Spring AOP

Code for DAO

Public void addAcountId(Acount acc,String userId)

{

// add code for loggin code

// add code for security chcek

// get session

//session.save

}

New Requirement

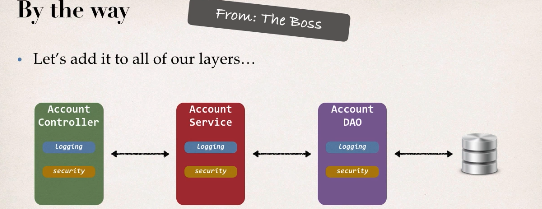
Add Logging methods to DAO : loggiin stament before the strt of dao method

New Requirement

Add Security methods to DAO : make user is authorized before running of dao method

TPSreport?

If requiremrent to add to all layers



Two main problme

1 code tangling : fro given method logging and security will be tnagled

2 code scattering : need to change looging and security we need to update all classes (think 100 classes)

Other possible solutions

Inheritance: every class would need to inheri from a base class

Can all classs extend from base class ? plus no multiple inheritnace

Delegation : classes would delegete logging,security calls

- still would need to update classes if we wnated to

- add / remove logging or security

- add new feature like auditing,APi management, instrumentation

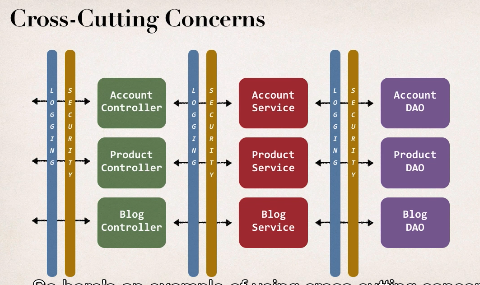
SOlution

AOP :Aspect

1 programming technique based on ASPECT

2 ASPECt encapsulet cross cutting logic :

**cross Cutting concerns :** means logic or functionlity : basic infrastructure that our all applicaion is required



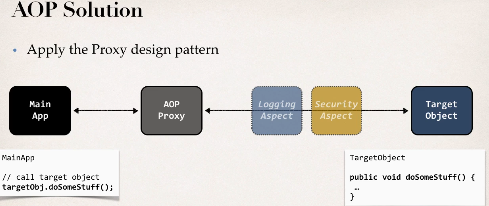
Common logic on respective layers

Aspect

1 can be reused at multiple loaction

2 same Aspect/class applied on base configuration



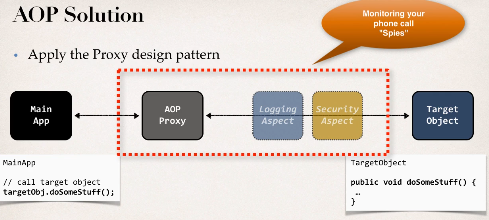


AOP proxy design pattern

- main app calls the target object

-method will be having some logic in target object

-Main app have no idea about AOP and no idea about any aspect behind secnce we dnt know its working will call just method



Befits of AOP

1 code for ASPECT is deisgned in a sinlge class

-much better then being scatrred every where

- Promtes code reuse and easier to change

2 Bussiness code in our application is cleaner

- only applies to bussiness funtionlity: addAccount

-reduce code complexity

3 Configurable

-based on configuration, apply aspects slectively to differt parts of app

-no need to make changes to main application code very important

Additonal use case

Most common: logging , security , tranaction

Audit loggin: who, what when ,where

Exception Handling:Log exception and notify DevOps team via SMS/email

API management:

-how many times has a method been called user

-analytics what are the peak times? Wt is Avg load ? who is top user

Advanteges and Disadvantages

Advantage: reusable modules, resolve code tangling, resolve code scatter, Applied slectively based on confiurtaion

Disadvantage: too many aspects and app flow is hard to follow, minor performance cost for aspect execution(run -tome -weaving)

Comparing SPringAOP and ASPECTJ

**AOP Terminology**

Aspect: module for code for a cross cutting concern(logging , security)

Advice: what actions should be taken and when it should applied

Join Point : when to apply code during program execution

Point Cut: A predict expression for where advice should be applied

**Advice Types**

Before Advice: run before method

After Finally Advice: run after the method (finally)

After returning Advice: run after the method (success excutio)

After Throwing Advice: run after method (if exception thrown)

Around Advice: run before and after method

**Weaving**

**-** connecting aspects to target objects to create an advice object

- differnet types of weaving : compile , run ,load -time

- regarding performance: run time weaving is the slowest,

AOP Frameworks

2 leading Framework SPRINGAOP and ASPECTJ

Spring AOP support

-key components: Security,tranaction ,cacheing etc

- uses run -time weaving aspects : usesr proxy pattern

ASPECTJ

- orinal AOP framwork relased in 2001

- provides compler suppport of AOP

- Rich support for AOP : Join points : method -level,constructor, field

Code weaing: compile time,post compile -time and laod time

SPRIGN AOPComparision

Spring AOP advanatage

- similat to use then ASPECTJ

- use proxy pattern

- can migrate to AspectJ when using @Aspect annotation

Disadvantage

- only support method level join points

- can apply to aspects to beans created by Spring app Context

- minor performance cost for aspect execution(run time weaving)

ASPECTJ Comparision

Advantages:

-Support all join points

-works with many pojo not just bean from app context

- faster compared to Spring AOP

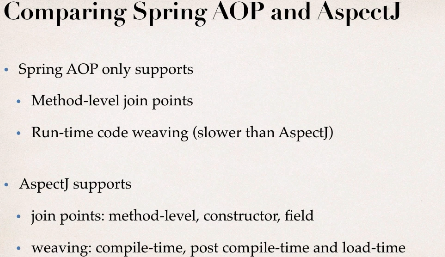
- complet AOP support

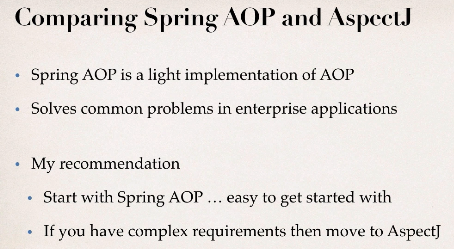
Disadvantage

-Compile time waving requires extra compilation steps

- AspectJ pointCut syntax can become complex

Comparing Both





Road Map

1 create Aspect

2 Develop Advice

- before,After returning ,after throwing

-after finally , Around

3 create point cut expression

4 apply to cRM project