**Spring Retry**

**Introduction**

Spring Retry provides an ability to automatically re-invoke a failed operation. To make processing more robust and less prone to failure, sometimes it helps to automatically retry a failed operation in case it might succeed on a subsequent attempt.

This is helpful where the errors may be transient in nature For example a remote call to a web service or RMI service that fails because of a network glitch or a **DeadLockLoserException** in a database update may resolve themselves after a short wait.  Spring Retry provides declarative control of the process and policy-based behavior that is easy to extend and customize.

**Maven Dependency**

**<dependency>**

**<groupId>org.springframework.retry</groupId>**

**<artifactId>spring-retry</artifactId>**

**<version>1.1.5.RELEASE</version>**

**</dependency>**

**Enable Spring Retry in your project**

To enable Spring Retry in an application, we need to add the **@EnableRetry** annotation to our **@Configuration** class.

e.g.

**@Configuration**

**@EnableRetry**

**public class ApplicationConfiguration {}**

**Approaches to incorporate Spring-retry logic**

**1: Custom Aspect to incorporate Spring-retry**

This approach should be fairly intuitive as the retry logic can be considered a cross cutting concern and a good way to implement a cross cutting concern is using Aspects.

@Aspect

public class RetryAspect {

@Autowired

private RetryTemplate retryTemplate;

@Pointcut("execution(\* retry.service..\*(..))")

public void serviceMethods() { }

@Around("serviceMethods()")

public Object aroundServiceMethods(ProceedingJoinPoint joinPoint) {

try {

return retryTemplate.execute(retryContext -> joinPoint.proceed());

} catch (Throwable e) {

throw new RuntimeException(e);

}

}

@Aspect

public class RetryAspect {

@Autowired

private RetryTemplate retryTemplate;

@Pointcut("execution(\* retry.service..\*(..))")

public void serviceMethods() {

//

}

@Around("serviceMethods()")

public Object aroundServiceMethods(ProceedingJoinPoint joinPoint) {

try {

return retryTemplate.execute(retryContext -> joinPoint.proceed());

} catch (Throwable e) {

throw new RuntimeException(e);

}

}

### 2: Using Spring-retry provided advice

Spring-retry project provides an advice which takes care of ensuring that targeted services can be retried.

The xml configuration looks like this:

**<aop:config>**

**<aop:pointcut id="transactional"**

**expression="execution(\* retry.service..\*(..))" />**

**<aop:advisor pointcut-ref="transactional"**

**advice-ref="retryAdvice" order="-1"/>**

**</aop:config>**

### 3: Declarative retry logic

This is the **recommended** approach, you will see that the code is far more concise than with the previous two approaches.

**public interface RemoteCallService {**

**@Retryable(maxAttempts = 3, backoff = @Backoff(delay = 2000))**

**String call() throws Exception;**

**}**