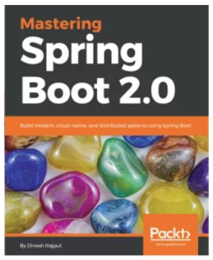
Dinesh Rajput
Spring 5 Design Patterns
Master efficient application development with patterns such as proxy, singleton, the template method, and more
Packt



Spring AOP is elegant feature of Spring Framework. It provides powerful to the cross cutting concern area into application. In this article I have collect Spring AOP (Aspect Oriented Programming) interview questions and answers. In my previous article I have introduced Spring Core interview questions answers, Spring Security interview questions answers and Spring MVC Interview questions answers. I could also follow my Spring AOP tutorial for depth understanding about Spring AOP framework.

Spring AOP Interview Questions

1. What is the concept of AOP? Which problem does it solve?
Aspect-Oriented Programming (AOP) is another way of thing to some areas of application i.e. cross cutting concern like security, logging and transaction. AOP is simple complement of OOP programming for different concerns. In OOP, the key unit of modularity is the class, whereas in AOP the unit of modularity is the aspect.

Aspect-Oriented Programming (AOP) enables modularization of cross-cutting concerns to solve following problems.

- To avoid tangling
- To eliminate scattering

Following Generic functionality that is needed in many places in your application

- Logging and Tracing
- Transaction Management

- Security
- Caching
- Error Handling
- Performance Monitoring
- Custom Business Rules

AOP terminologies

- Aspect
- Joint Point
- Advice
- Pointcut
- Introduction
- Target Object
- AOP Proxy
- Weaving

2. What is a pointcut, a join point, an advice, an aspect, weaving, Introduction, Target Object, AOP Proxy?

Pointcut
– An expression that selects one or more Join Points
Join Point
– A point in the execution of a program such as a method call or exception thrown
Advice
– Code to be executed at each selected Join Point
Aspect
– A module that encapsulates pointcuts and advice
Weaving
– Technique by which aspects are combined with main code
Introduction
-Spring AOP allows to introduce new interfaces (and a corresponding application) to any object advises.
Target Object
-An object is assisted by one or more respects. Also known as the object advised.
AOP Proxy
-AOP proxy is an object used to perform the contract area. This object is created by the AOP framework. In Spring AOP proxy is part of JDK dynamic proxy or proxy CGLIB.

3. How does Spring solve (implement) a cross cutting concern?

- Implement your mainline application logic
  - – Focusing on the core problem
- Write aspects to implement your cross-cutting concerns
  - – Spring provides many aspects out-of-the-box
- Weave the aspects into your application
  - – Adding the cross-cutting behaviors to the right places

4. Which are the limitations of the two proxy-types?

Spring will create either **JDK** or **CGLib** proxies

1. **JDK Proxy**
   1. Also called dynamic proxies
   2. API is built into the JDK
   3. Requirements: Java interface(s)
   4. All interfaces proxied
2. **CGLib Proxy**
   1. NOT built into JDK
   2. Included in Spring jars
   3. Used when interface not available
   4. Cannot be applied to final classes or methods

5. How many advice types does Spring support. What are they used for?

- **Before advice:** Advice that executes before a join point, but which does not have the ability to prevent execution flow proceeding to the join point (unless it throws an exception).
- **After returning advice:** Advice to be executed after a join point completes normally: for example, if a method returns without throwing an exception.
- **After throwing advice:** Advice to be executed if a method exits by throwing an exception.
- **After advice:** Advice to be executed regardless of the means by which a join point exits (normal or exceptional return).
- **Around advice:** Advice that surrounds a join point such as a method invocation. This is the most powerful kind of advice. Around advice can perform custom behavior before and after the method invocation. It is also responsible for choosing whether to proceed to the join point or to shortcut the advised method execution by returning its own return value or throwing an exception.

6. What do you have to do to enable the detection of the @Aspect annotation?

To use @AspectJ aspects in a Spring configuration you need to enable Spring support for configuring Spring AOP based on @AspectJ aspects, and autoproxying beans based on whether or not they are advised by those aspects.

Enabling @AspectJ Support with Java configuration

To enable @AspectJ support with Java @Configuration add the @EnableAspectJAutoProxy annotation:

```
@Configuration
@EnableAspectJAutoProxy
public class AppConfig {

}
```

Enabling @AspectJ Support with XML configuration

To enable @AspectJ support with XML based configuration use the <aop:aspectj-autoproxy/> element:

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

7. Name three typical cross cutting concerns?

1. Logging
2. Security
3. Transaction

8. What two problems arise if you don't solve a cross cutting concern via AOP?

Implementing Cross Cutting Concerns Without Modularization

• **Failing to modularize cross-cutting concerns leads to two things**

– Code tangling

• **A coupling of concerns**

– Code scattering

• **The same concern spread across modules**

Problem #1: Tangling

```
public class RewardNetworkImpl implements RewardNet
public RewardConfirmation rewardAccountFor(Dining d

//Non productive code or non functional code for bu
if (!hasPermission(SecurityContext.getPrincipal())
throw new AccessDeniedException();
}

Account a = accountRepository.findByCreditCard(…
Restaurant r = restaurantRepository.findByMerchantN
MonetaryAmount amt = r.calculateBenefitFor(account,
…
}
}
```

Problem #2: Scattering

```
public class JpaAccountManager implements AccountMa
public Account getAccountForEditing(Long id) {

//Non productive code or non functional code for bu
if (!hasPermission(SecurityContext.getPrincipal())
throw new AccessDeniedException();
}

…
public class JpaMerchantReportingService
implements MerchantReportingService {
public List<DiningSummary> findDinings(String merch
DateInterval interval) {

//Non productive code or non functional code for bu
if (!hasPermission(SecurityContext.getPrincipal())
throw new AccessDeniedException();
}

…
```

9. What does @EnableAspectJAutoProxy do?

To enable @AspectJ support with Java @Configuration add the @EnableAspectJAutoProxy annotation:

```
@Configuration
@EnableAspectJAutoProxy
public class AppConfig {

}
```

10. What is a named pointcut?

A named pointcut can be declared inside an <aop:config> element, enabling the pointcut definition to be shared across several aspects and advisors.

```
<aop:config>
    <aop:pointcut id="businessService" expression="
</aop:config>
```

11. How do you externalize pointcuts? What is the advantage of doing this?

Externalize the pointcut to a named pointcut. Avoid to writing complex pointcut expression across the application.

12. What is the JoinPoint argument used for?

Context provided by the JoinPoint parameter and Context about the intercepted point.

13. What is a ProceedingJoinPoint?
An around advice is a special advice that can control when and if a method (or other join point) is executed. This is true for around advices only, so they require an argument of type ProceedingJoinPoint, whereas other advices just use a plain JoinPoint. ProceedingJoinPoint is used as an argument of the methods which hints for before, after, after throwing and around. ProceedingJoinPoint has the methods like getKind, getTarget, proceed etc.

14. What are the five advice types called?

- Before
- After
- AfterThrowing
- AfterReturning
- Around

15. Which advice do you have to use if you would like to try and catch exceptions?
AfterThrowing

16. Limitations of Spring AOP?

- Can only advise non-private methods
- Can only apply aspects to Spring Beans
- Limitations of weaving with proxies
  - When using proxies, suppose method a() calls method b() on the same class/interface
- advice will never be executed for method b()

17. What are the supported AspectJ pointcut designators in Spring AOP?

- Execution
- This
- Target
- Args
- @target
- @args
- @within

- @annotation

18. How to declare aspect in Spring AOP?
In XML.

```
<bean class="com.doj.aop.LoggingAspect" id="logging
<!-- configure properties of aspect here -->
</bean>
```

In Java

```
@Aspect
@Component
class LoggingAspect{
//advice
//pointcut
}
```

19. How to declare a pointcut in Spring AOP?
Find the below code snippet.

```
@Pointcut("execution(* save(..))")
private void dataSave {}
```

20. What do you understand by Load-time weaving (LTW) in Spring?
Load-time weaving (LTW) or Run time weaving is a process of weaving AspectJ aspects into the classes of the application when the classes are being loaded in JVM.

Spring AOP Related Posts

1. Spring AOP-Introduction to Aspect Oriented Programming
2. @Aspect Annotation in Spring
3. Advices in Spring AOP
4. Spring AOP JoinPoints and Advice Arguments
5. Spring AOP-Declaring pointcut Expressions with Examples
6. Spring AOP XML configuration