Spring AOP

...

Introduction

What is AOP?

- Aspect Oriented Programming
- Aims to help with the separation of cross-cutting concerns to increase modularity
- Solves two problems:
  - Avoid tangling (Mixing business logic and cross-cutting concern)
  - Avoid scattering (Code duplication in several modules)

Cross Cutting Concerns

1. Logging
2. Auditing
3. Security
4. Transaction Management
5. Caching
6. Internationalization
7. Performance Monitoring

AOP Lingo

AOP Terminlogies

- Aspect
- Joinpoint
- Pointcut
- Advice
- Weaving

Aspect

- An aspect is a Java class that modularizes a set of concerns that cuts across multiple types and objects.
- You define an aspect by decorating a Java class with the @Aspect annotation.

JoinPoint

- The join point marks the execution point where aspect behavior and base behavior join.
- In Spring AOP, a join point always represents a method execution.
- In Spring AOP, only public method invocation join points are supported

Pointcut

Pointcut is a predicate or an expression used to identify join points.

It represents a point in the code where new behavior will be injected.

Pointcuts can be combined using the logical operators && (and), || (or) and ! (not).

Advice

- An action taken by an aspect at a particular join point.
- You can use five types of advice annotations: @Before, @After, @AfterReturning, @AfterThrowing, and @Around.
- Spring models an advice as an interceptor, maintaining a chain of interceptors around the join point.

Weaving

- Weaving is the process of applying aspects to your target advice objects.
- Spring AOP happens at runtime through dynamic proxies.

How does Spring solve a cross cutting concern?

Implementing a cross-cutting concern

- Spring uses proxy objects.
- Such proxy objects wrap the original Spring bean and intercepts method invocations specified by pointcuts defined by the cross cutting concern.

Proxy Techniques

JDK Dynamic Proxy

- default proxy mechanism used by Spring AOP
- require no additional libraries
- Proxies are created at runtime by generating a class that implements all the interfaces that the target object implements

CGLib Proxy

- To instruct Spring AOP to use CGLIB proxies by default

  @EnableAspectJAutoProxy(proxyTargetClass = true)

- It's included in the spring-core JAR
- Proxies are created by generating a subclass of the class implementing the target object

Proxy Techniques - Limitations

JDK Dynamic Proxy

- Must implement an interface
- Only public methods will be proxied
- If a method in the proxy calls another method in the proxy, and both match the pointcut expression of an advice, the advice will be executed only for the first method

CGLib Proxy

- Class and Methods cannot be final
- Only public and protected methods can be proxied

Advice Types

Advice Types

- @Before
- @After
  - Run advice after the method execution, regardless of its outcome
- @AfterReturning
  - Run advice after the a method execution only if method completes successfully
- @AfterThrowing
  - Run advice after the a method execution only if method exits by throwing an exception
- @Around

@EnableAspectJAutoProxy

- AopAutoConfiguration class in spring-boot-autoconfigure-x.y.z.RELEASE.jar will enable AOP configuration by default
- @EnableAspectJAutoProxy is not explicitly required

PointCut Designators

PointCut Designators

- A pointcut expression starts with a pointcut designator (PCD), which is a keyword telling Spring AOP what to match.

PointCut Designators

1. execution
    - execution(* com.sts.portal.repository.*.*(..))
2. within
    - within(com.wiley..*)
3. this
    - this(com.sts.repository.BaseRepository)
4. target
    - target(com.sts.portal.repository.CityRepository)
5. args

PointCut Designators

6. @target
7. @args
8. @within
9. @annotation
10. bean(*Repository)

ProceedingJoinPoint

ProceedingJoinPoint

- ProceedingJoinPoint class is a parameter to an around advice, only.
- When it's ready to pass control to the advised method, it will call ProceedingJoinPoint's proceed() method, which is used to execute the actual method.