How to Use Spring Retry

by Chris Shayan · Jul. 12, 18 · Java Zone · Tutorial

A few days ago, I noticed that there is a group of people asking how to use Spring Retry. Before I go into the sample code, let me quickly explain the purpose behind Spring Retry. Spring Retry provides the ability to automatically re-invoke a failed operation. This is helpful when errors may be transient in nature (like a momentary network glitch). Spring Retry provides a declarative control of the process and policy-based behavior that is easy to extend and customize.

You can find the complete source code in here.

Maven Dependencies

```
<dependencies>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter</artifactId>
</dependency>
<dependency>
<groupId>org.springframework.retry</groupId>
<artifactId>spring-retry</artifactId>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
</dependency>
<dependency>
<groupId>org.aspectj</groupId>
<artifactId>aspectjweaver</artifactId>
</dependency>

<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>
```

Enable Retry

```
package com.chrisshayan.example.springretry;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.retry.annotation.EnableRetry;

@EnableRetry
@SpringBootApplication
public class SpringRetryApplication {

public static void main(String[] args) {
SpringApplication.run(SpringRetryApplication.class, args);
}

}
```

Using Retry With Annotations

```
package com.chrisshayan.example.springretry;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.retry.annotation.Backoff;
import org.springframework.retry.annotation.Recover;
import org.springframework.retry.annotation.Retryable;
import org.springframework.stereotype.Service;

@Service
public class SampleRetryService {
    private static final Logger LOGGER = LoggerFactory.getLogger(SampleRetryService.class);

    private static int COUNTER = 0;

    @Retryable(
            value = {TypeOneException.class, TypeTwoException.class},
            maxAttempts = 4, backoff = @Backoff(2000))
    public String retryWhenException() throws TypeOneException, TypeTwoException {
        COUNTER++;
        LOGGER.info("COUNTER = " + COUNTER);

        if(COUNTER == 1)
            throw new TypeOneException();
        else if(COUNTER == 2)
            throw new TypeTwoException();
        else
            throw new RuntimeException();
    }

    @Recover
    public String recover(Throwable t) {
```

```
        LOGGER.info("SampleRetryService.recover");
        return "Error Class :: " + t.getClass().getName();
    }
}
```

In order for your test class to work, the retry needs to be in the proper context. This is because we need to have another service that wraps around the retry. This is called:

```
package com.chrisshayan.example.springretry;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class SampleRetryClientService {

    @Autowired
    private SampleRetryService sampleRetryService;


    public String callRetryService() throws TypeOneException, TypeTwoException {
        return sampleRetryService.retryWhenException();
    }
}
```

Test Class

```
package com.chrisshayan.example.springretry;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)
@SpringBootTest
public class SpringRetryApplicationTests {

    private static final Logger LOGGER = LoggerFactory.getLogger(SpringRetryApplicationTests.class);
    @Autowired
    private SampleRetryClientService client;

@Test
public void contextLoads() {
}

@Test
    public void sampleRetryService() {
        try {
            final String message = client.callRetryService();
```

```
            LOGGER.info("message = " + message);
        } catch (TypeOneException | TypeTwoException e) {
            e.printStackTrace();
        }
    }

}
```

Console

```
2018-07-10 23:42:45.528  INFO 14583 --- [           main] c.c.e.springretry.SampleRetryService     : CC
2018-07-10 23:42:47.534  INFO 14583 --- [           main] c.c.e.springretry.SampleRetryService     : CC
2018-07-10 23:42:49.538  INFO 14583 --- [           main] c.c.e.springretry.SampleRetryService     : CC
2018-07-10 23:42:49.539  INFO 14583 --- [           main] c.c.e.springretry.SampleRetryService     : Sa
2018-07-10 23:42:49.539  INFO 14583 --- [           main] c.c.e.s.SpringRetryApplicationTests      : me
```

There are more capabilities in Spring Retry, such as Stateless Retry, Stateful Retry, and different retry policies and listeners. You can read more here.

Like This Article? Read More From DZone

Using Spring Data JPA Specification

Spring XML-Based DI and Builder Pattern

Command Patterns in Spring Framework

Free DZone Refcard
Java 13

Topics: SPRING , RETRY , RETRY PATTERN , JAVA , TUTORIAL