Configuring Resilience4J Circuit Breakers

Starters

There are two starters for the Resilience4J implementations, one for reactive applications and one for non-reactive applications.

- `org.springframework.cloud:spring-cloud-starter-circuitbreaker-resilience4j` - non-reactive applications
- `org.springframework.cloud:spring-cloud-starter-circuitbreaker-reactor-resilience4j` - reactive applications

Auto-Configuration

You can disable the Resilience4J auto-configuration by setting `spring.cloud.circuitbreaker.resilience4j.enabled` to `false`.

Default Configuration

To provide a default configuration for all of your circuit breakers create a `Customizer` bean that is passed a `Resilience4JCircuitBreakerFactory` or `ReactiveResilience4JCircuitBreakerFactory`. The `configureDefault` method can be used to provide a default configuration.

```java
@Bean
public Customizer<Resilience4JCircuitBreakerFactory> defaultCustomizer() {
        return factory -> factory.configureDefault(id -> new Resilience4JConfig
                        .timeLimiterConfig(TimeLimiterConfig.custom().timeoutDu
                        .circuitBreakerConfig(CircuitBreakerConfig.ofDefaults()
                        .build());
}
```

Reactive Example

```java
@Bean
public Customizer<ReactiveResilience4JCircuitBreakerFactory> defaultCustomizer(
        return factory -> factory.configureDefault(id -> new Resilience4JConfig
                        .circuitBreakerConfig(CircuitBreakerConfig.ofDefaults()
                        .timeLimiterConfig(TimeLimiterConfig.custom().timeoutDu
}
```

Customizing The ExecutorService

If you would like to configure the `ExecutorService` which executes the circuit breaker you can do so using the `Resilience4JCircuitBreakerFactor`.

For example if you would like to use a context aware `ExecutorService` you could do the following.

```java
@Bean
public Customizer<ReactiveResilience4JCircuitBreakerFactory> defaultCustomizer(
        return factory -> {
                ContextAwareScheduledThreadPoolExecutor executor = ContextAware
                        .build();
                factory.configureExecutorService(executor);
        };
}
```

Specific Circuit Breaker Configuration

Similarly to providing a default configuration, you can create a `Customizer` bean this is passed a `Resilience4JCircuitBreakerFactory` or `ReactiveResilience4JCircuitBreakerFactory`.

```java
@Bean
public Customizer<Resilience4JCircuitBreakerFactory> slowCustomizer() {
        return factory -> factory.configure(builder -> builder.circuitBreakerCo
```

Convert web pages and HTML files to PDF in your applications with the Pdfcrowd HTML to PDF API Printed with Pdfcrowd.com

```
                         .timeLimiterConfig(TimeLimiterConfig.custom().timeoutDu
}
```

In addition to configuring the circuit breaker that is created you can also customize the circuit breaker after it has been created but before it is returned to the caller. To do this you can use the `addCircuitBreakerCustomizer` method. This can be useful for adding event handlers to Resilience4J circuit breakers.

```java
@Bean
public Customizer<Resilience4JCircuitBreakerFactory> slowCustomizer() {
        return factory -> factory.addCircuitBreakerCustomizer(circuitBreaker ->
        .onError(normalFluxErrorConsumer).onSuccess(normalFluxSuccessConsumer),
}
```

Reactive Example

```java
@Bean
public Customizer<ReactiveResilience4JCircuitBreakerFactory> slowCustomizer() {
        return factory -> {
                factory.configure(builder -> builder
                .timeLimiterConfig(TimeLimiterConfig.custom().timeoutDuration(D
                .circuitBreakerConfig(CircuitBreakerConfig.ofDefaults()), "slow
                factory.addCircuitBreakerCustomizer(circuitBreaker -> circuitBr
                .onError(normalFluxErrorConsumer).onSuccess(normalFluxSuccessCo
     };
}
```

Circuit Breaker Properties Configuration

You can configure `CircuitBreaker` and `TimeLimiter` configs or instances in your application's configuration properties file. Property configuration has higher priority than Java `Customizer` configuration.

Descending priority from top to bottom.

- Method(id) config - on specific method or operation
- Service(group) config - on specific application service or operations

Convert web pages and HTML files to PDF in your applications with the Pdfcrowd HTML to PDF API | Printed with Pdfcrowd.com

- Global default config

```java
ReactiveResilience4JCircuitBreakerFactory.create(String id, String groupName)
Resilience4JCircuitBreakerFactory.create(String id, String groupName)
```

Global Default Properties Configuration

```
resilience4j.circuitbreaker:
    configs:
        default:
            registerHealthIndicator: true
            slidingWindowSize: 50

resilience4j.timelimiter:
    configs:
        default:
            timeoutDuration: 5s
            cancelRunningFuture: true
```

Configs Properties Configuration

```
resilience4j.circuitbreaker:
    configs:
        groupA:
            registerHealthIndicator: true
            slidingWindowSize: 200

resilience4j.timelimiter:
    configs:
        groupC:
            timeoutDuration: 3s
            cancelRunningFuture: true
```

Instances Properties Configuration

```
resilience4j.circuitbreaker:
 instances:
```

Convert web pages and HTML files to PDF in your applications with the Pdfcrowd HTML to PDF API | Printed with Pdfcrowd.com

```
      backendA:
          registerHealthIndicator: true
          slidingWindowSize: 100
      backendB:
          registerHealthIndicator: true
          slidingWindowSize: 10
          permittedNumberOfCallsInHalfOpenState: 3
          slidingWindowType: TIME_BASED
          recordFailurePredicate: io.github.robwin.exception.RecordFailurePredic

 resilience4j.timelimiter:
  instances:
      backendA:
          timeoutDuration: 2s
          cancelRunningFuture: true
      backendB:
          timeoutDuration: 1s
          cancelRunningFuture: false
```

- `ReactiveResilience4JCircuitBreakerFactory.create("backendA")` or `Resilience4JCircuitBreakerFactory.create("backendA")` will apply `instances backendA properties`
- `ReactiveResilience4JCircuitBreakerFactory.create("backendA", "groupA")` or `Resilience4JCircuitBreakerFactory.create("backendA", "groupA")` will apply `instances backendA properties`
- `ReactiveResilience4JCircuitBreakerFactory.create("backendC")` or `Resilience4JCircuitBreakerFactory.create("backendC")` will apply `global default properties`
- `ReactiveResilience4JCircuitBreakerFactory.create("backendC", "groupC")` or `Resilience4JCircuitBreakerFactory.create("backendC", "groupC")` will apply `global default CircuitBreaker properties and config groupC TimeLimiter properties`

For more information on Resilience4j property configuration, see Resilience4J Spring Boot 2 Configuration.

Bulkhead pattern supporting

If `resilience4j-bulkhead` is on the classpath, Spring Cloud CircuitBreaker will wrap all methods with a Resilience4j Bulkhead. You can disable the Resilience4j Bulkhead by setting `spring.cloud.circuitbreaker.bulkhead.resilience4j.enabled` to `false`.

Convert web pages and HTML files to PDF in your applications with the Pdfcrowd HTML to PDF API Printed with Pdfcrowd.com

Spring Cloud CircuitBreaker Resilience4j provides two implementation of bulkhead pattern:

- a `SemaphoreBulkhead` which uses Semaphores
- a `FixedThreadPoolBulkhead` which uses a bounded queue and a fixed thread pool.

By default, Spring Cloud CircuitBreaker Resilience4j uses `FixedThreadPoolBulkhead`. To modify the default behavior to use `SemaphoreBulkhead` set the property `spring.cloud.circuitbreaker.resilience4j.enableSemaphoreDefaultBulkhead` to `true`.

For more information on implementation of Bulkhead patterns see the Resilience4j Bulkhead.

The `Customizer<Resilience4jBulkheadProvider>` can be used to provide a default `Bulkhead` and `ThreadPoolBulkhead` configuration.

```java
@Bean
public Customizer<Resilience4jBulkheadProvider> defaultBulkheadCustomizer() {
    return provider -> provider.configureDefault(id -> new Resilience4jBulkhead
        .bulkheadConfig(BulkheadConfig.custom().maxConcurrentCalls(4).build())
        .threadPoolBulkheadConfig(ThreadPoolBulkheadConfig.custom().coreThreadP
        .build()
);
}
```

Specific Bulkhead Configuration

Similarly to proving a default 'Bulkhead' or 'ThreadPoolBulkhead' configuration, you can create a `Customizer` bean this is passed a `Resilience4jBulkheadProvider`.

```java
@Bean
public Customizer<Resilience4jBulkheadProvider> slowBulkheadProviderCustomizer(
    return provider -> provider.configure(builder -> builder
        .bulkheadConfig(BulkheadConfig.custom().maxConcurrentCalls(1).build())
        .threadPoolBulkheadConfig(ThreadPoolBulkheadConfig.ofDefaults()), "slow
}
```

In addition to configuring the Bulkhead that is created you can also customize the bulkhead and thread pool bulkhead after they have been created but before they are returned to caller. To do this

Convert web pages and HTML files to PDF in your applications with the Pdfcrowd HTML to PDF API | Printed with Pdfcrowd.com

you can use the `addBulkheadCustomizer` and `addThreadPoolBulkheadCustomizer` methods.

Bulkhead Example

```java
@Bean
public Customizer<Resilience4jBulkheadProvider> customizer() {
    return provider -> provider.addBulkheadCustomizer(bulkhead -> bulkhead.getE
        .onCallRejected(slowRejectedConsumer)
        .onCallFinished(slowFinishedConsumer), "slowBulkhead");
}
```

Thread Pool Bulkhead Example

```java
@Bean
public Customizer<Resilience4jBulkheadProvider> slowThreadPoolBulkheadCustomize
    return provider -> provider.addThreadPoolBulkheadCustomizer(threadPoolBulkh
        .onCallRejected(slowThreadPoolRejectedConsumer)
        .onCallFinished(slowThreadPoolFinishedConsumer), "slowThreadPoolBulkhea
}
```

Bulkhead Properties Configuration

You can configure ThreadPoolBulkhead and SemaphoreBulkhead instances in your application's configuration properties file. Property configuration has higher priority than Java `Customizer` configuration.

```
resilience4j.thread-pool-bulkhead:
    instances:
        backendA:
            maxThreadPoolSize: 1
            coreThreadPoolSize: 1
resilience4j.bulkhead:
    instances:
        backendB:
            maxConcurrentCalls: 10
```

Convert web pages and HTML files to PDF in your applications with the Pdfcrowd HTML to PDF API Printed with Pdfcrowd.com

For more inforamtion on the Resilience4j property configuration, see Resilience4J Spring Boot 2 Configuration.

Collecting Metrics

Spring Cloud Circuit Breaker Resilience4j includes auto-configuration to setup metrics collection as long as the right dependencies are on the classpath. To enable metric collection you must include `org.springframework.boot:spring-boot-starter-actuator`, and `io.github.resilience4j:resilience4j-micrometer`. For more information on the metrics that get produced when these dependencies are present, see the Resilience4j documentation.

Note

You don’t have to include `micrometer-core` directly as it is brought in by `spring-boot-starter-actuator`

Convert web pages and HTML files to PDF in your applications with the Pdfcrowd HTML to PDF API | Printed with Pdfcrowd.com