Kresil - Kotlin Multi-Platform Library for Fault-Tolerance

Francisco Engenheiro, n.º 49428, e-mail: a49428@alunos.isel.pt, tel.: 928051992

Orientadores: Pedro Félix, e-mail: pedro.felix@isel.pt

Março de 2023

1 Contexto

1.1 Necessidade de Desenho de Software Resiliente

Grande parte dos sistemas contemporâneos são sistemas distribuídos, ou seja, representam um conjunto de computadores independentes entre si, ligados através de uma rede de dados, que se apresentam aos utilizadores como um sistema único e coerente [1]. No entanto, essa interdependência traz consigo o risco de falhas, e quando um desses componentes falha, toda ou parte da funcionalidade do sistema pode ser comprometida. Tal pode resultar em

perda de dados, indisponibilidade de serviços e outros problemas, dependendo da criticidade do(s) componente(s) afetado(s) [2]. É neste contexto que surge a necessidade de desenhar software resiliente, capaz de lidar com falhas e manter a sua funcionalidade mesmo quando um ou mais dos seus componentes falham ou estão temporariamente indisponíveis.

1.2 Bibliotecas como Mecanismos de Resiliência

Fornecem mecanismos para lidar com as eventuais falhas inerentes a componentes de um sistema distribuído, tentando garantir ao máximo a disponibilidade e confiabilidade dos serviços que estes disponibilizam.

Biblioteca	**Linguagem**	**Plataforma**
Netflix's Hystrix [3]	Java	JVM
Resilience4j [4]	Java/Kotlin	JVM
Polly [5]	C#	.NET

Tabela 1: Exemplos de bibliotecas como mecanismos de resiliência

Exemplos de mecanismos de resiliência disponibilizados por estas bibliotecas:

- **Retry**: Tenta novamente uma operação que falhou, aumentando a probabilidade de sucesso;
- **Rate Limiter**: Limita a taxa de requisições que um serviço pode receber;
- **Circuit Breaker**: Interrompe, temporariamente, a comunicação com um serviço que está a falhar, de forma a evitar que o mesmo sobrecarregue o sistema. Semelhante a um disjuntor elétrico;
- **Fallback**: Fornecer um valor ou executa uma ação alternativa caso uma operação falhe.

A biblioteca Polly [5] divide os mecanismos de resiliência que disponibiliza em duas categorias:

- **Resiliência Reativa**: Reage a falhas e mitiga o seu impacto;
- **Resiliência Proativa**: Previne que as falhas aconteçam.

De referir que, as bibliotecas mencionadas anteriormente serão exploradas ao longo da fase de planeamento e desenvolvimento do projeto, de forma a que se possa tirar proveito do conhecimento inerente à sua implementação. Com especial atenção para a biblioteca *Resilience4j* [4], uma vez que apresenta um módulo de interoperabilidade com *Kotlin* para a plataforma *JVM*.

1.3 Kotlin Multiplatform

A tecnologia Kotlin Multiplatform [6] possibilita a partilha do código nativo da aplicação entre diversas plataformas, tornando-o independente de qualquer plataforma específica.

Para cada plataforma alvo, regularmente denominada como *target*, poderão ter que existir implementações adicionais porque:

- Um determinado *target* não suporta diretamente o *KMP*, como é o caso do *Node.js*, e por isso é necessário criar um *adapter* para a comunicação com o código comum (*CommonMain* - que está em *Kotlin*);
- Uma determinada funcionalidade não consegue ser implementada de forma comum porque:
  - é necessário saber detalhes especificos do *target* para a sua implementação. Usando o padrão *expect/actual* é possível criar definições comuns (*expect*) e implementações específicas para cada *target* (*actual*);
  - as bibliotecas disponíveis para código comum (*Kotlin Stdlib + Kotlinx*) não cobrem a funcionalidade(s) pretendida(s) e não se quer usar outra(s) biblioteca(s) *KMP* como dependência(s) do projeto.

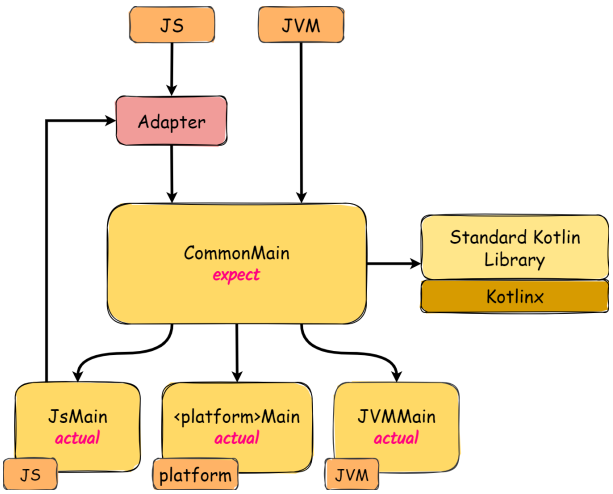

Figura 1: Arquitetura do KMP

Portanto, o objetivo principal do *KMP* é maximizar a reutilização de código, minimizando o código específico de cada plataforma visto que terá que ser replicado para cada plataforma suportada.

1.4 Ktor

É uma framework para *KMP* desenhada para criar aplicações assíncronas de servidor e cliente, que se tornou popular devido à sua simplicidade e facilidade de utilização.

Utiliza a linguagem *Kotlin* e o sistema de *Coroutines*, permitindo este último, de forma simplificada, a execução assíncrona de código de forma sequencial e sem bloqueio de threads, tirando maior proveito do sistema computacional disponível.

Referências

[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] Wikipedia contributors. Cap theorem. https://en.wikipedia.org/wiki/CAP_theorem, 2024. [Online; accessed 5-March-2024].

[3] Netflix contributors. Hystrix: Latency and fault tolerance for distributed systems. https://github.com/Netflix/Hystrix, 2024. [Online; accessed 6-March-2024].

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

[5] App-vNext contributors. Polly: Resilience strategies. https://github.com/App-vNext/Polly#resilience-strategies, 2024. [Online; accessed 6-March-2024].

[6] JetBrains contributors. Kotlin multiplatform. https://kotlinlang.org/docs/multiplatform.html, 2024. [Online; accessed 7-March-2024].