Programación Funcional en Java

Círculo Siete Capacitación

Clase 10 de 12

19 Marzo 2025

Uso de Future en Vavr para Programación Asíncrona

- En Java, la ejecución de tareas en paralelo y asincrónicas tradicionalmente se maneja con ***Threads***, ***ExecutorService***, y ***CompletableFuture***.
- Vavr proporciona ***Future<T>***, una abstracción funcional que facilita la ejecución de operaciones en segundo plano, sin la complejidad de manejar hilos manualmente.

¿Qué es Future<T> en Vavr?

- ***Future<T>*** representa una operación que se ejecuta en paralelo y que puede completarse en el futuro.
- Beneficios:
  - No bloqueante → No detiene el hilo principal.
  - Evita ***try-catch*** → Maneja errores con ***Try<T>***.
  - Encadenable → Soporta ***map(), flatMap(), onSuccess(), onFailure()***.
  - Mejor que ***CompletableFuture*** → Más funcional y composable.
- Ver Lab01.java

Encadenamiento de Future<T> con map() y flatMap()

- ***map()*** transforma valores sin modificar el estado del ***Future<T>***.
- ***flatMap()*** evita anidaciones innecesarias y permite combinaciones más expresivas.
- Ver Lab02.java

Manejo de Errores con onFailure() y recover()

- Manejo de errores sin ***try-catch***, de manera declarativa y funcional.
- Ver Lab03.java

Sincronización de Future<T> con get() y await()

- ***await()*** evita bloqueos indefinidos al permitir tiempos máximos de espera.
- Ver Lab04.java

Combinación de Múltiples Future<T>

- Permite ejecutar múltiples tareas en paralelo y combinar sus resultados.
- Ver Lab05.java

Reducción y filtrado de Futures

- Ver Lab06.java

Try y Future

- Es posible integrar ambas monadas para mejorar manejo de errores.
- Ver Lab07.java

Como se ejecutan los Futures

- Vavr necesita un ***ExecutorService*** para encolar los hilos.
- Por omision, ***Vavr*** decide el tamaño y tipo de este ***ExecutorService***.
- Pero es posible cambiar este comportamiento para poder controlar el ***ExecutorService*** completamente.
- Ver Lab 08.java

Uso de Future en Vavr con Virtual Threads (Project Loom)

- Desde Java 19 (en preview) y 21 (production ready), ***Virtual Threads*** (***Thread.ofVirtual()***) permiten ejecutar tareas concurrentes de manera más eficiente que los hilos tradicionales, gracias a su ligereza y menor sobrecarga.
- ¿Cómo combinar ***Future<T>*** de Vavr con Virtual Threads?
  - Ejecutar tareas de manera asíncrona y concurrente.
  - Aprovechar el escalado masivo sin bloquear el sistema.
  - Reducir el consumo de memoria y mejorar la eficiencia.

Creación de un Future<T> Usando Virtual Threads

- Future<T> ejecuta la tarea dentro de un Virtual Thread, permitiendo mejor escalabilidad.
- Ver Lab09.java

Ejecutar Múltiples Future<T> en Virtual Threads

- Correr varias tareas concurrentemente con Virtual Threads
- Se ejecutan múltiples tareas sin bloquear el sistema, utilizando Virtual Threads.
- Ver Lab10.java

Future<T> con VirtualThreadFactory para Mayor Control

- Se puede personalizar la configuración de los Virtual Threads.
- Ver Lab11.java

VirtualThreads masivos

- Los hilos virtuales consumen alrededor de ~200 bytes de memoria, puede ser un poco mas dependiendo la arquitectura de la JVM.
- Los hilos de plataforma (hilos habituales que todos conocemos) consumen alrededor de 1MB, tal vez más dependiendo la plataforma.
- Es “barato” usarlo hilos virtuales y gracias al buen diseño de las APIs de Vavr, se pueden usar.
- Ver Lab12.java

Notas finales

- Future<T> en Vavr con Virtual Threads permite ejecutar operaciones asíncronas de manera segura, eficiente y no bloqueante.
- Beneficios clave:
  - Ejecución en Virtual Threads sin sobrecarga de memoria.
  - Encadenamiento funcional (map(), flatMap()).
  - Manejo de errores sin try-catch (recover(), onFailure()).
  - Más expresivo y funcional que CompletableFuture<T>.
- Para escribir código concurrente más eficiente en Java, usa Future<T> de Vavr con Virtual Threads, combo breaker!