SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SPRING: SPRING BATCH A SPRING INTEGRATION
ARCHITEKTÚRA SOFTVÉROVÝCH SYSTÉMOV

Bc. Sabina Daniela Pekareková
Bc. Dominika Olejarníková
Bc. Michaela Poláková
Bc. Peter Rigo
Bc. Lukáš Pribula

2022

Úvod

Tento dokument popisuje problematiku frameworku Spring. Budeme sa venovať tomu čo je to Spring. Čo je Spring Batch, ako a prečo sa používa, jeho výhody a nevýhody. Čo je Spring Integration, ako a prečo sa používa, jeho výhody a nevýhody. Vysvetlíme ako ich používať spolu s ukážkami jednoduchých implementácií.

1. Spring

1.1. Spring

Spring patrí medzi obľúbené open-source frameworky pre vývoj J2EE aplikácií. Prvá verzia vznikla už v roku 2002. Pod licenciou Apache 2.0 bol tento framework uvoľnený v roku 2003. Spring framework môže byť použitý v akejkoľvek Java aplikácií. Spring sa považuje za alternatívu alebo nadstavbu Enterprise Java Beans.

Jadro Springu je postavené na návrhovom vzore IoC kontajner, celým slovom Inversion of Control. Tento návrhový vzor funguje na princípe presunutia zodpovednosti za vytvorenie a previazanie objektov z aplikácie na framework. Objekty je možné získať prostredníctvom vytvárania závislostí, čo je špeciálny prípad Inversion of Control.

Spring je modulárny framework: Umožňuje využitie častí, ktoré potrebujeme na riešenie daného problému. Účelom Springu je zjednodušenie návrhu J2EE aplikácií so zameraním na architektúru aplikácie, na jednoduchú testovateľnosť, neinvazívnosť a modulárnosť.

Podľa Springu je vytváranie aplikácií pomocou tohto frameworku rýchlejšie, bezpečnejšie, jednoduchšie a produktívnejšie. [4]

Základné moduly:

- Core Container
- Data Access/Integration
- Web
- Aspect Oriented Programming (AOP)
- Instrumentation
- Test

Core Container sa skladá z Core, Beans, Context a Expression Language.

- Moduly Beans a Core poskytujú základné časti frameworku, vrátane IoC a Dependency Injection. BeanFactory je sofistikované prevedenie návrhového vzoru factory.
- Modul Context stavia na pevnom základe poskytnutom modulmi Core a Beans. Je to prostriedok na prístup k objektom spôsobom, ktorý je podobný rozhraniu JNDI. Modul Kontext dedí vlastnosti z modulu Beans a pridáva podporu pre internacionalizáciu a Java EE funkcie, ako sú EJB a JMX. Ústredným bodom modulu Context je rozhranie ApplicationContext.

- Modul Expression je rozšírenie jednotného jazyka pre vytváranie výrazov (unified expression language), ako je uvedené v špecifikácii JSP 2.1.

Data Acess/Integration sa skladá z JDBC, ORM, OXM, JMS a modulu Transactions

- Modul JDBC poskytuje abstraktnú JDBC vrstvu, ktorá odstraňuje potrebu robiť JDBC kódovanie a odchytávanie chybových kódov špecifických pre danú databázu.
- Modul ORM poskytuje integračné vrstvy pre populárne API objektovo relačného mapovania, vrátane SPS, JDO, Hibernate a iBatis. Pomocou ORM balíčka môžeme využiť všetky uvedené frameworky pre O/R mapovanie v kombinácii so všetkými ostatnými funkciami, ktoré SpringFramework ponúka, ako napríklad jednoduchý nástroj na riadenie transakcií.
- Modul OXM poskytuje abstraktnú vrstvu, ktorá podporuje implementáciu Object/XML mapovania pre JAXB, Castor, XMLBeans, JiBX a Xstream.
- Modul JMS (Java Messaging Service) obsahuje funkcie pre tvorbu a príjem správ.
- Modul Transactions podporuje programové a deklaratívne riadenie transakcií pre triedy implementujuce špeciálne rozhranie a pre všetky POJO (Plain Old Java objekty).

Web sa skladá z Web, Web-Servlet, Web-Struts, a Web-portlet.

- Modul Web v Spring Frameworku poskytuje základné webovo orientované funkcie, ako je funkcia multipart file-upload.
- Modul Web-Servlet obsahuje Springovú implementáciu model-view-controller (MVC) pre webové aplikácie.
- Modul Web-Struts obsahuje podporné triedy pre integráciu Struts do webových aplikácií v rámci Springu.
- Modul Web-Portlet umožňuje implementáciu MVC, ktoré majú byť použité v prostredí portletu a zrkadlia funkcie Web-Servlet modulu.

AOP je modul implementujúci podporu pre aspektovo orientované programovanie. Umožňuje separovať časti kódu prelínajúce sa celou aplikáciou (autorizácia, logovanie, transakcie) do takzvaných aspektov a ich následnú aplikáciu na akýkoľvek POJO objekt. Využitie AOP modulu sa prelína celým frameworkom a jedná sa o jednu z najsilnejších vlastností Springu.

Modul **Test** podporuje testovanie komponentov Springu pomocou JUnit.[1]

2. Spring Batch

Spring batch je framework, ktorý sa používa na dávkové spracovanie vykonávaním série úloh. Dizajnovaný je pre robustné batch aplikácie, dôležité pre denno dennú prevádzku podnikových systémov. Tento framework je efektívny len v prípade, že je potrebné spracovať veľký objem dát, ktoré môžu zahŕňať spracovanie štatistík alebo resource management. Spring Batch obsahuje rôzne znovu použiteľné funkcie vhodné pre spracovanie týchto veľko objemových dát. [3]

Vráťme sa ale k tomu čo je to dávkové spracovanie alebo Batch processing. Je to proces spracovania veľkého množstva dát, ktorý beží dlhú dobu. tento process býva dátovo a výpočtovo zložitý a môže byť spúšťaný plánovanie, na požiadavku, … Typicky je tento proces hromadne orientovaný, neinteraktívny, dlhotrvajúci a spúšťaný na pozadí.

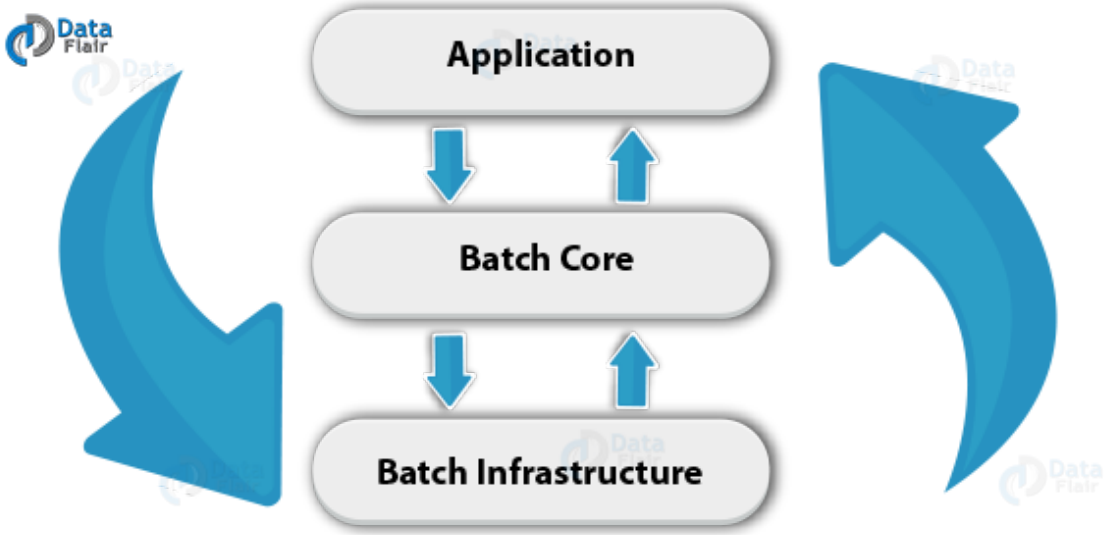
Vymenujme a vysvetlime si vlastnosti tohto frameworku. Prvá je flexibilita, čo znamená, že pre zmenu spracovania aplikácie stačí zmeniť XML súbor. Ďalej tu máme škálovateľnosť, kde možno pomocou portioning techniques (techniky porciovania) vykonávať každý krok úlohy paralelne. Následne tu máme udržateľnosť. To znamená, že každá úloha má krok, ktorý je možné oddeliť a testovať bez ovplyvnenia ostatných krokov. Tento framework je aj spoľahlivý nakoľko je možné reštartovať akúkoľvek úlohu od bodu, kde aplikácia zlyhala. Vykonáva sa to na základe oddelenia krokov. Posledná z vlastností je viacero možností spustenia úlohy. Úlohu je možné spustiť cez webovú aplikáciu, Java program príkazový riadok alebo pomocou automatickej úlohy.

Dôvody pre používanie frameworku Spring Batch je viacej. Začnime jedným s najdôležitejších a to je reštartovateľnosť. Znamená to, že keď pri spustení úloh nastane niekde chyba, je možné tento program reštartovať od miesta , kde nastala chyba. Jednou z ďalších výhod je možnosť použiť rôzne readeri a writeri, ako napríklad JDBC (Java Database Conectivity), JMS (Java Message Service), Hibernate, a rôzne ďalšie. Výhodou je aj paralelne spracovanie. S jednoduchou implementáciou je možné nakonfigurovať rôzne kroky aby bežali paralelne. Potom tu máme chunk processing, ktorý nám umožňuje rozdeliť dataset na rôzne menšie časti. Napríklad je možné rozdeliť milión riadkov dát na chunky po tisíc riadkoch. Tu nastáva výhoda aj pri chybách, že chyba nastane len pri jednej tisícke a táto sa začne vykonávať odznova.

```
<job id="job1">
    <split id="split1" task-executor="taskExecutor" next="step4">
        <flow>
            <step id="step1" parent="s1" next="step2"/>
            <step id="step2" parent="s2"/>
        </flow>
        <flow>
            <step id="step3" parent="s3"/>
        </flow>
    </split>
    <step id="step4" parent="s4"/>
</job>
```

Obrázok: úlohy a ich spracovanie

Na obrázku vyššie vidíme úlohu, ktorá obsahuje 4 úlohy. Flow, ktorý obsahuje krok 1 a 2, kde krok 2 sa môže vykonať až po vykonaní kroku 1, sa spúšťa paralelne s flowom, ktorý obsahuje krok 3. Následne až po dokončení týchto krokov sa môže spustiť krok 4.



Obrázok: architektúra Spring Batch

Na obrázku vyššie vidíme architektúru Spring Batch frameworku. Časť Aplikácia obsahuje v sebe všetky úlohy a kód pre spring framework. Batch core v sebe obashuje všetky API triedy potrebné pre kontrolovnie a spúšťanie batchov. Posledná časť Batch Infrastructure v sebe obsahuje všetky readre a writre spolu so servicami, ktoré sú používané týmito komponentami.

3. Spring Integration

Spring Integration poskytuje rozšírenie programovacieho modelu Spring na podporu Enterprise Integration Patterns. Spring Integration umožňuje zasielanie správ v rámci Spring aplikácií a podporuje integráciu s externými systémami prostredníctvom deklaratívnych adaptérov. Tieto adaptéry poskytujú vyššiu úroveň abstrakcie v porovnaní so Spring podporou pre vzdialenú komunikáciu, posielanie správ a plánovanie.

Hlavným cieľom Spring Integration je poskytnúť jednoduchý model na vytváranie enterprise integration riešení pri zachovaní oddelenia záujmov, čo je nevyhnutné na vytváranie udržateľného a testovateľného kódu.

Jednou z kľúčových tém frameworku Spring je IoC. V najširšom zmysle to znamená, že framework spracováva zodpovednosti v mene komponentov, ktoré sú spravované v jeho kontexte. Samotné komponenty sú zjednodušené. Napríklad funkcia dependency injection zbavuje komponenty zodpovednosti za vyhľadávanie alebo vytváranie ich závislostí. Podobne aspektovo orientované programovanie zbavuje business komponenty všeobecných prierezových problémov tým, že ich moduluje do opakovane použiteľných aspektov. V každom prípade je konečným výsledkom systém, ktorý sa ľahšie testuje, chápe, udržiava a rozširuje.

Tento framework rozširuje programovací model Spring na oblasť správ a stavia na existujúcej podpore Spring enterprise integration, aby poskytol ešte vyššiu úroveň abstrakcie. Podporuje architektúry riadené správami, v ktorých sa inverzia riadenia vzťahuje na problémy počas behu, napríklad kedy sa má spustiť určitá obchodná logika a kam sa má poslať odpoveď. Podporuje smerovanie a transformáciu správ, aby bolo možné integrovať rôzne transporty a rôzne formáty údajov bez toho, aby to malo vplyv na testovateľnosť.

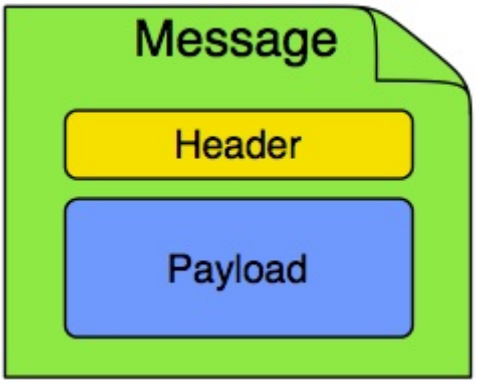
Ako rozšírenie programovacieho modelu Spring poskytuje Spring Integration širokú škálu možností konfigurácie vrátane anotácií, XML s podporou menných priestorov, XML s generickými prvkami "bean" a priameho použitia základného API. Toto API je založené na dobre definovaných strategických rozhraniach a neinvazívnych, delegujúcich adaptéroch.

3.1. Hlavné komponenty

Z vertikálneho hľadiska uľahčuje vrstvená architektúra oddelenie záujmov a interface-based contracts. Architektúry riadené správami pridávajú horizontálnu perspektívu. Tak ako je "vrstvená architektúra" extrémne všeobecná a abstraktná paradigma, systémy správ sa zvyčajne riadia podobne abstraktným modelom "pipes and filters". “Filtre” predstavujú všetky komponenty schopné vytvárať alebo prijímať správy a "potrubia" prenášajú správy medzi filtrami tak, aby samotné komponenty zostali voľne prepojené. Základná infraštruktúra na prenos správ, ktorá podporuje "potrubia", by mala byť stále zapuzdrená vo vrstve, ktorej zmluvy sú definované ako rozhrania. Podobne aj samotné "filtre" by sa mali spravovať vo vrstve, ktorá je logicky nad vrstvou služieb aplikácie a ktorá s týmito službami komunikuje prostredníctvom rozhraní podobne ako webová vrstva.

3.2. Message

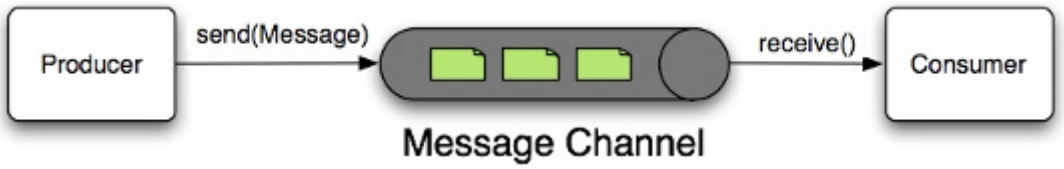
V Spring Integration je správa všeobecný obal pre akýkoľvek objekt v kombinácii s metadátami, ktoré framework používa pri práci s týmto objektom. Pozostáva z payloadov a headrov. Payload môže byť ľubovoľného typu a headre - hlavičky obsahujú bežne požadované informácie, ako sú ID, časová značka, korelačné ID a návratová adresa. Napríklad pri vytváraní správy z prijatého súboru môže byť v hlavičke uložený názov súboru, ku ktorému majú prístup nadväzujúce komponenty. Podobne, ak sa obsah správy nakoniec odošle adaptérom odchádzajúcej pošty, rôzne vlastnosti (to, from, cc, subject a iné) môžu byť nakonfigurované ako hodnoty hlavičky správy komponentom vyššieho prúdu. Vývojári môžu do hlavičiek ukladať aj ľubovoľné dvojice kľúč-hodnota.



Obrázok 1 : Message

3.3. Message Channel - Kanál správ

Kanál správ predstavuje "potrubie" architektúry pipes-and-filters. Producer posiela správy do kanála a Consumer prijíma správy z kanála. Kanál správ teda oddeľuje komponenty na zasielanie správ a zároveň poskytuje vhodný bod na zachytávanie a monitorovanie správ.



Kanál správ sa môže riadiť sémantikou point-to-point alebo publish-subscribe. Pri kanáli typu point-to-point môže každú správu odoslanú do kanála prijať najviac jeden spotrebiteľ. Na druhej strane, kanály typu publish-subscribe sa snažia vysielať každú správu všetkým účastníkom kanála. Konfiguráciou pollera môžeme obmedziť počet prijatých správ a zabrániť tým preťaženiu prijemcu.

3.4. Message Endpoint

Jedným z hlavných cieľov Spring Integration je zjednodušiť vývoj podnikových integračných riešení prostredníctvom inverzie riadenia. To znamená, že by ste nemali priamo implementovať konzumentov a producentov a dokonca by ste nemali ani vytvárať správy a vyvolávať operácie odosielania alebo prijímania na kanáli správ. Namiesto toho by ste mali mať možnosť zamerať sa na svoj špecifický doménový model s implementáciou založenou na jednoduchých objektoch. Potom môžete prostredníctvom deklaratívnej konfigurácie "pripojiť" svoj doménovo špecifický kód k infraštruktúre na zasielanie správ, ktorú poskytuje Spring Integration.

Message endpoint predstavuje "filter" architektúry pipes-and-filters. Ako už bolo spomenuté, hlavnou úlohou koncového bodu je pripojiť aplikačný kód k rámcu na zasielanie správ.

3.5. Message transformer

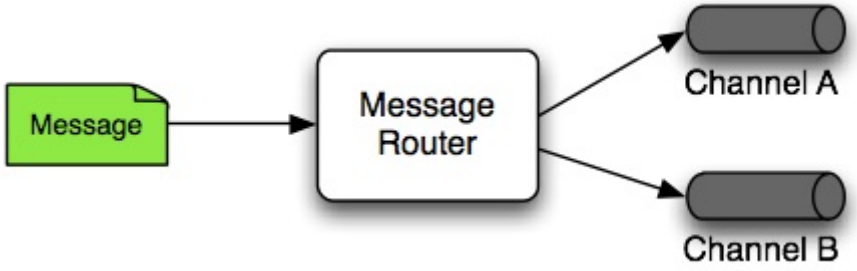
Message transformer je zodpovedný za konverziu obsahu alebo štruktúry správy a vrátenie upravenej správy. Pravdepodobne najbežnejším typom transformátora je ten, ktorý konvertuje payload z jedného formátu do iného (napríklad z XML do java.lang.String). Podobne môže transformátor pridať, odstrániť alebo upraviť hodnoty hlavičky správy.

3.6. Message Filter

Message Filter určuje, či sa má správa vôbec odovzdať výstupnému kanálu. Na to sa používa testovaciu metóda, ktorá môže kontrolovať konkrétny typ obsahu payloadu, property value, prítomnosť hlavičky alebo iné podmienky. Ak je správa prijatá, odošle sa do výstupného kanála. Ak nie, správa sa zahodí (alebo v prípade prísnejšej implementácie sa môže vyhodiť výnimka). Filtre správ sa často používajú v spojení s kanálom publish-subscribe, kde môže tú istú správu prijímať viacero príjemcov a na zúženie množiny správ, ktoré sa majú spracovať, sa používajú kritériá filtra.

3.7. Message Router

Message Router je zodpovedný za rozhodovanie o tom, ktorý kanál alebo kanály by mali prijať správu ako ďalšie. Toto rozhodnutie je zvyčajne založené na obsahu správy alebo na metadátach dostupných v hlavičkách správy. Message Router sa často používa ako dynamická alternatíva staticky nakonfigurovaného výstupného kanála na aktivátore služby alebo inom koncovom bode schopnom odosielať správy s odpoveďou.



3.8. Splitter

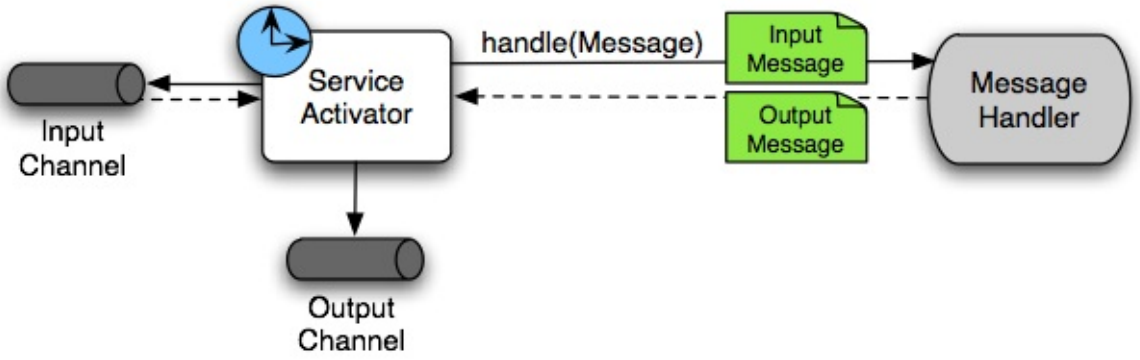
Splitter je ďalší typ message endpointu, ktorého úlohou je prijať správu zo vstupného kanála, rozdeliť ju na viacero správ a každú z nich odoslať do výstupného kanála. Zvyčajne sa používa na rozdelenie "zloženého" objektu.

3.9. Aggregator

Agregátor je v podstate zrkadlový obraz splittera (rozdeľovača), je to typ koncového bodu správy, ktorý prijíma viacero správ a spája ich do jednej správy.

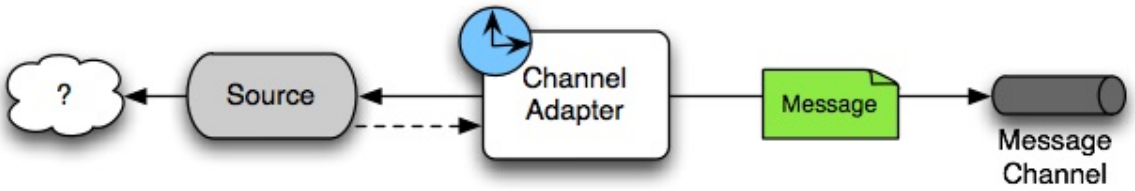
3.10. Service Activator

Service Activator je generický koncový bod na pripojenie inštancie služby k systému zasielania správ. Musí sa nakonfigurovať vstupný kanál správ, a ak metóda služby, ktorá sa má zavolať, dokáže vrátiť hodnotu, môže sa poskytnúť aj výstupný kanál správ.

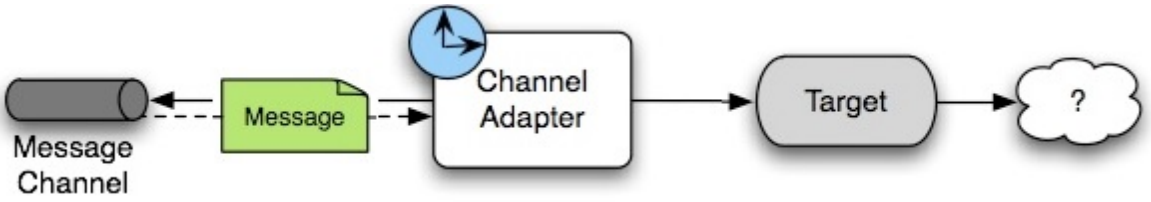


3.11. Channel Adapter

Channel Adapter je koncový bod, ktorý spája kanál správ s iným systémom alebo transportom. Adaptéry kanálov môžu byť buď prichádzajúce, alebo odchádzajúce. Channel adapter zvyčajne vykonáva určité mapovanie medzi správou a akýmkoľvek objektom alebo zdrojom, ktorý je prijatý z iného systému alebo odoslaný do iného systému (súbor, požiadavka HTTP, správa JMS a iné).[2]



Inbound channel adapter



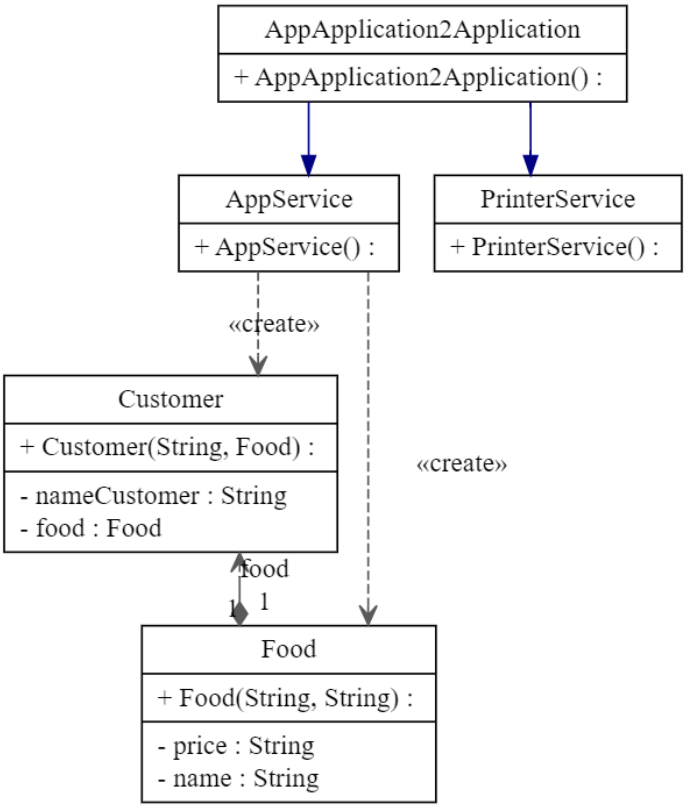
Outbound channel adapter

4. Implementácia

4.1. Implementácia Spring Integration

Na ukážku implementácie Spring Integration sme simulovali situáciu kedy si zákaznik vyberá jedlo z obedového menu, systém ho následne vyzve aby zaplatil požadovanú čiastku. Implementáciu tvoria tieto triedy: *Customer*, *Food*, *AppService*, *PrinterService* a spustiteľná Spring Boot trieda *AppApplication2Application*. Trieda *Customer* má 2 atribúty - *food*, teda jedlo ktoré si zákazník zvolí a *nameCustomer*, teda meno zákazníka. Trieda *Food* obsahuje atribúty *name* a *price*, teda názov a cena daného jedla.

Aplikácia využíva 2 kanály - vstupný a výstupný. Na vstupný kanál posielame správu, ktorej obsahom je načítané meno zákazníka. V xml súbore pre spring-config sme si nastavili referenciu na triedu *AppService*, ktorá spracuje správu na vstupnom kanáli a posunie ju na výstupný kanál. Systém vyzve zákazníka aby si vybral z ponuky jedál, následne sa vytvorí objekt typu *Customer*, obsahujúci meno zákazníka a jedlo ktoré si zvolil. *AppService* potom vráti tento objekt, v xml je definované že výstup z *AppService* sa má uložiť do výstupného kanála *outputCustomerChannel* . Ďalej máme v xml definované že vstup na kanáli *outputCustomerChannel* sa spracuje v triede *PrinterService*. Táto trieda má jedinú metódu - *printPayValue*, ktorá vypíše výzvu zákaznikovi, aby zaplatil za ním zvolené jedlo požadovanú sumu. Keďže *printPayValue* nič nevracia, v xml súbore nemáme zadefinovaný žiadny výstupný kanál kam by sa mal uložiť výsledok.

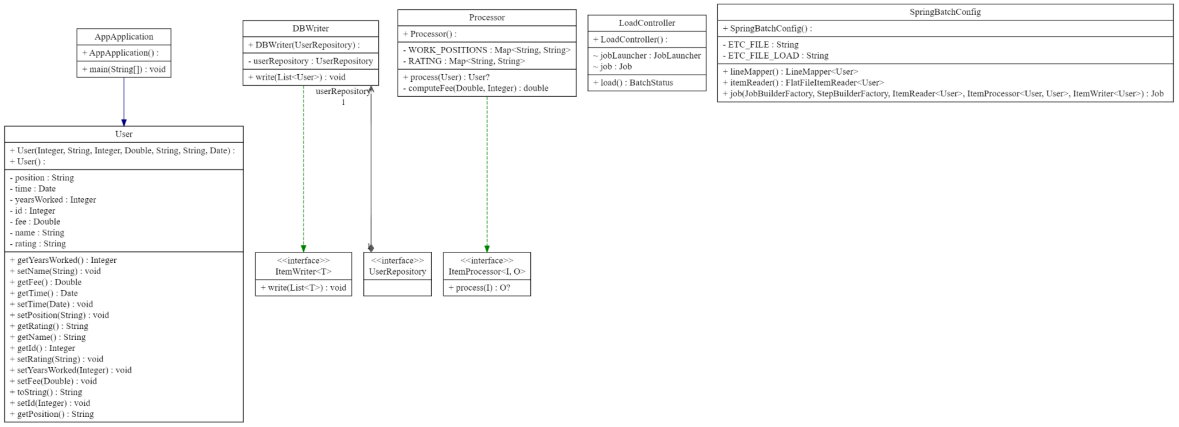


4.2. Implementácia Spring Batch

Na to aby sme mohli Spring Batch implementovať potrebujeme si simulovať vhodnú situáciu na jeho využitie. Máme skupinu zamestnancov a nadriadeného, ktorý chce rozdať koncoročné odmeny. K dispozícii má databázu zamestnancov, ktorá je vo formáte csv.

V csv súbore sú údaje o zamestnancovi, ako dlho pracuje vo firme, aký má plat, pozíciu a ako je ohodnotený.Používame reštartovateľný reader FlatFileItemReader, ktorý číta vstup zo súbora, ktorý môžeme nastaviť pomocou metódy setResource(). Riadok je definovaný pomocou metódy setRecordSeparatorPolicy() a mapuje sa podľa lineMappera – funkcia setLineMapper(). Pokiaľ sa niečo pokazí vyhodí výnimku FlatFileParseException. Skipujeme prvý riadok, pretože tam sa nachádzajú názvy stĺpcov. Vo funkcii setName() nastavíme názov komponentu, ktorý sa použije ako kmeň pre kľúče v ExecutionContext. V kroku nastavíme reader, processor a writer následne vykonáme pomocou jobBuildera úlohu, dáme jej nové id. Vytvorili sme si mapy s kľúčmi a jednoduchými hodnotami na simulovanie spracovania dát a výpočtovú funkciu s platom. Overridnuta metóda

ItemProcessora process nam slúži na vykonanie spracovania. Pomeníme podľa kľúča pozíciu a rating zamestnanca a potom mu vypočítame podľa nami náhodne zvoleného jednoduchého výpočtu na uvedenie príkladu. Na konci metódy vraciame objekt spracovaný ako user. Pri zapisovaní jednoducho iba píšeme list users to databázy, tým aj všetkých userov s rovnakým id prepisujeme ak už existujú a majú iné parametre okrem id.



Potrebujeme zavolať JobExecution objekt a mapu s parametrami, ktorú vložíme do objektu JobParameters a následne zavoláme JobExecution s danými parametrami.

Záver

Framework Spring ponúka veľké množstvo využitia. Záleží od užívateľa a funkcionality aplikácie aky typ(modul) spring frameworku je potrebné využiť.

Výhodou Spring Batch je že poskytuje znovupoužiteľné funkcie ktoré sú nevyhnutné pri spracovaní veľkého množstva dát, čiastočnom spracovaní objemových dát, opakovateľných úlohách a podobne. Nevýhodou je že všetky informácie ktoré Spring Batch spracováva musia byť korektné.

Spring Integration umožňuje ľahké zasielanie správ v rámci aplikácií založených na Spring a podporuje integráciu s externými systémami prostredníctvom deklaratívnych adaptérov. Výhodou Spring Integration je poskytovanie širokého výberu kanálových adaptérov a brán na komunikáciu s externými systémami. Je vhodný pre aplikácie/služby, ktoré sú založené na riadení správami.

Zoznam použitej literatúry

[1]https://cs.wikipedia.org/wiki/Spring_Framework

[2]https://docs.spring.io/spring-integration/reference/html/overview.html

[3]https://spring.io/projects/spring-batch

[4]https://spring.io/why-spring