# Защита приложений с помощью АОП

## *Использование аспектно-ориентированного программирования для реализации системы защиты WEB приложений.*

Ренат Зубаиров

Аспектно-ориентированное программирование, сейчас является, пожалуй, одной из самых обсуждаемых тем, когда речь заходит о крупных информационных системах. Несмотря на то, что сегодня это скорее академические исследования, множество компаний таких как BEA, IBM, Microsoft a так же open source объединения как JBoss, ObjectWeb, Eclipse уже работают в этом направлении с видимыми результатами. В данной статье я постараюсь показать как можно применять АОП и какие преимущества даёт это применение на примере реализации системы разграничения доступа в WEB приложениях.

## 1. Введение

Существует множество способов защиты информационных систем, но все реализации строятся на следующих базисных принципах:

- **Идентификация** — совместно с системами аутентификации дают ответ на вопрос кто есть пользователь.
- **Авторизация** — аутентифицированный пользователь может выполнять определённые действия.
- **Целостность** — данные могут быть модифицированы только заранее определённым способом.
- **Секретность** — данные доступны только авторизированным пользователям заранее определённым путём (очень часто идёт подмена терминов секретность и защищённость).
- **Аудит** — система сохраняет отчёт о своей деятельности для дальнейшего анализа.



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

В данной статье приводится пример простейшего, не защищённого WEB приложения реализованного в терминах MVC, и последовательность применения аспектов для реализации защиты данного приложения. Я уверен, что аспектно-ориентированные техники дают реальную возможность реализовать систему защиты, удовлетворяющую всем принципам, изложенным выше, но при этом оставить аспекты защиты слабо связанными с основной частью системы.

# 2. Описание системы

## 2.1. Прецеденты использования системы

Для примера рассмотрим реализацию простейшего фотоальбома с доступом через WEB.



*Copyright © 1999 - 2007, IT-archiv. All rights reserved.*

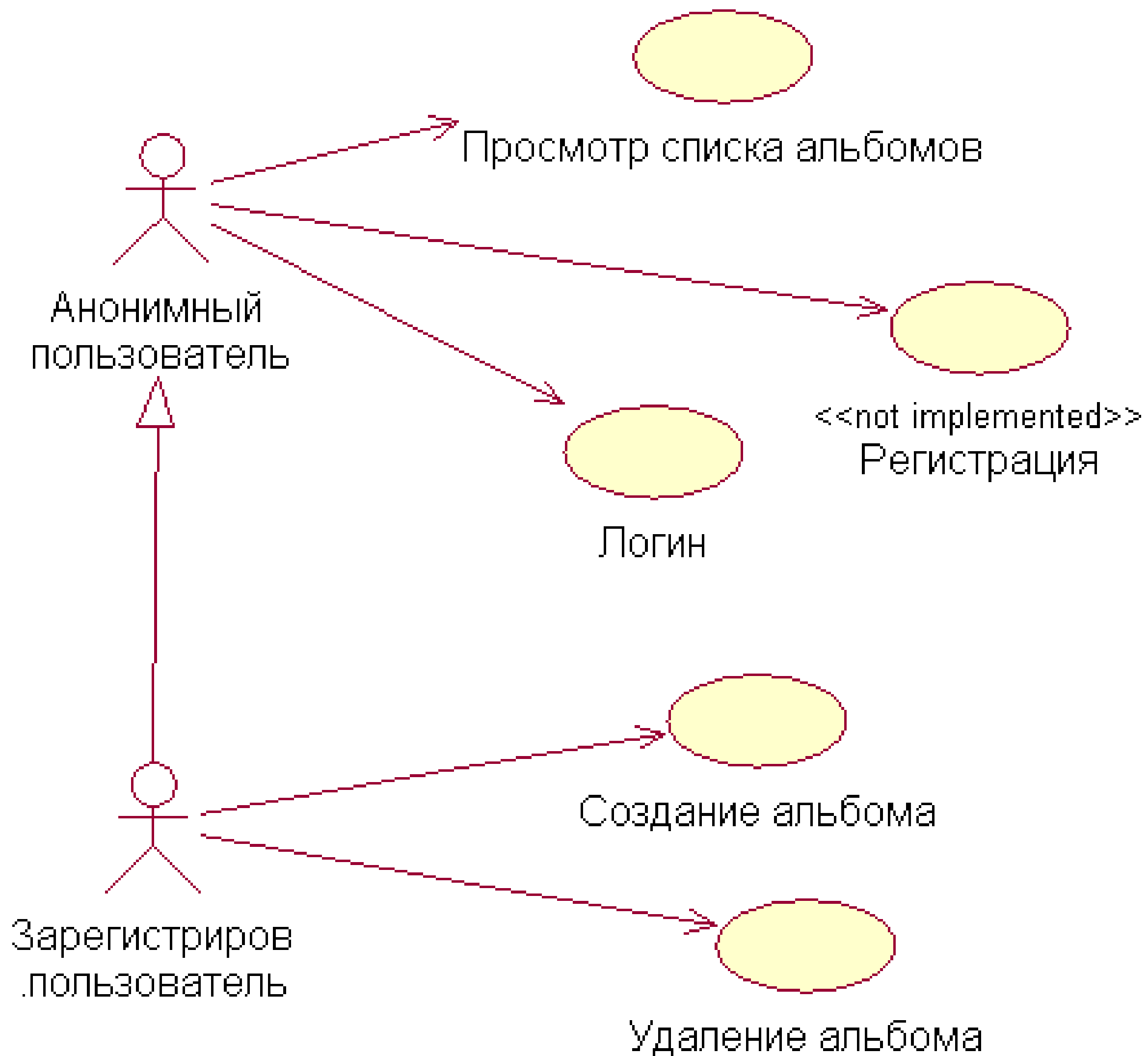


**Рисунок 1. Use cases.**

Как видно из рисунка, в системе существуют пользователи различного типа:

- Анонимный пользователь — любой посетитель сайта с фотоальбомом.
- Зарегистрированный пользователь — аутентифицированный пользователь системы имеющий регистрационное имя и пароль.

## 2.2. Техническая реализация системы



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Система представляет собой простейшее веб-приложение реализованное с использованием паттерна проектирования (design pattern) Модель-Вид-Контроллер (название по GOF). В данном приложении:

- Объект пользователь (либо `User` либо `AnonymousUser`) хранится в сессии.
- В сессии сохранён массив объектов `Album`, доступный для модификации.

Я использую упрощённую процедуру аутентификации, чтобы уменьшить и упростить код приложения.

Итак, описание системы в терминах Модель-Вид-Контроллер:

## 2.3. Модель

В нашей системе используется следующая модель:

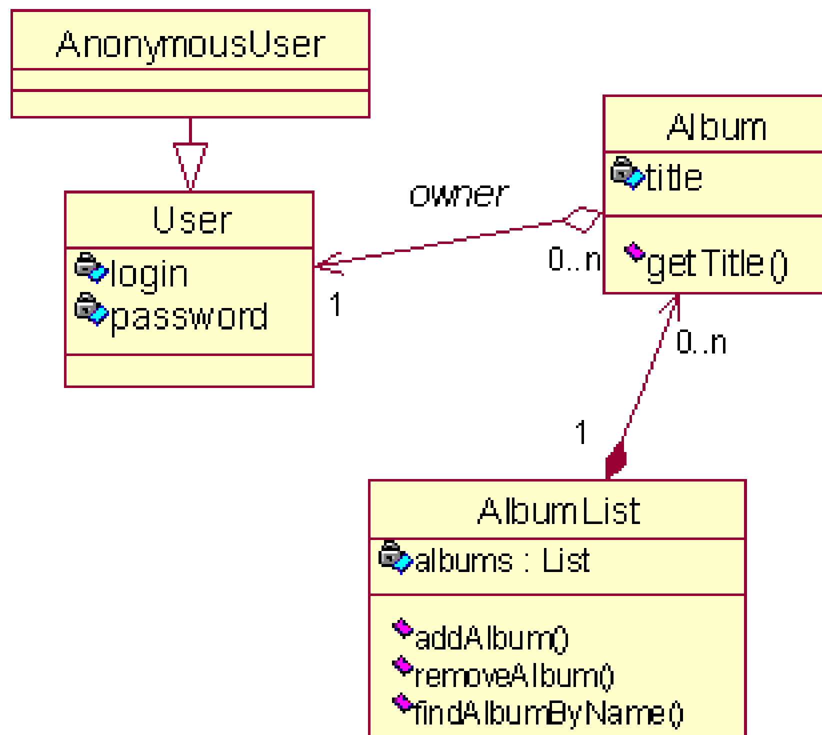


**Рисунок 2. Model.**



*Copyright © 1999 - 2007, IT-archiv. All rights reserved.*

Заметим что, класс `User` расширяется (наследуется) классом `AnonymousUser`. Каждый альбом (класс `Album`) имеет атрибут `title` и ссылку на владельца альбома - класс `User`, следовательно первое ограничение - анонимный пользователь не может быть владельцем альбома. Альбомы хранятся в объекте `AlbumList` реализующем методы добавления, удаления и поиска альбомов.

## 2.4. Вид

Отображение осуществляется с помощью Velocity шаблонов. Данный подход позволяет нам разделить отображение от остальных частей системы, кроме того, эти шаблоны достаточно просты для интуитивного понимания. В приложении используется 2 шаблона:

- view.vm — отображает логин текущего пользователя (или `Anonymous`), ссылки на login/logout, список альбомов, форму для добавления нового альбома



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

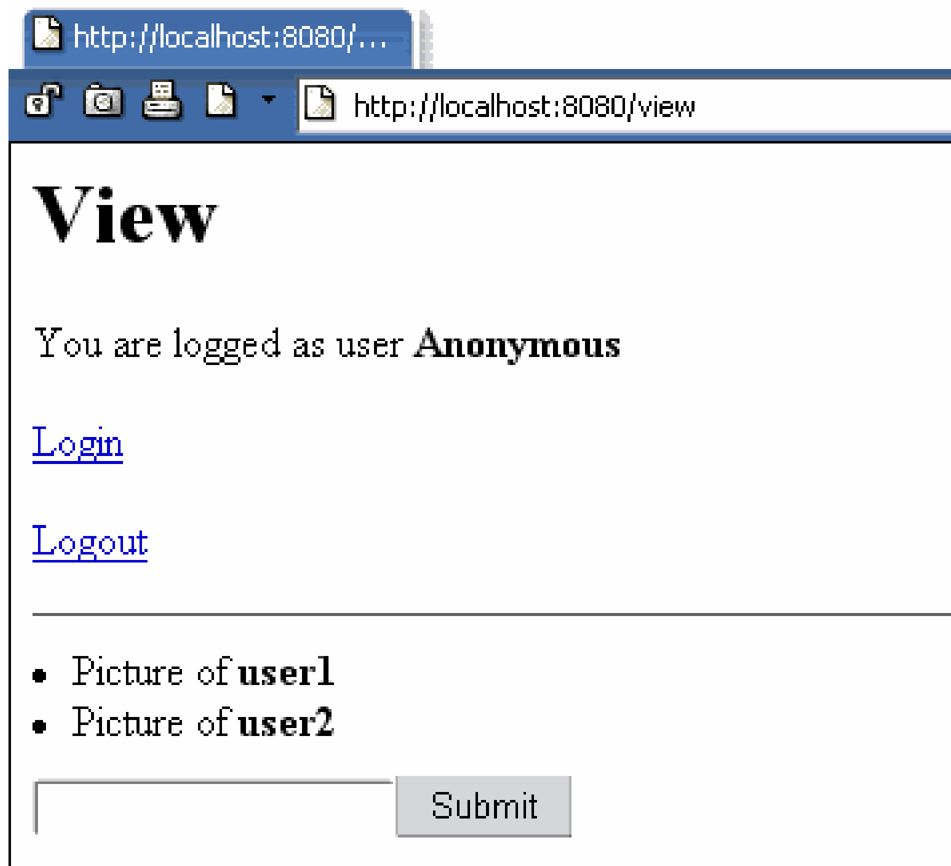


**Рисунок 3. View**

- login.vm — отображает 2 ссылки для регистрации под разными пользователями.



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

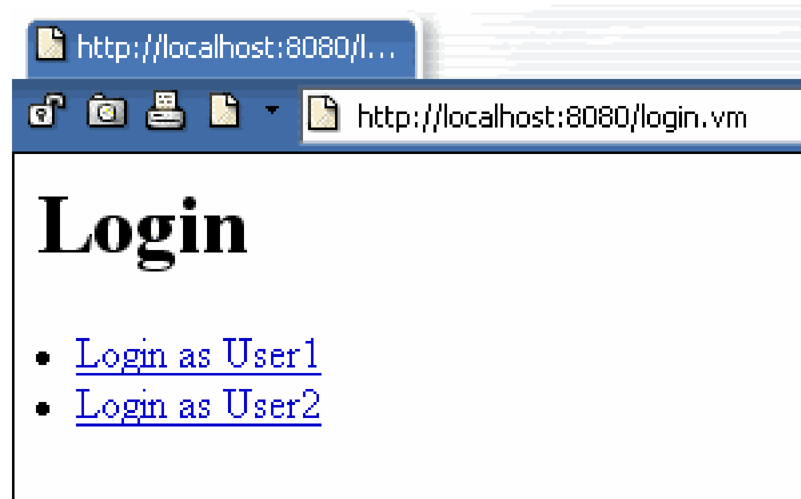


**Рисунок 4. Login**

## 2.5. Контроллер

В качестве контроллеров выступают 2 сервлета (я намеренно не использую Struts, так как считаю его использование в подавляющем большинстве случаев антипаттерном).

- `LoginServlet` — сервлет осуществляющий логин пользователя.

**Рисунок 5. LoginServlet.**

- `ViewServlet` — сервлет осуществляет подготовку списка альбомов и обрабатывает добавление/удаление альбома по переданному `title`.

**Рисунок 6. ViewServlet**

Кроме того, реализован фильтр (`EntranceFilter`) который запускается перед каждым запросом.



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

**Рисунок 7. Entrance filter.**

На данный фильтр возложена обязанность установления пользователя в AnonymousUser если пользователь не найден в сессии, и начальной инициализации массива альбомов.

# 3. Запуск системы

Приложение можно взять здесь. Для того что бы запустить приложение необходимо распаковать архив и запустить выполняемый jar архив startme.jar: `java -jar startme.jar` после чего запустится Servlet Container jetty. Приложение доступно с помошью любого веб браузера по адресу `http://localhost:<port>/` где <port> это 8080 или первый свободный порт после 8080. По умолчанию приложение сконфигурированно со всеми описанными аспектами, но кроме того, в папке `configurations` располагаются различные версии системы:

- allaspects.jar — система со всеми аспектами (уже скопирован в WEB-INF/lib).
- auditaspect.jar — только аспект аудита.
- authentaspect.jar — только аутентификационный аспект.
- authoraspectpart.jar — только пост-проверка в авторизационном аспекте.
- authoraspectfull.jar — пост и пред-проверка в авторизационном аспекте.
- noaspects.jar — исходная система без аспектов.

Для того что бы запустить одну из версий системы необходимо скопировать один из вышеперечисленных jar файлов в дирректории WEB-INF/lib (по умолчанию в WEB-INF/lib скопирован allaspects.jar)



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

# 4. Терминология

Перед тем как рассматривать методы защиты приложений, разберёмся с терминологией:

- **Идентификация** — процесс определения «личности» (во внутрисистемном понимании) пользователя. В данном случае пользователь, предоставляющий правильное (существующее) регистрационное имя, идентифицирует себя.
- **Аутентификация** — процесс подтверждения того, что пользователь является тем, кем, он себя объявляет (с кем он себя идентифицирует). В нашем случае средство аутентификации — уникальный пароль, который известен только конкретному пользователю.
- **Авторизация** — процесс определения прав пользователя по отношению к объекту внутри системы. В данном случае, очевидно, что даже зарегистрированный пользователь не должен иметь возможность удалить фотографии другого зарегистрированного пользователя.

Специфика WEB приложения работающего через HTTP 1.1 такова, что все аспекты защиты достаточно сложны в реализации. Например, самый казалось бы очевидный аспект — идентификация — совсем не так прост, учитывая отсутствие поддержки состояния соединения в HTTP. Каждый последующий запрос пользователя никак не связан с предыдущим его запросом и для сервера представляется как абсолютно независимый. Даже если мы аутентифицировали пользователя, то необходимо обозначить все запросы этого пользователя к системе. Для этого обычно используются так называемые сессии.

# 5. Реализация защиты без АОП

Для того, что бы явно показать преимущества АОП подхода, для начала бегло рассмотрим реализацию защиты приложения обычными способами.

## 5.1. Идентификация/аутентификация

Обычно WEB приложение запрашивает регистрационное имя и пароль с помощью формы, данные проверяются (сравниваются с базой данных, например) и на основании



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

сравнения принимается решение об аутентификации текущего пользователя. Далее в сессию клиента попадает объект, инкапсулирующий пользователя (а в cookie браузера отправляется идентификатор сессии).

Данный способ представляет разработчику приложения полный контроль над процессом аутентификации.

В нашем приложении аутентификация не реализована в полной мере, так как она не представляет интереса с точки зрения данной статьи. Существуют по умолчанию два пользователя **User1** и **User2**, логин каждого пользователя осуществляется декларативно со страницы login.vm;

## 5.2. Авторизация

Самая интересная часть системы защиты. В большинстве случаев применения ООП получаются сложные, сильно связные подсистемы, не позволяющие повторное использование в других проектах. Давайте разберемся почему.

Итак, авторизация — разрешение/запрет действий, выполняемых аутентифицированными пользователями. Вполне логично было бы разместить авторизацию в компоненте «Контроллер», но если, например, несколько контроллеров оперируют с одними и теми же объектами модели, то получается дублирование функций, поэтому логично было бы разместить авторизацию непосредственно в объектах модели. Для примера рассмотрим удаление альбома по названию (в нашей простой версии нет ID альбома):



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

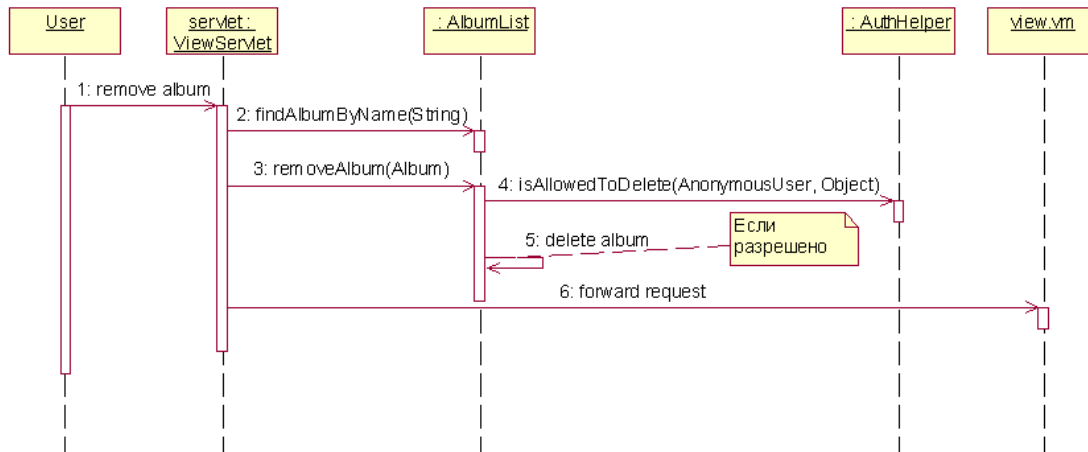


**Рисунок 8. Auth sequence.**

На диаграмме изображён запрос пользователя, обрабатываемый сервлетом, который с начала находит альбом, который нужно удалить (вызов номер 2), а потом пытается его удалить из хранилища альбомов (AlbumList — класс модели). Далее, хранилище альбомов проверяет, может ли текущий пользователь удалить выбранный альбом, используя класс системы авторизации (AuthHelper) и если такое разрешение получено, то удаляет альбом. Вызов номер 6 перенаправляет запрос на слой отображения.

На даиграмме взаимодействия видно, что для проверки авторизации нам необходимо знать пользователя (вызов номер 4 имеет 2 параметра — пользователь и объект который он пытается удалить), то есть нам необходимо передать объект пользователь в методе номер 3. Следовательно, в каждый метод модели нам необходимо дополнительно передавать текущего пользователя, а если контроллер не является объектом, непосредственно получающим запрос (а как мы помним, объект User храниться в сессии), то и в каждый метод контроллера нужно добавить такой параметр.

Кроме того, мы получаем связь модели и системы авторизации, что недопустимо, так как при смене системы авторизации модель не должна меняться.

Предположим, что мы рассматриваем реальную систему, в которой десятки, если не сотни объектов модели. В данном случае никто не сможет дать твёрдой гарантии, что все вызовы методов приводящих к изменению модели окружены такими проверками.



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

Есть один не маловажный нюанс — ссылка «Удалить» будет всё ещё отображаться напротив всех альбомов, в том числе не принадлежащих текущему пользователю. Безусловно, при нажатии на эту ссылку удаления чужых альбомов происходить не будет, но всё же ссылка на удаление не должна отображаться.

## 5.3. Целостность

Решение проблемы целостности, в общем, более сложно, чем все остальные аспекты защиты, в частности для целостности и не противоречивости данных обычно используются транзакции. Реализация этого механизма ООП методами представляет не тривиальную задачу, а попытки отделить этот механизм от основной функциональности системы могут привести к решению такому же сложному как EJB (под сложностью я понимаю не только сложность в реализации, но так же сложность в тестировании, поддержке, расширении и переносимости).

## 5.4. Секретность

Предположим, что у нас существует часть параметров объектов модели, которые не должны быть видимы для определённого типа пользователей, но в тоже время видны для остальных. Для примера рассмотрим тот же самый альбом с фотографиями, но некоторые фотографии в каждом альбоме могут быть помечены как приватные, то есть видны только «друзьям» создателя альбома, в этом случае  получаем следующее взаимодействие:



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

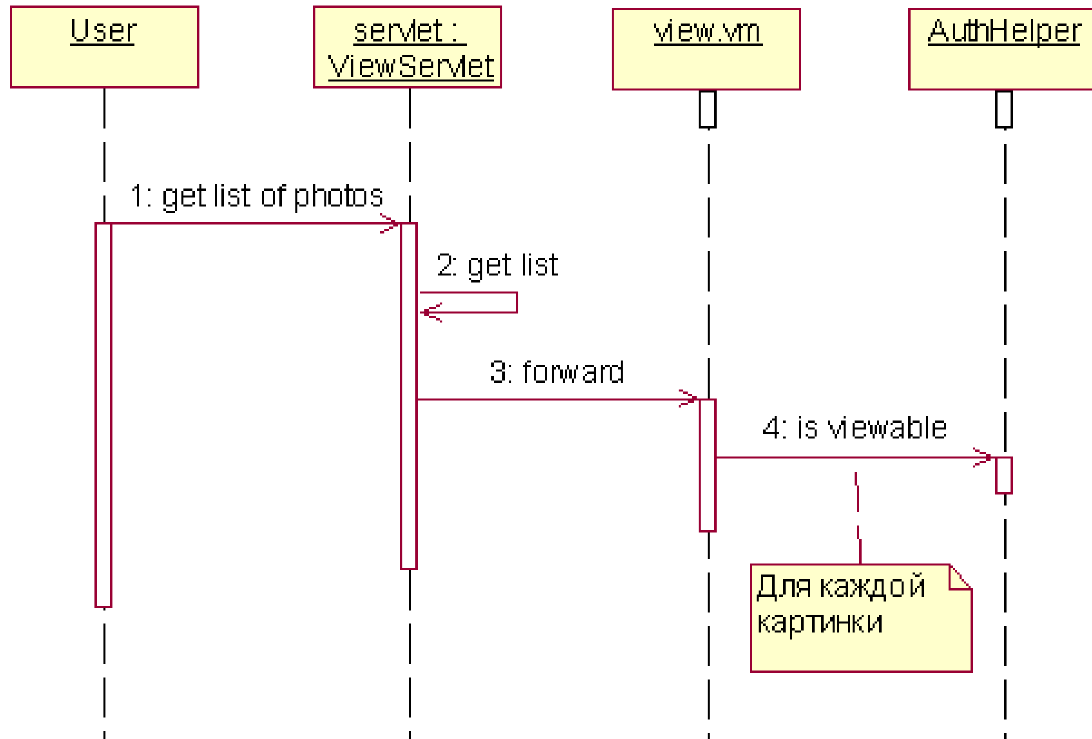


**Рисунок 9. View private pictures sequence.**

Как видно, здесь мы опять получили зависимость между слоем отображения и слоем авторизации. Кроме того, опять же необходимо контролировать, что все вызовы к методам возвращающим «секретные» данные должны быть окружены проверками пользователями, что очень трудно проконтролировать (представьте себе какой урон безопасности системы может нанести не окружённый проверками метод).

## 5.5. Аудит

Реализации аудита могут существенно отличаться, например можно фиксировать каждый вызванный «контроллер» с помощью добавления в каждый метод контроллера вызова метода записи в журнал. Или можно отслеживать посетителей и их путь по сайту с помощью сессии и специального фильтра (такой подход реализован в Open Symphony Click Stream) но возникает проблема установления соответствия между выполнением бизнес-логики и URL на сайте. В любом случае вызовы системы аудита



*Copyright © 1999 - 2007, IT-archiv. All rights reserved.*

распределяются по всему коду системы, и тем самым существенно затрудняют сопровождение системы и её эволюцию.

### 5.6. Проблемы

В рассмотренной реализации защиты приложения обычными способами явно видны следующие недостатки:

- Реализация защиты очень сильно привязана к конкретному приложению.
- Так как код проверок внедряется в код системы то никак нельзя надёжно гарантировать, что все места раскрытия секретных данных окружены такими проверками.
- Код с внедрёнными вызовами системы авторизации трудно расширять и тестировать.

Очевидно, что все вышеперечисленные проблемы возникают из-за того, что основная функциональность системы переплетается с функциональностью реализующую защиту. Схожие проблемы всегда существуют при создании сложных информационных систем, и происходит это из-за того, что доступные ООП средства не позволяют нам удобно и просто совмещать пересекающиеся функциональности.

## 6. Реализация защиты средствами АОП

Термин аспектно-ориентированное программирование впервые был упомянут в работе Gregor Kiczales et al. "Aspect-oriented programming" в 1997 году (здесь перевод на русский язык). АОП ставит своей целью разработать механизм реализации сквозной функциональности в ООП системах. АОП не в коей мере не заменяет собой ООП, данная техника всего лишь достраивает концепции ООП. Для реализации примеров статьи использовался AspectJ — реализация АОП для Java. В общих словах АОП предоставляет новый механизм композиции отличный от имеющихся в ООП. Для примера рассмотрим термины, в которых работает AspectJ:

- **JoinPoint** — определённая точка в выполнении программы, это может быть выполнение метода, изменение атрибута класса, вызов метода, выбрасывание исключения и т.д.
- **Pointcut** — набор (0..N) точек выполнения программы, удовлетворяющих определенному условию. Например, подобным срезом точек может являтся



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

выполнение всех методов начинающихся с «get» классов определённого пакета, или вызов методов классов реализующих некоторый интерфейс.
- **Advice** — Java код выполняемый до (before advice), после (after advice) или вместо (around advice) каждой точки выполнения входящих в определённый pointcut.
- **Aspect** — модуль в терминах АОП, аналог класса в Java может содержать публичные/приватные pointcut и advice кроме того обычные методы класса, могут наследоваться и быть абстрактными.
- **Introduction** — метод изменения структуры наследовании и реализаций существующей системы, применяется например для того что бы добавить дополнительный интерфейс к существующему классу, или изменить цепь наследников.

В общем, аспекты добавляют дополнительную функциональность в точки выполнения программы (pointcut) через advice. Важно то, что pointcut могут собирать не только точки выполнения, но и контекст в которых эти точки находятся, например, если pointcut определяет все вызовы методов `someMethod` всех объектов класса `SomeClass`, то контекст выполнения это:

- Объект, метод которого вызван,
- Параметры метода,
- Объект, вызвавший метод.

## 6.1. Идентификация/аутентификация

Очевидно, что аутентификация нужна не для всех областей нашего сайта, следовательно, нам необходимо сделать следующее — перед вызовом контроллеров, для которых аутентификация необходима, мы будем проверять пользователя, и в случае если он не аутентифицирован, то перенаправляем запрос на страницу с формой аутентификации.

Для того, что бы реализовать это в АОП нам необходимо сделать следующее:

1. Создаём pointcut, определяющий те контроллеры, вызов которых может осуществлять только аутентифицированный пользователь. Данный pointcut должен включать в себя HttpRequest необходимый для проверки аутентификации.
2. Создаём before advice на основе созданного pointcut который будет проверять HttpRequest на наличие аутентифицированного пользователя. Если таковой не



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

найден то advice будет выбрасывать AuthnificationException.

3. Создаём AuthnificationException наследуясь от RuntimeException. Очень важно что мы создаём именно unchecked exception так как компилятор позволяет не объявлять методы, которые могут выбрасывать такой exception.
4. Теперь нам необходимо отловить выброшенный Exception и сделать forward на login.vm, для этого нам необходимо выбрать точку в программе, которая даёт нам доступ к ServletContext для доступа к RequestDispatcher, после чего окружить эту точку around advice, который содержит proceed метод окружённый try/catch конструкцией.

Созданный аспект приведён ниже:

```
package aop.example;

import aop.example.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

/**
 * Аутентификационный аспект
 * обеспечивает проверку того что все контроллеры
 * нуждающиеся в пользователе его получат :)
 * @author Zubairov Renat
 */
public aspect AuthenticationAspect {

        // pointcut определяющий метод где мы будем отлавливать исключение
        // если пользователь не аутентифицирован
        // в нашем случае это метод фильтра
        public pointcut doFilterMethod(ServletRequest srequest,
          ServletResponse sresponse, EntranceFilter filter) :
        execution(void aop.example.EntranceFilter.doFilter
         (ServletRequest, ServletResponse, FilterChain))
           && args(srequest, sresponse, FilterChain) && this(filter);


        // pointcut определяющий методы для которых необходима аутентификация
        // в нашем случае это метод сервлета ViewServlet doGet
        // так же захватываем контекст а именно HttpServletRequest
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
        public pointcut authenticationNeeded(HttpServletRequest request) :
                execution(* aop.example.ViewServlet.doGet(
                HttpServletRequest, HttpServletResponse))
                   && args(request, HttpServletResponse);

        /**
         * Advice запускающийся перед методами нуждающимися в аутентификации
         * и если пользователь не найдён в сесси, то выбрасывающий исключение
         */
        before(HttpServletRequest request) : authenticationNeeded(request) {
                // В нашем случае только тогда когда есть хоть один параметр
                // у сервлета, только тогда необходима аутентификация
                if (request.getParameterMap().size() > 0) {
                        if (request.getSession().getAttribute(EntranceFilter.USER_KEY)
                                throw new AuthenticationException("User not logged in")
                        }
                        if (!(request.getSession().getAttribute(EntranceFilter.USER_KEY
                                throw new AuthenticationException("Anonymous user acces
                        }
                }
        }

        /**
         * Around advice (окружает pointcut — метод фильтра doFilter)
         * ловящий исключение и отравляющий запрос пользователя
         * на страницу с логином если исключение вылетело
         */
        void around(ServletRequest srequest, ServletResponse sresponse, EntranceFilter
                        throws IOException, ServletException
                          : doFilterMethod(srequest, sresponse, filter) {

                try {
                        // Выолняем метод
                        proceed(srequest, sresponse, filter);
                } catch (AuthenticationException e) {
                        // Сообщение об ошибке если вдруг понадобиться
                        srequest.setAttribute("error_message", e.getMessage());
                        // Отправляем на страницу с логином
                        filter.getConfig().getServletContext().getRequestDispatcher("lo
                        forward(srequest, sresponse);
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
                    }
          }

}
```

Заметим что данное решение имеет следующие преимущества:

- Используя мощный синтаксис определения pointuct, мы можем гарантировать, что все контроллеры, которые требуют аутентифицированный доступ, будут доступны только зарегистрированным пользователям. Например, мы можем создать интерфейс `AuthnificationNeeded` без методов, и определить pointcut, который будет включать в себя все классы, реализующие данный интерфейс. Можно определить pointcut, который будет включать в себя все классы в некотором пакете, или, например, по маске имени класса.
- Действие аспекта будет автоматически распространяться на все новые контроллеры, добавленные в систему в ходе эволюции (при условии, что они будут созданы по определённым в проекте правилам).
- Никаких изменений кода контроллеров не потребовалось, то есть контроллеры совершенно не осведомлены об особенностях аутентификации приложения.
- Вся логика аутентификации как генерация исключения, и перенаправление запроса реализована в одном аспекте, а не разбросана по коду системы на разных уровнях.

В тестовой системе попробуйте, как анонимный пользователь, добавить альбом — попытка добавления альбома приведёт вас на страницу авторизации, т.к. добавлять альбом могут только авторизованные пользователи. К сожалению, ссылки удаления альбомов не показываются, т.к. система уже содержит все аспекты которые мы будем реализовывать далее.

## 6.2. Авторизация

Наиболее интересная часть решения. Идеальное решение авторизации должно быть слабо связанно со всей системой, и должно реализовывать два различных типа авторизации:

- Пост-проверка — запрет не авторизованных действий. Например, запрет удаления альбома или запрет на чтение поля объекта.
- Пред-проверка — оповещение о том, что конкретное действие не разрешено. Например, слой отображения должен быть знать отображать ли ссылку «удалить»



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

около альбома, то есть можно ли удалить данный альбом.

Решение в терминах АОП мы разберём последовательно для различных типов.

### 6.2.1. Пост-проверка

Перед каждым методом, считывающим или записывающим данные, нам необходимо проверять пользователя, и в случае если операция не доступна, мы будем показывать пользователю сообщение об ошибке. В нашем примере каждый альбом имеет владельца, следовательно, правило простое — пользователь может удалять только принадлежащие ему альбомы. Определим три различных действия, нуждающиеся в авторизации:

- Чтение — вызов любых методов начинающихся с get (по умолчанию считаем, что наша модель удовлетворяет требованиям JavaBeans по именованию методов доступа к данным класса).
- Добавление — добавление нового альбома. В нашей системе это метод `addAlbum` в классе `AlbumList`.
- Удаление — удаление альбома. В нашей системе это метод `removeAlbum` в классе `AlbumList`.

Т.к. мы не будем реализовывать возможность редактирования имени альбома, то методы изменения данных нас не интересуют, хотя в реальных системах этот тип доступа, безусловно, нуждается в авторизации. Перед вызовом методов каждого типа мы будем обращаться к системе авторизации для проверки. Посмотрим более детально на ход работы системы:



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

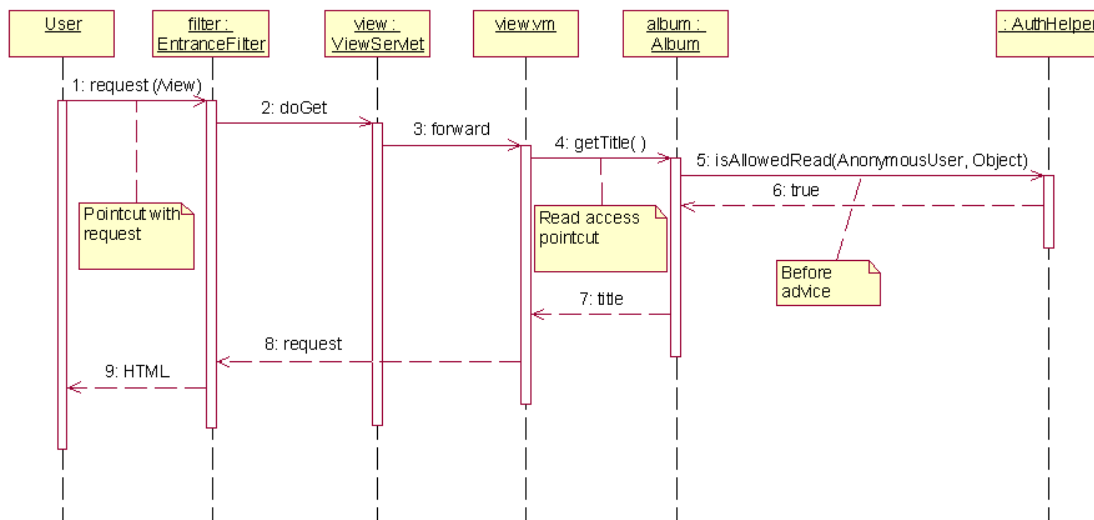


**Рисунок 10. AOP Auth sequence.**

На диаграмме изображена последовательность действий по отображению списка альбомов. На первом шаге пользователь запрашивает страницу /view. Запрос с начала поступает на созданный фильтр `EntranceFilter`, после чего обрабатывается сервлетом. Так как запрос без параметров, то сервлет ничего не делает и направляет запрос на слой отображения view.vm. Скрипт отображения вынимает из сессии список альбомов и выводит их названия.

Здесь видно, что «Before advice» созданный на основе «Read access pointcut» перехватывает операцию `getTitle` объекта модели (вызов номер 5 уже производится аспектом) и взывает класс, инкапсулирующий правила авторизации для проверки операции чтения: `boolean isAllowedRead(AnonymousUser, Object)`. Как видно из сигнатуры метода для проверки нам необходим текущий пользователь. Здесь нам на помощь приходит паттерн "червоточина" (Wormhole) (Статья на английском [Aspect-oriented refactoring] раздел "Replace argument trickle by wormhole"). В двух словах, этот паттерн позволяет нам неявно передавать параметры контекста одного из одной части системы в другую часть, находящуюся, например, в одном потоке выполнения. Реализуется он следующим образом:

1. Создаём pointcut в точках выполнения методов чтения данных таким образом, что бы захватить объект, у которого этот метод вызван. На рисунке такой метод `getTitle()`.



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

2. Создаём pointcut который бы гарантированно был на пути следования запроса к системе. В нашем примере это метод `doFilter(ServletRequest request, ServletResponse response, FilterChain chain)` класса `EntranceFilter`. Из параметров метода захватываем `ServletRequest`.
3. Создаём третий pointcut, который объединяет первый и второй, при условии, что вызов метода чтения производится в потоке выполнения второго pointcut, и захватываем все существующие параметры.

В результате мы получили pointcut, который даёт нам доступ как к объекту текущего запроса, так и к объекту над которым производится интересующее нас действие. Обратите внимание, что перехват события будет производиться только в потоке выполнения, включающем в себя фильтр, то есть если мы, например, будем запускать unit тесты модели, тестирующие классы напрямую, то никакой проверки происходить не будет (что естественно, т.к. не откуда будет взять пользователя), что сильно упрощает тестирование.

Итак, с перехватом выполнения методов разобрались, теперь необходимо обработать исключительные ситуации, то есть когда пользователь не авторизован выполнить действие. В данном случае before advice выбросит unchecked exception `AuthorizationException` и тем самым не даст выполниться не разрешённому методу. Для того, что бы отобразить пользователю цивилизованное сообщение об ошибке, или обработать эту ситуацию другим способом, мы перехватываем исключение, точно так же как и в предыдущем аспекте.

Полученный код аспекта приведён ниже:

```
package aop.example;

import aop.example.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

/**
 * Авторизационный аспект
 * @author Zubairov Renat
 */
public aspect AuthorizationAspect {
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
/**
 * pointcut включающий в себя метод с ServletRequest для
 * того что бы потом можно было бы получить его в объединении
 * с другим pointcut (паттерн "червоточина")
 */
pointcut requestMethod(ServletRequest request) :
execution(* aop.example.EntranceFilter.doFilter(ServletRequest, ServletResponse
&& args(request, ServletResponse, FilterChain);

/**
 * pointcut определяющий метод фильтра в котором мы будем отлавливать
 * исключение
 */
public pointcut doFilterMethod(ServletRequest srequest, ServletResponse srespon
EntranceFilter filter) :
execution(void aop.example.EntranceFilter.doFilter(ServletRequest, ServletRespo
&& args(srequest, sresponse, FilterChain) && this(filter);


// все методы производящие чтение информации объектов модели
pointcut readMethods(Object object) : execution (public * aop.example.model.*.g
 && this(object);

// все методы производящие добавление объектов модели
pointcut addMethods(Object object) : execution (public * aop.example.model.*.ad
 && this(object);

// все методы производящие удаление объектов модели
pointcut deleteMethods(Album album) : execution (public * aop.example.model.Alb
 deleteAlbum(Album)) && args(album);

// Вызов методов чтения произошедшие в потоке выполнения
// следующим за вызовом метода фильтра
// мы объеденили два pointcut — реализация паттерна "червоточина"
pointcut readAccess(ServletRequest request, Object object) :
cflow(requestMethod(request)) && readMethods(object);

// то же самое только для добавления
pointcut addAccess(ServletRequest request, Object object) :
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
        cflow(requestMethod(request)) && addMethods(object);

        // то же самое только для удаления
        pointcut deleteAccess(ServletRequest request, Album album) :
        cflow(requestMethod(request)) && deleteMethods(album);
        /**
         * Before advice проверки на чтение
         */
        before(ServletRequest request, Object object) : readAccess(request, object) {
                if (!AuthHelper.isAbleToRead(extractUser(request), object)) {
                        throw new AuthorizationException("Read access not allowed");
                        }
                }

        /**
         * Before advice проверки на добавление
         */
        before(ServletRequest request, Object object) : addAccess(request, object) {
                if (!AuthHelper.isAbleToAdd(extractUser(request), object)) {
                        throw new AuthorizationException("Add access not allowed");
                        }
                }

        /**
         * Before advice проверки на удаление
         */
        before(ServletRequest request, Album album) : deleteAccess(request, album) {
                if (!AuthHelper.isAbleToDelete(extractUser(request), album)) {
                        throw new AuthorizationException("Delete access not allowed");
                        }
                }

        /**
         * Around advice отлавливающий исключение
         * и отправляющий запрос на страницу с ошибкой
         */
        void around(ServletRequest srequest, ServletResponse sresponse, EntranceFilter
        throws IOException, ServletException
        :
        doFilterMethod(srequest, sresponse, filter) {
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
                try {
                        // выполняем метод фильтра
                        proceed(srequest, sresponse, filter);
                        } catch (AuthorizationException e) {
                        // ловим исключение
                        srequest.setAttribute("error_message", e.getMessage());
                        // вперёд на страницу с сообщением об ошибке
                        filter.getConfig().getServletContext().getRequestDispatcher("er
                        forward(srequest, sresponse);
                        }
                }

        /**
         * Приватная фунция которая вынимает пользователя из запроса
         */
        private AnonymousUser extractUser(ServletRequest request) {
                return (AnonymousUser)((HttpServletRequest) request).getSession().
                getAttribute(EntranceFilter.USER_KEY);
                }

}
```

Обратите внимание, что для удаления альбома мы используем немного отличный от остальных pointcut, это связанно с тем, что метод удаляющий альбомы принадлежит не классу `Album`, а классу `AlbumList`, кроме того, альбом, подлежащий удалению, передаётся как параметр метода. Как видно из решения язык определения pointcut AspectJ с лёгкостью справился и с такой задачей.

Для примера попробуйте ввести в строку браузера следующий запрос:

```
http://localhost:8080/view?delete=
  Picture%20of%20%3Cb%3Euser2%3C/b%3E
```

Будучи не зарегистрированным в системе (должна отобразиться страница с логином), или под пользователем User1 (должна отобразиться страница с ошибкой, т.к. производится попытка удалить альбом, не принадлежащий текущему пользователю).

### 6.2.2. Пред-проверка

Для того, что бы реализовать оповещение слоя отображения о действиях которые



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

разрешено производить с объектом модели, мы создадим дополнительный интерфейс который будет содержать 3 метода — **boolean** `isReadable()`, **boolean** `isDeletable()` и **boolean** `isAddable()` после чего создадим в классе `Album` методы реализующие данный интерфейс и возвращающие всегда true. Слой отображения будет отображать ссылку «удалить» только если метод `isDeletable()` даст добро. А в нашем авторизационном аспекте создадим pointuct перехватывающие соответствующие методы всех классов реализующих данный интерфейс. После чего создадим around advice, который будет возвращать результат вызова системы авторизации. Захват параметров производится точно так же, как это было сделано при пост-проверке.

Изменённый аспект:

```
package aop.example;

import aop.example.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.IOException;

/**
 * Авторизационный аспект
 * @author Zubairov Renat
 */
public aspect AuthorizationAspect {

        /**
         * pointcut включающий в себя метод с ServletRequest для
         * того что бы потом можно было бы получить его в объединении
         * с другим pointcut (паттерн "червоточина")
         */
        pointcut requestMethod(ServletRequest request) :
                execution(* aop.example.EntranceFilter.doFilter(ServletRequest,
                ServletResponse, FilterChain))
                && args(request, ServletResponse, FilterChain);

        /**
         * pointcut определяющий метод фильтра в котором мы будем отлавливать
         * исключение
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
 */
public pointcut doFilterMethod(ServletRequest srequest,
 ServletResponse sresponse, EntranceFilter filter) :
        execution(void aop.example.EntranceFilter.doFilter(ServletRequest,
        ServletResponse, FilterChain))
                && args(srequest, sresponse, FilterChain) && this(filter);


// все методы производящие чтение информации объектов модели
pointcut readMethods(Object object) : execution (public * aop.example.model.*.g
  && this(object);

// все методы производящие добавление объектов модели
pointcut addMethods(Object object) : execution (public * aop.example.model.*.ad
  && this(object);

// все методы производящие удаление объектов модели
pointcut deleteMethods(Album album) : execution (public *
  aop.example.model.AlbumList.deleteAlbum(Album))
                                      && args(album);

// методы проверки на доступность чтения (пред-проверка)
pointcut controlledRead(Object object) : execution(public
  boolean aop.example.model.Controlled+.isReadable())
                                      && this(object);

// методы проверки на доступность добавления (пред-проверка)
pointcut controlledAdd(Object object) : execution(public
  boolean aop.example.model.Controlled+.isAddable())
                                      && this(object);

// методы проверки на доступность удаления (пред-проверка)
pointcut controlledDelete(Object object) : execution(public
  boolean aop.example.model.Controlled+.isDeletable())
                                      && this(object);

// Вызов методов чтения произошедшие в потоке выполнения
// следующим за вызовом метода фильтра
// мы объеденили два pointcut — реализация паттерна "червоточина"
pointcut readAccess(ServletRequest request, Object object) :
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
            cflow(requestMethod(request)) && readMethods(object);

    // то же самое только для добавления
    pointcut addAccess(ServletRequest request, Object object) :
            cflow(requestMethod(request)) && addMethods(object);

    // то же самое только для удаления
    pointcut deleteAccess(ServletRequest request, Album album) :
            cflow(requestMethod(request)) && deleteMethods(album);

    // пред-проверка на чтение
    pointcut readCheck(ServletRequest request, Object object) :
            cflow(requestMethod(request)) && controlledRead(object);

    // пред-проверка на добавление
    pointcut addCheck(ServletRequest request, Object object) :
            cflow(requestMethod(request)) && controlledAdd(object);

    // пред-проверка на удаление
    pointcut deleteCheck(ServletRequest request, Object object) :
            cflow(requestMethod(request)) && controlledDelete(object);

    /**
     * Around advice отлавливающий исключение
     * и отправляющий запрос на страницу с ошибкой
     */
    void around(ServletRequest srequest, ServletResponse sresponse, EntranceFilter
                    throws IOException, ServletException
                    : doFilterMethod(srequest, sresponse, filter) {

            try {
                    // выполняем метод фильтра
                    proceed(srequest, sresponse, filter);
            } catch (AuthorizationException e) {
                    // ловим исключение
                    srequest.setAttribute("error_message", e.getMessage());
                    // вперёд на страницу с сообщением об ошибке
                    filter.getConfig().getServletContext().getRequestDispatcher("er
                    forward(srequest, sresponse);
            }
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
    }


    /**
     * Before advice проверки на чтение
     */
    before(ServletRequest request, Object object) : readAccess(request, object) {
            if (!AuthHelper.isAbleToRead(extractUser(request), object)) {
                    throw new AuthorizationException("Read access not allowed");
            }
    }

    /**
     * Before advice проверки на добавление
     */
    before(ServletRequest request, Object object) : addAccess(request, object) {
            if (!AuthHelper.isAbleToAdd(extractUser(request), object)) {
                    throw new AuthorizationException("Add access not allowed");
            }
    }

    /**
     * Before advice проверки на удаление
     */
    before(ServletRequest request, Album album) : deleteAccess(request, album) {
            if (!AuthHelper.isAbleToDelete(extractUser(request), album)) {
                    throw new AuthorizationException("Delete access not allowed");
            }
    }

    /**
     * Around advice пред-проверки, здесь мы игнорируем возвращаемое методом
     * значение, и всё время возвращаем то которое удовлетворяет правилам
     * авторизации
     * Мы не обрабатываем остальные пред-проверки т.к. по умолчанию любой может
     * читать, и все аутентифицированные пользователи могут добавлять
     */
    boolean around(ServletRequest request, Object object) : deleteCheck(request, ob
            return AuthHelper.isAbleToDelete(extractUser(request), object);
    }
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
        /**
         * Приватная фунция которая вынимает пользователя из запроса
         */
        private AnonymousUser extractUser(ServletRequest request) {
                return (AnonymousUser)((HttpServletRequest) request).getSession().
                getAttribute(EntranceFilter.USER_KEY);
        }

}
```

Достоинства АОП решения:

- Как видно из определения аспектов `readMethods`, `addMethods`, `controlledRead`, `controlledAdd` и `controlledDelete` мы не ссылаемся на определённые классы, следовательно, авторизация будет автоматически распространяться на все новые классы модели помещённые в пакет `model` и реализующие интерфейс `Controllable`.
- Как выбрасывание исключения, так и его обработка реализована в пределах одного аспекта, что упрощает дальнейшую эволюцию и сопровождение системы.
- Все вызовы методов классов модели (созданные согласно правилам) будут объектом для применения системы авторизации.
- Слой отображения не зависит от системы авторизации.
- Система полностью работоспособна без применения аспекта авторизации, что говорит об отсутствии связи между системой в целом и аспектом авторизации (в свою очередь аспект авторизации зависит от системы, но эта связь не существенна т.к. происходит лишь на уровне определения pointcut).

Для наглядного примера можно создать несколько новых альбомов под одним из пользователей, а потом зарегистрироваться под другим. Ссылки «удалить» будут проставлены только у «своих» альбомов, кроме того попытки удаления чужих альбомов через непосредственное редактирование get запросов приведут к странице с ошибкой.

## 6.3. Целостность

Проблемы целостности, как было сказано выше, достаточно сложны для реализации, но всё же необходимо отметить, что паттерн "червоточина", например, прекрасно



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

подходит для распространения контекста транзакции

## 6.4. Секретность

Усовершенствовав приведенный выше авторизационный аспект, мы с лёгкостью можем точно разграничить доступ даже к отдельным полям объекта данных, тем самым реализовывать такую систему доступа данным, которая подходит именно для нашего приложения. Достоинства АОП очевидны, т.к. обычно никакого изменения в исходной системе не производится.

Кроме того, AspectJ даёт нам ещё один очень интересный инструмент — ограничения на статическую структуру приложения. Так как внедрение аспектов производится во время компиляции специальным компилятором, то существует возможность создать определённые правила, накладывающие ограничения на структуру и внутреннее взаимодействие объектов системы (статическую структуру). Например, мы можем определить правило — вызовы методов объектов модели приводящих к изменению состояния доступны только из «контроллеров». Или, например, что бы классы модели не были связанны с классами уровня контроллеров. В случае не соблюдения правил AspectJ компилятор может либо просто выдать предупреждение, либо прервать компиляцию с фатальной ошибкой (в зависимости от определения правила).

## 6.5. Аудит

Аудит, пожалуй, является самым наглядным примером применения АОП, т.к. его реализация обычным способом обычно обязывает разработчиков добавлять один и тот же код занесения в журнал во все части системы. Например, для ведения журнала действий каждого пользователя нам необходимо включить в каждый контроллер вызов метода ответственного за занесение данных в журнал событий, опять же гарантии что такие вызовы включены в каждый контроллер, обеспечить трудно. Или предположим, что первая версия системы поддерживала простой механизм аудита, в то время как для следующей версии, требования усложнились, и теперь необходимо изменить все включения вызовов старого типа на вызовы новой системы журналирования.

Красота АОП решения в том, что сквозная функциональность аудита изолирована в пределах аспекта. Все изменения в типе и детализации событий могут быть с



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

лёгкостью проведены не затрагивая основную функциональность.

Для примера рассмотрим простейший аспект, который будет перехватывать все вызовы контроллеров, определяя пользователя, и записывать информацию в журнал.

Создадим новый аспект, с pointcut перехватывающий все выполнения контроллеров, и захватывающий объект запроса. После чего на основе созданного pointcut, напишем before advice выводящий сообщение на `System.out`.

Код аспекта:

```
package aop.example;

import org.aspectj.lang.*;
import javax.servlet.http.*;

/**
 * Аспект обеспечивающий аудит
 * @author Zubairov Renat
 */
public aspect AuditAspect {

        /**
         * Все методы которые необходимо заносить в журнал
         * Так же собираем запрос для извлечения от туда сесии
         */
        public pointcut controllerMethods(HttpServletRequest request) :
                // Выполнение всех методов параметрами которых являются
                // HttpServletRequest и HttpServletResponse
                // для всех классов наследников HttpServlet
                execution(* javax.servlet.http.HttpServlet+.*
                 (HttpServletRequest, HttpServletResponse))
                        && args(request, HttpServletResponse);


        /**
         * Advice который обеспечивает печать на стандартный вывод
         * информации какой метод будет выполнен и пользователь
         * который выполняет метод
         */
```



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

```
        before(HttpServletRequest request) : controllerMethods(request) {
                // Собираем статическую информацию о точке вплетения
                Signature sig = thisJoinPointStaticPart.getSignature();
                // Выводим доступную информацию
                System.out.println("User " +
                        request.getSession().getAttribute
                        (EntranceFilter.USER_KEY).toString() +
                        " executing " + sig.getDeclaringType().getName() +
                        "." + sig.getName());
        }

}
```

Обратите внимание, что в теле advice мы используем статическую информацию о точке вплетения, для того, что бы отобразить имя класса объекта и название метода.

В нашем примере стандартный вывод системы будет отображать строки вида:

```
User Anonymous executing aop.example.LoginServlet.doGetUser user1 executing aop.example
```

## 7. Заключение

В данной статье я попытался показать достоинства аспектно-ориентированного программирования на примере реализации системы защиты WEB приложения и наглядно продемонстрировать, как можно применять AspectJ на практике.

Итак, АОП при правильном использовании может следующее:

- Уменьшить объем кода системы (следовательно, снизить вероятность программных ошибок)
- Улучшить дизайн системы с точки зрения реализации сквозной функциональности, улучшить модульность.
- Упростить код системы, благодаря локализации кода, не относящегося к основной функциональности.
- Упростить тестирования системы (можно тестировать различные аспекты отдельно, а только потом вплетённые в систему). Улучшить управляемость кода, как следствие простота эволюции и сопровождения.
- Увеличить количество повторно используемых модулей благодаря слабой связности подсистем.



Copyright © 1999 - 2007, IT-archiv. All rights reserved.

С другой стороны не правильное применение АОП может привести к следующим последствиям:

- Затруднения в ходе отладки (Какой из аспектов выполняется сейчас?)
- Трудности с пониманием концепции зачастую приводят к грубым ошибкам в дизайне аспектов.
- Новая технология всегда риск.

## 7.1. Ссылки

- Приложение описанное в статье с AIP документацией и исходным кодом aopexample.zip
- AOP по русски http://www.javable.com/columns/aop/
- Статья Валентина Павлова http://www.javable.com/columns/aop/workshop/01/
- Перевод на русский язык части статьи с первым упоминанием термина АОП http://www.javable.com/columns/aop/workshop/02/
- AspectJ http://www.aspectj.org/
- Несколько параграфов из книги "AspectJ in action" доступных для свободного скачивания http://www.theserverside.com/resources/article.jsp?l=AspectJReview
- AOP blog http://www.darkwolf.ws/blog/blojsom/aop/
- Eclipse бесплатное IDE работающее в том числе и с AspectJ http://www.eclipse.org/eclipse/index.html
- AspectJ plugin для Eclipse http://www.eclipse.org/ajdt/



Copyright © 1999 - 2007, IT-archiv. All rights reserved.