Berner Fachhochschule

Spring AOP

Stephan Fischli

Introduction

- Aspect-oriented programming (AOP) enables the modularization of cross-cutting tasks and the separation of technical aspects from application logic
- AOP is used by the Spring Framework to provide declarative enterprise services (e.g. transactions, security), but can also be used by Spring applications (e.g. logging, profiling)
- There are several AOP libraries, the most widely used being **AspectJ**

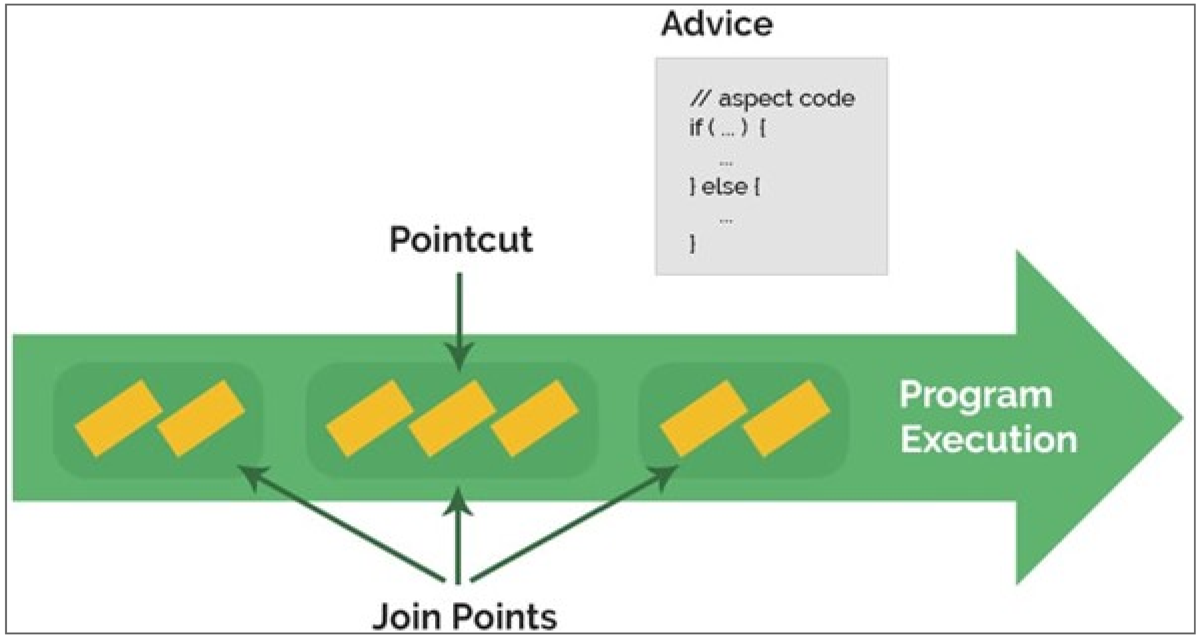
Spring AOP

Spring AOP

- is conceptually based on AspectJ but is simpler and less powerful
- can only be used with Spring beans and supports only method execution pointcuts
- uses JDK or GCLIB proxies to implement aspects (no code weaving)

AOP Concepts

- An *aspect* is a modularized concern that cuts over multiple components
- A *join point* is a point during the execution of a program
- An *advice* is an action that is executed at a join point
- A *pointcut* is a predicate that is used to match an advice to particular join points



Advice Types

Spring AOP supports the following types of advice:

- `@Before` advices run before a method invocation
- `@AfterReturning` advices run after a normal return from method
- `@AfterThrowing` advices run after a method throwed an exception
- `@After` advices run after a method regardless of its outcome
- `@Around` advices wrap a method invocation

Pointcuts

Spring AOP supports the following pointcuts designators:

- `execution` matches method executions with a specific signature pattern
- `within` matches join points within certain types (e.g. a package)
- `this` matches join points where the bean reference has a given type
- `target` matches join points where the target object has a given type
- `args` matches join points where the arguments have given types

Examples

```
@Aspect
@Component
public class LoggingAspect {
    @Before("within(org.example.*)")
    public void before(JoinPoint joinPoint) {
        System.out.println("Invoking method " +
joinPoint.getSignature());
    }
}
```

```
@Aspect
@Component
public class ProfilingAspect {
    @Around("execution(* org.example.*Service.*(..))")
    public Object around(ProceedingJoinPoint joinPoint) throws
Throwable {
        long startTime = System.currentTimeMillis();
        try {
            return joinPoint.proceed();
        } finally {
            long duration = System.currentTimeMillis() - startTime;
            System.out.println("Method took " + duration + " ms");
            }
        }
    }
```