Fruzenshtein's notes

Java blog with articles about programming, spring, hibernate and other web technologies, QA automation, MySQL, etc

About me **Site map**

Spring JPA Data + Hibernate + MySQL + MAVEN

POSTED 05/28/2013

Like 28 Tweet 9 +1 3 Share

Development of web-applications with the help of Spring MVC implies creation of several logical layers of architecture. One of the layers is a DAO (Repository) layer. It is responsible for communication with a database. If you developed the DAO layer at least once, you should know that it involves a lot of boilerplate code. A Spring Data take a part of the routine job related to the DAO on itself.

In the post I'm going to provide an example of application which will demonstrate **Spring Data (JPA)** in conjunction with **Spring MVC, MySQL and Maven**. **Hibernate** will be used as implementation of the JPA. As you probably know, I'm a real fan of java based

configurations, so I will use this approach to configure the Spring Data. In the end of the tutorial you can find a link to the sample project on GitHub.

Preparation

In the article I want to concentrate on the Spring Data, so all stuff which is out topic I will omit. But in the start I want provide a bulk of links which can be helpful for you in context of this tutorial.

- Creation of dynamic web project in Eclipse with Maven.
- Simple Spring MVC application with the java based configuration.
- Spring MVC + Hibernate sample application.

These links should give answers on 90% of questions which can occur during reading the post. Let's start with table creation in the MySQL:

```
view plain  copy to clipboard  print  ?
01. CREATE TABLE `shops` (
02.   `id` int(6) NOT NULL AUTO_INCREMENT,
03.   `name` varchar(60) NOT NULL,
04.   `employees_number` int(6) NOT NULL,
05.   PRIMARY KEY (`id`)
06. ) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;
```

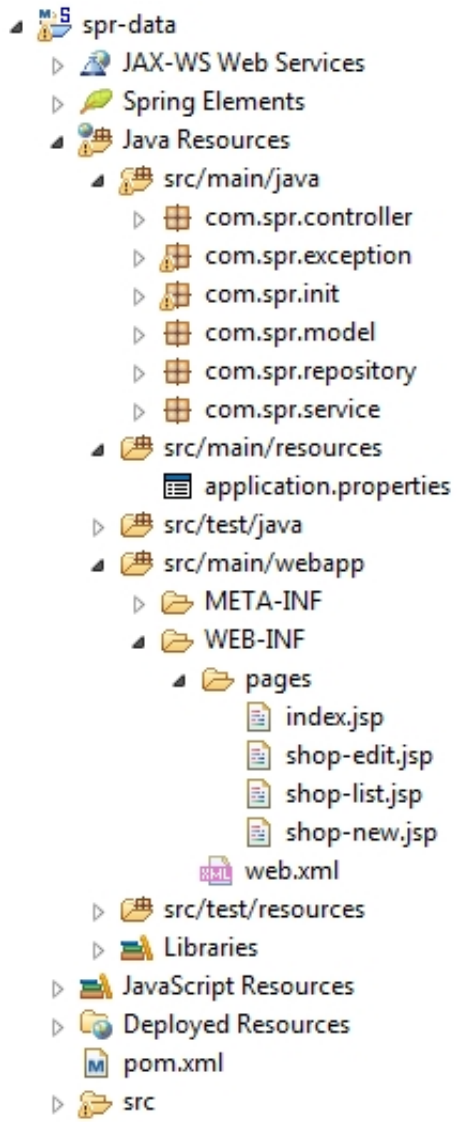
Now we can go ahead with a java code:

```
1  @Entity
2  @Table(name = "shops")
3  public class Shop {
4
5      @Id
6      @GeneratedValue
7      private Integer id;
8
9      private String name;
10
11     @Column(name = "employees_number")
12     private Integer emplNumber;
13
14     public Integer getId() {
15         return id;
16     }
17
18     public void setId(Integer id) {
```

```
19          this.id = id;
20      }
21
22      public String getName() {
23          return name;
24      }
25
26      public void setName(String name) {
27          this.name = name;
28      }
29
30      public Integer getEmplNumber() {
31          return emplNumber;
32      }
33
34      public void setEmplNumber(Integer emplNumber) {
35          this.emplNumber = emplNumber;
36      }
37  }
```

Configuration of Spring Data

I believe that a screenshot of the project will help you to understand what's going on.



In the property file concentrated all configuration data:

view plain copy to clipboard print ?

```
01. #DB properties:
02. db.driver=com.mysql.jdbc.Driver
03. db.url=jdbc:mysql://localhost:3306/hibnatedb
04. db.username=hibuser
05. db.password=root
06.
```

```
#Hibernate Configuration:
hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
hibernate.show_sql=true
entitymanager.packages.to.scan=com.spr.model
```

The WebAppConfig class contains all java based configurations:

```
@Configuration
@EnableWebMvc
@EnableTransactionManagement
@ComponentScan("com.spr")
@PropertySource("classpath:application.properties")
@EnableJpaRepositories("com.spr.repository")
public class WebAppConfig {

    private static final String PROPERTY_NAME_DATABASE_DRIVER = "db.driver";
    private static final String PROPERTY_NAME_DATABASE_PASSWORD = "db.password";
    private static final String PROPERTY_NAME_DATABASE_URL = "db.url";
    private static final String PROPERTY_NAME_DATABASE_USERNAME = "db.username";

    private static final String PROPERTY_NAME_HIBERNATE_DIALECT = "hibernate.dialect";
    private static final String PROPERTY_NAME_HIBERNATE_SHOW_SQL = "hibernate.show_sql";
    private static final String PROPERTY_NAME_ENTITYMANAGER_PACKAGES_TO_SCAN = "entitymanager.packag

    @Resource
    private Environment env;

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataSource();

        dataSource.setDriverClassName(env.getRequiredProperty(PROPERTY_NAME_DATABASE_DRIVER));
        dataSource.setUrl(env.getRequiredProperty(PROPERTY_NAME_DATABASE_URL));
        dataSource.setUsername(env.getRequiredProperty(PROPERTY_NAME_DATABASE_USERNAME));
        dataSource.setPassword(env.getRequiredProperty(PROPERTY_NAME_DATABASE_PASSWORD));

        return dataSource;
    }

    @Bean
    public LocalContainerEntityManagerFactoryBean entityManagerFactory() {
        LocalContainerEntityManagerFactoryBean entityManagerFactoryBean = new LocalContainerEntityMa
        entityManagerFactoryBean.setDataSource(dataSource());
        entityManagerFactoryBean.setPersistenceProviderClass(HibernatePersistence.class);
        entityManagerFactoryBean.setPackagesToScan(env.getRequiredProperty(PROPERTY_NAME_ENTITYMANAG

        entityManagerFactoryBean.setJpaProperties(hibProperties());

```

```
42          return entityManagerFactoryBean;
43      }
44
45      private Properties hibProperties() {
46          Properties properties = new Properties();
47          properties.put(PROPERTY_NAME_HIBERNATE_DIALECT, env.getRequiredProperty(PROPERTY_NAME_HIBERN
48          properties.put(PROPERTY_NAME_HIBERNATE_SHOW_SQL, env.getRequiredProperty(PROPERTY_NAME_HIBER
49          return properties;
50      }
51
52      @Bean
53      public JpaTransactionManager transactionManager() {
54          JpaTransactionManager transactionManager = new JpaTransactionManager();
55          transactionManager.setEntityManagerFactory(entityManagerFactory().getObject());
56          return transactionManager;
57      }
58
59      @Bean
60      public UrlBasedViewResolver setupViewResolver() {
61          UrlBasedViewResolver resolver = new UrlBasedViewResolver();
62          resolver.setPrefix("/WEB-INF/pages/");
63          resolver.setSuffix(".jsp");
64          resolver.setViewClass(JstlView.class);
65          return resolver;
66      }
67
68  }
```

Pay your attention at *@EnableJpaRepositories* annotation. It enables usage of JPA repositories. The *com.spr.repository* package will be scaned to detect repositories. In the *entityManagerFactory* bean I determined that Hibernate will be used as JPA implementation.

Initializer class will be omitted.

DAO & Service layers

The repository for the Shop entity:

```
1  package com.spr.repository;
2
3  import org.springframework.data.jpa.repository.JpaRepository;
4
5  import com.spr.model.Shop;
6
7  public interface ShopRepository extends JpaRepository<Shop, Integer> {
8
```

```
}
```

Definitely it is the most simplest code snippet in the tutorial. But it requires the most high attention. The JpaRepository interface contains the basic operations which can be performed with any entity (CRUD operations). More information you can find on the official documentation page.

Here is a code of the ShopService interface:

```
public interface ShopService {

    public Shop create(Shop shop);
    public Shop delete(int id) throws ShopNotFound;
    public List<Shop> findAll();
    public Shop update(Shop shop) throws ShopNotFound;
    public Shop findById(int id);

}
```

And the implementation of the service interface:

```
import java.util.List;

import javax.annotation.Resource;

import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import com.spr.exception.ShopNotFound;
import com.spr.model.Shop;
import com.spr.repository.ShopRepository;

@Service
public class ShopServiceImpl implements ShopService {

    @Resource
    private ShopRepository shopRepository;

    @Override
    @Transactional
    public Shop create(Shop shop) {
        Shop createdShop = shop;
        return shopRepository.save(createdShop);
    }

    @Override
    @Transactional
    public Shop findById(int id) {
```

```
28          return shopRepository.findOne(id);
29      }
30
31      @Override
32      @Transactional(rollbackFor=ShopNotFound.class)
33      public Shop delete(int id) throws ShopNotFound {
34          Shop deletedShop = shopRepository.findOne(id);
35
36          if (deletedShop == null)
37              throw new ShopNotFound();
38
39          shopRepository.delete(deletedShop);
40          return deletedShop;
41      }
42
43      @Override
44      @Transactional
45      public List<Shop> findAll() {
46          return shopRepository.findAll();
47      }
48
49      @Override
50      @Transactional(rollbackFor=ShopNotFound.class)
51      public Shop update(Shop shop) throws ShopNotFound {
52          Shop updatedShop = shopRepository.findOne(shop.getId());
53
54          if (updatedShop == null)
55              throw new ShopNotFound();
56
57          updatedShop.setName(shop.getName());
58          updatedShop.setEmplNumber(shop.getEmplNumber());
59          return updatedShop;
60      }
61
62  }
```

In this way the ShopRepository is used.

Controller

Finally I can use ShopSrviceImpl class in the controller. All JSP pages will be omitted, so you can find them source code on the GitHub.

```
1   @Controller
2   @RequestMapping(value="/shop")
3   public class ShopController {
4
```

```
    @Autowired
    private ShopService shopService;

    @RequestMapping(value="/create", method=RequestMethod.GET)
    public ModelAndView newShopPage() {
        ModelAndView mav = new ModelAndView("shop-new", "shop", new Shop());
        return mav;
    }

    @RequestMapping(value="/create", method=RequestMethod.POST)
    public ModelAndView createNewShop(@ModelAttribute Shop shop,
            final RedirectAttributes redirectAttributes) {

        ModelAndView mav = new ModelAndView();
        String message = "New shop "+shop.getName()+" was successfully created.";

        shopService.create(shop);
        mav.setViewName("redirect:/index.html");

        redirectAttributes.addFlashAttribute("message", message);
        return mav;
    }

    @RequestMapping(value="/list", method=RequestMethod.GET)
    public ModelAndView shopListPage() {
        ModelAndView mav = new ModelAndView("shop-list");
        List<Shop> shopList = shopService.findAll();
        mav.addObject("shopList", shopList);
        return mav;
    }

    @RequestMapping(value="/edit/{id}", method=RequestMethod.GET)
    public ModelAndView editShopPage(@PathVariable Integer id) {
        ModelAndView mav = new ModelAndView("shop-edit");
        Shop shop = shopService.findById(id);
        mav.addObject("shop", shop);
        return mav;
    }

    @RequestMapping(value="/edit/{id}", method=RequestMethod.POST)
    public ModelAndView editShop(@ModelAttribute Shop shop,
            @PathVariable Integer id,
            final RedirectAttributes redirectAttributes) throws ShopNotFound {

        ModelAndView mav = new ModelAndView("redirect:/index.html");
        String message = "Shop was successfully updated.";

        shopService.update(shop);

```

```
54          redirectAttributes.addFlashAttribute("message", message);
55          return mav;
56      }
57
58      @RequestMapping(value="/delete/{id}", method=RequestMethod.GET)
59      public ModelAndView deleteShop(@PathVariable Integer id,
60              final RedirectAttributes redirectAttributes) throws ShopNotFound {
61
62          ModelAndView mav = new ModelAndView("redirect:/index.html");
63
64          Shop shop = shopService.delete(id);
65          String message = "The shop "+shop.getName()+" was successfully deleted.";
66
67          redirectAttributes.addFlashAttribute("message", message);
68          return mav;
69      }
70
71  }
```

Shop List page

id	company	employees	actions
1	Water Market	3	Edit Delete
2	Fruzenshtein INC	44	Edit Delete
3	Alex & Co	7	Edit Delete
4	Green Garden	85	Edit Delete

Home page

Summary

The Spring Data is very powerful weapon, it helps you develop an application more faster and avoid hundreds of boilerplate strings of code. Usage of Spring Data is the most convenient way to create a DAO layer in an application, so don’t ignore it in your projects.

Share and Enjoy

4 Comments Fruzenshtein's notes Login

Recommend Share Sort by Best

Join the discussion…

Vishal Prajapati • 7 months ago

It wil be even more helpful if you also put a link of source download.

8 • Reply • Share ›

James Gardiner • 2 months ago

Really like this example, but I am having a massive issue..

When I run this demo, tomcat keeps printing..

[2015-02-13 03:59:15,034] Artifact spr-data:war: Deploy took 4,803 milliseconds
15:59:15.373 [http-bio-8080-exec-1] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:15.393 [http-bio-8080-exec-2] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:15.898 [http-bio-8080-exec-3] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:15.901 [http-bio-8080-exec-4] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:16.406 [http-bio-8080-exec-5] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:16.409 [http-bio-8080-exec-6] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:16.914 [http-bio-8080-exec-7] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:16.917 [http-bio-8080-exec-8] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:17.421 [http-bio-8080-exec-9] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported

15:59:17.424 [http-bio-8080-exec-10] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported
15:59:17.928 [http-bio-8080-exec-10] WARN o.s.web.servlet.PageNotFound - Request method 'HEAD' not supported

---

see more

• Reply • Share ›

Guest • a year ago

thanks, it's an awesome complement to the quickstart guide :)

• Reply • Share ›

rajan sellapan • 7 months ago

hello sir, thank you for your tutorial, I download and convert eclipse web project, and export to eclipse. when I run No index/home page coming, I could not call any page.
please show one full url for how to access index, shop-list, shop-edit everything.
http://localhost:8080/spr-mvc-hib/pages/index.jsp, when I start run the tomcat server automatically the index/ home page should come,
please sorry for ignorance knowlede in spring mvc, I could not access. please help

• Reply • Share ›

ALSO ON **FRUZENSHTEIN'S NOTES** WHAT'S THIS?

Portrait of success Java developer in 2015
1 comment • 3 months ago
Avat **hemanth** — Great inputs. Thanks for sharing :)

Spring MVC: REST application with CNVR vol. 1
2 comments • 2 years ago
Avat **from scratch** — Why is your pom.xml incorrect? groupid should be groupId, same with artifactid.

5 minutes for Java
3 comments • 2 years ago
Avat **vkopchenin** — int[] array = {1,3,2,6,5,7,8,10,9}; // sum of arithmetic progression int sum_all = ((1 + 10) * 10) / 2; int sum_part = 0; for(int i: array) { …

Spring MVC: Ajax & JQuery
13 comments • 2 years ago
Avat **shams** — Thanks bro, it is quite useful for me

Subscribe Add Disqus to your site Privacy

Alexey Zvolinskiy aka Alex Fruzenshtein
Using of all materials from this site implies back link to www.fruzenshtein.com
Kiev, Ukraine 2012 - 2015 (C)