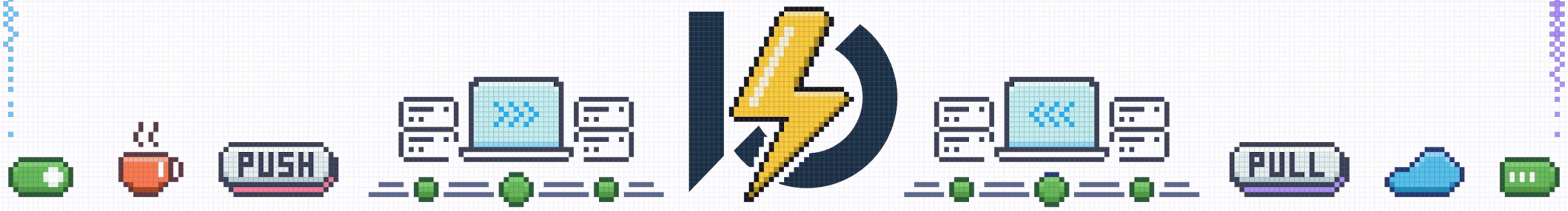
Core Resilience Features in Spring Framework 7

Juergen Hoeller



Agenda

- **Common Annotations**
  - Retryable
  - ConcurrencyLimit
- **Programmatic APIs**
  - RetryTemplate
  - SyncTaskExecutor

Design Goals

- **Virtual Threads friendly**
  - Blocking the current thread when necessary
  - Designed for high-concurrency scenarios

- **Aligned with BackOff and Reactor**
  - BackOff as used for JMS listener containers
  - Reactor's RetryBackoffSpec for Mono/Flux

Design Limits

- **Building blocks at java.util.concurrent level**
  - Empowering architectures on Virtual Threads
  - Analogous to common reactive operators

- **For component resilience, not cloud deployment**
  - Internal to the application
  - In addition to rate limiter, circuit breaker etc. at server/microservice boundary level.

Common Annotations

@Retryable with Defaults

```
@Service
public class MyService {

    @Retryable  // default: 1 initial attempt + maximum of 3 retries
    public void sendNotification() {
        this.jmsClient.destination("notifications").send(...);
    }
}
```

@Retryable with Max Retries

```
@Service
public class MyService {

    @Retryable(maxRetries = 4)  // 1 initial + maximum of 4 retries
    public void sendNotification() {
        this.jmsClient.destination("notifications").send(...);
    }
}
```

@Retryable with Exception Type

```
@Service
public class MyService {

    @Retryable(MessageDeliveryException.class)
    public void sendNotification() {
        this.jmsClient.destination("notifications").send(...);
    }
}
```

@Retryable with Custom Settings

```
@Service
public class MyService {

    @Retryable(includes = MessageDeliveryException.class,
            maxRetries = 4, delay = 100, jitter = 10,
            multiplier = 2, maxDelay = 1000)
    public void sendNotification() {
        this.jmsClient.destination("notifications").send(...);
    }
}
```

@Retryable for Reactive

```
@Service
public class MyService {

    @Retryable(maxRetries = 4, delay = 100)
    public Mono<Void> sendNotification() {
        return Mono.from(...);
    }
}
```

Events triggered by @Retryable

```
@Service
public class MyService {

    @EventListener
    public void logException(MethodRetryEvent event) {
        if (!event.isRetryAborted() {
            // event.getMethod() / event.getFailure() / etc.
        }
    }
}
```

@ConcurrencyLimit as a Throttle

```
@Service
public class MyService {

    @ConcurrencyLimit(10)
    public void sendNotification() {
        this.jmsClient.destination("notifications").send(...);
    }
}
```

@ConcurrencyLimit as a Lock

```
@Service
public class MyService {

    @ConcurrencyLimit(1)
    public void sendNotification() {
        this.jmsClient.destination("notifications").send(...);
    }
}
```

@EnableResilientMethods

```
@Configuration
@EnableResilientMethods  // covering @Retryable and @ConcurrencyLimit
public class MyConfig {

    ...
}
```

Programmatic APIs

RetryTemplate with Defaults

```
public void sendNotification() {
    var retryTemplate = new RetryTemplate();

    retryTemplate.invoke(
            () -> jmsClient.destination("notifications").send(...));
}
```

RetryTemplate with Max Retries

```
public void sendNotification() {
    var retryTemplate = new RetryTemplate(RetryPolicy.withMaxRetries(4));

    retryTemplate.invoke(
            () -> jmsClient.destination("notifications").send(...));
}
```

RetryTemplate with Exception Type

```
public void sendNotification() {
    var retryPolicy = RetryPolicy.builder()
        .includes(MessageDeliveryException.class)
        .excludes(...)
        .build();

    var retryTemplate = new RetryTemplate(retryPolicy);

    retryTemplate.invoke(
            () -> jmsClient.destination("notifications").send(...));
}
```

RetryTemplate with Custom Settings

```
public void sendNotification() {
    var retryPolicy = RetryPolicy.builder()
        .includes(MessageDeliveryException.class)
        .maxRetries(4)
        .delay(Duration.ofMillis(100))
        .jitter(Duration.ofMillis(10))
        .multiplier(2)
        .maxDelay(Duration.ofSeconds(1))
        .build();
    ...
}
```

RetryTemplate with Exception Handling

```
public void sendNotification() {
    try {
        retryTemplate.invoke(
                () -> jmsClient.destination("notifications").send(...));
    }
    catch (MessageDeliveryException ex) {
        // coming out of the original JmsClient send method
    }
}
```

RetryTemplate with Return Value

```
public void sendNotification() {
    try {
        var result = retryTemplate.invoke(() -> {
            jmsClient.destination("notifications").send(...);
            return "result";
        });
    }
    catch (MessageDeliveryException ex) {
        // coming out of the original JmsClient send method
    }
}
```

RetryTemplate with RetryException

```
public void sendNotification() {
    try {
        var result = retryTemplate.execute(() -> {
            jmsClient.destination("notifications").send(...);
            return "result";
        });
    }
    catch (RetryException ex) {
        // ex.getExceptions() / ex.getLastException() ...
    }
}
```

RetryTemplate with RetryListener

```
public void sendNotification() {
    var retryTemplate = new RetryTemplate();
    retryTemplate.setRetryListener(new RetryListener() {
        @Override
        public void onRetryableExecution(RetryPolicy retryPolicy,
                Retryable<?> retryable, RetryState retryState) {
            ...
        }
    });
    retryTemplate.invoke(
        () -> jmsClient.destination("notifications").send(...));
}
```

SyncTaskExecutor with Concurrency Limit

```
public void sendNotification() {
    var executor = new SyncTaskExecutor();
    executor.setConcurrencyLimit(10);

    executor.execute(
            () -> jmsClient.destination("notifications").send(...));
}
```

SyncTaskExecutor with Return Value

```
public void sendNotification() {
    var executor = new SyncTaskExecutor();
    executor.setConcurrencyLimit(10);

    var result = executor.execute(() -> {
        jmsClient.destination("notifications").send(...);
        return "result";
    });
}
```

SimpleAsyncTaskExecutor with Concurrency Limit

```
public void sendNotification() {
    var executor = new SimpleAsyncTaskExecutor();
    executor.setConcurrencyLimit(10);

    executor.execute(
            () -> jmsClient.destination("notifications").send(...));
}
```

SimpleAsyncTaskExecutor with Virtual Threads

```
public void sendNotification() {
    var executor = new SimpleAsyncTaskExecutor();
    executor.setConcurrencyLimit(10);
    executor.setVirtualThreads(true);

    executor.execute(
            () -> jmsClient.destination("notifications").send(...));
}
```

Wrapping Up

Availability

- **All available in Spring Framework 7.0.6**
  - Superseding the former Spring Retry project
  - Used by Spring Integration 7.0 and co.

- **Check out Spring's Virtual Threads support**
  - All across Spring Framework and Spring Boot
  - SimpleAsyncTaskExecutor/Scheduler etc.

Q&A