MicroProfile Fault Tolerance

This cheat sheet covers the basics of MicroProfile Fault Tolerance specification uses.

DEFINED INTERCEPTOR BINDINGS

@Timeout

`@Timeout` prevents the execution from waiting forever. It is recommended that microservice invocations should have an associated timeout.

Usage

```
@Timeout(4000)
public void timeout() {
    // ...
}

@Timeout(value = 4, unit = ChronoUnit.SECONDS)
public void timeout() {
    // ...
}
```

@Retry

In order to recover from a brief network glitch, `@Retry` can be used to invoke the same operation again.

Usage

```
@Retry(maxRetries = 90, maxDuration= 1000)
public void retry() {
    // ...
}

@Retry(delay = 400, maxDuration= 3200, jitter= 400,
maxRetries = 10)
public void retry() {
    // ...
}
```

@Fallback

A Fallback method is invoked if a method annotated with `@Fallback` completes exceptionally.

Usage - FallbackHandler class

```
@Fallback(MyFallbackHandler.class)
public String fallbackedMethod() {
    // ...
    throw new RuntimeException();
}

public class MyFallbackHandler implements
FallbackHandler<String> {
    @Override
    public String handle(ExecutionContext
executionContext) {
        return "fallback value";
    }
}
```

Usage - Fallback method

```
@Fallback(fallbackMethod = "fallbackMethod")
public String fallbackedMethod() {
    // ...
    throw new RuntimeException();
}

public String fallbackMethod() {
    return "fallback method value";
}
```

`fallbackMethod` must be in the same class as `@Fallback` annotated method.

@CircuitBreaker

A Circuit Breaker prevents repeated failures, so that dysfunctional services or APIs fail fast.

Circuit states

`closed`: normal request processing

`open`: requests fail immediately

`half-open`: some trial requests are processed

Transitions

`closed > open`: on failure (exception)

`open > half-open`: after specified delay

`half-open > open`: if trial requests fail

`half-open > closed`: if trial requests pass

Usage

```
@CircuitBreaker(successThreshold = 10,
requestVolumeThreshold = 4, failureRatio=0.5,
delay = 1000)
public Connection serviceA() {
   // ...
}
```

@Bulkhead

The Bulkhead pattern is to prevent faults in one part of the system from cascading to the entire system by limiting the number of concurrent requests accessing an instance.

Usage

```
@Bulkhead(5)
public void bulkhead() {
    // ...
}
```

@Asynchronous

`@Asynchronous` means the execution of the client request will be on a separate thread. Use together with other fault tolerance annotations.

Usage

The method annotated with `@Asynchronous` must return a Future or a `CompletionStage` from the `java.util.concurrent` package.

```
@Asynchronous
public CompletionStage<Void> asynchronous() {
    // ...
}
```

CONFIGURATION

Default schemes

```
<classname>/<methodname>/<annotation>/<parameter>:
(com.acme.test.MyClient/doSomeOp/Retry/
maxRetries=100)
```

```
<classname>/<annotation>/<parameter>
(com.acme.test.MyClient/Retry/maxRetries=100)
```

```
<annotation>/<parameter> (Retry/maxRetries=30)
```

Disabling individual Fault Tolerance policies

For example:

```
com.acme.test.MyClient/methodA/CircuitBreaker/
enabled=false
```

Author Martin Stefanko
Senior Software Engineer
Middleware Runtimes Sustaining Engineering Team