
# Spring Framework
# Module 3 – AOP

Evgeniy Krivosheev
Andrey Stukalenko
Vyacheslav Yakovenko
Last update: Feb, 2012

# Содержание

- Введение
- Активация AOP в Spring контейнере
- Язык Pointcat’ов
- Типы Advice’ов
- Пример использования AOP
- Дополнительная информация
- Упражнение

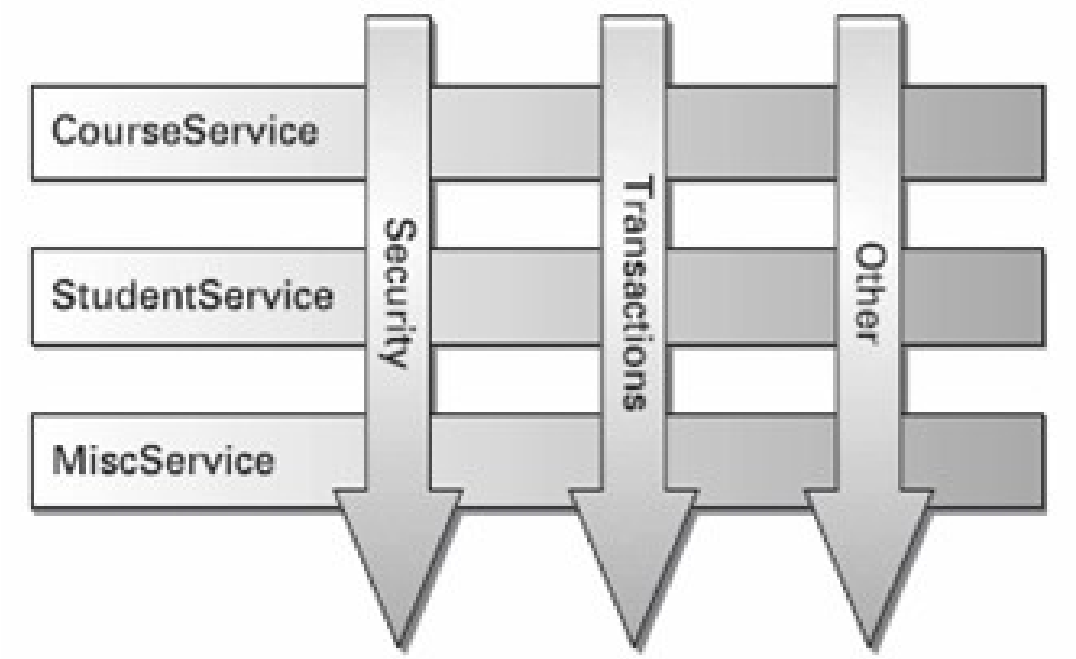
LUXOFT TRAINING

"Let us begin by defining some central AOP concepts and terminology. These terms are not Spring-specific… unfortunately, AOP terminology is not particularly intuitive; however, it would be even more confusing if Spring used its own terminology."

*Spring Framework 3.1.0M1, 7.1.1. AOP Concepts*

# Spring Framework :: AOP :: Введение

- Aspect Oriented Programming (AOP) -> Аспектно-Ориентированное Программирование
- АОП предоставляет средства для реализации ортогональной (crosscutting) функциональности



LUXOFT TRAINING

Для того чтобы вам было легче разобраться с AOP давайте подумаем, как подобную «ортогональную» бизнес-логику можно реализовать на базе RDBMS?

LUXOFT TRAINING

# Пример «ортогонального» логирования на базе триггеров в RDBMS:

```
/* Триггер на уровне таблицы */
 CREATE OR REPLACE TRIGGER DistrictUpdatedTrigger
 AFTER UPDATE ON district
 BEGIN
  INSERT INTO info VALUES ('table "district" has changed');
 END;
```

LUXOFT TRAINING

- В Spring Framework, AOP реализуется с помощью создания прокси-объекта на интересующий вас сервис;
- Прокси-объект может быть создан с использованием следующих библиотек:
  - Используя стандартный механизм создания динамических прокси из J2SE;
  - Используя CGLIB прокси.

LUXOFT TRAINING

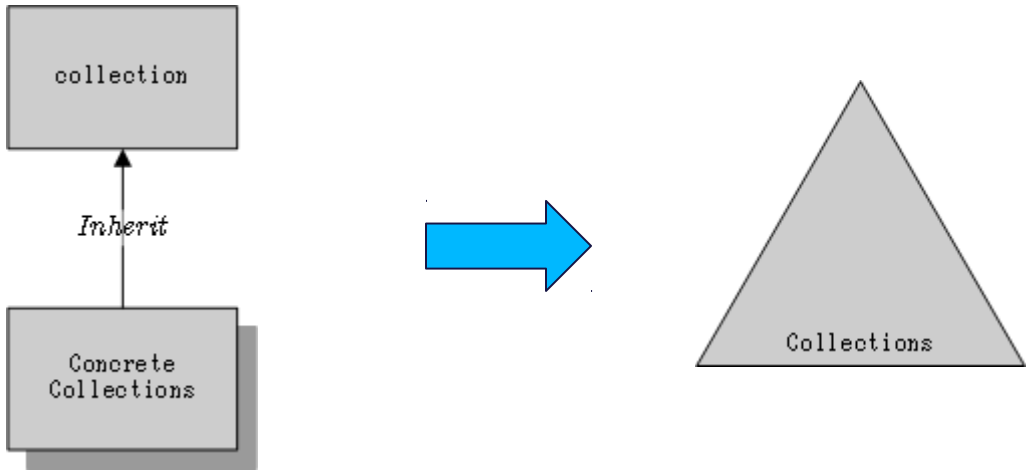
- Наш тренинг будет построен на использовании стандартного J2SE-механизма создания динамических прокси. Поэтому, от слушателей требуется понимание этого механизма;
- Дополнительная информация по этой теме может быть найдена вами по следующей ссылке: http://download.oracle.com/javase/1.3/docs/guide/refl (http://goo.gl/ucfM0)

LUXOFT TRAINING

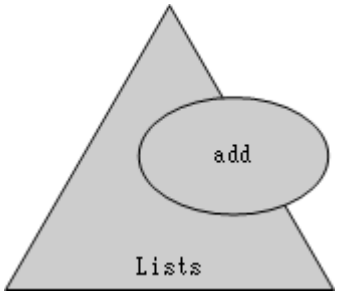
- Важным моментом в понимании работы Spring AOP является то, что динамический прокси может быть создан только для класса имплементирующего интерфейс;
- Если наличие интерфейса в иерархии классов, по каким либо причинам, не возможно, необходимо использовать механизм создания прокси с помощью CGLIB (не рассматривается в рамках этого тренинга).

LUXOFT TRAINING

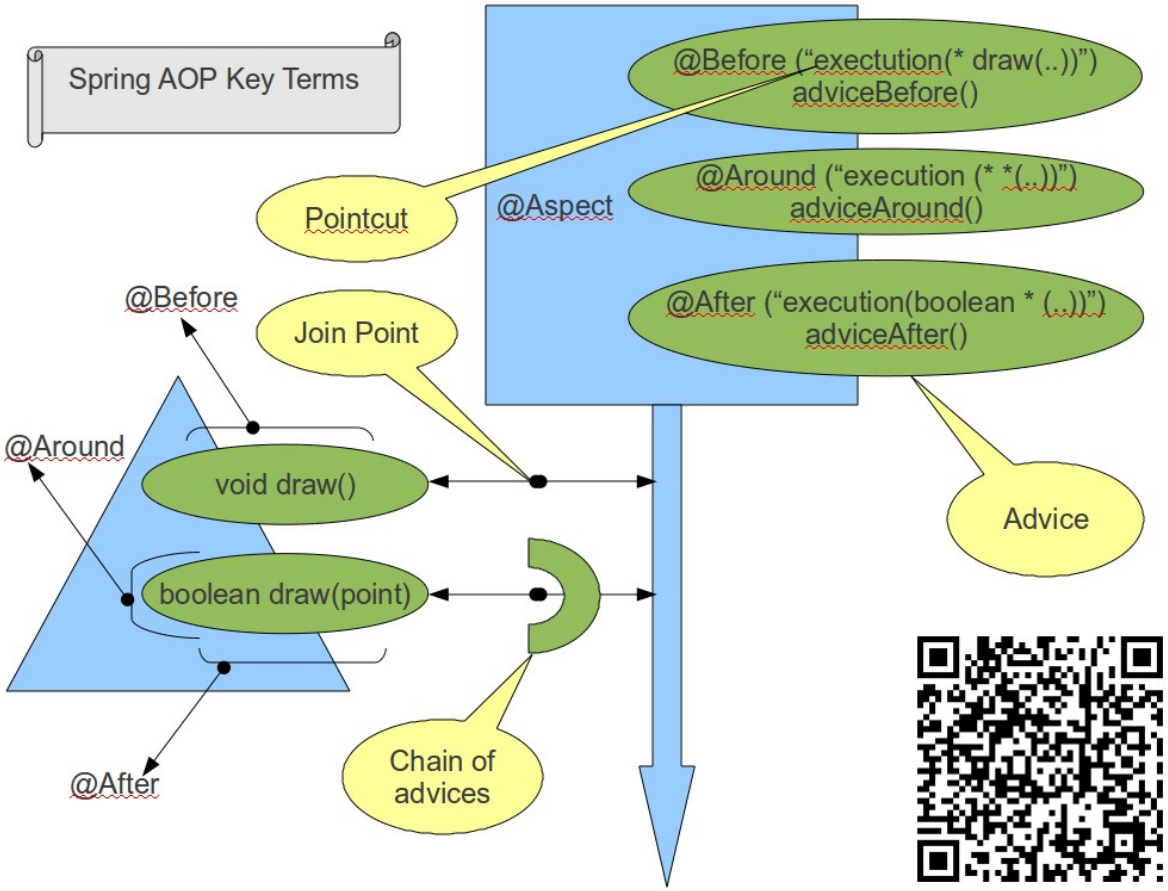
В дальнейшем иерархии объектов, имеющих в своей вершине интерфейс, мы будем изображать в виде треугольника, как в LePUS3 диаграммах, в то время как отдельные классы в виде привычного для UML прямоугольника:



Методы, общие для всей иерархии классов (унаследованные из интересующего нас интерфейса) мы будем изображать в виде овала выступающего за пределы прямоугольника либо треугольника, в зависимости от контекста:




Схематически все основные элементы AOP и их взаимодействие изображены ниже:



LUXOFT TRAINING

- Aspect – ортогональная функциональность:
  - В Spring AOP сфокусирована в классах аннотированных с помощью @Aspect (на нашей диаграмме синий прямоугольник с вертикальной стрелкой вниз);
- Примеры использования:
  - Логирование;
  - Ограничение прав доступа;
  - Транзакции, etc.

Join Point - Точка во время выполнения приложения, где Aspect может быть применен:

- Точка пересечения «основной» и «ортогональной» функциональности;
- В Spring Framework в качестве Join Point может выступать только метод (в нашей диаграмме это методы синего треугольника в иерархии классов).

Advice - Реализация Aspect'а в конкретном join point'е, дополняет приложение новой функциональностью.

В Spring AOP в качестве Advice выступает метод @Aspect-а, аннотированный как @Before, @Around, @After, etc. (см. диаграмму).

LUXOFT TRAINING

Pointcut - предикат, определяющий в каких join point'ах должен быть применен advice.

Spring AOP использует @AspectJ язык описания pointcut'ов по умолчанию.

Сам язык описания pointcut'ов мы рассмотрим подробнее ниже.

Weaving: связывание – процесс применения aspect'а к target объекту для создания нового proxy объекта;

Для осуществления связывания в Run Time в classpath Spring Framework использует две дополнительные зависимости:

- aspectjrt.jar
- aspectjweaver.jar

Также необходимо инициировать создание динамических прокси в файле конфигурации: <aop:aspectj-autoproxy />

# Spring Framework :: Активация AOP

```
<?xml version="1.0" encoding="UTF-8"?>
   <beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:aop="http://www.springframework.org/schema/aop"
   xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/aop
   http://www.springframework.org/schema/aop/spring-aop-3.0.xsd">


    <aop:aspectj-autoproxy/>
</beans>
```

LUXOFT TRAINING

В общем случае pointcut в Spring AOP задается выражением:

execution(

modifiers-pattern?

ret-type-pattern

declaring-type-pattern?

name-pattern(param-pattern)

throws-pattern?)

? – опциональные параметры

LUXOFT TRAINING

Примеры:

- execution (* *(..)) – связывание с любым методом, вне зависимости от возвращаемого типа или параметров;
- execution (int *(..)) – связывание с любым методом, возвращающим int, вне зависимости от параметров;
- execution (!static * *(..)) – связывание с любым не статическим методом, вне зависимости от параметров;

# Язык Pointcut'ов

- bean – связывание с join points определенного Spring бина (или набора бинов, если используются wildcards)
- within – связывание с любым методом в рамках соответствующего класса
- this – связывание с join points в случае если бин AOP Proxy является объектом заданного типа
- target – связывание с join points когда целевой объект (т.е. объект, который обернут прокси) является объектом заданного типа
- args – связывание с join points где аргументами являются объекты заданных типов
- @target – связывание с join point, где класс исполняемого объекта имеет аннотацию соответствующего типа
- @args - связывание с join points где типы переданных аргументов (в runtime) имеют аннотации заданных типов
- @within – связывание с любыми join points в типах, имеющих аннотацию заданного типа
- @annotation – связывание с join points где субъект вызова (вызываемый метод) имеет заданную аннотацию

# Язык Pointcut'ов
## Примеры использования

LUXOFT TRAINING

- execution(public * *(..))
- execution(* set*(..))
- execution(* com.xyz.service.AccountService.*(..))
- execution(* com.xyz.service.*.*(..))
- execution(* com.xyz.service..*.*(..))
- within(com.xyz.service.*)
- within(com.xyz.service..*)
- this(com.xyz.service.AccountService)
- target(com.xyz.service.AccountService)
- args(java.io.Serializable)
- @target(org.springframework.transaction.annotation.Transactional)
- @within(org.springframework.transaction.annotation.Transactional)
- @annotation(org.springframework.transaction.annotation.Transactional)
- @args(com.xyz.security.Classified)
- bean(tradeService)
- bean(*Service)

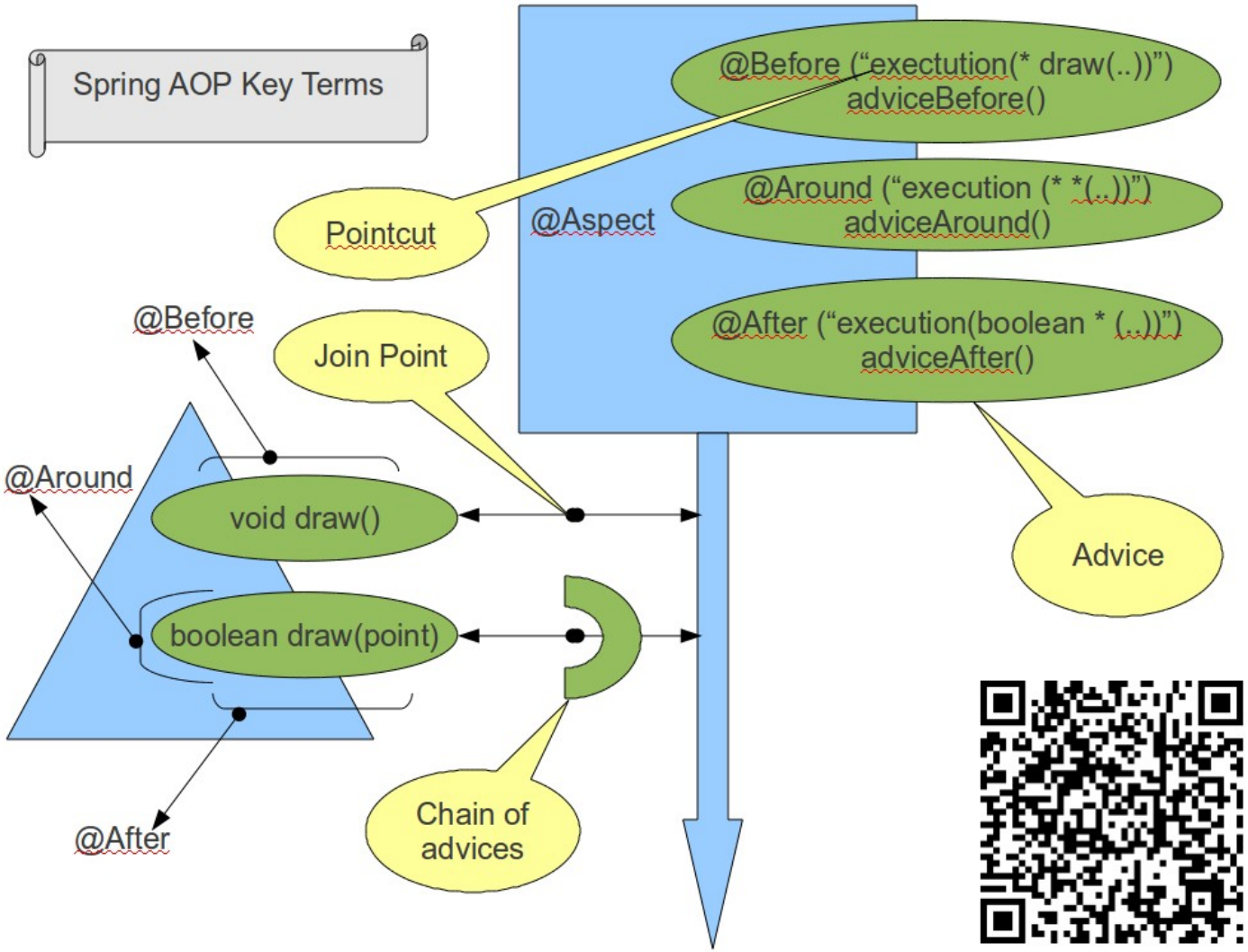
## Spring :: AOP :: Язык Pointcut'ов

LUXOFT TRAINING

Примеры:

- execution (void Test.foo(..)) – связывание с методом foo, класса Test, вне зависимости от параметров;
- execution (void Test.foo(int, String)) – связывание с методом foo, класса Test, принимающим в качестве параметров int и String;
- execution (* foo.bar.*.dao.update*(..)) – связывание с любым методом начинающимся на «update» субпакета dao;
- Подробнее: http://www.eclipse.org/aspectj/doc/released/progguide/language-joinPoints.html (http://goo.gl/Vm79a)

# Spring :: AOP :: Типы Advice'ов



## Spring :: AOP :: Типы Advice'ов

LUXOFT TRAINING

@Before - выполняется перед joinpoint'ом

Нет возможности отменить вызов joinpoint'а, кроме как выбросить исключение

LUXOFT TRAINING

@AfterReturning – после успешного исполнения joinpoint'а , например, метод выполнился, не выбросив исключение;

@AfterThrowing – в случае выброшенного исключения в joinpoint'е;

@After (finally) – в любом случае после исполнения joinpoint'а;

** Подробнее: Spring 3.1.RELEASE M1 Reference manual, 8.2.4 Declaring advice*

LUXOFT TRAINING

# @Around - окружает joinpoint

# Наиболее мощный из всех типов advice’ов;

# Может решать, исполнять joinpoint или вернуть собственное значение;

```
@Around("com.xyz.myapp.SystemArchitecture.businessService()")
public Object doBasicProfiling(ProceedingJoinPoint pjp) throws Throwable {
        // start stopwatch
        Object retVal = pjp.proceed();
        // stop stopwatch
        return retVal;
}
```

LUXOFT TRAINING

Алгоритм:

- Покупатель приходит в бар
- Бармен здоровается
- Покупатель заказывает напиток
- Если у покупателя есть деньги
- Ему продают напиток
- Бармен спрашивает его мнение
- Если нет денег (isBroke)
- Ему что-то говорят
- Бармен прощается с покупателем

LUXOFT TRAINING

# Мини-опрос: Какие из шагов алгоритма относятся к базовой, а какие к ортогональной функциональности?

- Покупатель приходит в бар
- Бармен здоровается
- Покупатель заказывает напиток
- Если у покупателя есть деньги
- Ему продают напиток
- Бармен спрашивает его мнение
- Если нет денег (isBroke)
- Ему что-то говорят
- Бармен прощается с покупателем

LUXOFT TRAINING

- Основная функция:
  - покупка напитка;
- Вспомогательные:
  - приветствие;
  - прощание;
  - вопросы.

LUXOFT TRAINING

# Бизнес объекты:

```
public interface Bar {
    Squishee buySquishee(Customer customer);
}
public class ApuBar implements Bar {
    public Squishee buySquishee(Customer customer)  {
        if (customer.isBroke()){
            throw new CustomerBrokenException();
        }
        System.out.println("Here is your Squishee");
        return new Squishee("Usual Squishee");
    }
}
```

LUXOFT TRAINING

# Ортогональная, вспомогательная функциональность:

```
@Aspect
public class Politeness {

    @Before(value = "execution(* buySquishee(..))")
    public void sayHello() {
        System.out.println("Hello!");
    }
}
```

```
<bean id="politeness" class="aop.Politeness"/>
```

```
<bean id="bar" class="aop.ApuBar"/>
```

LUXOFT TRAINING

# Ортогональная, вспомогательная функциональность:

```
@AfterReturning(pointcut = "execution(* buySquishee(..))",
                returning = "retVal", argNames = "customer")
public void askOpinion(Object retVal) {
    System.out.println("Is " +
            ((Squishee)retVal).getName() + " Good Enough?");
}

@AfterThrowing("execution(* buySquishee(..))")
public void sayYouDontHaveEnoughMoney() {
    System.out.println("Hmmm...");
}

@After("execution(* buySquishee(..))")
public void sayGoodBye() {
    System.out.println("Good Bye!");
}
```

LUXOFT TRAINING

- Особенности использования @Around advice'ов детально описаны в разделах 8.2.4.5, 6 Spring 3.1.RELEASE Reference Manual;
- Важно понимать, что «срабатывание» аспекта, в случае использования динамических прокси будет выполняться только в том случае, если обращение к таржет-бину будет производится через IoC контейнер;

# Упражнения

- №5: Использование Spring AOP. @AspectJ style. :
  - 30..45 мин – самостоятельная работа;
  - 15 мин – обсуждение;

LUXOFT TRAINING

# Вопросы!?