (типа *System.Object*) – это ссылка на объект в целевой программе, к которому применяется целевой метод в точке присоединения. Например, если вызов целевого метода в точке присоединения имеет вид *p.M(),* то *%TargetObject* обозначает ссмылку на объект *p*. Если целевой метод статический, *%TargetObject* равно *null*.

%TargetMemberInfo (типа *System.Reflection.MemberInfo*) – это ссылка на объект, представляющий целевой метод из точки присоединения в традиционном стиле, принятом при использовании рефлексии .NET (*System.Reflection*). Данная конструкция позволяет получить значительный объем информации о целевом методе и использовать ее в аспекте. В частности, *%TargetMemberInfo.Name* – это public-свойство, содержащее имя целевого метода.

%RetValue (типа *System.object*) – это значение, возвращаемое целевым методом. Пример его использования – *RetValue*, входящий в поставку Aspect.NET 2.1.

%WithinType (типа *System.Type*) – это ссылка на определение типа, в котором расположен вызов целевого метода.

%WithinMethod (типа *System.Reflection.MethodBase*) – это ссылка на объект, представляющий метод, в котором расположен вызов целевого метода.

%SourceFilePath – это строка, представляющая путь к целевому файлу исходного кода, в котором расположен вызов целевого метода.

%SourceFileLine – это строка, представляющая номер строчки в исходном коде, в файле, где расположен вызов целевого метода.

%This – это ссылка на объект, представляющий аспект в целом. В соответ-

Листинг 6

```
TargetAppContextItem =
      %TargetObject |
      %TargetMemberInfo |
      %This |
      %RetValue |   //  – дополнение, реализованное в Aspect.NET 2.1
      %WithinType |
      %WithinMethod |
      %SourceFilePath |
      %SourceFileLine .
```

ствии с реализацией Aspect 2.1, каждый аспект представлен дочерним классом абстрактного класса *Aspect*, определенного в пространстве имен *AspectDotNet*, части реализации Aspect.NET. Родственная связь класса, задающего определение аспекта, и класса *Aspect* по исходному коду *.an.cs* – файла, генерируемого конвертором Aspect.NET. Все другие ключевые слова Aspect.NET.ML, описанные в данном пункте, реализованы как public static – свойства класса, реализующего аспект.

7.4. САМОДОКУМЕНТИРОВАНИЕ АСПЕКТОВ: *ASPECTDESCRIPTION*

Исходный код определения аспекта может быть документирован обычными комментариями в стиле // (от символа // до конца строчки исходного кода).

Aspect.NET обеспечивает преобразование таких комментариев в значение специализированного атрибута *AspectDescription*. Более подробно: если комментарий в стиле «//» дан перед заголовком аспекта, либо перед действием аспекта, то конвертор Aspect.NET.ML преобразует такие комментарии, как показано в приведенном ниже примере.

Код на Aspect.NET.ML (см. листинг 7).

Соответствующий код на C# после конвертирования (см. листинг 8).

Самодокументирующие описания аспектов полезны тем, что они воспроизводятся Aspect.NET Framework при визуализации открытого аспекта в окне Aspect.NET, что улучшает понимаемость аспектов их пользователями.

Листинг 7

```
// This is my aspect
%aspect MyAspect
{

// This is my aspect's action
%before %call * %action
public static void A()
{ Console.WriteLine("Action A");
}
...
```

7.5. ИСПОЛЬЗОВАНИЕ СПЕЦИАЛИЗИРОВАННЫХ АТРИБУТОВ ASPECT.NET НЕПОСРЕДСТВЕННО

Как объяснено выше, Aspect.NET позволяет избежать использования метаязыка Aspect.NET.ML для реализации аспектов и вместо этого использовать непосредственно код на языке C# с определениями специализированных атрибутов АОП. Пример файла *.an.cs* приведен в разделе 5.

Имеется два специализированных атрибута АОП для представления аспектов – *AspectDescription* и *AspectAction*.

Атрибут *AspectDescription* содержит определенную пользователем строку, описывающую (аннотирующую) определение аспекта в целом и его действия. Данная строка воспроизводится Aspect.NET Framework при визуализации информации об аспекте. Данный специализированный атрибут не является обязательным. Определение аспекта может иметь либо не иметь явных пользовательских описаний (в виде строк) аспекта и его действий, хотя их наличие и рекомендуется.

Листинг 8

```
[AspectDescription("This is my aspect")]
    public class Aspect1 : Aspect {

[AspectDescription("This is my aspect's action")]
[AspectAction("%before %call *")]
public static void A()
{ Console.WriteLine("Action A"); }
 …
```

Атрибут *AspectAction* – ключевой. Он помечает любое действие аспекта и содержит (в форме строки) соответствующее условие внедрения. Данный атрибут – *обязательный* для аспектов в Aspect.NET. Если методы, реализующие действия аспекта, не помечены данным атрибутом, Aspect.NET «не поймет» правильно структуру аспекта, и Aspect.NET Framework не позволит загрузить данную сборку как аспект.

Использование описанных атрибутов АОП иллюстрируется следующим примером кода на C# (см. листинг 9).

Внимание! При непосредственном использовании специализированных атрибутов АОП не следует забывать явно указывать наследование от класса *Aspect*. Иначе Aspect.NET Framework «не поймет» Ваш код как определение аспекта.

8. ПРИМЕРЫ АСПЕКТОВ, ВХОДЯЩИЕ В ПОСТАВКУ ASPECT.NET 2.1

С версией Aspect.NET 2.1 поставляются следующие примеры аспектов и целевых программ для их внедрения:

Aspect2 – очень простой пример определения аспекта, демонстрирующий основные понятия АОП и Aspect.NET и использование *%TargetMemberInfo*. Он аналогичен рассмотренному выше примеру *ConsoleApplication1 / Aspect1*.

Пример состоит из целевой программы *ConsoleApplication1* и аспекта *Aspect2*, внедряемого в нее.

TestArgs – аспект, демонстрирующий фильтрацию целевых методов и захват аргументов целевого метода. Состоит из целевой программы *TestArgs* и аспекта *TestArgsAspect*.

BankManagement – упрощенная программа для управления банковскими счетами и два аспекта, расширяющие ее возможности некоторыми дополнительными проверками, связанными с безопасностью и лицензированием: например, баланс должен быть больше или равен нулю и т. д. Данный пример может также рассматриваться как демонстрация возможностей контрактного (*Design-by-Contract*) стиля программирования, который обеспечивает повышенную надежность программ. Пример состоит из целевой программы *BankManagement*, класса *BankAccount*, используемого в данной программе, и двух аспектов – *BankAccountContractAspect* и *UsageLicensingAspect*. Для демонстрационных целей примеры этих аспектов написаны непосредственно на C# с явным использованием специализированных атри-

Листинг 9

```
namespace BankManagement
{
   [AspectDescription("Design By Contract aspect for verifying BankAccount logic")]
    public class BankAccountContractAspect: Aspect
     {
         [AspectDescription("Invariant: this.Balance >= 0")]
              [AspectAction("%before %call *BankAccount.*(..) || %after %call
*BankAccount.*(..)")]
         static public void CheckInvariant()
          {
             if (((BankAccount)TargetObject).Balance < 0)
                 Console.WriteLine
              ("ERROR INVARIANT: Balance should not be negative at {0} line {1}",
                 SourceFilePath, SourceFileLine);
          }
      }
}
```

бутов АОП (фаза конвертирования пропускается). Отметим, что Вы можете загрузить оба аспекта в Aspect.NET Framework в одном вызове (экземпляре) Visual Studio и выполнить цикл «Find Join Points / Weave Aspects» лишь один раз, одним вызовом подсистемы внедрения аспектов Aspect.NET.

MAddNopCounter – наш скромный подарок от группы Aspect.NET группе Phoenix из Microsoft. Целевая программа в данном примере – это программа *addnop-tool* из примеров, поставляемых с версией Phoenix RDK [15], расположенная в поддиректории *src/samples/AddNop-tool/csharp* Phoenix RDK (март 2007). Эта программа, использующая Phoenix, вставляет инструкцию *nop* (пустую операцию) после каждой инструкции MSIL-кода файла сборки .NET, заданного параметром. Но утилита *addnop-tool* очень «молчалива». В нашем примере в дополнение к ней, мы предлагаем аспект *MAddNopCounter*, внедрение которого в программу *addnop-tool*, расширяет ее функциональность возможностями «инструментовки» ее кода добавлением дополнительных операторов вывода, например, аспект подсчитывает и выводит фактическое число вставленных инструкций *nop*. Программа вызывается с двумя аргументами командной строки – имя исходного файла сборки и имя файла результата.

RetTest and *RetTest2* – примеры, демонстрирующие новую возможность Aspect.NET 2.1– *%RetValue,* а также конструкции *WithinType / WithinMethod functionality* (см. 7.3).

Каждый из примеров состоит из определений аспектов и целевых программ для их внедрения. Аспекты самодокументированы с использованием комментариев и атрибута *AspectDescription*.

Решения Visual Studio 2005 для аспектов и их целевых программ размещены в поддиректориях, имена которых совпадают с именами примеров.

Кроме описанных примеров, для освоения Aspect.NET могут быть использованы примеры аспектов, улучшающих надежность и безопасность программ, иллюстрирующие мою монографию [2], которые доступны для скачивания на сайте проекта Aspect.NET [11].

9. СОПРОВОЖДЕНИЕ И РАЗВИТИЕ ASPECT.NET

Группа Aspect.NET: проф. Сафонов Владимир Олегович (научный руководитель и главный архитектор проекта), Дмитрий Григорьев, Михаил Грачев, Александр Масленников, Руслан Муханов (аспиранты математико-механического факультета СПбГУ) будет благодарна уважаемым читателям за любые предложения и замечания по нашей системе.

Присылайте их проф. В.О. Сафонову по email: *v_o_safonov@mail.ru*

Удачи в использовании Aspect.NET!

Мы надеемся, что наша система вам понравится, и, благодаря Aspect.NET и данной статье, Вы изучите и оцените огромные возможности и перспективы АОП и научитесь применять их практически к Вашим задачам.

Мы очень заинтересованы также в заказчиках, которые могли бы финансировать нашу разработку, развитие Aspect.NET, его применение для разработки программного обеспечения в различных областях, и открыты к Вашим предложениям.

Литература

1. *Сафонов В.О.* Aspect.NET – инструмент аспектно-ориентированного программирования для разработки надежных и безопасных программ // Компьютерные инструменты в образовании, 2007, № 5.

2. *Safonov V.O.* Using aspect-oriented programming for trustworthy software development. Wiley Interscience. John Wiley & Sons, 2008.

3. Aspect.NET 2.1: *http://www.msdnaa.net/curriculum/?id=6801*

4. *Safonov V.O.* Aspect.NET: a new approach to aspect-oriented programming. .NET Developer's Journal, 2003, #4.

5. *Safonov V.O.* Aspect.NET: concepts and architecture. .NET Developer's Journal, 2004, # 10.

6. *Safonov V.O., Grigoryev D.A.* Aspect.NET: aspect-oriented programming for Microsoft.NET in practice. NET Developer's Journal, 2005, # 7

7. *Safonov V.O., Gratchev M.K., Grigoryev D.A., Maslennikov A.I.* Aspect.NET – aspect-oriented toolkit for Microsoft.NET based on Phoenix and Whidbey. «.NET Technologies 2006» International Conference Proceedings, Pilsen, Czech Republic, 2006. *http://dotnet.zcu.cz*

8. Aspect.NET 1.0: *http://www.msdnaa.net/curriculum/?id=6219*

9. Aspect.NET 1.1: *http://www.msdnaa.net/curriculum/?id=6334*

10. Aspect.NET 2.0: *http://www.msdnaa.net/curriculum/?id=6595*

11. Web-сайт проекта Aspect.NET: *http://www.aspectdotnet.org*

12. Web-сайт, посвященный аспектно-ориентированной разработке программ: *http://aosd.net*

13. Web-сайт AspectJ: *http://www.aspectj.org*

14. Web-сайт Microsoft Phoenix: *http://research.microsoft.com/phoenix*

15. Web-страница Microsoft Phoenix RDK, March 2007, используемого в Aspect.NET 2.1: *https://connect.microsoft.com/Downloads/DownloadDetails.aspx?SiteID=214&DownloadID=5538*

Abstract

The article describes features and use of Aspect.NET – aspect-oriented programming toolkit for .NET platform. Samples of aspects are provided, AOP meta-language is described, practical information on using the system is given.

Сафонов Владимир Олегович,
доктор технических наук,
профессор кафедры информатики
СПбГУ, руководитель лаборатории
Java-технологии,
v_o_safonov@mail.ru