Spring Boot Event Timer

Einführung

In der Praxis werden je nach Anwendungsfall Arbeiten in synchrone und asynchrone Aufgaben unterteilt. Als Beispiel dient uns z.B. ein UI Formular, welches mit jedem erfolgreichen Request ein PDF Dokument zum Zweck der Nachverfolgbarkeit zu erstellen ist. Dieses PDF Dokument wird asynchron erstellt und damit vom synchronen Request entkoppelt.Die asynchrone Aufgabe wird z.B. über eine Datenbank Tabelle (e.g. «events») verwaltet. Der Scheduled Event Timer liest im Intervall nach gespeicherten Events in der Datenbank. Jedes Event wird je nach Typ via Application Event Publisher synchron ausgelöst.

Scheduled Event Timer

Das folgende Listing zeigt die Klasse Event Timer mit 3 statischen Member Klassen EventOne, EventTwo, EventThree:package ch.std.event.timer;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.context.ApplicationEventPublisher;
import org.springframework.context.event.EventListener;
import org.springframework.scheduling.annotation.Scheduled;
import org.springframework.stereotype.Component;

@Component
public class EventTimer {

 Logger logger = LoggerFactory.getLogger(EventTimer.class);

 private ApplicationEventPublisher applicationEventPublisher;

 public EventTimer(ApplicationEventPublisher applicationEventPublisher) {
 this.applicationEventPublisher = applicationEventPublisher;
 }

 @Scheduled(initialDelay = 5000, fixedRate = 2500)
 public void runTimer() {
 int randomNumber = this.getRandomNumber(1, 4);
 switch (randomNumber) {
 case 1:
 logger.info("before publishEvent One");
 this.applicationEventPublisher.publishEvent(new EventOne());
 logger.info("after publishEvent One");
 break;
 case 2:
 logger.info("before publishEvent Two");
 this.applicationEventPublisher.publishEvent(new EventTwo());
 logger.info("after publishEvent Two");
 break;
 case 3:
 logger.info("before publishEvent Three");
 this.applicationEventPublisher.publishEvent(new EventThree());
 logger.info("after publishEvent Three");
 break;
 default:
 break;
 }
 }

 private int getRandomNumber(int min, int max) {
 return (int) ((Math.random() * (max - min)) + min);
 }

 @Component
 public static class EventOne {

 Logger logger = LoggerFactory.getLogger(EventOne.class);

 @EventListener
 public void runEvent(EventOne eventOne) {
 logger.info("runEvent()");
 }
 }

 @Component
 public static class EventTwo {

 Logger logger = LoggerFactory.getLogger(EventTwo.class);

 @EventListener
 public void runEvent(EventTwo eventTwo) {
 logger.info("runEvent()");
 }
 }

 @Component
 public static class EventThree {

 Logger logger = LoggerFactory.getLogger(EventThree.class);

 @EventListener
 public void runEvent(EventThree eventThree) {
 logger.info("runEvent()");
 }
 }

}Damit das Scheduling funktioniert muss solche via Application Klasse enabled werden:package ch.std.event.timer;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class SpringbootEventTimerApplication {

 public static void main(String[] args) {
 SpringApplication.run(SpringbootEventTimerApplication.class, args);
 }

}Mit dem erfolgreichen Start der Anwendung wird der Event Timer aktiv:In der Praxis würde der Random Ansatz durch die Datenbank Query ersetzt.

Full Sample Download

Das gesamte Beispiel finden Sie unter dem Link springbooteventtimer.zip.

Feedback

War dieser Blog für Sie wertvoll. Wir danken für jede Anregung und Feedback

Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

Impressum

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsauschluss.
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

https://www.simtech-ag.ch/«events»