KONKURENTNÉ PROGRAMOVANIE

Cvičenie 10 : Reaktívne programovanie 1

Producent-konzument

- Klasický producent-konzument a Queue
  - Producent pracuje "sám od seba" a posiela dátové balíčky na koniec radu úloh
  - Konzument čaká na začiatku radu a postupne berie dátové balíčky cez Queue.take() a spracúva ich
    - Nijak neoslovuje producenta, čo a koľko dát by chcel
- Callable<X> a Future
  - "Budúci konzument" zadá jednu úlohu worker vláknu a dostane Future a po čase čaká na výsledok
    - Konzumer vyzýva producenta, aby začal pracovať
  - Vlákno vyráta úlohu a ako producent pošle výsledok – dátový balíček
  - Dátový balíček je prevzatý konzumentom cez Future.get() a potom ho spracuje

Udalosťami riadené programovanie

- Producentom dát (= udalostí) je nejaký senzor, ktorý je ovplyvnený vonkajším prostredím
- "Konzumentom" je listener – záchytná funkcia, ktorá sa pustí, keď dotečú dáta od senzora
  - Konzument nijako neoslovuje producenta, aby začal produkovať
  - Registrácia listenera je rýchla

Prúdové spracovanie

- Vytvoríme kanál, cez ktorý bude tiecť prúd objektov
  - Po ceste sa môžu meniť na iné
  - Zjednodušený pohľad: postupnosť operátorov ~ listenerov,
    - Prvý producent v kanály je pripravený poslať dáta do kanála
    - Každý operátor v strede robí nasledovné:
      - Zaregistrujem funkciu/listener, ktorá sa má spustiť, keď prídu ku mne dáta a v ktorej môžem dáta čítať a spracovať
      - Po spracovaní vstupu pošlem nejaké dáta (rovnaké alebo zmenené) ďalšiemu operátoru v kanáli, ktorý už pred tým zaregistroval funkciu na spracovanie týchto mojich dát
- Dve fázy:
  1. Vybudujem **celý** kanál od prvého producenta po posledného konzumenta.
     - poprepájanie operátorov pripravené na spracovanie budúcich dát z prúdu a preposlanie spracovaných dát ďalej do prúdu
     - vytvorenie kanála je VŽDY rýchle, t.j. bez blokovania/čakania
  2. Cez posledného konzumenta zaregistrujem poslednú záchytnú funkciu a oznámim cez kanál, že prvý producent môže začať posielať dáta a koľko.
     - subscribe(záchytná_funkcia)
       - Posledný listener je záchytná funkcia, ktorá sa spustí, keď prídu dáta z prúdu až k nemu
     - spracovanie dát v operátoroch prúdu už môže byť pomalé aj dočasne blokované

Prúdové spracovanie (project Reactor)

- 4 nosné rozhrania (my budeme používať už iba existujúce implementácie):
  - Publisher
    - Má funkciu subscribe(subscriber), cez ktorú si zaregistruje/zapamätá príjemcu (~listener) a je pripravený poslať mu objekty prúdu (volaním niektorých z metód príjemcu)
  - Subscriber (~ listener)
    - prijíma objekty prúdu z kanála
    - má metódy na spracovanie toho, čo príde cez prúd – volá ich ten, čo mu ich posiela
      - onNext(dáta) – predchodca posiela dáta
      - onError(výnimka) – predchodca posiela výnimku
      - onComplete() – predchodca už nič nepošle (prúd skončil, kanál sa zavrel)
      - onSubscribe(subscription) – predchodca ma informuje, že posledný v kanáli zavolal subscribe, teda, že sa kanál otvoril, a spúšťa sa prúd dát
        - Je to vlastne listener na požiadavky od budúceho príjemcu mojich dát
        - Cez subscription si príjemca môže žiadať objekty (viď nižšie)
  - Subscription
    - požiadavka od príjemcu
      - request(n) – pošli maximálne n objektov (n je často Long.MAX_VALUE)
      - cancel() – už neposielaj nič (publisher to však môže ignorovať)
  - Processor
    - Publisher a Subscriber v jednom – na prácu uprostred prúdu

Príklad: ako to funguje vo vnútri...

- Flux<Long> kanál = fibGen.filter(x -> x%2==0) ;
  - vytvorenie kanála

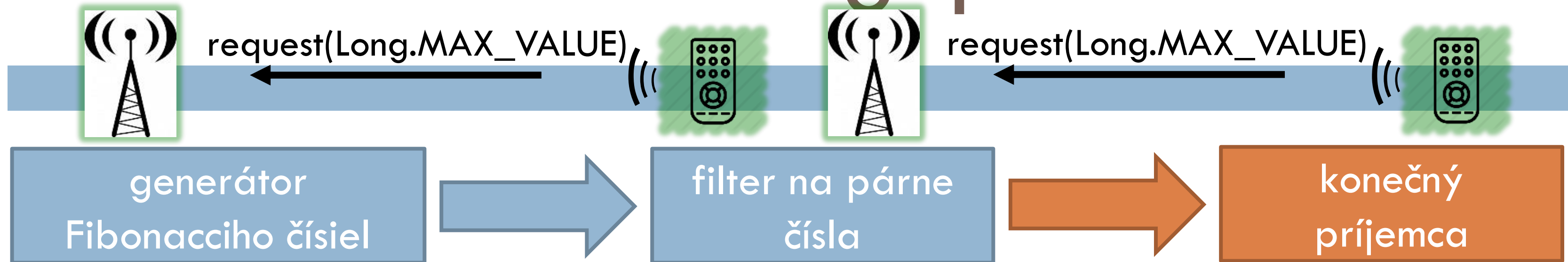
Príklad: ako to funguje vo vnútri...

- Flux<Long> kanál = fibGen.filter(x -> x%2==0) ;
  - vytvorenie kanála – modrá časť obrázka
- kanál.subscribe(x -> System.out.println(x));
  - Udeje sa nasledovné:
    - Subscriber, ktorého dodala metóda subscribe t.j. konečný príjemca zavolá subscribe na filtri
    - Filter zavolá subscribe na fibGen

Príklad: ako to funguje vo vnútri...

- Flux<Long> kanál = fibGen.filter(x -> x%2==0) ;
  - vytvorenie kanála – modrá časť obrázka
- kanál.subscribe(x -> System.out.println(x));
  - Udeje sa nasledovné:
    - Subscriber, ktorého dodala metóda subscribe t.j. konečný príjemca zavolá subscribe na filtri
    - Filter zavolá subscribe na fibGen
    - fibGen zavolá onSubscribe na filtri, t.j. pošle subscription
    - filter zavolá onSubscribe na konečnom príjemcovi, t.j. pošle subscription

Príklad: ako to funguje vo vnútri...



- Flux<Long> kanál = fibGen.filter(x -> x%2==0) ;
  - vytvorenie kanála – modrá časť obrázka
- kanál.subscribe(x -> System.out.println(x));
  - Udeje sa nasledovné:
    - Subscriber, ktorého dodala metóda subscribe t.j. konečný príjemca zavolá subscribe na filtri
    - Filter zavolá subscribe na fibGen
    - fibGen zavolá onSubscribe na filtri, t.j. pošle subscription
    - filter zavolá onSubscribe na konečnom príjemcovi, t.j. pošle subscription
    - konečný príjemca si zapýta hodnoty cez subscription.request(Long.MAX_VALUE)

Príklad: ako to funguje vo vnútri...

- Flux<Long> kanál = fibGen.filter(x -> x%2==0) ;
  - vytvorenie kanála – modrá časť obrázka
- kanál.subscribe(x -> System.out.println(x));
  - Udeje sa nasledovné:
    - Subscriber, ktorého dodala metóda subscribe t.j. konečný príjemca zavolá subscribe na filtri
    - Filter zavolá subscribe na fibGen
    - fibGen zavolá onSubscribe na filtri, t.j. pošle subscription
    - filter zavolá onSubscribe na konečnom príjemcovi, t.j. pošle subscription
    - konečný príjemca si zapýta hodnoty cez subscription.request(Long.MAX_VALUE)
    - fibGen pošle prvú hodnotu nula zavolaním onNext(0) na filtri
    - filter vyhodnotí, že je to párna hodnota a zavolá onNext(0) na konečnom príjemcovi
    - Konečný príjemca zavolá System.out.println(0)

Príklad: ako to funguje vo vnútri...

- Flux<Long> kanál = fibGen.filter(x -> x%2==0) ;
  - vytvorenie kanála – modrá časť obrázka
- kanál.subscribe(x -> System.out.println(x));
  - Udeje sa nasledovné:
    - Subscriber, ktorého dodala metóda subscribe t.j. konečný príjemca zavolá subscribe na filtri
    - Filter zavolá subscribe na fibGen
    - fibGen zavolá onSubscribe na filtri, t.j. pošle subscription
    - filter zavolá onSubscribe na konečnom príjemcovi, t.j. pošle subscription
    - konečný príjemca si zapýta hodnoty cez subscription.request(Long.MAX_VALUE)
    - fibGen pošle prvú hodnotu nula zavolaním onNext(0) na filtri
    - filter vyhodnotí, že je to párna hodnota a zavolá onNext(0) na konečnom príjemcovi
    - Konečný príjemca zavolá System.out.println(0)
    - fibgen pošle druhú hodnotu 1 zavolaním onNext(1) na filtri
    - filter vyhodnotí, že nejde o párnu hodnotu, nepošle nič
    - fibgen pošle tretiu hodnotu 1 zavolaním onNext(1) na filtri...

Implementácie rozhrania Publisher

- Schovávajú réžiu okolo požiadaviek a dátového prúdu
- Mono
  - Publisher, ktorý pošle do prúdu maximálne jeden dátový objekt
  - Porovnateľné s Callable úlohou akurát funguje asynchrónne
    - Callable
      - spustím úlohu generujúcu hodnotu,
      - zavolám future.get() a zaspím,
      - keď úloha vráti dátový objekt, tak sa zobudím a idem ho spracovať
    - Mono
      - zaregistrujem záchytnú metódu na spracovanie dátového objektu z prúdu (alebo ich dokonca zreťazím viacero za sebou)
      - zavolám subscribe, čím spustím prúd dát cez kanál t.j. spustím úlohu generujúcu hodnotu na jeho začiatku
      - záchytná metóda je zavolaná, keď dotečie dátový objekt
- Flux
  - Publisher, ktorý postupne pošle do prúdu viac dátových objektov
    - Nasledovník prichystá metódu, ktorá sa spustí vždy, keď dôjde ďalší dátový objekt

Generovanie dát do prúdu Flux (výber)

- Flux<String> prúd1 = Flux.just("Hello","world");
  - Generovanie statických vymenovaných hodnôt
- Flux<Integer> prúd2 = Flux.fromArray(new Integer[]{1, 2, 3});
- Flux<Integer> prúd3 = Flux.fromIterable(Arrays.asList(1, 2, 3));
- Flux<Integer> prúd4 = Flux.range(2020, 5);
  - Pošle postupne 5 hodnôt: 2020, 2021, 2022, 2023, 2024
- Flux<String> empty = Flux.empty();
  - Posiela iba informáciu complete o uzavretí prúdu
- Flux<String> never = Flux.never();
  - Neposiela absolútne nič
- Flux<Integer> prúd5 = Flux.create(sink -> {…})
  - sink.next(x) posielame ďalšiu hodnotu
  - sink.complete() uzatvára prúd
  - sink.error(e) pošle výnimku
  - + ďalšie metódy, napr. na odchytenie udalostí
- Flux<Integer> prúd6 = Flux.generate(() -> init_state, (state, sink) -> {…})
  - Druhá metóda sa volá pri každej požiadavke na ďalšiu hodnotu

Generovanie dát do prúdu Mono (výber)

- Mono<String> prúd1 = Mono.just("Hello");
  - Generovanie statickej hodnoty
- Mono<String> empty = Mono.justOrEmpty(null);
  - Ak je null uzavrie prúd, inak generuje hodnotu
- Mono<String> empty2 = Mono.justOrEmpty(Optional.empty());
  - Ak optional.isPresent(), generuje hodnotu, inak uzavrie prúd
- Mono<String> prúd2 = Mono.fromCallable(*Callable*);
  - Spustí úlohu, keď sa subscribne prúd a odosiela výsledok
- Mono<String> prúd3 = Mono.fromFuture(*ComputableFuture*);
  - Úloha spojená s týmto Future je už spustená pred tým a keď vráti hodnotu, odošle sa do prúdu
  - podobne aj fromCompletionStage(CompletionStage)
- Mono<Void> prúd4 = fromRunnable(Runnable)
  - Keď úloha dobehne, uzavrie prúd
- Mono<String> err = Mono.error(new RuntimeException("mono error"));
- Mono<String> prúd4 = Mono.defer(
  () -> (podmienka) ? Mono.just("x") : Mono.error(new Exception("neplatí podmienka")));
  - Vnútorná funkcia vracia Mono, no tá sa spustí, až keď nastane subscribe
  - V tomto príklade sa podmienka overuje až v čase, keď sa urobí subscribe

Spustenie prúdu: subscribe()

- Metódy spustiteľné na inštancii Mono alebo Flux – na poslednom operátore v kanáli:
  - subscribe()
    - Spustíme prúd s požiadavkou na Long.MAX_VALUE dátových objektov, nespracujeme žiadne prichádzajúce dáta, výnimky ani ukončenie prúdu
  - subscribe(Consumer<T> dataConsumer)
    - Spustíme prúd s požiadavkou na Long.MAX_VALUE dátových objektov, dáta spracujeme v dataConsumer-i, nespracujeme výnimky a ukončenie prúdu
  - subscribe(Consumer<T> dataConsumer, Consumer<Throwable> errorConsumer)
    - Spustíme prúd s požiadavkou na Long.MAX_VALUE dátových objektov, dáta spracujeme v dataConsumer-i, výnimky spracujeme v errorConsumeri, ignorujeme ukončenie prúdu
  - subscribe(Consumer<T> dataConsumer, Consumer<Throwable> errorConsumer, Runnable completeConsumer)
  - subscribe(Subscriber<T> subscriber)
    - Spustíme prúd, spracovanie udalostí aj požiadavky na dátové objekty prenechávame na subscriber
    - Najbezpečnejšie je rozšíriť BaseSubscriber<T>, aby sme nezabudli implementovať všetku réžiu požiadaviek a spracovania dát.

BaseSubscriber

- Môžeme prekryť iba tie metódy, ktoré chceme

```
BaseSubscriber<XXX> subscriber = new BaseSubscriber() {
        protected void hookOnSubscribe(Subscription s) { }

        protected void hookOnNext(XXX value) { }

        protected void hookOnError(Throwable throwable) { }

        protected void hookOnComplete() { }

        protected void hookOnCancel() { }
}

nejakýFlux.subscribe(subscriber);
```

Ak požadujeme N dátových objektov zavoláme s.request(N). Prúd môžeme ukončiť cez subscription.cancel()

Iné generátory dát

- knižnice pre MongoDB, Cassandra, Redis, Couchbase, relačné DB (R2DBC)
  - zasielanie hodnôt zo select-ov
  - získanie odozvy ukončenia insert-ov, update-ov a ďalších operácií
- WebFlux
  - reaktívne REST API cez eventsQueue
    - Vlákna nespia, kým dôjdu dáta z pezistentnej vrstvy
  - reaktívny WebSocket
- WebClient
  - reaktívny HTTP klient
- Processory (Sinks)
  - Keď generovanie dát je závislé od externých udalostí