Spring Boot + Hibernate 5 + Mysql Example

By Dhiraj Ray (https://plus.google.com/112360792925347143122), 13 March,2017 | Last updated on: 03 January,2018 9590

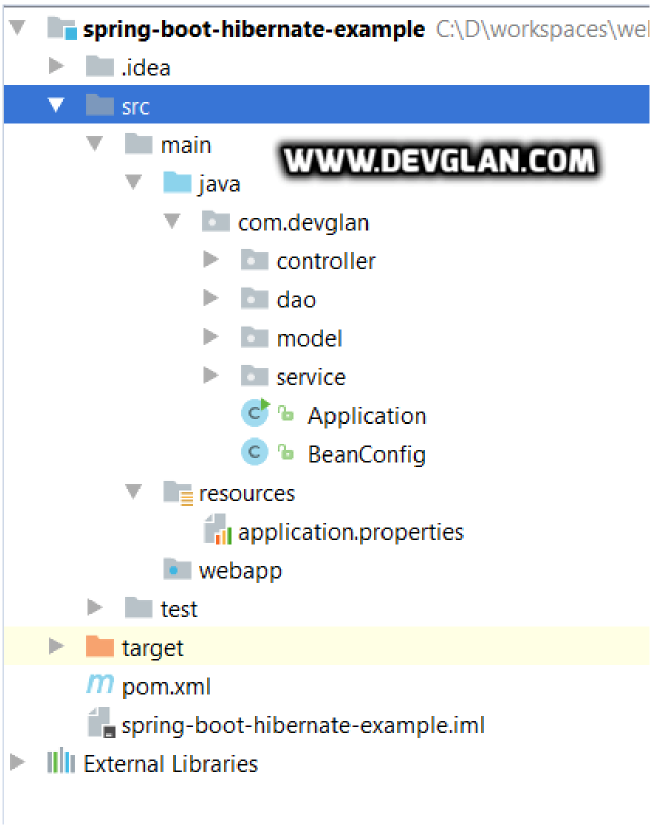
This article is about integrating spring boot with hibernate. Here, we will be using `spring boot 1.5.1` and hence by default it will be `hibernate 5` configurations. We will be creating sample spring boot hibernate example having some rest endpoints exposed through spring controller. The dao class will have `sessionFactory` injected which will be used to create hibernate session and connect to database. We will be using mysql database.

Let's get started.

Table of Contents

Project Structure

Following is the project structure. We have controllers, service and dao layers. We have application.properties defined that contains configurations related to our datasource.



(http://imgur.com/GL4U2d4)

Maven Dependencies

`spring-boot-starter-parent` : It provides useful Maven defaults. It also provides a dependency-management section so that you can omit version tags for existing dependencies.

`spring-boot-starter-web` : It includes all the dependencies required to create a web app. This will avoid lining up different spring common project versions.

`spring-boot-starter-tomcat` : It enable an embedded Apache Tomcat 7 instance, by default.This can be also marked as provided if you wish to deploy the war to any other standalone tomcat.

`spring-boot-starter-data-jpa` : It provides key dependencies for Hibernate, Spring Data JPA and Spring ORM.

pom.xml

```
<parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>1.5.1.RELEASE</version>
</parent>

    <dependencies>
            <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-web</artifactId>
            </dependency>
            <dependency>
                    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-starter-tomcat</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-security</artifactId>
         </dependency>
         <dependency>
             <groupId>org.springframework.boot</groupId>
             <artifactId>spring-boot-starter-data-jpa</artifactId>
              <exclusions>
                  <exclusion>
                      <groupId>org.apache.tomcat</groupId>
                      <artifactId>tomcat-jdbc</artifactId>
                   </exclusion>
               </exclusions>
          </dependency>
          <dependency>
              <groupId>mysql</groupId>
              <artifactId>mysql-connector-java</artifactId>
          </dependency>
           <dependency>
               <groupId>commons-dbcp</groupId>
               <artifactId>commons-dbcp</artifactId>
          </dependency>

    </dependencies>
```

Spring Boot Configuration

@SpringBootApplication enables many defaults. It is a convenience annotation that adds @Configuration, @EnableAutoConfiguration, @EnableWebMvc, @ComponentScan

The `main()` method uses Spring Boot SpringApplication.run() method to launch an application.

Application.java

```
package com.devglan;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

Other Interesting Posts

Spring Data JPA Example (http://www.devglan.com/spring-boot/spring-data-jpa-example)

Spring Hibernate Integration Example (hhttp://www.devglan.com/spring-mvc/spring-hibernate-integration-example-javaconfig)

Spring Boot Actuator Complete Guide (http://www.devglan.com/spring-boot/spring-boot-actuator-tutorial-guide)

Spring Boot Actuator Rest Endpoints Example (http://www.devglan.com/spring-boot/spring-boot-atuator-rest-endpoints-example)

Spring 5 Features and Enhancements (http://www.devglan.com/spring-mvc/spring-5-features-and-enhancements)

Spring Boot Thymeleaf Example (http://www.devglan.com/spring-boot/spring-boot-thymeleaf-example)

Spring Boot Security Hibernate Example with complete JavaConfig (http://www.devglan.com/spring-security/spring-boot-security-hibernate-login-example)

Securing REST API with Spring Boot Security Basic Authentication (http://www.devglan.com/spring-security/spring-boot-security-rest-basic-authentication)

Spring Boot Security Password Encoding using Bcrypt Encoder (http://www.devglan.com/spring-security/spring-boot-security-password-encoding-bcrypt-encoder)

Spring Security with Spring MVC Example Using Spring Boot (http://www.devglan.com/spring-security/spring-boot-security-login-example)

Websocket spring Boot Integration Without STOMP with complete JavaConfig (http://www.devglan.com/spring-boot/spring-websocket-integration-example-without-stomp)

Basic Datasource Configurations in Spring Boot

The most convenient way to define datasource parameters in spring boot application is to make use of `application.properties` file. Following is our sample application.properties. Here we are using JPA based configurations and hibernate as a JPA provider.

The following configuration creates a DriverManagerDataSource which opens and closes a connection to the database when needed.It means no connection pooling is achieved.While doing so, you may have performance issues in the production. In production, it is always recommended to have datasource that supports connection pooling and to create this connection pooling datasource we require to configure custom datasource bean programatically. We will create it in next section.

```
spring.datasource.url=jdbc:mysql://localhost:3306/test
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

Hibernate supports 2 different naming strategies.To use Hibernate 5 default naming strategy, we have used PhysicalNamingStrategyStandardImpl. Keep a note that SpringPhysicalNamingStrategy is the default naming strategy used by spring boot.

Hikari Datasource Configurations with Hibernate

In production, it is always recommended to use datasource that supports connection pooling because database connection creation is a slow process.Here in the example we will be using HikariDatasource instead. It provides many advanced features while configuring our datasource in comparison to other datasources such as connectionTimeout, idleTimeout, maxLifetime, connectionTestQuery, maximumPoolSize and very important one is leakDetectionThreshold.It is as advanced as detecting connection leaks by itself.It is also faster and lighter than other available datasource.Following is the configuration for HikariDatasource.Make sure you comment the datasource confguration in properties file.

HikariDatasource Config

```
@Bean
  public DataSource dataSource() {
    HikariDataSource ds = new HikariDataSource();
    ds.setMaximumPoolSize(100);
    ds.setDataSourceClassName("com.mysql.jdbc.jdbc2.optional.MysqlDataSource");
    ds.addDataSourceProperty("url", "jdbc:mysql://localhost:3306/test");
    ds.addDataSourceProperty("user", "root");
    ds.addDataSourceProperty("password", "password");
    ds.addDataSourceProperty("cachePrepStmts", true);
    ds.addDataSourceProperty("prepStmtCacheSize", 250);
    ds.addDataSourceProperty("prepStmtCacheSqlLimit", 2048);
    ds.addDataSourceProperty("useServerPrepStmts", true);
    return ds;
  }
```

We can also create Hikaridatasource using DataSourceBuilder as follow.While doing so the datasource related properties can be still there in proerties file.I like this way.

```
@Bean
@ConfigurationProperties("spring.datasource")
```

```
public HikariDataSource dataSource() {
	return DataSourceBuilder.create().type(HikariDataSource.class).build();
}
```

In order to use HikariDataSource, you must include following maven dependency. Checkout the latest version here - Hikari Maven (https://mvnrepository.com/artifact/com.zaxxer/HikariCP)

```
<dependency>
		<groupId>com.zaxxer</groupId>
		<artifactId>HikariCP</artifactId>
			<version>2.7.3</version>
			</dependency>
```

In this case, we need to explicitly tell spring boot to use our custom datasource while creating EntityManagerfactory.Following is a sample example.

```
@Bean(name = "entityManagerFactory")
public EntityManagerFactory entityManagerFactory() {
	LocalContainerEntityManagerFactoryBean emf = new LocalContainerEntityManagerFactoryBean();
	emf.setDataSource(dataSource);
	emf.setJpaVendorAdapter(jpaVendorAdapter);
	emf.setPackagesToScan("com.mysource.model");
	emf.setPersistenceUnitName("default");
	emf.afterPropertiesSet();
	return emf.getObject();
}
```

Hibernate Related Configurations

Spring boot focusses on using JPA to persist data in relational db and it has ability to create repository implementations automatically, at runtime, from a repository interface. But here we are trying to use hibernate as a JPA provider. Hence, following configuration is required to autowire sessionFactory in our DAO class.

BeanConfig.java

```
package com.devglan;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

import javax.persistence.EntityManagerFactory;


@Configuration
public class BeanConfig {

	@Autowired
	private EntityManagerFactory entityManagerFactory;

	@Bean
	public SessionFactory getSessionFactory() {
```

```
            if (entityManagerFactory.unwrap(SessionFactory.class) == null) {
                throw new NullPointerException("factory is not a hibernate factory");
            }
            return entityManagerFactory.unwrap(SessionFactory.class);
        }

}
```

Hibernate Entity Class

Following is the entity class. The class is annotated as hibernate entity.

UserDetails.java

```
package com.devglan.model;

@Entity
@Table
public class UserDetails {

        @Id
        @Column
        @GeneratedValue(strategy = GenerationType.IDENTITY)
        private int id;
        @Column
        private String firstName;
        @Column
        private String lastName;
        @Column
        private String email;
        @Column
        private String password;

        //getters and setters goes here
```

Spring Server Implementation

Let us define our controller. It has one url mapping that intercepts request at /list and returns all users present in db.

UserController.java

```
package com.devglan.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import com.devglan.model.UserDetails;
import com.devglan.service.UserService;

@Controller
public class UserController {

	@Autowired
	private UserService userService;

	@RequestMapping(value = "/list", method = RequestMethod.GET)
	public ResponseEntity> userDetails() {

		List userDetails = userService.getUserDetails();
		return new ResponseEntity>(userDetails, HttpStatus.OK);
	}

}
```

Defining Service Class

UserServiceImpl.java

```
package com.devglan.service.impl;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.devglan.dao.UserDao;
import com.devglan.model.UserDetails;
import com.devglan.service.UserService;

@Service
public class UserServiceImpl implements UserService {

	@Autowired
	private UserDao userDao;

	public List getUserDetails() {
		return userDao.getUserDetails();
	}

}
```

Defining Dao Implementation

Let us define the dao.

UserDaoImpl.java

```
package com.devglan.dao.impl;

import java.util.List;
import org.hibernate.Criteria;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;

import com.devglan.dao.UserDao;
import com.devglan.model.UserDetails;

@Component
public class UserDaoImpl implements UserDao {

	@Autowired
	private SessionFactory sessionFactory;

	public List getUserDetails() {
		Criteria criteria = sessionFactory.openSession().createCriteria(UserDetails.class);
		return criteria.list();
	}

}
```

Note:

We can also get hibernate session in following way using JPA entitymanager. But since this article is about spring boot and hibernate integration, we are injecting hibernate sessionfactory and getting session out of it. In next post we will be discussing about spring data with spring boot.

UserDaoImpl.java

```
@Component
public class UserDaoImpl implements UserDao {

    @PersistenceContext
    private EntityManager entityManager;

	public List getUserDetails() {
		Criteria criteria = entityManager.unwrap(Session.class).createCriteria(UserDetails.class);
		return criteria.list();
	}
}
```

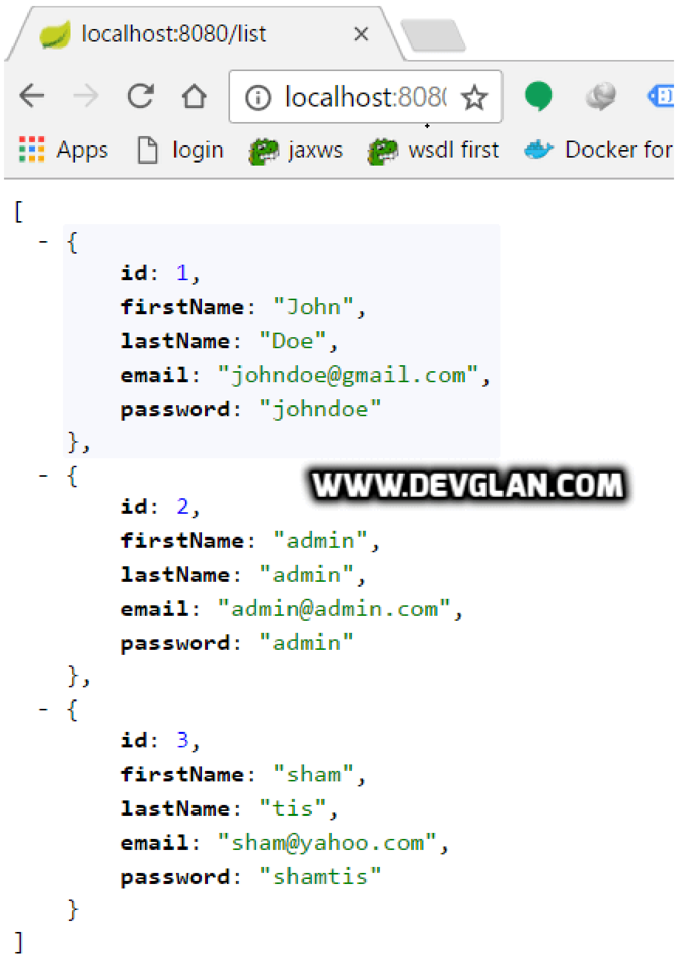
Sample Script

Following are some sample DML. We will be creating some dummy user details using following insert statements.

```
create table User_Details (id integer not null auto_increment, email varchar(255), first_Name varchar(255), last_Name varchar(255), password varchar(255), primary key (id)) ENGINE=InnoDB;

INSERT INTO user_details(email,first_Name,last_Name,password) VALUES ('admin@admin.com','admin','admin','admin');

INSERT INTO user_details(email,first_Name,last_Name,password) VALUES ('john@gmail.com','john','doe','johndoe');

INSERT INTO user_details(email,first_Name,last_Name,password) VALUES ('sham@yahoo.com','sham','tis','shamtis');
```

Run Application

1. Run Application.java as a java application.
2. Hit the url - http://localhost:8080/list (http://localhost:8080/list). Following screen will appear.



```
[
  - {
        id: 1,
        firstName: "John",
        lastName: "Doe",
        email: "johndoe@gmail.com",
        password: "johndoe"
    },
  - {
        id: 2,
        firstName: "admin",
        lastName: "admin",
        email: "admin@admin.com",
        password: "admin"
    },
  - {
        id: 3,
        firstName: "sham",
        lastName: "tis",
        email: "sham@yahoo.com",
        password: "shamtis"
    }
]
```

(http://imgur.com/Nx4wIKI)

Conclusion

I hope this article served you that you were looking for. If you have anything that you want to add or share then please share it below in the **comment section**.

R#
ReSharper
THE LEGENDARY EXTENSION TO VISUAL STUDIO
*WARNING! Prolonged use may cause addiction.
Get a free trial
JET BRAINS

Download the source

()

Further Reading:

1. Spring Boot Multiple Database Configuration (http://www.devglan.com/spring-boot/spring-boot-multiple-database-configuration)
2. Spring Boot H2 Database Example (http://www.devglan.com/spring-boot/spring-boot-h2-database-example)
3. Spring Data Jpa Example (http://www.devglan.com/spring-boot/spring-data-jpa-example)
4. Spring Boot Actuator Rest Endpoints Example (http://www.devglan.com/spring-boot/spring-boot-actuator-rest-endpoints-example)
5. Spring Boot Mvc App With Jsp (http://www.devglan.com/spring-boot/spring-boot-mvc-app-with-jsp)
6. Spring Boot Security Hibernate Login Example (http://www.devglan.com/spring-security/spring-boot-security-hibernate-login-example)
7. Spring Boot Websocket Integration Example (http://www.devglan.com/spring-boot/spring-boot-websocket-integration-example)

Suggest more topics (user/suggest) in suggestion section or write your own article (user/write-for-us) and share with your colleagues.

San Salvador de Jujuy - Departamento 2 habitaciones

$ 1 031

San Salvador de Jujuy - Munay San Salvador de Jujuy

Bien 7.7

References

Accessing data JPA (https://spring.io/guides/gs/accessing-data-jpa/)

Sspring Boot features (https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-sql.html/)

spring Boot Datasource (https://stackoverflow.com/questions/28821521/configure-datasource-programmatically-in-spring-boot)

Data Access (https://docs.spring.io/spring-boot/docs/current-SNAPSHOT/reference/html/howto-data-access.html)

Share this post

FACEBOOK (HTTPS://WWW.FACE[illegible] U=HTTP://WWW.DEVGLAN.COM/SPRING-BOOT/SPRING-BOOT-HIBERNATE-5-EXAMPLE)
LINKEDIN ([illegible] BOOT/SPRING-BOOT-HIBERNATE-5-EXAMPLE&TITLE=SPRING BOOT HIBERNATE 5 WITH MYSQL EXAMPLE - DEVGLAN&SUMMARY=SPRING-BOOT-HIBERNATE-5-EXAMPLE)
GOOGLE+ ([illegible] BOOT/SPRING-BOOT-HIBERNATE-5-EXAMPLE)
REDDIT ([illegible] BOOT/SPRING-BOOT-HIBERNATE-5-EXAMPLE&TITLE=SPRING BOOT HIBERNATE 5 WITH MYSQL EXAMPLE - DEVGLAN&TEXT=SPRING-BOOT-HIBERNATE-5-EXAMPLE)
TWITTER ([illegible] BOOT/SPRING-BOOT-HIBERNATE-5-EXAMPLE)
TUMBLR ([illegible] BOOT/SPRING-BOOT-HIBERNATE-5-EXAMPLE)
PINTEREST ([illegible] BOOT/SPRING-BOOT-HIBERNATE-5-EXAMPLE&MEDIA=BOOT-HIBERNATE-5-EXAMPLE)

9 Comments Devglan Login

Recommend Share Sort by Best

Join the discussion...

LOG IN WITH OR SIGN UP WITH DISQUS ?

Name

ismail • 2 months ago
Where to put the Application.properties ? The img url is broken :(
Reply • Share ›

Dhiraj Ray ➦ ismail • 2 months ago
src/main/resources
Reply • Share ›

dudhat dhaval • 6 months ago
Awesome :)
Reply • Share ›

Maheswara Reddy • 6 months ago
my Spring boot application running fine, but while running on browser it was asking credentials like username and password. plz solve my issue
Reply • Share ›

Dhiraj Ray ➦ Maheswara Reddy • 6 months ago
In application.properties add this entry - security.basic.enabled=false
Reply • Share ›

Dmytro Martyniuk • 10 months ago
Hi, I have got problem with entityManagerFactory in BeanConfig class, The problem is: Could not autowired. No beans of 'entityManagerFactory' type found. Can you help me? Thank you :)
Reply • Share ›

Dhiraj Ray Mod ➦ Dmytro Martyniuk • 10 months ago
I don't see any issue. I tried importing it as a maven project and it is working for me.It may be your workspace problem. You just delete it from the workspace and try importing it again as a maven project and run the Application.java. Or you can post the logs here.
Reply • Share ›

Dmytro Martyniuk ➦ Dhiraj Ray • 10 months ago

I don't know why, but now it works. Thank you very much for this atricle

Reply • Share ›

Dhiraj Ray Mod → Dmytro Martyniuk • 10 months ago

you are welcome

Reply • Share ›

ALSO ON **DEVGLAN**

Spring Boot H2 Database Example With Hibernate - DevGlan

9 comments • 10 months ago

Mrinmoy Majumdar — For those who are looking for some example implementation and source code, …

spring mvc pdf and excel view example - DevGlan

4 comments • 9 months ago

Dhiraj Ray — Sure but it will take some time as I m busy with other stuffs. Bdw thanks for your …

Spring Boot Actuator Tutorial Guide

1 comment • a year ago

5ofo5 — Very nice and pretty straightforward tutorial.Thank you!

Spring Boot Security Redirect After Login

3 comments • a year ago

Pete — And thank you for the tutorial!

Subscribe **Add Disqus to your siteAdd DisqusAdd** **Privacy**