Mini Web Framework

Domaći 2

1. Pregled zadatka

Zadatak je napraviti web framework po uzoru na JAX-rs/Spring. Potrebno je da se framework bazira na kontrolerima koji sadrže metode za obradu dolazećih zahteva i kreiranje JSON odgovora. Pored toga, neophodno je podržati dependency injection na nivou kontrolera.

Očekuje se da programer može definisati dostupne rute u svojoj aplikaciji tako što će anotirati metode kontrolera anotacijom @Path koja zahteva obavezan parametar - putanju. Pored toga, potrebno je programeru obezbediti način da definiše HTTP metodu koja odgovara prethodno navedenoj ruti - @GET ili @POST. Ako na primer imamo metodu kontrolera označenu sa @Path("/test") i @GET, to znači da kada klijent uputi GET HTTP zahtev ka putanji */test,* to će pokrenuti navedenu metodu kontrolera koja će biti zadužena za vraćanje odgovora klijentu.

Prilikom instanciranja kontrolera potrebno je izvršiti inicijalizaciju njegovih atributa upotrebom dependency injection-a.

2. Zahtevi

Registracija ruta. Potrebno je kreirati anotaciju @Controller kojom se može anotirati klasa (kontroler) i koja će biti znak da se u toj klasi nalaze metode koje obrađuju zahteve klijenta. Metoda kontrolera će predstavljati akciju koja je vezana za jednu rutu, te je neophodno specificirati HTTP metodu i rutu za koju će se ta akcija (metod) izvršiti. Kako bi to postigli neophodne su dve anotacije koje će detaljnije opisati tu metodu kontrolera:

1. @GET/@POST - Anotacija koja označava koja moguće HTTP metode koji će biti jedan od uslova za aktivaciju metode kontrolera. Ove anotacije su bez argumenata.
2. @Path anotacija koja prima samo jedan argument - putanju na koju će se metoda kontrolera aktivirati.

Pri pokretanju programa, potrebno je izanalizirati sve klase projekta (kreirati discovery mehanizam) i izdvojiti one anotirane sa @Controller. Nakon toga potrebno je proći kroz sve metode tih klasa i na osnovu anotacija @GET/@POST i @Path, globalno namapirati rutu (kao kombinaciju HTTP metode i putanje) na kontroler i metodu koja će se aktivirati pri pozivu te rute.

Neophodno je obezbediti da framework samo jednom instancira kontrolere, te da za svaki novi request već ima spremnu instancu kontrolera.

Dakle, klijent poziva rutu, framework iz rute zaključuje koju metodu (i iz kog kontrolera) će pozvati, metoda izvrši neku poslovnu logiku i klijentu vraća odgovor.

Nije moguće imati više metoda u kontroleru koje obrađuju istu putanju sa istom HTTP metodom.

Dependency injection. Ideja je da se postigne inversion of control (IOC) počev od atributa kontrolera. To ćemo postići tako što ćemo dislocirati mesto inicijalizacije svih potrebnih dependency-ja jednog kontrolera i to prepustiti da uradi framework pri inicijalizaciji kontrolera. Logiku koja će inicijalizovati kontrolere i sve njihove dependency-je potrebno je odvojiti u zasebnu klasu - ***DI Engine***.

Inicijalizacija dependency-ja će se dešavati na atributima kontrolera koji su anotirani sa @Autowired. Takođe, postoji mogućnost da i sam dependency ima atribute (druge dependency-je) anotirane sa @Autowired. U tom slučaju ćemo morati prvo njih da rešimo rekurzivno - počev od dna stabla.

Imamo dva slučaja pri inicijalizaciji atributa anotiranih sa @Autowired:

1. Anotiran je atribut koji za tip ima konkretnu klasu.
2. Anotiran je atribut koji za tip ima interfejs. Za ovaj slučaj je potrebno implementirati ***Dependency Container*** koji će predstavljati mapu specifikacije (interfejsa) na određenu implementaciju (klasu).

Za implementaciju dependency injection-a, potrebno je pružiti sledeće anotacije: @Autowired, @Bean, @Service, @Component i @Qualifier.

1. **Autowired** anotacija služi da markira sve atribute klase koje želite da se inject-uju tokom inicijalizacije kontrolera/drugog dependency-ja. Ova anotacija će imati jedan parametar, boolean verbose() koji opisuje da li će se prilikom inicijalizacije anotiranog atributa u konzoli ispisati dodatne informacije o samom objektu. Ispis treba da poštuje sledeću strukturu:

   “Initialized <param.object.type> <param.name> in <param.parentClass.name> on <localDateTime.now()> with <param.instance.hashCode>”

2. **Bean** anotacija služi za markiranje klasa. Sama po sebi ova anotacija nema nikakvu upotrebnu vrednost ali upotrebom njenog jedinog parametra scope() možemo definisati da li ce se prilikom inicijalizacije koristiti već inicijalizovana instanca (imitira singleton) ili će se svaki put kreirati nova instanca te klase prilikom inject-ovanja nekog dependency-ja. Vrednosti scope-a su singleton (default) i prototype.

3. **Service i Component** anotacije se ponašaju kao Bean koji za scope ima: scope = singleton i scope = prototype, respektivno. Oponašaju kompletno Bean anotaciju.

   Ideja je da ne moramo da navodimo parametar nego da se on podrazumeva upotrebom ovih anotacija kao neka vrsta olakšice.

4. **Qualifier** može da anotira i atribut i klasu i razrešava koji tačno Bean treba inject-ovati. Prima jedan parametar: String value().

   Kada anotira klasu, predstavlja jedinstvenu vrednost po kojoj ćemo prepoznati ovu implementaciju. Vrednost registrovati u Dependency Container-u.

   Kada anotira atribut koji je interfejs, na tom mestu treba inject-ovati implementaciju (klasu) registrovanu u Dependency Container-u sa navedenim Qualifier-om. Iz ovoga sledi da svaki atribut čiji tip je neki interfejs i koji je anotiran sa @Autowired mora imati i @Qualifier kako bi se navela određena implementacija.

Dependency Container

Prilikom inicijalizacije nekog dependency-ja može se desiti da nije poznat konkretan tip koji treba instancirati, jer je korišćen interfejs. Dependency Container je struktura u kojoj ćemo moći da registrujemo konkretnu implementaciju (klasu) za neki interfejs.

Dependency Container baca izuzetak ako neki tip nije naveden, a zahteva se iz container-a.

DI Engine

DI Engine je zadužen da upotrebom refleksije inicijalizuje sve anotirane dependency-je rekurzivno polazeci od kontrolera.

On će se baviti ispisom ako je “verbose” parametar na anotaciji @Autowire podešen na true.

DI Engine je zadužen da ima spisak svih instanci koje su anotirane sa @Bean(scope=”singleton”) ili sa @Service. Da ih inicijalizuje samo jednom tokom izvršavanja aplikacije i upotrebljava pri svakom inject-ovanju.

DI Engine je zadužen da prilikom inicijalizacije neke klase najpre injectuje, tj. inicijalizuje sve njene atribute koristeći rekurziju.

DI Engine je zadužen da konsultuje Dependency Container, svaki put kad naiđe na interfejs umesto na konkretnu klasu. Iz container-a, pomoću Qualifier-a dobija klasu koja predstavlja implementaciju i nju koristi za inicijalizaciju samog atributa.

Baca nekoliko izuzetaka:

1. Postoji vise @Bean-ova sa @Qualifier-om iste vrednosti
2. Atribut tipa Interface je anotiran sa @Autowired a nije sa @Qualifier
3. @Autowired na atributu koji nije @Bean (ili @Service ili @Component)

3. Tehnički zahtevi

Rešenje implementirati uz oslonac na refleksiju ili na AOP. Nije dozvoljeno korišćenje pomoćnih biblioteka za dependency injection.

Podrazumevati:

1. Da svaka klasa ima prazan konstruktor!
2. Da atribut nije static, niti final
3. Da ni jedna klasa ne nasleđuje drugu klasu
4. Da se ni u jednom slučaju neće pojaviti ciklična zavisnost dependency-ja.

4. Predaja zadatka

Studenti koji slušaju vežbe četvrtkom imaju rok za predaju 9.11. do 23:59. Studenti koji slušaju vežbe petkom imaju rok za predaju 10.11. do 23:59.

Domaći nosi 10 poena.

Domaći se šalje na:

- santic@raf.rs za grupe: 406, 422, 402a, 402b, 404, 423
- nredzic@raf.rs za grupe: 401, 405

Subject mail-a mora da bude u obliku: “Domaci2 <Ime> <Prezime> <Indeks>”. Npr. “Domaci2 Stefan Antić RN19/16”. U telu mejla navesti grupu kojoj pripadate po zvaničnom spisku kao i sam rad koji može biti u obliku:

- Zip
- Link ka Drive-u
- Link ka GitHub privatnom repozitorijumu.

Srećno! :)