Kresil - Kotlin Resilience

Kotlin Multiplatform Library for Fault-Tolerance

Francisco José Barbosa Engenheiro

Supervisor: Pedro Félix

Final report written for Project and Seminary
BSc in Computer Science and Engineering

July 2024

Instituto Superior de Engenharia de Lisboa

Kresil - Kotlin Resilience

Kotlin Multiplatform Library for Fault-Tolerance

49428 Francisco José Barbosa Engenheiro

Supervisor: Pedro Félix, ISEL

Final report written for Project and Seminary
BSc in Computer Science and Engineering

July 2024

Abstract

Text of the abstract. Brief description of the project, important results, and conclusions: the goal is to provide the reader with an overview of the project (should not exceed one page).

Keywords: list of keywords separated by ;.

Resumo

Texto do resumo. Breve descrição do projeto, dos resultados importantes e das conclusões: o objetivo é dar ao leitor uma visão global do projeto (não deve exceder uma página).

Palavras-chave: lista de palavras-chave separadas por ;.

Contents

Chapter 1

Introduction

1.1 Context

In the modern era, our reliance on digital services has grown exponentially, driving the need for these services to be highly reliable and available at all times. Whether it's financial transactions, healthcare systems, or social media platforms, users expect uninterrupted access and seamless experiences. This expectation places significant pressure on the underlying infrastructure to handle failures gracefully and maintain service continuity. Achieving this level of reliability requires sophisticated mechanisms to manage and mitigate faults effectively.

Most of these services are built on top of a distributed system, which consist of independent networked computers that present themselves to users as a single, coherent system [1]. Given the complexity of these systems, they are susceptible to failures caused by a variety of factors, such as hardware malfunctions, software bugs, network issues, communication problems, or even human errors. As such, its is crucial to ensure that services within distributed systems are resilient and fault-tolerant.

Fault tolerance and fault resilience are key concepts in this context, and while they are related and sometimes used interchangeably, they have subtle differences [2]:

- **Fault Tolerance**: A fault-tolerant service is a service that is able to maintain all or part of its functionality, or provide an alternative, when one or more of its associated components fail. The user does not observe see any fault except for some possible delay during which failover occurs.

- **Fault Resilience**: A fault-resilient service acknowledges faults but ensures that they do not impact committed data (i.e., the database may respond with an error to the attempt to commit a transaction, etc.).

These distinctions are important, because it is possible to regard a fault-tolerant service as suffering *no* downtime even if the machine it is running on crashes, whereas the potential data fault in a fault resilient service counts toward downtime.

1.2 Resilience Mechanisms

Over the years, several resilience mechanisms have been developed to help implemented build more robust and reliable systems. These mechanisms provide a set of tools and strategies to handle the inevitable occurrence of failures. Some of the most common mechanisms are described in table 1.1.

Table 1.1: Resiliency mechanisms from *Resilience4j* [3] documentation

Name	**Funcionality**	**Description**
Retry	Repeats failed executions.	Many faults are transient and may self-correct after a short delay.
Circuit Breaker	Temporary blocks possible failures.	When a system is seriously struggling, failing fast is better than making clients wait.
Rate Limiter	Limits executions/period.	Limit the rate of incoming requests.
Time Limiter	Limits duration of execution.	Beyond a certain wait interval, a successful result is unlikely.
Bulkhead	Limits concurrent executions.	Resources are isolated into pools so that if one fails, the others will continue working.
Cache	Memorizes a successful result.	Some proportion of requests may be similar.
Fallback	Defines an alternative value to be returned (or action to be executed) on failure.	Things will still fail - plan what you will do when that happens.

1.3 Technologies

Kotlin Multiplatform (not in depth)

1.4 Project Goal

Multiplatform library for kmp with resilience mechanisms

1.5 Related Work

1.5.1 Ktor

Mention plugin integration

1.5.2 Other solutions

Other libraries with resilience mechanisms: Polly, resilience4j, Hystrix, Arrow

1.6 Document Structure

Chapter 2

Kotlin Multiplatform

2.1 Project Structure

Mention gradle project (divide in modules, gradle build file)

2.2 Platform-Dependent Code

2.3 Running Tests

2.4 Other Aspects

What was done to have concurrency, logging, CI integration, etc

Chapter 3

Common Design and Implementation Strategy

For all mechanisms

3.1 Design Aspects

All the design and implementation aspects that are common to all mechanisms - use mechanism model

3.2 Implementation Aspects

- configuration - decoration - ktor pipeline plugin integration

Chapter 4

Retry

4.1 Introduction

- Why it exists (1) - Functional characterization (2)

4.2 Configuration

- mention default values and why they were chosen

4.3 Implementation Aspects

4.4 Ktor Integration

Chapter 5

Circuit Breaker

5.1 Introduction

- Why it exists (1) - Functional characterization (2)

5.2 Configuration

- mention default values and why they were chosen

5.3 Implementation Aspects

5.4 Ktor Integration

Bibliography

[1] FreeCodeCamp contributors. A thorough introduction to distributed systems. https://www.freecodecamp.org/news/a-thorough-introduction-to-distributed-systems-3b91562c9b3c, 2024. [Online; accessed 5-March-2024].

[2] James Bottomley. Fault tolerance vs fault resilience. https://www.usenix.org/legacy/publications/library/proceedings/usenix04/tech/sigs/full_papers/bottomley/bottomley_html/node21.html, 2004. [Online; accessed 21-May-2024].

[3] resilience4j contributors. Resilience4j: User guide. https://resilience4j.readme.io/docs/getting-started, 2024. [Online; accessed 6-March-2024].

Appendix A

Appendix Example

Etiam ac leo a risus tristique nonummy. Donec dignissim tincidunt nulla. Vestibulum rhoncus molestie odio. Sed lobortis, justo et pretium lobortis, mauris turpis condimentum augue, nec ultricies nibh arcu pretium enim. Nunc purus neque, placerat id, imperdiet sed, pellentesque nec, nisl. Vestibulum imperdiet neque non sem accumsan laoreet. In hac habitasse platea dictumst. Etiam condimentum facilisis libero. Suspendisse in elit quis nisl aliquam dapibus. Pellentesque auctor sapien. Sed egestas sapien nec lectus. Pellentesque vel dui vel neque bibendum viverra. Aliquam porttitor nisl nec pede. Proin mattis libero vel turpis. Donec rutrum mauris et libero. Proin euismod porta felis. Nam lobortis, metus quis elementum commodo, nunc lectus elementum mauris, eget vulputate ligula tellus eu neque. Vivamus eu dolor.

Nulla in ipsum. Praesent eros nulla, congue vitae, euismod ut, commodo a, wisi. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Aenean nonummy magna non leo. Sed felis erat, ullamcorper in, dictum non, ultricies ut, lectus. Proin vel arcu a odio lobortis euismod. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Proin ut est. Aliquam odio. Pellentesque massa turpis, cursus eu, euismod nec, tempor congue, nulla. Duis viverra gravida mauris. Cras tincidunt. Curabitur eros ligula, varius ut, pulvinar in, cursus faucibus, augue.

Nulla mattis luctus nulla. Duis commodo velit at leo. Aliquam vulputate magna et leo. Nam vestibulum ullamcorper leo. Vestibulum condimentum rutrum mauris. Donec id mauris. Morbi molestie justo et pede. Vivamus eget turpis sed nisl cursus tempor. Curabitur mollis sapien condimentum nunc. In wisi nisl, malesuada at, dignissim sit amet, lobortis in, odio. Aenean consequat arcu a ante. Pellentesque porta elit sit amet orci. Etiam at turpis nec elit ultricies imperdiet. Nulla facilisi. In hac habitasse platea dictumst. Suspendisse viverra aliquam risus. Nullam pede justo, molestie nonummy, scelerisque eu, facilisis vel, arcu.