SPRING FRAMEWORK 3.0

Dmitry Noskov

Aspect Oriented Programming with Spring

Aspect Oriented Programming

What is AOP?

- is a programming **paradigm**
- extends **OOP**
- enables **modularization** of crosscutting concerns
- is second heart of Spring Framework

A simple service method

```java
public Order getOrder(BigDecimal orderId) {
    return (Order) factory.openSession()
                            .get(Order.class, orderId);
}
```

Add permissions check

```
public Order getOrder(BigDecimal orderId) {
    if (hasOrderPermission(orderId)) {
        return (Order) factory.openSession()
                              .get(Order.class, orderId);
    } else {
        throw new SecurityException("Access Denied");
    }
}
```

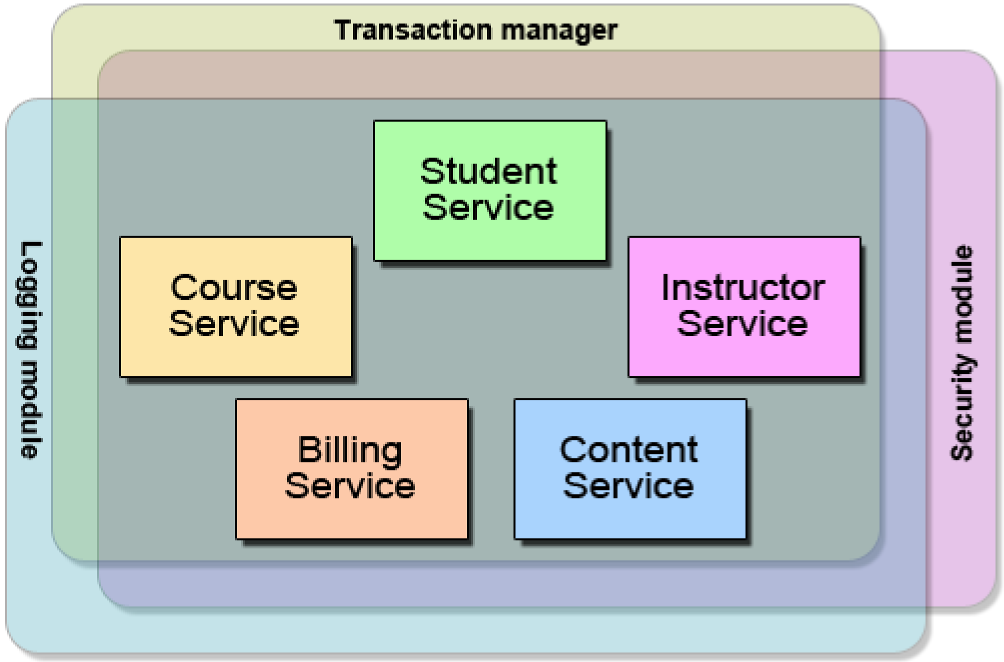
Add transaction management

```
public Order getOrder(BigDecimal orderId) {
    if (hasOrderPermission(orderId)) {
        Order order;
        Session session = factory.openSession();
        Transaction tx = session.beginTransaction();

         try {
             order = (Order) session.get(Order.class, orderId);
             tx.commit();
         } catch (RuntimeException e) {if (tx!=null) {tx.rollback();}
         } finally {session.close();}

        return order;
    } else { throw new SecurityException("Access Denied");}
}
```

Add cache

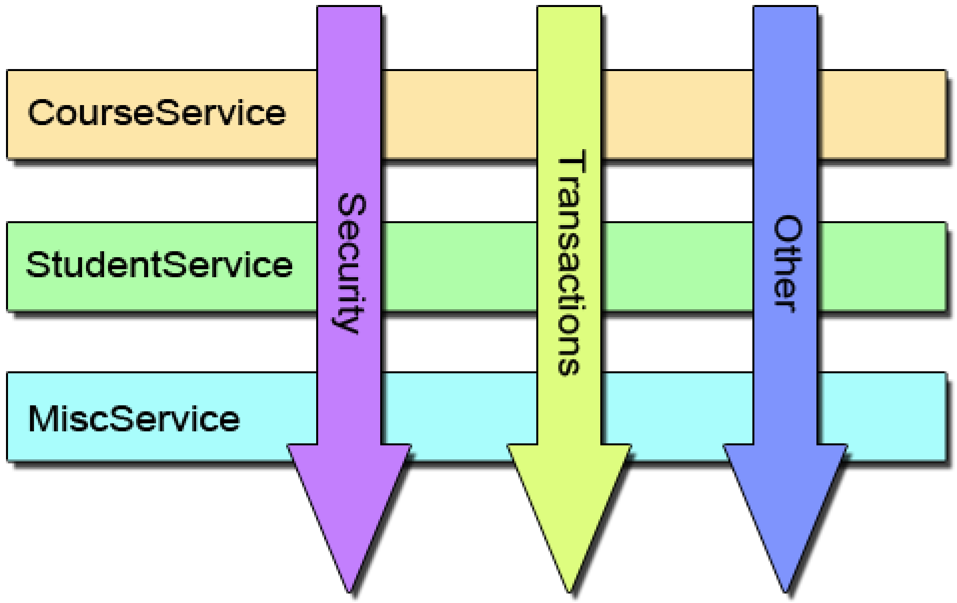
```
public Order getOrder(BigDecimal orderId) {
    if (hasOrderPermission(orderId)) {
        Order order = (Order)cache.get(orderId);
        if (order==null) {
          Session session = factory.openSession();
          Transaction tx = session.beginTransaction();

          try {
              order = (Order) session.get(Order.class, orderId);
              tx.commit();
              cache.put(orderId, order);
          } catch (RuntimeException e) {if (tx!=null) {tx.rollback();}
          } finally {session.close();}
        }

        return order;
    } else { throw new SecurityException("Access Denied");}
}
```

A similar problem at enterprise level



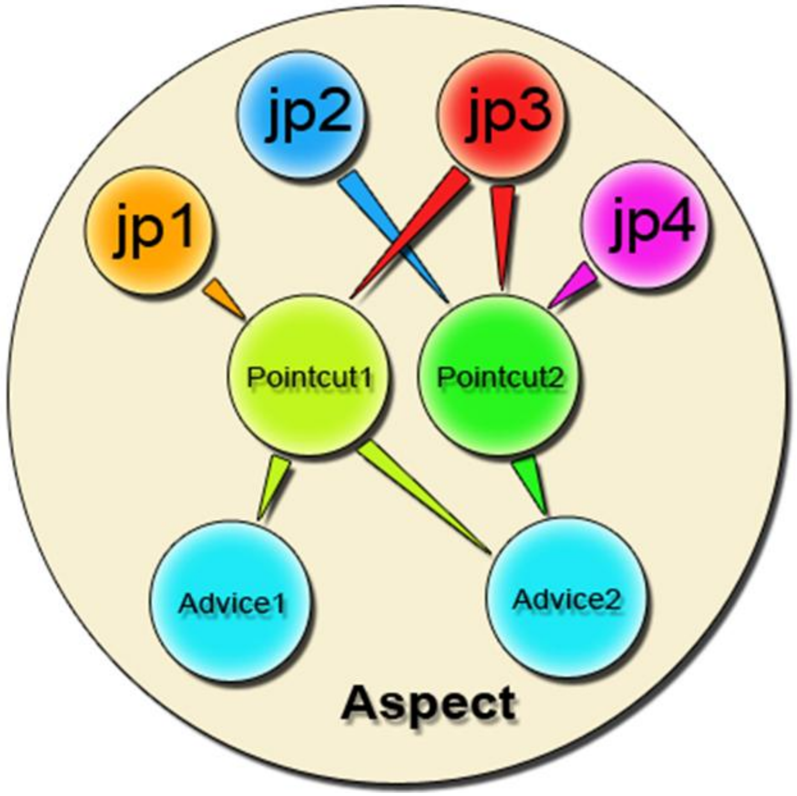
What does AOP solve?

- Logging
- Validation
- Caching
- Security
- Transactions
- Monitoring
- Error Handling
- Etc…



AOP concepts

- aspect
- advice
- pointcut
- join point



AOP and OOP

AOP	OOP
1. **Aspect** – code unit that encapsulates pointcuts, advice, and attributes	1. **Class** – code unit that encapsulates methods and attributes
2. **Pointcut** – define the set of entry points (triggers) in which advice is executed	2. **Method signature** – define the entry points for the execution of method bodies
3. **Advice** – implementation of cross cutting concern	3. **Method bodies** –implementation of the business logic concerns
4. **Weaver** – construct code (source or object) with advice	4. **Compiler** – convert source code to object code

AOP concepts(2)

- introduction
- target object
- AOP proxy
- weaving
  - compile time
  - load time
  - runtime

Spring AOP

- implemented in pure java
- no need for a special compilation process
- supports only method execution join points
- only runtime weaving is available
- AOP proxy
  - JDK dynamic proxy
  - CGLIB proxy
- configuration
  - `@AspectJ` annotation-style
  - Spring XML configuration-style

@AspectJ

Declaring aspect

```
@Aspect
public class EmptyAspect {
}
```

```
<!--<context:annotation-config />-->
<aop:aspectj-autoproxy proxy-target-class="false | true"/>

<bean
class="org.springframework.aop.aspectj.annotation.AnnotationAwareAspectJAutoProxyCreator">
</bean>

<bean class="example.EmptyAspect"/>
```

Declaring pointcut

Pointcut designators

- code based
  - execution
  - within
  - target
  - this
  - args
  - bean

Pointcut designators(2)

- annotation based
  - @annotation
  - @within
  - @target
  - @args

Format of an execution expression

execution(

  modifiers-pattern

  returning-type-pattern

  declaring-type-pattern

  name-pattern(param-pattern)

  throws-pattern

)

Simple pointcut expressions

```
@Aspect
public class ItemStatusTracker {

    @Pointcut("execution(* approve(..))")
    public void ifApprove() {}

    @Pointcut("execution(* reject(..))")
    public void ifReject() {}

    @Pointcut("ifApprove() || ifReject()")
    public void ifStateChange() {}
}
```

Execution examples

any public method

```
execution(public * * (..))"
```

any method with a name beginning with "get"

```
execution(* get*(..))
```

any method defined by the appropriate interface

```
execution(* bank.BankService.*(..))
```

any method defined in the appropriate package

```
execution(* com.epam.pmc.service.*.*(..))
```

other examples

http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/aop.html#aop-pointcuts-examples

Declaring advice

Advice

- associated with a pointcut **expression**
  - a simple **reference** to a named pointcut
  - a pointcut **expression** declared in place
- runs
  - before
  - after returning
  - after throwing
  - after (finally)
  - around

Before advice

```
@Aspect
public class BankAspect {

    @Pointcut("execution(public * * (..))")
    public void anyPublicMethod() {}

    @Before("anyPublicMethod()")
    public void logBefore(JoinPoint joinPoint) {
        //to do something
    }
}
```

After returning advice

```
@Aspect
public class BankAspect {

    @AfterReturning(
        pointcut="execution(* get*(..))",
        returning="retVal")
    public void logAfter(JoinPoint joinPoint, Object retVal) {
        //to do something
    }
}
```

After throwing advice

```
@Aspect
public class BankAspect {

    @AfterThrowing(
        pointcut = "execution(* bank..*ServiceImpl.add*(..))",
        throwing = "exception")
    public void afterThrowing(Exception exception) {
        //to do something
    }
}
```

After finally advice

```
@Aspect
public class BankAspect {

    @Pointcut("execution(public * * (..))")
    public void anyPublicMethod() {}

    @After(value="anyPublicMethod() && args(from, to)")
    public void logAfter(JoinPoint jp, String from, String to) {
        //to do something
    }
}
```

Around advice

```
@Aspect
public class BankCacheAspect {

    @Around("@annotation(bank.Cached)")
    public Object aroundCache(ProceedingJoinPoint joinPoint){
        //to do something before
        Object retVal = joinPoint.proceed();
        //to do something after
    }
}
```

Aspect and advice ordering

- order of advice in the **same** aspect
  - before
  - around
  - after finally
  - after returning or after throwing
- Spring **interface** for ordering **aspects**
  - org.springframework.core.Ordered
- Spring **annotation**
  - org.springframework.core.annotation.Order

XML based AOP

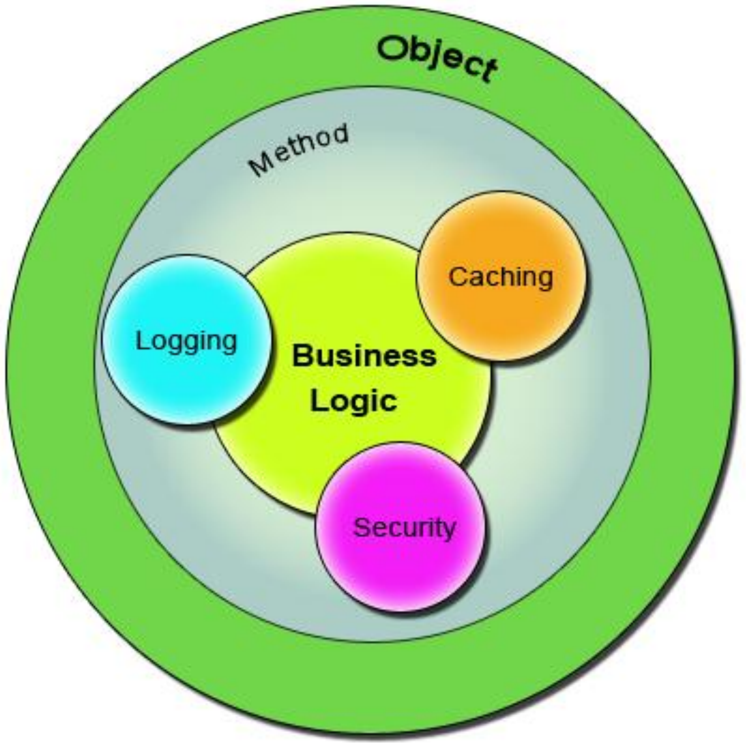
Declaring an aspect

```
<?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="…">

  <aop:config>
    <aop:aspect id="bankAspectId" ref="bankAspect">
      <aop:pointcut id="anyPublicMethod"
                    expression="execution(public * *(..))"/>

      <aop:before pointcut-ref="anyPublicMethod" method="logBefore"/>
    </aop:aspect>
  </aop:config>

  <bean id="bankAspect" class="bank.BankAspect"/>
</beans>
```

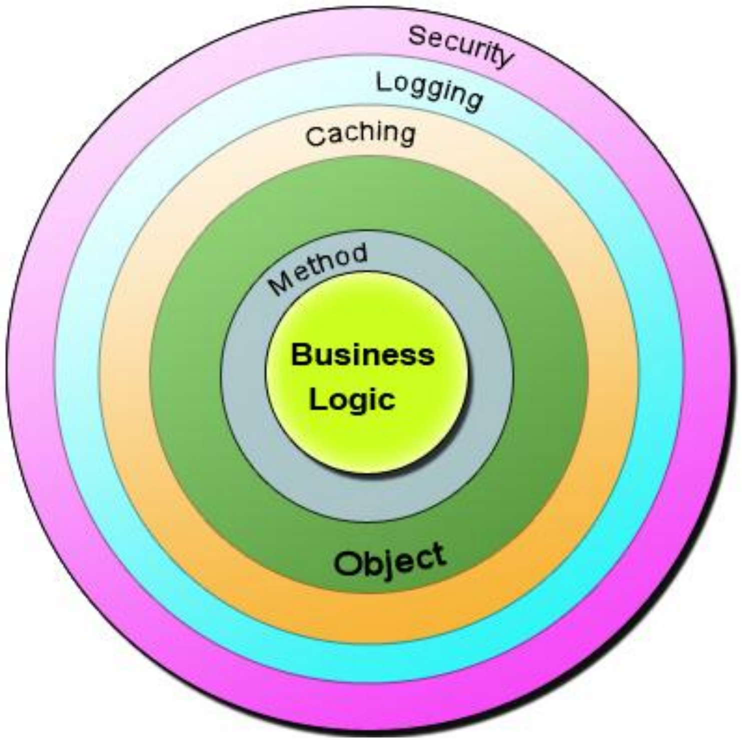
How it all works

Bean in Spring container

Standard OOP implementation

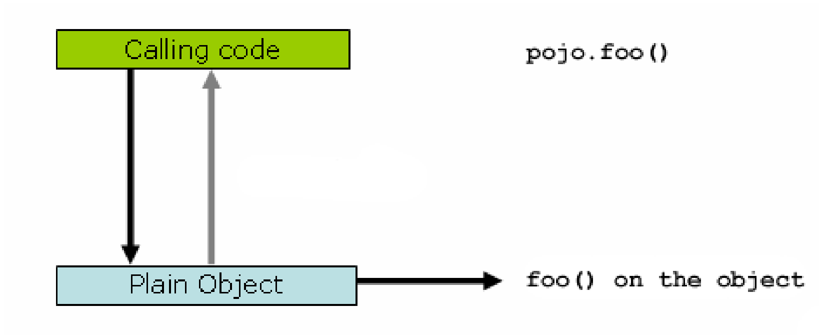


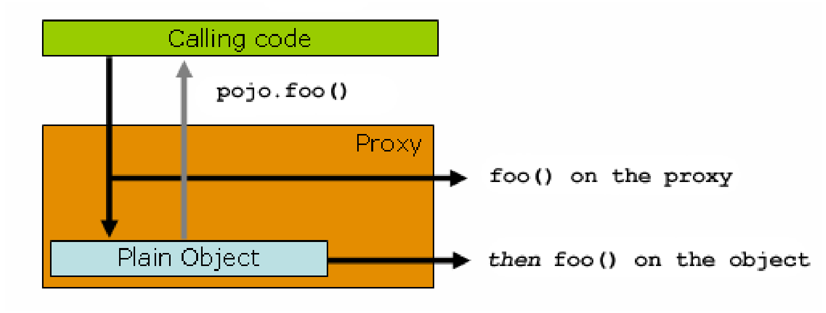
Implementation with AOP



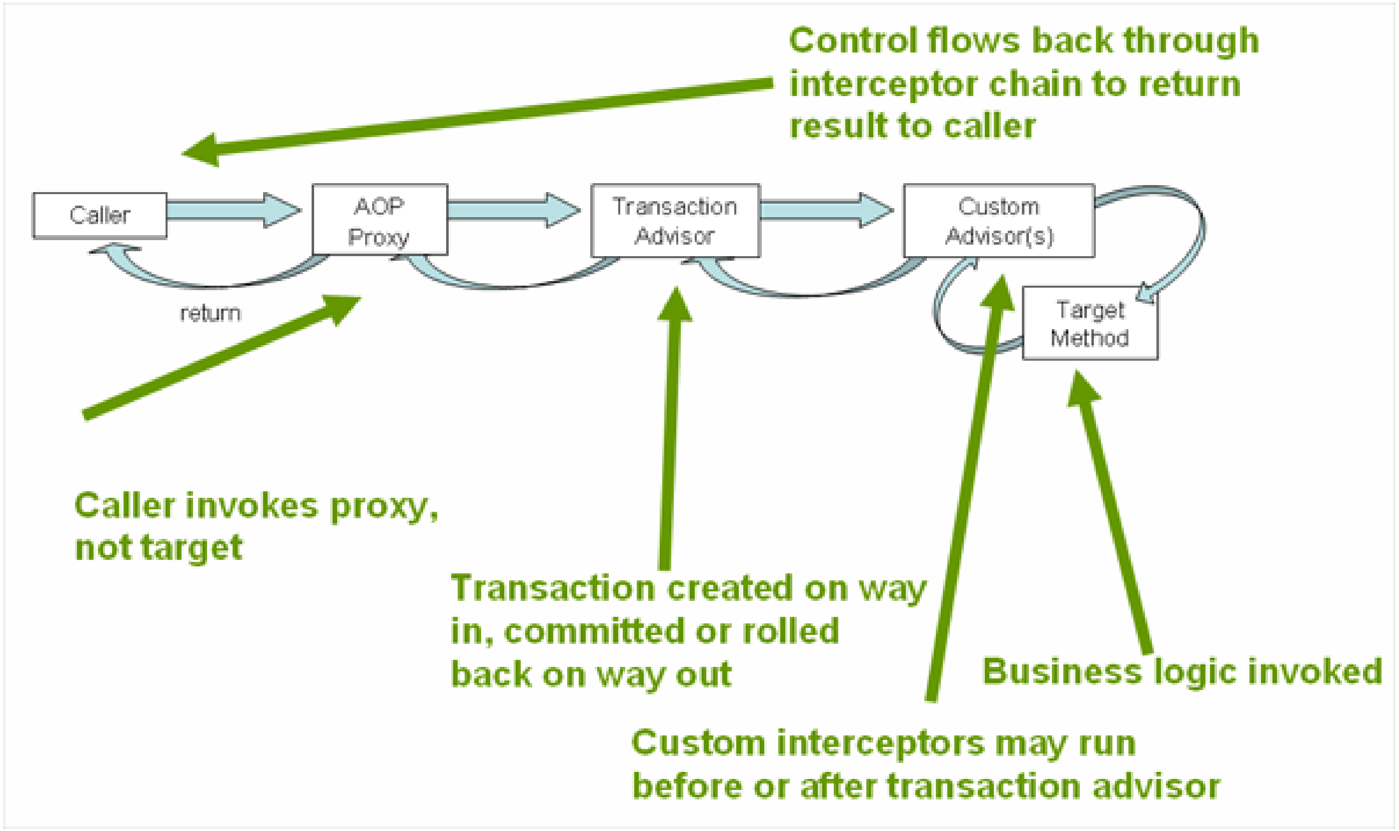
AOP proxies

Invoke directly	Invoke via proxy





How it really works


Control flows back through interceptor chain to return result to caller
Caller
AOP Proxy
Transaction Advisor
Custom Advisor(s)
Target Method
return
Caller invokes proxy, not target
Transaction created on way in, committed or rolled back on way out
Business logic invoked
Custom interceptors may run before or after transaction advisor

Introductions

Introduction behaviors to bean

```
@Aspect
public class CalculatorIntroduction {

    @DeclareParents(
        value = "calculator.ArithmeticCalculatorImpl",
        defaultImpl = MaxCalculatorImpl.class)
    public MaxCalculator maxCalculator;

    @DeclareParents(
        value = "calculator.ArithmeticCalculatorImpl",
         defaultImpl = MinCalculatorImpl.class)
    public MinCalculator minCalculator;
}
```

Introduction states to bean

```
@Aspect
public class BankServiceIntroductionAspect {
    @DeclareParents(
        value="bank.BankServiceImpl",
        defaultImpl=DefaultCounterImpl.class)
    public Counter mix;

    @Before("execution(* get*(..)) && this(auditable)")
    public void useBusinessService(Counter auditable) {
        auditable.increment();
    }
}
```

Spring AOP vs AspectJ

Spring AOP	AspectJ
□ no need for a special compilation process	□ need AspectJ compiler or setup LTW
□ support only method execution pointcuts	□ support all pointcuts
□ advise the execution of operations on Spring beans	□ advice all domain objects

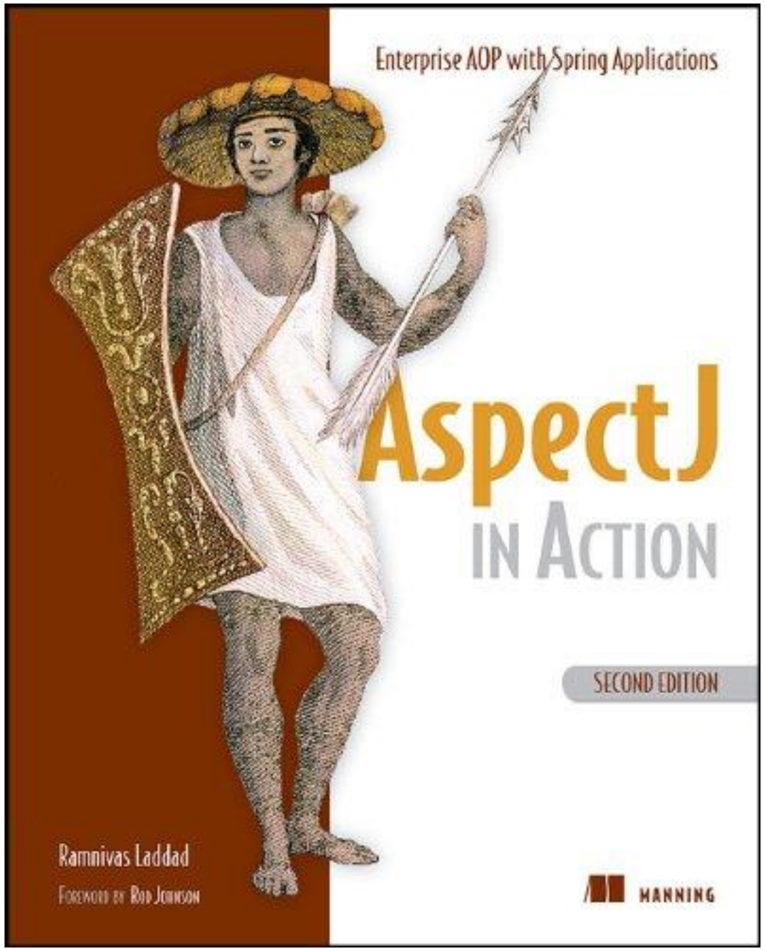
@AspectJ vs XML

@AspectJ	XML
has more opportunities, such as combine named pointcuts	can be used with any JDK level
encapsulate the implementation of the requirement it addresses in a single place	good choice to configure enterprise services

Links

- Useful links
  - Wiki: Aspect-oriented programming
    http://en.wikipedia.org/wiki/Aspect-oriented_programming
  - Spring Reference
    http://static.springsource.org/spring/docs/3.0.x/spring-framework-reference/html/aop.html
  - AspectJ home site
    http://www.eclipse.org/aspectj/

Books


Enterprise AOP with Spring Applications
AspectJ
IN ACTION
SECOND EDITION
Ramnivas Laddad
FOREWORD BY ROD JOHNSON
MANNING

Questions

The end

noskov.d@Gmail.com

my Linked in profile

http://www.linkedin.com/in/noskovd

slideshare

http://www.slideshare.net/analizator/presentations