**I Глава**

**1.1. Spring AOP**

**1.1.1. 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.

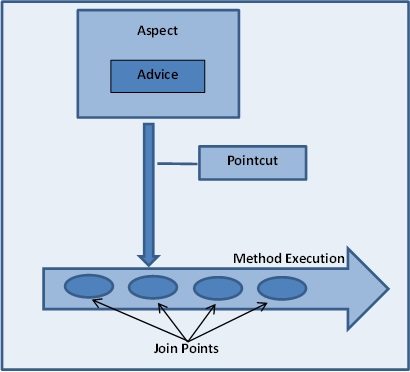
В ходе доработок получается метод в 10 раз превышающий исходный размер. Объём бизнес-логики в нём не изменился – это всё та же 1 строка. Остальной код реализует некоторую общую служебную функциональность приложения: логирование, обработку ошибок, проверку прав доступа, кеширование и так далее.

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.

**1.1.2. Терминология АОП**



*Рисунок 1.1. Основные компоненты Spring AOP*

*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;

**1.1.3. Описании имплементации принципа АОП.**

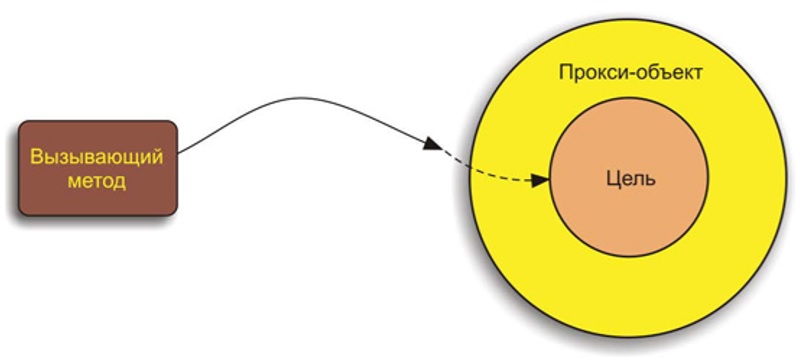
Не все фреймворки AOP равноценны. Они могут отличаться богатством моделирования точек сопряжения. Некоторые позволяют применять советы на уровне изменения полей, тогда как другие предлагают точки сопряжения на уровне вызовов методов. Они могут также отличаться порядком и особенностями вплетения аспектов. 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.



*Рисунок 1.2. 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.

**1.1.4. 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.

**1.2. TypeScript**

**1.2.1. Причины использования TypeScript.**

JavaScript — очень динамичный язык программирования, который многие противопоставляют С-подобным языкам. Его не типизированная природа и уникальное прототипное наследование послужили переходом большого количества разработчиков на JavaScript.

Динамически типизированные языки не требуют указывать тип, но и не определяют его сами. Типы переменных неизвестны до тех пор, пока они не проинициализированы . Большинство динамических языков выдадут ошибку, если типы используются некорректно (JavaScript — известное исключение; он пытается вернуть значение для любого выражения, даже когда оно не имеет смысла). При использовании динамически типизированных языков даже простая ошибка вида "a" + 1 может обернуться большой проблемой.

TypeScript больше оправдывает себя в крупных проектах. Это связано с тем, что разработка на нем занимает больше времени чем на JavaScript из-за того, что приходится помимо методов и классов описывать и их декларации. Но тем не менее, пока в JavaScript нет статической типизации, TypeScript является отличной альтернативой.

**1.2.2. Детальное описание технологии.**

TypeScript - это строго типизированный и компилируемый язык, чем, возможно, будет ближе к программистам Java, C# и других строго типизированных языков. Хотя на выходе компилятор создает все тот же JavaScript, который затем исполняется браузером, так как ориентируется прежде всего на стандарт ECMAScript 3, хотя TS также поддерживает и стандарты ECMAScript 5 и ECMAScript 2015 / 2017. Хотя в процессе разработки мы можем сами задать целевой стандарт ECMAScript.. В TS можно использовать все те конструкции, которые применяются в JS - те же операторы, условные, циклические конструкции. Более того код на TS компилируется в Javascript. В конечном счете, TS - это всего лишь инструмент, который призван облегчить разработку приложений. Однако строгая типизация уменьшает количество потенциальных ошибок, которые могли бы возникнуть при разработке на JavaScript.

Потенциал TypeScript-а позволяет быстрее и проще писать большие сложные комплексные программы, соответственно их легче поддерживать, развивать, масштабировать и тестировать, чем на стандартном JavaScript.

TypeScript возник из-за предполагаемых недостатков JavaScript в крупномасштабных приложениях как в Microsoft, так и у прочих пользователей JavaScript. Проблемы с разработкой сложных программ на JavaScript привели к необходимости облегчения разработки компонентов языка.

Разработчики TypeScript искали решение, которое не будет нарушать совместимость со стандартом и его кроссплатформенной поддержкой. Зная, что только стандарт ECMAScript предлагает поддержку в будущем для программирования на базе классов (Class-based programming), TypeScript был основан на этом предположении. Это привело к созданию компилятора JavaScript с набором синтаксических языковых расширений, увеличенным на основе предложения, которое трансформирует расширения в JavaScript. В этом смысле TypeScript является представлением того, что ожидать от ECMAScript 6. Уникальный аспект не в предложении, а в добавлении в TypeScript статической типизации, что позволяет статически анализировать язык, облегчая оснастки и IDE Для написания кода на языке TypeScript можно использовать любой самый простейший текстовый редактор. Многие текстовые редакторы и среды разработки, например, Visual Code Studio, Atom, Sublime, Visual Studio, Netbeans, WebStorm и другие, имеют поддержку TypeScript на уровне плагинов, что позволяет воспользоваться рядом преимуществом, например, подцветкой кода или всплывающей подсказкой по типам и конструкциям языка.

TypeScript реализует многие концепции ООП, такие как наследование, полиморфизм, инкапсуляция и модификаторы доступа. В нем есть классы, интерфейсы и абстрактные классы.

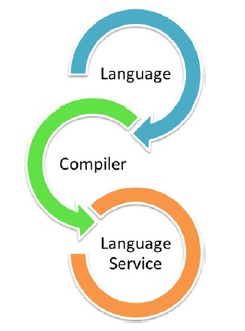
Потенциал языка позволяет быстрее и проще писать сложные комплексные решения, которые легче развивать и тестировать в дальнейшем, чем на стандартном JavaScript.

Поскольку данный язык является OpenSource, то все его инструменты доступны для всех желающих. Адрес репозитория - https://github.com/Microsoft/TypeScript. Кроме того, он является кроссплатформенным, а это значит, для работы с TypeScript мы можем использовать как Windows, так и Linux и MacOS.

*Typescript имеет не так много недостатков:*

* В процессе разработки имеем дело с файлами \*.ts, \*.d.ts, \*.map, \*.js. Слишком много дополнительных файлов, что бывает неудобно, если ваш проект небольшой.
* Не все браузеры поддерживают отладку TypeScript в консоли без лишних настроек.
* Множество нетривиальных классов. Чтобы писать код, опираясь на классы, приходится держать в голове какое свойство где находится. Например, вместо одного класса Event существуют еще такие как MouseEvent, TouchEvent, KeyboardEvent и другие...
* Неявная статическая типизация. Всегда можно описать тип как any, что по факту отключит приведение к конкретному типу этой переменной.
* Это транспайлер, что подразумевает, что мы должны всегда иметь под рукой tsc
* d.ts декларации поддерживаются сообществом DefinitelyTyped и часто не соответствуют текущей версии библиотеки. Либо не учитывают сложных вариантов (generic-функции, возвращаемые значения нескольких типов)

В основе TypeScript лежат три компонента:



*Рисунок 1.3. Основные три компонента TypeScript(Язык, Компилятор, Языковая служба)*

* Язык – Он включает в себя синтаксис, ключевые слова и сигнатуры типа.
* Компилятор TypeScript − Компилятор TypeScript (tsc) конвертирует инструкции, написанные в TypeScript для его JavaScript эквивалентов.
* Языковая служба TypeScript – «Языковая служба» предоставляет дополнительный уровень вокруг конвейера основного компилятора, которым являются редакторы. Языковая служба поддерживает стандартный набор стандартных операций по редактированию, например, автодополнение инструкций, автопоподстановки для сигнатур, форматирования и т.д.

**1.2.3. Обоснование выбора TypeScript.**

Основное достоинство TypeScript – это отличный инструментарий. Здесь обеспечивается сложное авто-завершение, навигация и рефакторинг. В больших проектах без таких инструментов практически не обойтись. Своими силами вечно не решаешься окончательно изменить код, вся база кода пребывает в «сыром» виде, и любой крупный рефакторинг становится рискованным и затратным.

TypeScript во многом превосходит все аналоги, к примеру, такие языки программирования как CoffeeScript и Dart, в том плане, что TypeScript – это расширенный JavaScript. Для сравнения, Dart и CoffeeScript являются новыми языками, и для них требуется среда выполнения, ориентированная на конкретный язык.

Angular построен на TypeScript. Рекомендуется писать приложения Angular на TypeScript ,а так же писать код на современном JS (ES6).

**1.3. Angular**

**1.3.1. Причины использования Angular.**

Современные сайты сильно полагаются на JS, поэтому необходимость правильной организации и тестирования кода возросла. Невозможно создать сложный, эффективный и удобный в обслуживании интерфейс на чистом JavaScript. В JavaScript тяжело поддерживать синхронизацию интерфейса и состояние приложения. Поэтому клиентские фреймворки стали популярны, и на данный момент их, как минимум, десяток.

Angular позволяет из "коробки" создавать большие и сложные по части бизнес-логики приложения. В Angular уже есть много инструментов для создания приложения.

Angular подготовлен к мобильным устройствам и десктопу – один фреймворк под множество платформ.

Angular представляет не только инструменты, но и шаблоны дизайна для создания обслуживаемого проекта. В Angular представление и логика приложения не связаны, что сильно очищает и упрощает разметку, т.е. код удобно структурирован.

Angular построен на TypeScript, который, в свою очередь, полагается на ES6. Angular убрал жесткую связь между различными компонентами приложения. Инъекция проходит подобно NodeJS, что позволяет легко заменять компоненты.

Angular активно обслуживается и имеет большое сообщество и экосистему. По фреймворку можно найти много материалом и полезных сторонних инструментов.

Angular тщательно протестирован и поддерживает юнит тесты и сквозное тестирование с помощью инструментов типа Jasmine и Protractor.

**1.3.2. Детальное описание технологии.**

Angular (версия 2 и выше) — это открытая и свободная платформа для разработки веб-приложений, написанная на языке TypeScript, разрабатываемая командой из компании Google, а также сообществом разработчиков из различных компаний. Angular — это полностью переписанный фреймворк от той же команды, которая написала AngularJS.

Изначально создавался как вторая версия AngularJS. Angular 2 был переписан с нуля на TypeScript, обладает иной архитектурой и не является обратно совместимым с AngularJS, в связи с чем для предотвращения путаницы было решено развивать его как отдельный фреймворк, нумерация версий которого начинается с 2.

Необходимо знать множество дополнительных технологий для работы с Angular:

Рекомендуется писать приложения Angular на TypeScript. Необходимо писать код на современном JS (ES6).

Для ускорения процесса разработки, нужно использовать Angular CLI. Angular CLI — это интерфейс командной строки, который позволяет быстро создавать проекты, добавлять файлы и выполнять множество определенных задач, такие как тестирование, сборка и развертывание. Для установки Angular и других компонентов активно используется npm или yarn.

***Angular приспособления для облегчения разработки UI:***

* ***Компоненты***

Компонент управляет отображением представления на экране, в ее основе используется Shadow DOM по умолчанию (для создания инкапсулированного визуального поведения). Как правило, компоненты используются для создания простого виджета в пользовательском интерфейсе, в то же время они могут представлять из себя набор еще более простых компонентов внутри себя (для увеличения абстракции и создания простых функциональных виджетов внутри приложения).

Каждый компонент имеет свой собственный детектор изменений, который гарантирует проверку привязок данных, определенных шаблоне.

* ***Шаблоны***

Шаблон — это html-разметка, в которой вы можете описывать ваши взаимодействия с DOM на основе модели данных и событий вашего класса компонента.

* ***Внедрение зависимостей***

Внедрение зависимостей (англ. Dependency Injection) — это композиция структурных шаблонов проектирования, при которой за каждую функцию приложения отвечает один, условно независимый объект (сервис), который может иметь необходимость использовать другие объекты (зависимости), известные ему интерфейсами. Зависимости передаются (внедряются) сервису в момент его создания.

* ***Директивы***

Директивы позволяют получать прямой доступ к DOM элементов. Они бывают двух видов: структурные и атрибутные. Благодаря директивам которые определяют набор инструкций, которые применяются при рендеринге html-кода т.е. HTML элементы могут вести себя динамически. Структурные директивы изменяют структуру DOM с помощью добавления или удаления html-элементов. Существует минимум три встроенных структурных директивы: ngIf, ngSwitch и ngFor.

* ***Пайпы***

Пайп (pipe) представляет собой особый обработчик, который позволяет форматировать отображаемые значения. Каждое приложение начинается с простой задачи: получить данные, преобразовать их и показать пользователям. Получение данных может быть таким же простым, как создание локальной переменной, или таким же сложным, как потоковая передача данных.

Как только данные поступают, можно поместить их необработанные значения toString непосредственно в представление, но это редко дает хороший пользовательский опыт. В большинстве случаев пользователи предпочитают видеть дату в простом формате, таком как 15 апреля 1988 года, а не в формате необработанной строки Пт 15 апреля 1988 00:00:00 GMT-0700 (тихоокеанское летнее время).

Очевидно, что некоторые значения выигрывают от небольшого редактирования. Можно применить их в своих шаблонах HTML, как и стили. Пайп предоставляет возможность писать преобразования отображаемых значений, которые можно объявить в своем HTML.

* ***Класс HttpClient***

Для взаимодействия с сервером и отправки запросов по протоколу http применяется класс HttpClient. Этот класс определяет ряд методов для отправки различного рода запросов: GET, POST, PUT, DELETE. Данный класс построен поверх стандартного объекта в JavaScript - XMLHttpRequest. Angular HttpClient довольно прост. Необходимо вызвать метод get и передать ему url. Данный метод get возвращает объект Observable. Этот класс является частью библиотеки RxJS ("Reactive Extensions"), которая используется во многих местах Angular'а.

* ***Angular-CLI***

Angular CLI облегчает создание приложения, которое уже работает, прямо из коробки. Генерация компонентов, маршрутов, сервисов и каналов с помощью простой команды. CLI также создаёт простые тестовые оболочки вышеописанного. Легко тестировать приложение локально во время разработки.

**1.3.3. Обоснование выбора Angular.**

Встроенный поток данных, безопасность типов и модульный интерфейс командной строки (CLI) являются отличительными признаками того, что делает Angular такой самодостаточной средой. Однако, учитывая, что это структура, которая помогает даже крупнейшим компаниям управлять своими корпоративными приложениями, в сочетании с обязательством Google не вносить более серьезных изменений, Angular будет только развиваться.

**Книжные источники:**

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

Spring in action. Third Edition. стр. 198

TypeScript Deep Dive стр. 9 – 22

Pro TypeScript: Application-Scale JavaScript Development cтр. 47

Angular для профессионалов стр. 418

Angular и TypeScript. Сайтостроение для профессионалов стр. 22

**Интернет источники:**

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>

Руководство по TypeScript<https://metanit.com/web/typescript/1.1.php>

Перевод официальной документации Typescript <http://typescript-lang.ru/docs/>

Documentation TypeScript <https://www.typescriptlang.org/docs/home.html>

Статическая и динамическая типизация <https://habr.com/ru/post/308484/>

JavaScript vs TypeScript: что в каких случаях лучше использовать <https://proglib.io/p/javascript-vs-typescript/>

Мысли вслух о TypeScript <https://habr.com/ru/post/272055/>

TypeScript – Изучаем с нуля <https://webformyself.com/typescript-izuchaem-s-nulya/>

Введение в Angular: что это за фреймворк и зачем его использовать

<https://webformyself.com/vvedenie-v-angular-chto-eto-za-frejmvork-i-zachem-ego-ispolzovat/>

Просто Angular <https://habr.com/ru/post/341688/#s1_1>

Официальный сайт Angular <https://angular.io/>

Руководство по Angular 7 <https://metanit.com/web/angular2/>

Удивительный Angular <https://habr.com/ru/post/348818/#s1>