**Spring AOP**

**Книги**

AspectJ in Action: Enterprise AOP with Spring Applications стр.3

Spring in action. Third Edition. Стр.198

**Сайты**

Знакомство с AOP

<https://habr.com/ru/post/114649/>

Introduction to Spring AOP

<https://www.baeldung.com/spring-aop>

Aspect Oriented Programming with Spring

<https://docs.spring.io/spring/docs/2.5.x/reference/aop.html>

* **Причины использования AOP.**

Рассматривая web-сервис, реализующий чтение информации о некотором объекте по её идентификатору. Данный метод занимает одну строчку чистой бизнес-логики. Для такого метода необходимо задуматься о добавлении логирования, реализовать обработку исключений, добавить проверку прав доступа т.е. безопасность. В некоторых случаях имеет смысл кешировать результат работы.

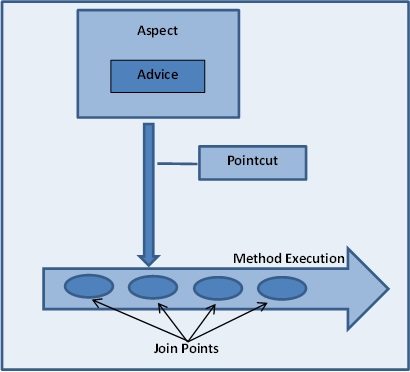
В ходе доработок получается метод в 10 раз превышающий исходный размер. Объём бизнес-логики в нём не изменился – это всё та же 1 строка. Остальной код реализует некоторую общую служебную функциональность приложения: логирование, обработку ошибок, проверку прав доступа, кеширование и так далее.

Переплетение бизнес-логики со служебным функционалом не является большой проблемой, пока приложение невелико. Однако, чем сложнее становится программа, тем более тщательно следует подходить к её архитектуре в целом и выделении общей функциональности в частности. Такую функциональность называют «сквозной» или «разбросанной», в виду того, что её реализация действительно разбросана по разным частям приложения. Примерами сквозной функциональности могут служить:

* логирование,
* обработка транзакций,
* обработка ошибок,
* авторизация и проверка прав,
* кэширование,
* элементы контрактного программирования.

Решением вышеперечисленных проблем занимается аспектно-ориентированное программирование. Основной задачей аспектно-ориентированного программирования (АОП) является модуляризация сквозной функциональности, выделение её в аспекты. Для этого языки, поддерживающие концепцию АОП, реализуют средства для выделения сквозной функциональности.

* **Терминология АОП**



https://cdn2.howtodoinjava.com/wp-content/uploads/2015/01/spring-aop-diagram.jpg

*Advice(Совет)*

Аспекты также имеют свою цель – работу, которую они призваны делать. В терминах AOP работа аспекта называется совет.

Совет определяет, что и когда делает аспект. В дополнение к описанию работы, выполняемой аспектом, совет учитывает, когда следует ее выполнять.

Аспекты Spring могут работать с пятью типами советов:

* до – работа выполняется перед вызовом метода;
* после – работа выполняется после вызова метода, независимо от результата;
* после успешного вызова – работа выполняется после вызова метода, если его выполнение завершилось успешно;
* после исключения – работа выполняется после того, как вызванный метод возбудит исключение;
* вокруг – аспект обертывает метод, обеспечивая выполнение некоторых операций до и после вызова метода.

*Join points (Точки сопряжения)*

Также и приложение может иметь тысячи точек применения совета. Эти точки известны как точки сопряжения (join points). Точка сопряжения – это точка в потоке выполнения приложения, куда может быть внедрен аспект. Это может быть вызов метода, возбуждение исключения или даже изменение поля. Все это – точки, куда может быть внедрен аспект для добавления новой особенности поведения.

*Points cut (Срезы множества точек сопряжения)*

Аналогично от аспекта не требуется воздействовать на все точки сопряжения в приложении. Срезы множества точек сопряжения помогают сузить множество точек для внедрения аспекта.

Если совет отвечает на вопросы что и когда, то срезы множества точек сопряжения отвечают на вопрос «где». Срез содержит одну или более точек сопряжения, куда должны быть вплетены советы. Часто срезы множества точек сопряжения определяются за счет явного указания имен классов и методов или через регулярные выражения, определяющие шаблоны имен классов и методов. Некоторые фреймворки, поддерживающие AOP, позволяют создавать срезы множества точек сопряжения динамически, определяя необходимость применения совета, опираясь на решения, принимаемые во время выполнения, такие как значения параметров метода.

*Aspects (Аспекты)*

Аспект объединяет в себе совет и срез множества точек сопряжения. Взятые вместе, они определяют все, что нужно знать об аспекте, – что он делает, где и когда.

*Introduction (Внедрение)*

Внедрение позволяет добавлять новые методы или атрибуты в существующие классы. Например, можно создать класс-совет Auditable, хранящий информацию о том, когда объект был изменен в последний раз. Это может быть очень простой класс, состоящий из единственного метода, например setLastModified(Date), и переменной экземпляра для хранения этой информации. В дальнейшем новый метод и переменная могут быть внедрены в существующие классы без их изменения, добавляя новые черты поведения и информацию.

*Weaving (Вплетение)*

Вплетение – это процесс применения аспектов к целевому объекту для создания нового, проксированного объекта. Аспекты вплетаются в целевой объект в указанные точки сопряжения. Вплетение может происходить в разные моменты жизненного цикла целевого объекта.

* Во время компиляции – аспекты вплетаются в целевой объект, когда тот компилируется. Это требует специального компилятора, такого как AspectJ, вплетающего аспекты на этапе компиляции.
* Во время загрузки класса – вплетение аспектов выполняется в процессе загрузки целевого класса виртуальной машиной JVM. Это требует специального загрузчика, который дополняет байт-код целевого класса перед внедрением его в приложение, например, механизм load-time weaving (LTW) в AspectJ 5.
* Во время выполнения – вплетение аспектов производится вовремя выполнения приложения. В этом случае контейнер AOP обычно динамически генерирует объект с вплетенным аспектом, представляющий целевой объект.

*Target (Цель)*

Цель – объект, к которому будут применяться советы;

* **Описании имплементации принципа АОП.**

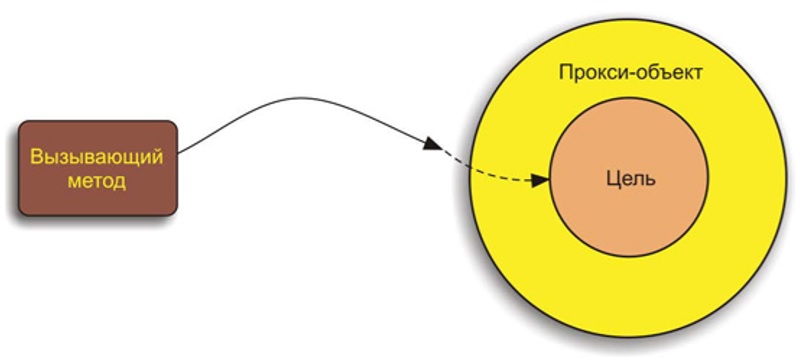
Не все фреймворки AOP равноценны. Они могут отличаться богатством моделирования точек сопряжения. Некоторые позволяют применять советы на уровне изменения полей, тогда как другие предлагают точки сопряжения на уровне вызовов методов. Они могут также отличаться порядком и особенностями вплетения аспектов. Но не эти особенности делают фреймворки фреймворками с поддержкой AOP, а возможность определения срезов множества точек сопряжения аспектов.

Для .NET – используется PostSharp. Для Java AOP доступен через проект AspectJ. Наиболее простая и проверенная реализация AOP – Spring AOP.

Тем не менее проекты Spring и AspectJ тесно взаимодействуют друг с другом, и реализация поддержки AOP в Spring многое заимствует из AspectJ. Фреймворк Spring поддерживает четыре разновидности AOP:

* классическое аспектно-ориентированное программирование на основе промежуточных объектов;
* аспекты, создаваемые с применением аннотаций @AspectJ;
* аспекты на основе POJO;
* внедрение аспектов AspectJ (доступно во всех версиях Spring).

При использовании фреймворка Spring, аспекты вплетаются в компоненты во время выполнения посредством обертывания их прокси-классами. Прокси-класс играет роль целевого компонента, перехватывая вызовы методов и передавая эти вызовы целевому компоненту.



*Аспекты Spring реализуются в виде прокси-объектов, обертывающих целевые объекты. Прокси-объект обрабатывает вызовы методов, выполняет дополнительную логику аспекта и затем вызывает целевой метод.*

Spring поддерживает точки сопряжения только для методов Как упоминалось выше, различные реализации AOP обеспечивают несколько моделей точек сопряжения. Так как поддержка AOP в Spring основана на использовании динамических прокси-объектов, точками сопряжения могут служить только методы, в отличие от некоторых других фреймворков AOP, таких как AspectJ и JBoss, где помимо методов роль точек сопряжения могут играть поля и конструкторы. Отсутствие в Spring возможности применения аспектов к полям препятствует созданию высокоизбирательных аспектов, например для отслеживания изменений полей в объектах. Невозможность применить аспект к конструктору также препятствует реализации логики, которая должна выполняться в момент создания экземпляра компонента. Однако возможности перехватывать вызовы методов должно быть достаточно для удовлетворения если не всех, то большинства потребностей. Если потребуется нечто большее, чем возможность перехвата вызовов методов, всегда можно воспользоваться расширением AspectJ.

Создаётся аспект – класс, помеченный аннотацией Aspect. Далее определяются среза точек соединения. В завершении объявляется совет, который выполняется вместо точек соединения, удовлетворяющих срезу.

В коде совета происходит получение информации о текущем методе (точке соединения), начала выполнения метода, непосредственный вызов запрошенного метода и возвращение результата работы.

* **Обоснование выбора Spring AOP.**

Spring Framework реализует ограниченную AOP-функциональность на чистом Java без использования сторонних библиотек с помощью создания прокси-объектов (JDK Dynamic Proxy, CGLIB). Другими словами, в Spring AOP можно использовать только точки соединения типа «выполнение метода». Однако, как показывает практика, данное ограничение не играет значительной роли, так как для решения большинства задач, требуется точки соединения именно этого типа.

Кроме того, Spring Framework поддерживает конфигурирование приложений c помощью @AspectJ аннотаций, а также интеграцию аспектов, скомпилированных непосредственно с помощью AspectJ.