Osnove web programiranja

Servisni sloj

Termin 7

Sadržaj

1. Arhitektura Spring MVC
2. Servisni sloj uvod
3. Implementacija servisnog sloja
4. Odabir implementacije servisa
5. Korišćenje servisa u kontroleru
6. Korišćenje servisnog sloja prednosti
7. Case study –servisi u bioskop veb aplikaciji

Sutup

Pokretanje primera

Importovati u Eclipse projekat u BioskopVebAplikacijaT6T7.zip

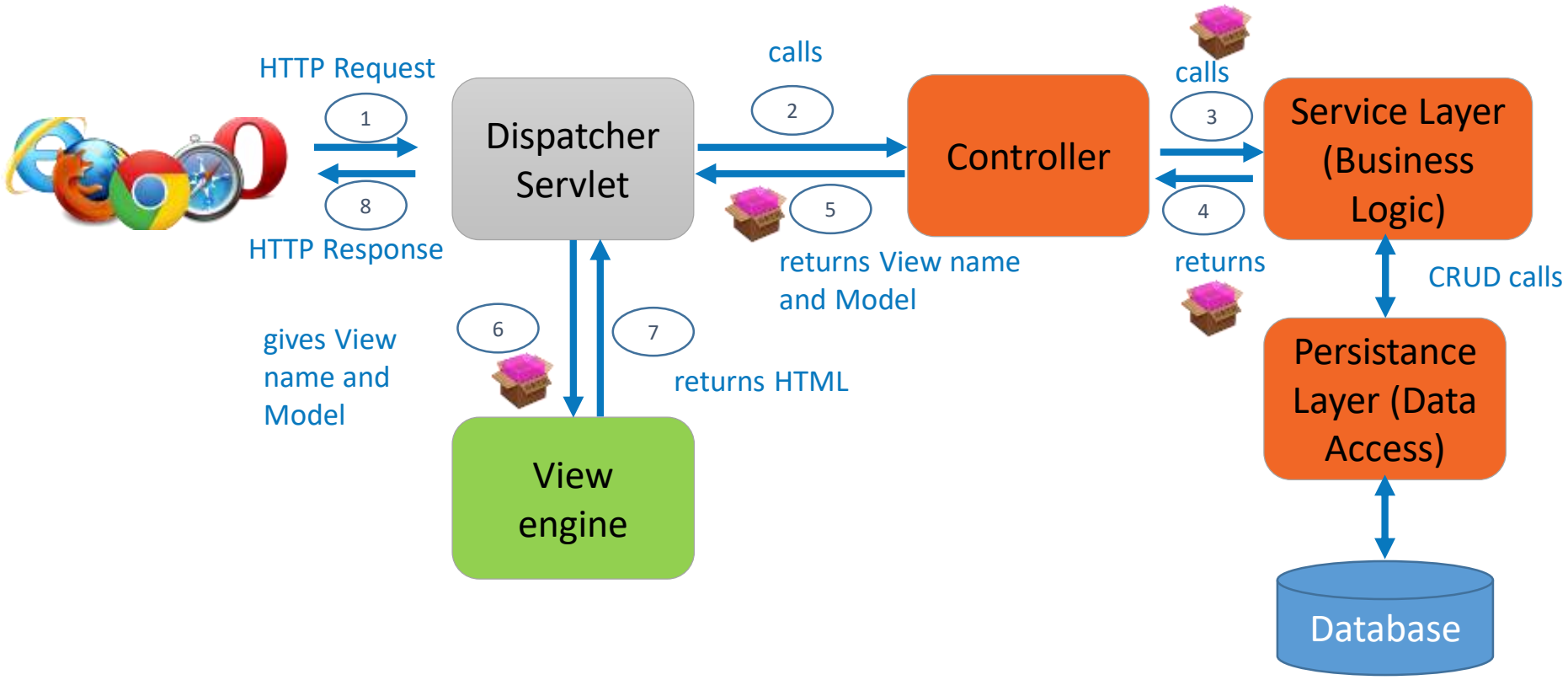
Otići u application.properties fajl i izmeniti kod tako da ukazuje na Vašu lokaciju na disku, i navedenoj putanji postaviti filmovi.txt fajl

```
#putanja do lokacije gde se nalaze filmovi txt
filmovi.pathToFile=c:\\ImePrezime\\Podaci\\filmovi.txt

filmovi.pathToFile=/student/home/ImePrezime/Podaci/filmovi.txt
```

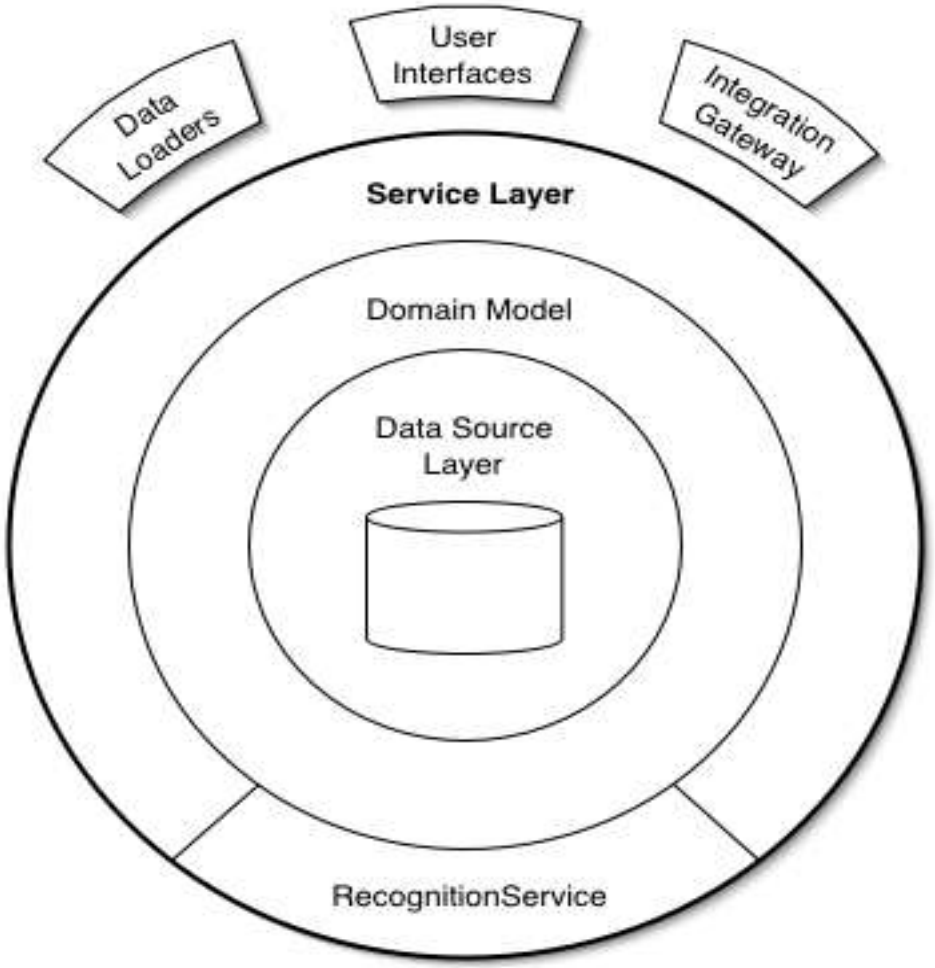
Arhitektura Spring MVC

Arhitektura Spring MVC aplikacije sa procesom rada– bez detalja



Arhitektura Spring MVC

N-layer application



This approach is typical for *client-server architecture* (*web applications*). This architectural design allows you to separate presentation (user interface), application processing (service layer) and data management (data access layer/data source layer). All of this allows developers to create flexible and reusable application parts

Once you develop one service in service layer it can be used on any other place, which means it maximize reusability of software components.
Last feature is that it provides *loose coupling*. This means that components can be replaced with alternative implementations that provide the same services.

Servisni sloj uvod

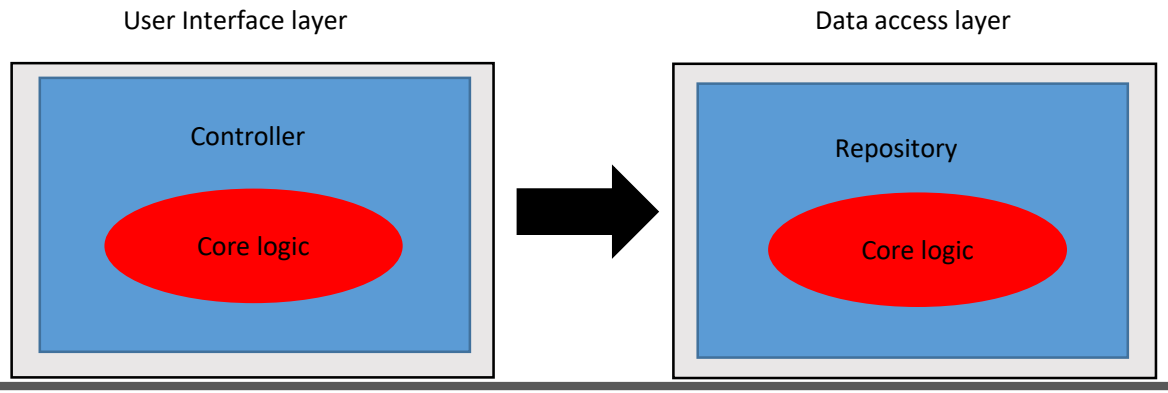
Uvod

U Spring radnom okviru da bi se podaci perzistirali u bazu podataka klase koje su kontroleri trebaju da komuniciraju sa klasama iz DAO sloja.

DAO sloj treba da je lagan i treba da omogući komunikciju sa bazom podataka.

Postavlja se pitanje gde onda treba da se nalazi poslovna logika upravljanja podacima?

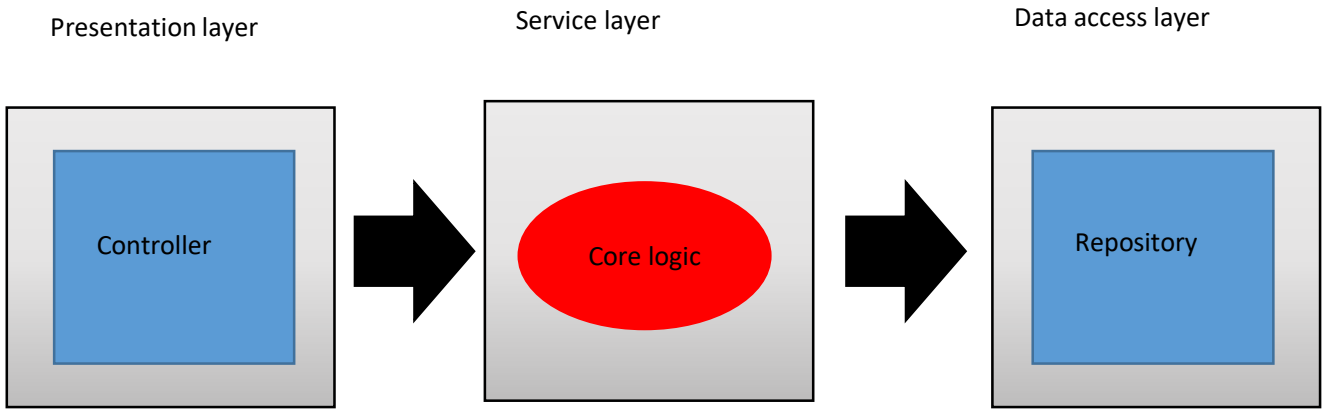
Šta bi se desilo ako bi poslovna logika (*core logic*) čuvala u kontrolerima User Interface sloja i/ili u DAO klasama Data Accsses sloja.



Servisni sloj uvod

Uvod

Servisni sloj je tu da pruža poslovnu logiku nad podacima koji se šalju ka i od DAO sloja ka kontrolerima.



Servisni sloj uvod

Poslovna logika

Servisi su komponente koje implementiraju **poslovnu logiku** vezanu za neki entitet.

Servisni sloj se koristi za manipulaciju podacima. Te manipulacije najčešće podrazumevaju CRUD operacije.

Servisni sloj se koristi kao omotač (wrapper) DAO sloja i predstavlja opšteprihvaćen način za korišćenje DAO sloja. Radi sa **Modelom**.

- U servisni sloj se stavljaju sve CRUD (create, retrieve, update, delete) metode koje perzistencioni sloj nudi
- Može biti izostavljen tako što ćemo koristiti DAO sloj direktno, ali njegovo uvođenje može doneti mnoge pogodnosti:
- **Zašto ne koristiti direktno perzistencioni sloj**? Zato što perzistencioni sloj nudi samo osnovne operacije. Ako želimo da proverimo poslovnu logiku u tim podacima to nije moguće. Takve provere možemo vršiti u metodama u servisnom sloju.
- **Preporuka je da se u Service sloju piše poslovna logika sistema**

Implementacija servisnog sloja

Interfejsi

U **Spring Boot** postoji konvencija za strukturu paketa, i davanje imena paketima i klasama za određene slojeve.

Za servisni sloj se po konvenciji koristi paket **service**.

- Paket će sadržati interfejse sa metodama koje će predstavljati **željene CRUD operacije nad podacima**.
- Za jedan entitet se obično kreira jedan interfejs.
- Imena interfejsa u servisnom sloju uglavnom imaju sufiks "Service".

Implementacija servisnog sloja

Interfejsi

Za entitet Film bi primer servisa bio FilmService.

```
public interface FilmService {

	Film findOne(Long id);
	List<Film> findAll();
	Film save(Film film);
	List<Film> save(List<Film> films);
	Film update(Film film);
	List<Film> update(List<Film> films);
	Film delete(Long id);
	void delete(List<Long> ids);
	List<Film> findByNaziv(String naziv);
	List<Film> findByTrajanje(int trajanje);
	List<Film> findByTrajanjeFrom(int trajanjeFrom);
	List<Film> findByTrajanjeTo(int trajanjeTo);
	List<Film> findByTrajanjeFromAndTrajanjeTo(int trajanjeFrom, int trajanjeTo);
}
```

Implementacija servisnog sloja

Implementacije interfejsa

Paket **service.impl** sadrži klase sa implementacijama interfejsa servisa.

Ideja je da se u interfejsima navedu sve metode za manipulaciju sa entitetom, a da se konkretna implementacija servisa navede u klasama koje implementiraju navedeni interfejs.

Prethodno ima za posledicu da se za 1 interfejs može kreirati N implementacija gde bi svaka od implementacija sadržala različitu poslovnu logiku za različite načine perzistencije podataka.

Te implementacije servisa mogu da čuvaju podatke u npr. memoriji, fajl sistemu, bazi podatka, itd.

Implementacija servisnog sloja

Implementacije interfejsa

Za servis FilmService kreirane su dve implementacije, jedna koja čuva podatke u memoriji,a druga u fajl sistemu.

Sve implementacije servisa anotirane su sa @Service

```
/** filmovi se prezistiraju u fajl sistem */
@Service
public class FileFilmService implements FilmService {..}

/** filmovi se prezistiraju u memoriji */
@Service
public class InMemoryFilmService implements FilmService {..}
```

Implementacija servisnog sloja

Pozivanje implementacije servisa iz drugih klasa

Aplikacija se implementacijama obraća preko interfejsa, oslanjajući se na DI mehanizam i @Autowired anotaciju.

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController{

        @Autowired
        private FilmService filmService;
```

```
@Controller
@RequestMapping(value="/Korisnici")
public class KorisnikController {

@Autowired
private KorisnikService korisnikService;
```

Odabir implementacije servisa

Postavljanje primerne implementacije za servis

- Koristeći DI mehanizam @Autowired anotaciju Spring može automatski da prepozna i poveže implementaciju servisa sa kodom koji poziva interfejs za taj servis.
- Postavlja se pitanje šta raditi u slučaju kada postoji više od 1 implementacije za određeni interfejs tj. šta ako postoje više kandidata za @Autowired anotaciju?
- Kako i da li će Spring znati prepozna koju od datih implementacija treba injektovati?

Odabir implementacije servisa

Postavljanje primerne implementacije za servis

Ukoliko postoji više od 1 implementacije za određeni servis, potrebno je Spring **eksplictno** naznačiti koja implementacija servisa treba da se koristiti.

Jedan od načina bi bio da se definiše primarna implementacija za servis.

Impelemtacija se anotira kao primarna sa `@Primary`.

Anotacija `@Primary` ukazuje da datoj bean klasi treba dati prednost ukoliko postoje više bean klasa koji su kandidati za automatsko povezivanje sa `@Autowired` anotacijom.

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController{

        @Autowired
        private FilmService filmService;
```

```
/** filmovi se prezistiraju u fajl sistem */
@Service
@Primary
public class FileFilmService implements FilmService
{..}

/** filmovi se prezistiraju u memoriji */
@Service
public class InMemoryFilmService implements FilmService
{..}
```

Odabir implementacije servisa

Postavljanje primerne implementacije za servis

Samo jedna implementacija servisa za određeni interfejs može biti anotirana sa @Primary

Na nivou cele aplikacije, na svim mestima gde se koristi automatsko povezivanje sa @Autowired anotacijom, injektovaće se ta primarna implementacije.

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController{

        @Autowired
        private FilmService filmService;
```

Implementacija FilmService interfejsa

```
/** filmovi se prezistiraju u fajl sistem */
@Service
@Primary
public class FileFilmService implements FilmService
{..}

/** filmovi se prezistiraju u memoriji */
@Service
public class InMemoryFilmService implements FilmService
{..}
```

Odabir implementacije servisa

Odabir implementacije za servis

Kako koristiti različite imeplemntacije u različitim delovima koda?
Kada je potrebno imati veći stepen kontrole nad procesom selekcije kandidata za DI mehanizam, moguće je koristiti @Qualifier anotaciju.

```
@Controller
@RequestMapping(value="/Filmovi")
public class FilmoviController{

        @Autowired
        @Qualifier("memorijaFilm")
        private FilmService filmService;
```

Implementacije FilmService i ProjekcijaService interfejsa

```
/** filmovi se prezistiraju u fajl sistem */
@Service
@Primary
@Qualifier("fajloviFilm")
public class FileFilmService implements FilmService
{..}

/** filmovi se prezistiraju u memoriji */
@Service
@Qualifier("memorijaFilm")
public class InMemoryFilmService implements FilmService
{..}
```

Korišćenje servisa u kontroleru

Metoda index u FilmoviController

Kod koji je omogućio pribavljanje fimova iz ServletContext i pozivanje metoda klase Filmovi zamenjen je pozivanjem metoda iz sevisnog sloja.

```
@Autowired
private FilmService filmService;

@GetMapping
@ResponseBody
public String index(HttpSession session, HttpServletResponse response) throws
IOException {

// preuzimanje vrednosti iz konteksta
// Filmovi filmovi =
(Filmovi)servletContext.getAttribute(FilmoviController.FILMOVI_KEY);
// List<Film> films = filmovi.findAll();
//preuzimanje vrednosti preko Servisnog sloja
List<Film> films = filmService.findAll();
for (int i=0; i < films.size(); i++) {
```

FilmoviController i upotreba servisa

Korišćenje servisnog sloja prednosti

Prednosti

Provides separation of concern

- Service layer provides code modularity, the business logic and rules are specified in the service layer which in turn calls DAO layer, the DAO layer is then only responsible for interacting with DB.

Provides Security

- If you provide a controllers that have no relation to the DB, then it is more difficult to gain access to the DB from the client except through the service. If the DB cannot be accessed directly from the client (and there is no trivial DAO module acting as the service) then an attacker who has taken over the client cannot have access to your data directly

Korišćenje servisnog sloja prednosti

Prednosti

Provide Loose Coupling

- Service layer can also be used to serve loose coupling in the application. Suppose your controller has 50 methods and in turn it calls 20 Dao methods, now at later point you decide to change the Dao methods serving these controllers. You need to change all the 50 methods in controller. Instead if you have 20 service methods calling those 20 Dao methods, you need to make change in only 20 Service methods to point to a new Dao.

Case study – servisi u bioskop veb aplikaciji

Filmovi i Projekcije

Pogledati korišćenje servisa za sve metode kontrolera FilmoviController, ProjekcijeController i KorisniciController

korićenje servisa u bioskop veb aplikaciji