|  |  |  |
| --- | --- | --- |
| Implementing persistence layer may need some effort to configure and develop some DAO classes.  Spring-Data-JPA framework can simplify the creation of JPA based repositories and to reduce the amount of code needed to communicate with a database |  |  |

This post is an introduction to Spring-Data-JPA. It describes, how you can configure the framewok when you are using Hibernate as your JPA provider.  
In this post I will take profit from some Spring 4 new feature as generics autowiring to create generic service.  
You can get the sample project from this [link](http://project4example7.googlecode.com/svn/trunk/) to start using Spring-Data-jpa and try some more advenced features.

1- Project dependencies

Spring-Data-JPA references a Spring 3 version. Maven will resolve dependencies by using Spring 3 version instead of version 4.

We need to exclude spring-core and spring-context and import Spring 4 version.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | <!-- Spring -->  <dependency>     <groupId>org.springframework</groupId>     <artifactId>spring-context</artifactId>     <version>4.0.4.RELEASE</version>  </dependency>  <dependency>     <groupId>org.springframework.data</groupId>     <artifactId>spring-data-jpa</artifactId>     <version>1.5.2.RELEASE</version>     <exclusions>        <exclusion>           <groupId>org.springframework</groupId>           <artifactId>spring-core</artifactId>        </exclusion>        <exclusion>           <artifactId>spring-context</artifactId>           <groupId>org.springframework</groupId>        </exclusion>    </exclusions>  </dependency> |

2- Add an Entity

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | @Entity  @Table(name = "users")  public class User {        @Id      @GenericGenerator(name = "generator", strategy = "increment")      @GeneratedValue(generator = "generator")      @Column(name = "id", nullable = false)      private Long id;        @Column(name = "first\_name", nullable = false)      private String firstName;        /\* Add some extra attributes \*/        /\* Add getters and setters \*/    } |

3- Create repository

Declare an interface extending Repository.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | import com.mycompany.myproject.persist.entity.User;  import org.springframework.data.jpa.repository.JpaRepository;    public interface UserRepo extends JpaRepository<User, Long> {        User findByLogin(String login);    } |

Spring-Data-JPA analyses the method name and generates a query using the JPA criteria.  
Query generation is base on key words like (And, Or, Is, Equals, Between …). For more details, you can read the official Spring-Data-JPA documentation in this [link](http://docs.spring.io/spring-data/jpa/docs/current/reference/html/jpa.repositories.html#jpa.query-methods.query-creation).

4- Create Service

We make a choice to create a generic service that will reference the repository created in previous step.

One of the new features of Spring4 is generic types qualifier. We will take profit from this feature to inject repository by its generic interface.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | public class GenericServiceImpl<T, D, ID extends Serializable> implements GenericService<T, D, ID> {        @Autowired      private JpaRepository<T, ID> repository;        @Autowired      private DozerBeanMapper mapper;        protected Class<T> entityClass;        protected Class<D> dtoClass;        @SuppressWarnings("unchecked")      public GenericServiceImpl() {          ParameterizedType genericSuperclass = (ParameterizedType) getClass().getGenericSuperclass();          this.entityClass = (Class<T>) genericSuperclass.getActualTypeArguments()[0];          this.dtoClass = (Class<D>) genericSuperclass.getActualTypeArguments()[1];      }        public D findOne(ID id) {          return mapper.map(repository.findOne(id), dtoClass);      }        public List<D> findAll() {          List<D> result = new ArrayList<D>();          for (T t : repository.findAll()) {              result.add(mapper.map(t, dtoClass));          }          return result;      }        public void save(D dto) {          repository.saveAndFlush(mapper.map(dto, entityClass));      }    } |

Next, we add UserService that extends GenericServcie.

|  |  |
| --- | --- |
| 1  2  3 | @Service  public class UserServiceImpl extends GenericServiceImpl<User, UserDto, Long> {  } |

5- Configure the project

* line 15 : EnableJpaRepositories annotation enables scanning package to look for repository interfaces (created in step 3).
* line 21 : We use HSQL embedded database. We need to provide schema script and data script.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | import javax.sql.DataSource;  import org.dozer.DozerBeanMapper;  import org.springframework.context.annotation.\*;  import org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor;  import org.springframework.data.jpa.repository.config.EnableJpaRepositories;  import org.springframework.jdbc.datasource.embedded.\*;  import org.springframework.orm.jpa.\*;  import org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter;  import org.springframework.transaction.PlatformTransactionManager;  import org.springframework.transaction.annotation.EnableTransactionManagement;    @Configuration  @EnableTransactionManagement  @ComponentScan({"com.mycompany.myproject.persist", "com.mycompany.myproject.service"})  @EnableJpaRepositories("com.mycompany.myproject.persist")  public class JPAConfig {        @Bean(name = "dataSource")      public DataSource dataSource() {          return new EmbeddedDatabaseBuilder().setType(EmbeddedDatabaseType.HSQL).setName("myDb")                  .addScript("classpath:schema.sql").addScript("classpath:data.sql").build();      }        @Bean(name = "entityManagerFactory")      public LocalContainerEntityManagerFactoryBean entityManagerFactoryBean() {          LocalContainerEntityManagerFactoryBean factoryBean = new LocalContainerEntityManagerFactoryBean();          factoryBean.setDataSource(dataSource());          factoryBean.setPackagesToScan(new String[] { "com.mycompany.myproject.persist" });          HibernateJpaVendorAdapter vendorAdapter = new HibernateJpaVendorAdapter();          vendorAdapter.setShowSql(true);          factoryBean.setJpaVendorAdapter(vendorAdapter);          return factoryBean;      }        @Bean      public PlatformTransactionManager transactionManager() {          JpaTransactionManager transactionManager = new JpaTransactionManager();          transactionManager.setEntityManagerFactory(entityManagerFactoryBean().getObject());          return transactionManager;      }        @Bean      public PersistenceExceptionTranslationPostProcessor exceptionTranslation() {          return new PersistenceExceptionTranslationPostProcessor();      }        @Bean      public DozerBeanMapper getMapper() {          return new DozerBeanMapper();      }  } |

6- Test your application

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | @RunWith(SpringJUnit4ClassRunner.class)  @ContextConfiguration(classes = {JPAConfig.class })  public class GenericServiceTest {        @Autowired      private GenericService<User, UserDto, Long> userService;        @Autowired      private GenericService<Authority, AuthorityDto, Long> authorityService;        @Test      public void testFindOneUser() {          UserDto userDto = userService.findOne(1L);          Assert.assertNotNull(userDto);          Assert.assertEquals(1, userDto.getId().longValue());      }        @Test      public void testFindOneAuthority() {          AuthorityDto authorityDto = authorityService.findOne(1L);          Assert.assertNotNull(authorityDto);          Assert.assertEquals(1, authorityDto.getId().longValue());      }        @Test      public void testFindAll() {          List<AuthorityDto> authorities = authorityService.findAll();          Assert.assertFalse(authorities.isEmpty());      }        @Test      public void testSave() {          AuthorityDto authorityDto = new AuthorityDto();          authorityDto.setName("test authority");      }  } |

*If you are a concise code adept you will appreciate using Spring-Data-JPA. To get your repositories you just need to add some interfaces for entities. You don’t need to write redundant basic code for each persistent bean.*

*The mix of Spring-DATA-JPA and Spring4 gives some extra possibilities to write generic code for the persistence layer.*