# Бесшовное аспектно-ориентированное программирование веб-приложений на платформе ASP.NET с помощью ASPECT.NET

Григорьева А.В., асп. кафедры информатики СПбГУ,
nastya001@mail.ru

## Аннотация

В работе рассматривается система аспектно-ориентированного программирования Aspect.NET, делается сравнение с существующими решениями и приводятся примеры бесшовного расширения функциональности веб-приложений на платформе ASP.NET.

## Технология АОП

Аспектно-ориентированное программирование (АОП) — парадигма программирования, в основе которой лежит идея выделения сквозной функциональности в отдельные модули, называемые аспектами. В таких модулях описываются действия, выполняемые в определенных точках программы и правила их "вплетения" в эти точки. К сквозной функциональности относятся: протоколирование, кэширование, сбор диагностической информации, авторизация и т.д. Инкапсуляция сквозной функциональности в отдельных модулях повышает качество архитектуры программного проекта в соответствии с принципом единственной ответственности SRP [1]. При этом основное приложение будет сосредоточено на бизнес-логике, в то время как аспекты на нефункциональных требованиях.

Для АОП-разработки применяется множество инструментов, самыми популярными из которых являются AspectJ [2] и PostSharp [3]. Однако, проведенный в процессе написания работы, анализ предметной области показал, что популярность АОП-инструментов уступает альтернативным технологиям повышения качества программного обеспечения, таким, например, как рефакторинг и IOC-контейнеры [4]. Существенной проблемой при изменении поведения программы с помощью современных АОП-инструментов является отсутствие бесшовной интеграции аспектов и целевого исходного кода системы. Термин "бесшовная интеграция" применительно к аспектам и целевому коду можно

сформулировать следующим образом – это такое расширение функциональности целевого кода, при котором не требуется вносить изменения в целевой проект, включая его код, файлы настроек и свойства проекта. При бесшовной интеграции в этом смысле у пользователя должна быть возможность отдельно компилировать, запускать и тестировать аспектную и целевую сборки, без необходимости иметь их исходный код. Данное определение отличается от "бесшовности" применения ряда АОП-инструментов (PostSharp, FuseJ [5] и др.), когда целевой код не зависит от аспектов, однако они тесно интегрированы в его проект, что препятствует тестированию бизнес-логики и затрудняет возможный отказ от применяемого АОП-инструмента.

В то время как пользователи AspectJ для Java-платформы имеют возможность создавать свои аспекты и правила внедрения в одном модуле и внедрять их в целевые сборки на бинарном уровне, то у платформы MS .NET до сих пор нет полноценной поддержки бесшовного АОП-программирования. Были разработаны несколько экспериментальных АОП-инструментов, которые позволяют бесшовно внедрять аспекты, например, проекты SheepAspect [6] и AOP.NET [7]. Оба эти проекта предлагают хранить определения аспектов вместе с правилами их внедрения и вставлять их в целевую сборку на этапе пост-компиляции, однако ни один из них не вышел из стадии альфа-версии и их развитие приостановлено.

Основные преимущества АОП проявляются при введении в процесс разработки с самого начала. Однако парадигма АОП позволяет проводить и АОП-рефакторинг [8] унаследованного кода (т.е. повторяющиеся куски кода, пригодные для повторного использования, выносятся в отдельный "аспектный" модуль и связываются с целевым кодом через инструмент АОП). Кроме возможности повторного использования кода со сквозной функциональностью, данный подход позволяет улучшить метрики качества целевых классов, когда вместо одной сложной сущности с появляются несколько более простых. При наличии инструмента бесшовной интеграции сквозная функциональность выносится в отдельный проект (solution), который зачастую не имеет ссылок на целевой проект и может быть затем повторно использован простым переконфигурированием правил внедрения аспектов.

Осталось упомянуть еще одно возможное применение АОП-инструментов в виде тонкой прослойки между целевым проектом и сторонней библиотекой, реализующей сквозную функциональность. С учетом того, что компания Microsoft выпустила библиотеку Enterprise Library (EL) [9], которая позиционируется как набор готовых функцио-

нальных блоков со сквозной функциональностью (Caching Application Block, Validation Application Block, Logging Application Block и т.п.), роль АОП-инструмента можно упростить и свести к вставке вызовов соответствующих функциональных блоков в нужных местах целевой программы.

## Основы системы Aspect.NET

Aspect.NET — это легковесный инструмент АОП для платформы .NET, требования и спецификации к которому были сформулированы профессором В. О. Сафоновым в 2004 году [10]. При создании Aspect.NET были поставлены следующие задачи:

- минимизировать время программиста на изучение системы;
- хранить код и правила внедрения аспектов отдельно от целевого проекта (для поддержки их бесшовной интеграции);
- дать возможность аспектам влиять на целевой код;
- снизить накладные расходы на вызов аспектов.

Aspect.NET вставляет действия на уровне MSIL-инструкций после этапа компиляции целевой сборки (аналогично PostSharp), что влечет повышение производительности целевого приложения по сравнению с IoC-контейнерами. Более того, такая "пост-обработка" дает возможность выбирать конкретные места применения действий аспектов.

## Перехват целевых методов действиями аспекта

Аспектом может быть любой класс, производный от класса Aspect (предопределенного в библиотеке Aspect.NET). Реализация аспекта осуществляется статическими методами ("действиями"), которые затем будут вставлены компоновщиком в заданные точки внедрения (joinpoints) в сборке целевого приложения. Требуемое множество точек внедрения задается в пользовательском атрибуте AspectAction() своего действия аспекта. Любое действие можно вставлять перед (ключевое слово %before), после (%after) или вместо (%instead) вызова заданного целевого метода. Название целевого метода задается с помощью регулярных выражений относительно его сигнатуры. В дополнение к свойствам контекста, компоновщик обеспечивает "захват" аргументов,

передавая их от целевого метода в аргументы действия аспекта. Подробно синтаксис “захвата” аргументов рассмотрен в [11].

Например, пусть в целевом классе DBFacade есть статический метод loadProduct(int id), который возвращает объект типа Product загруженный из базы данных. В свою очередь, кэширование — это сквозная функциональность и имеет смысл выделить ее в действие аспекта:

```
class CachedDBFacade : Aspect {
static DataCache cache = new DataCache("default");
/* Выполнить действие аспекта вместо DBFacade.loadProduct(int) и
    передать в него целевой аргумент */
[AspectAction("%instead %call *DBFacade.loadProduct(int))")]
  static public Product loadCachedProduct(int id)  {
    Product cachedProduct = (Product) cache.Get“(”Product +
    id);
    if (cachedProduct == null) {
      cachedProduct = DBFacade.loadProduct(id);
      cache.Put“(”Product + id, cachedProduct);
    }
    return cachedProduct;
  }
}
```

## Замещение аспектом целевого класса для перехвата событий

Кроме возможности вставлять действия аспектов перед, после или вместо вызова целевого метода, Aspect.NET предлагает концепцию “замещающего наследника” [12]. Современное программирование стремится к использованию различных каркасов (frameworks), которые предлагают общую архитектуру для ряда задач, а программист лишь уточняет поведение программы в методах обратного вызова (callbacks). В качестве примера можно привести платформу для создания веб-приложений MS ASP.NET. Предположим, у нас есть веб-страница Default.aspx, на которой расположена кнопка LogButton. При ее нажатии необходимо записать сообщение в функциональный блок EL Logging Application Block:

```
public partial class Default : System.Web.UI.Page {
protected void LogButton_Click(object sender, EventArgs e) {

    Microsoft.Practices.EnterpriseLibrary.Logging.Logger.
     Write("Message from the EL Logger");
    }
}
```

Как известно, протоколирование относится к сквозной функциональности, и его имеет смысл выносить в аспект. В этом случае метод LogButton_Click вызывается средой ASP.NET и у компоновщика нет доступа к вызывающему коду, чтобы обернуть его вызовом аспекта. В этом случае PostSharp предлагает применять правило внедрения OnMethodBoundaryAspect, при котором действия аспекта вставляются внутри тела целевого метода. Однако, если потребуется получить доступ к защищенным полям или методам целевого класса, придется использовать рефлексию .NET. Более оптимальным решением здесь было бы создать наследника класса Default с переопределенным целевым методом LogButton_Click:

```
//Проект с замещающим аспектным наследником
[ReplaceBaseClass]
public class AspectClass : Default {
protected void LogButton_Click(object sender, EventArgs e) {
//Расширенное поведение...
Microsoft.Practices.EnterpriseLibrary.Logging.
      Logger.Write("Message from the EL Logger");
        //Вызов целевого метода...
base.LogButton_Click(sender, e);
} }
//Исходный проект, после отделения зависимости от MS EL
public partial class Default : System.Web.UI.Page {
protected void LogButton_Click(
  object sender, EventArgs e) {}
}
```

Специальный пользовательский атрибут [ReplaceBaseClass] предписывает компоновщику Aspect.NET заменить целевой класс своим аспектным наследником. Следует отметить, что данное решение все же не типично для АОП, где аспект описывается максимально изолированно от целевого класса, в то время как наследование предполагает сильную связь. В то же время, выразительные средства АОП решают ограниченный круг задач и, если есть возможность решить задачу более элегантным способом, стоит им воспользоваться.

## Внедрение в конфигурационный файл веб-приложения настроек аспектов

Бесшовное расширение бизнес-логики нефункциональными требованиями с помощью сторонних библиотек (напр. Microsoft EL) заключается в создании отдельного проекта в виде библиотеки классов (dll),

добавлению к нему аспектных классов, а также интеграции в него сторонней библиотеки через менеджера зависимостей Nuget. В процессе интеграции в зависимости аспектного проекта будут добавлены необходимые ссылки и, возможно, будет создан конфигурационный XML-файл с настройками сторонней библиотеки. После слияния компоновщиком Aspect.NET аспектной сборки и целевой, объекты сторонней библиотеки будут обращаться к конфигурационному файлу целевого проекта, в котором отсутствуют необходимые настройки. Пользователю приходится вносить изменения в целевой конфигурационный файл самостоятельно, что затрудняет его редактирование, ведь теперь все возможные настройки всех аспектов и целевого проекта содержатся в одном файле. Особенно такая ситуация критична в веб-приложениях, поскольку их файл web.config содержит достаточно много узлов и назвать "бесшовным" такое применение аспектов нельзя.

Итак, сформулируем проблему по бесшовному слиянию целевой и аспектной конфигурации. Есть два проекта, в каждом из них есть свой конфигурационный файл web.config. Необходимо предложить решение, которое бы автоматически обеспечивало слияние аспектных настроек с целевыми, но при этом сами файлы с настройками оставались бы неизменными.

Для решения указанной проблемы в Aspect.NET было предложено в целевом проекте иметь два конфигурационных файла (напр. TargetWeb.config и web.config). В первом из них будут настройки целевого приложения, а во втором — результат объединения их с аспектными настройками. Пользователь будет редактировать и хранить настройки целевого проекта в TargetWeb.config, а, при построении сборки в Microsoft Visual Studio, специальное консольное приложение XMLMergerConsole (вызванное в событиях Pre-Build Events аспектного проекта) будет сливать аспектный web.config с TargetWeb.config в общий web.config целевого проекта. После завершения компиляции и развертывания в облаке аспекты и бизнес-логика будут иметь доступ к своим настройкам, содержащимся в общем web.config. Необходимость слияния конфигурационных файлов на этапе предкомпиляции (Pre-Build Events) вытекает из того, что компилятор использует некоторые настройки, заданные в web.config и, следовательно, слияние должно произойти перед стадией компиляции.

Далее будет рассмотрен требуемый алгоритм слияния аспектного web.config и целевого TargetWeb.config в результирующий web.config. Он должен удовлетворять следующим правилам:

1. вначале содержимое TargetWeb.config копируется в результирую-

щий web.config;

2. если в TargetWeb.config отсутствует узел, который есть в аспектном web.config, то этот узел добавляется в результирующий web.config;

3. если и в TargetWeb.config и в аспектном web.config есть узел с одинаковым атрибутом name, то весь этот узел берется из аспектного web.config.

Например, применение подобного алгоритма к файлу web.config аспектного проекта приведет к его интеграции в конфигурацию целевого проекта, что обеспечит сохранение веб-сессии облачного ASP.NET приложения в оперативной памяти сервера (In-Role caching):

```
<system.web>
<sessionState mode="Custom"
   customProvider="AFCacheSessionStateProvider">
   <providers>
      <add name="AFCacheSessionStateProvider"
         type="Microsoft.Web.DistributedCache.Distributed
            CacheSessionStateStoreProvider,
            Microsoft.Web.DistributedCache"
         cacheName="default"
         dataCacheClientName="default"/>
   </providers></sessionState></system.web>
```

## Литература


[1] Мартин Р. Принципы, паттерны и методики гибкой разработки на языке C# / Мартин Р., Мартин М. — Спб.: Символ-Плюс, 2011. — С. 768.

[2] Miles R. AspectJ Cookbook /Miles R. — Cambridge, USA: O'Reilly, 2004. — P. 354.

[3] Сайт проекта PostSharp. http://sharpcrafters.com [дата просмотра: 05.06.2016].

[4] Эспозито Д. Аспектно-ориентированное программирование, перехват и Unity 2.0 // MSDN журнал, — 12.2010.http://msdn.microsoft.com/ru-ru/magazine/gg490353.aspx [дата просмотра: 05.06.2016].

[5] Suvée D. et al. Evaluating FuseJ as a web service composition language //Web Services, 2005. ECOWS 2005. Third IEEE European Conference on. — IEEE, 2005.

[6] Сайт проекта SheepAspect. http://sheepaspect.codeplex.com [дата просмотра: 05.06.2016].

[7] Сайт проекта AOP.NET. https://sourceforge.net/projects/aopnet [дата просмотра: 05.06.2016].

[8] Григорьева А. В. Аспектно-ориентированный рефакторинг облачных приложений MS Azure с помощью системы Aspect.NET / Григорьева А.В. // Компьютерные инструменты в образовании. — СПб.: АНО “КИО”. — 2012. — № 1. — С. 21-30.

[9] Сайт проекта MS Enterpise Library. http://msdn.microsoft.com/en-us/library/ff648951.aspx [дата просмотра: 05.06.2016].

[10] Сайт проекта Aspect.NET. http://aspectdotnet.org [дата просмотра: 05.06.2016].

[11] Григорьев Д.А. Реализация и практическое применение аспектно-ориентированной среды программирования для Microsoft .NET /Григорьев Д.А. // Научно-технические ведомости, СПбГПУ. — 2009. — № 3. — С. 225-232.

[12] Григорьев Д.А. Бесшовная интеграция аспектов в облачные приложения на примере библиотеки Enterprise Library Integration Pack for Windows Azure и Aspect.NET / Григорьев Д.А., Григорьева А.В., Сафонов В.О. // Компьютерные инструменты в образовании. — СПб.: АНО “КИО”. — 2012. — № 4. — С. 3-15.