WebSystique

learn together

Spring 4 + Hibernate 4 + MySQL+ Maven Integration example (Annotations+XML)

August 20, 2014 websystiqueadmin

In this tutorial , we will integrate Spring 4 with `Hibernate 4` using annotation based configuration. We will develop a simple CRUD java application , creating hibernate entities, saving data in `MySQL database` , performing database CRUD operations within `transaction` , and learn how different layers interacts with each-other in typical enterprise application, all using `annotation based configuration` . We will also see corresponding XML configuration side-by-side for comparison.

For Spring MVC based application, checkout Spring4 MVC Hibernate and MySQL integration.

Other interesting posts you may like

- Spring MVC 4+Hibernate 4+MySQL+Maven integration example
- Spring MVC 4+Hibernate 4+MySQL+Maven integration + Testing example using annotations
- Spring MVC 4+Hibernate 4 Many-to-many JSP Example
- Spring MVC 4+AngularJS Example

Recent Posts

Spring 4 MVC+AngularJS CRUD Application using ngResource

Angularjs Server Communication using ngResource-CRUD Application

AngularJS Custom-Directives controllers, require option guide

AngularJS Custom-Directives transclude, ngTransclude guide

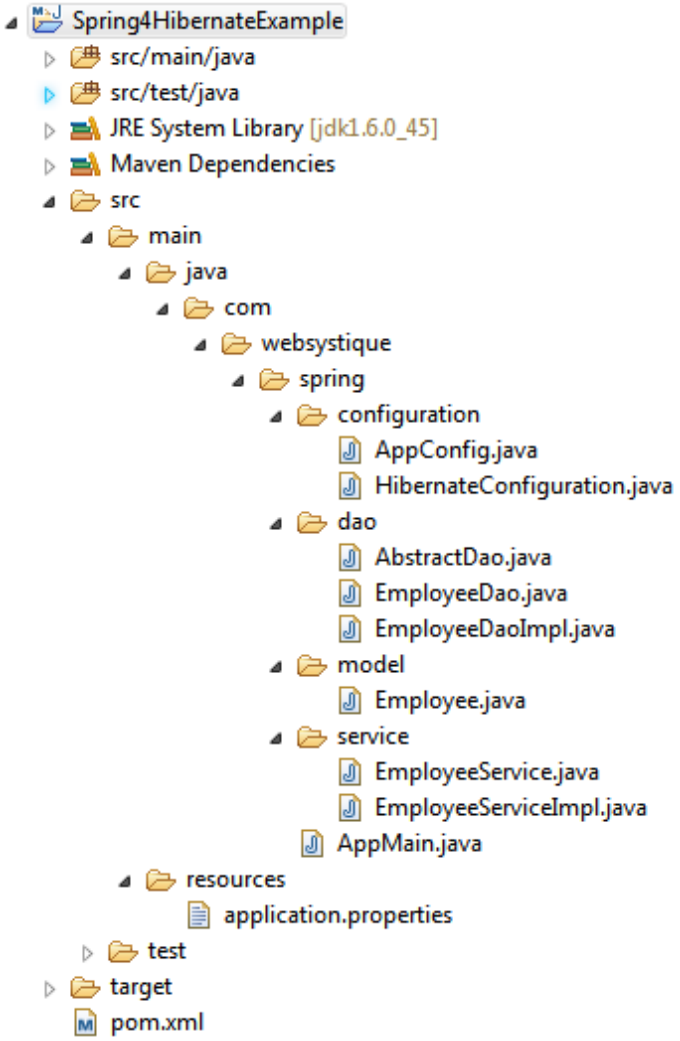
AngularJS Custom-Directives replace option guide

- Spring Security 4 + Hibernate Database Authentication Example

Following technologies being used:

- Spring 4.0.6.RELEASE
- Hibernate Core 4.3.6.Final
- MySQL Server 5.6
- Joda-time 2.3
- Maven 3
- JDK 1.6
- Eclipse JUNO Service Release 2

Project directory structure



Let's now add the content mentioned in above structure explaining each in detail.

Step 1: Update pom.xml to include required dependencies

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http:
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://ma

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.websystique.spring</groupId>
    <artifactId>Spring4HibernateExample</artifactId>
    <version>1.0.0</version>
    <packaging>jar</packaging>

    <name>Spring4HibernateExample</name>

    <properties>
        <springframework.version>4.0.6.RELEASE</springframework.ver
        <hibernate.version>4.3.6.Final</hibernate.version>
        <mysql.connector.version>5.1.31</mysql.connector.version>
        <joda-time.version>2.3</joda-time.version>
    </properties>

    <dependencies>

        <!-- Spring -->
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-core</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-context</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-tx</artifactId>
            <version>${springframework.version}</version>
        </dependency>
        <dependency>
            <groupId>org.springframework</groupId>
            <artifactId>spring-orm</artifactId>
            <version>${springframework.version}</version>
        </dependency>

        <!-- Hibernate -->
        <dependency>
            <groupId>org.hibernate</groupId>
            <artifactId>hibernate-core</artifactId>
            <version>${hibernate.version}</version>
        </dependency>

        <!-- MySQL -->
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>${mysql.connector.version}</version>
        </dependency>

        <!-- Joda-Time -->
        <dependency>
            <groupId>joda-time</groupId>
            <artifactId>joda-time</artifactId>
            <version>${joda-time.version}</version>
        </dependency>

        <!-- To map JodaTime with database type -->
        <dependency>
            <groupId>org.jadira.usertype</groupId>
            <artifactId>usertype.core</artifactId>
            <version>3.0.0.CR1</version>
        </dependency>
```

```
    </dependencies>

</project>
```

Spring, Hibernate & MySQL connector dependencies are pretty obvious. We have also included joda-time as we will use joda-time library for any date manipulation. usertype-core is included to provide the mapping between database date-type and joda-time LocalDate.

Step 2: Configure Hibernate

`com.websystique.spring.configuration.HibernateConfiguration`

```
package com.websystique.spring.configuration;

import java.util.Properties;

import javax.sql.DataSource;

import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.core.env.Environment;
import org.springframework.jdbc.datasource.DriverManagerDataSource;
import org.springframework.orm.hibernate4.HibernateTransactionManag
import org.springframework.orm.hibernate4.LocalSessionFactoryBean;
import org.springframework.transaction.annotation.EnableTransactior

@Configuration
@EnableTransactionManagement
@ComponentScan({ "com.websystique.spring.configuration" })
@PropertySource(value = { "classpath:application.properties" })
public class HibernateConfiguration {

    @Autowired
    private Environment environment;

    @Bean
    public LocalSessionFactoryBean sessionFactory() {
        LocalSessionFactoryBean sessionFactory = new LocalSessionFa
        sessionFactory.setDataSource(dataSource());
        sessionFactory.setPackagesToScan(new String[] { "com.websys
        sessionFactory.setHibernateProperties(hibernateProperties()
        return sessionFactory;
     }

    @Bean
    public DataSource dataSource() {
        DriverManagerDataSource dataSource = new DriverManagerDataS
        dataSource.setDriverClassName(environment.getRequiredProper
        dataSource.setUrl(environment.getRequiredProperty("jdbc.url
        dataSource.setUsername(environment.getRequiredProperty("jdb
        dataSource.setPassword(environment.getRequiredProperty("jdb
        return dataSource;
    }

    private Properties hibernateProperties() {
        Properties properties = new Properties();
        properties.put("hibernate.dialect", environment.getRequirec
        properties.put("hibernate.show_sql", environment.getRequire
        properties.put("hibernate.format_sql", environment.getRequi
```

```
        return properties;
    }

    @Bean
    @Autowired
    public HibernateTransactionManager transactionManager(SessionFa
       HibernateTransactionManager txManager = new HibernateTransac
       txManager.setSessionFactory(s);
       return txManager;
    }
}
```

`@Configuration` indicates that this class contains one or more bean methods annotated with `@Bean` producing beans manageable by spring container. In our case, this class represent hibernate configuration.
`@ComponentScan` is equivalent to `context:component-scan base-package="..."` in xml, providing with where to look for spring managed beans/classes.
`@EnableTransactionManagement` is equivalent to Spring's tx:* XML namespace, enabling Spring's annotation-driven transaction management capability.
`@PropertySource` is used to declare a set of properties(defined in a properties file in application classpath) in Spring run-time `Environment`, providing flexibility to have different values in different application environments.

Method `sessionFactory()` is creating a `LocalSessionFactoryBean`, which exactly mirrors the XML based configuration : We need a dataSource and hibernate properties (same as hibernate.properties). Thanks to `@PropertySource`, we can externalize the real values in a .properties file, and use Spring's `Environment` to fetch the value corresponding to an item. Once the SessionFactory is created, it will be injected into Bean method `transactionManager` which may eventually provide transaction support for the sessions created by this sessionFactory.

Below is the properties file used in this post.

Below is the properties file used in this post.
`/src/main/resources/application.properties`

```
jdbc.driverClassName = com.mysql.jdbc.Driver
jdbc.url = jdbc:mysql://localhost:3306/websystique
jdbc.username = myuser
jdbc.password = mypassword
hibernate.dialect = org.hibernate.dialect.MySQLDialect
hibernate.show_sql = false
hibernate.format_sql = false
```

Corresponding XML based Hibernate configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/contex
        xmlns:tx="http://www.springframework.org/schema/tx"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xsi:schemaLocation="http://www.springframework.org/schema/b
                            http://www.springframework.org/schema/c
                            http://www.springframework.org/schema/a
                            http://www.springframework.org/schema/t


    <context:property-placeholder location="classpath:application.p

    <context:component-scan  base-package="com.websystique.spring"

    <tx:annotation-driven transaction-manager="transactionManager"/

    <bean id="dataSource" class="org.springframework.jdbc.datasourc
        <property name="driverClassName" value="${jdbc.driverClassN
        <property name="url" value="${jdbc.url}"/>
        <property name="username" value="${jdbc.username}" />
        <property name="password" value="${jdbc.password}"/>

    </bean>

    <bean id="sessionFactory" class="org.springframework.orm.hiberr
        <property name="dataSource" ref="dataSource"/>
        <property name="packagesToScan">
            <list>
                <value>com.websystique.spring.model</value>
            </list>
        </property>
        <property name="hibernateProperties">
            <props>
                <prop key="hibernate.dialect">${hibernate.dialect}<
                <prop key="hibernate.show_sql">${hibernate.show_sql
                <prop key="hibernate.format_sql">${hibernate.format
            </props>
        </property>
    </bean>

    <bean id="transactionManager"  class="org.springframework.orm.h
        <property name="sessionFactory" ref="sessionFactory" />
    </bean>

    <bean id="persistenceExceptionTranslationPostProcessor"
        class="org.springframework.dao.annotation.PersistenceExcept

</beans>
```

Step 3: Configure Spring

`com.websystique.spring.configuration.AppConfig`

```
package com.websystique.spring.configuration;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan(basePackages = "com.websystique.spring")
public class AppConfig {

}
```

In our simple example, this class is empty and only reason for it's existence is `@ComponentScan` which provides beans auto-detection facility. You may completely remove above configuration and put the component scan logic in application context level (in Main). In full-fledged applications, you may find it handy to configure some beans (e.g. messageSource, PropertySourcesPlaceHolderConfigurer) in Configuration class.

Corresponding XML based Spring Configuration

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:context="http://www.springframework.org/schema/context"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/context http://www.spring

    <context:component-scan base-package="com.websystique.spring" /

</beans>
```

As for as Annotation based configuration goes, this is all we need to do. Now to make the application complete, we will add service layer, dao layer, Domain object, sample database schema and run the application.

Step 4: Add DAO Layer

`com.websystique.spring.dao.AbstractDao`

```
package com.websystique.spring.dao;

import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.springframework.beans.factory.annotation.Autowired;

public abstract class AbstractDao {

    @Autowired
    private SessionFactory sessionFactory;

    protected Session getSession() {
        return sessionFactory.getCurrentSession();
    }

    public void persist(Object entity) {
        getSession().persist(entity);
    }

    public void delete(Object entity) {
        getSession().delete(entity);
    }
}
```

Notice above, that SessionFactory we have created earlier in step 2, will be auto-wired here.This class serve as base class for database related

operations.

com.websystique.spring.dao.EmployeeDao

```
package com.websystique.spring.dao;

import java.util.List;

import com.websystique.spring.model.Employee;

public interface EmployeeDao {

    void saveEmployee(Employee employee);

    List<Employee> findAllEmployees();

    void deleteEmployeeBySsn(String ssn);

    Employee findBySsn(String ssn);

    void updateEmployee(Employee employee);
}
```

com.websystique.spring.dao.EmployeeDaoImpl

```
package com.websystique.spring.dao;

import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Query;
import org.hibernate.criterion.Restrictions;
import org.springframework.stereotype.Repository;

import com.websystique.spring.model.Employee;

@Repository("employeeDao")
public class EmployeeDaoImpl extends AbstractDao implements Employe

    public void saveEmployee(Employee employee) {
        persist(employee);
    }

    @SuppressWarnings("unchecked")
    public List<Employee> findAllEmployees() {
        Criteria criteria = getSession().createCriteria(Employee.cl
        return (List<Employee>) criteria.list();
    }

    public void deleteEmployeeBySsn(String ssn) {
        Query query = getSession().createSQLQuery("delete from Empl
        query.setString("ssn", ssn);
        query.executeUpdate();
    }


    public Employee findBySsn(String ssn){
        Criteria criteria = getSession().createCriteria(Employee.cl
        criteria.add(Restrictions.eq("ssn",ssn));
        return (Employee) criteria.uniqueResult();
    }

    public void updateEmployee(Employee employee){
        getSession().update(employee);
    }
```

```
}
```

Step 5: Add Service Layer

`com.websystique.spring.service.EmployeeService`

```
package com.websystique.spring.service;

import java.util.List;

import com.websystique.spring.model.Employee;

public interface EmployeeService {

    void saveEmployee(Employee employee);

    List<Employee> findAllEmployees();

    void deleteEmployeeBySsn(String ssn);

    Employee findBySsn(String ssn);

    void updateEmployee(Employee employee);
}
```

`com.websystique.spring.service.EmployeeServiceImpl`

```
package com.websystique.spring.service;

import java.util.List;

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

import com.websystique.spring.dao.EmployeeDao;
import com.websystique.spring.model.Employee;

@Service("employeeService")
@Transactional
public class EmployeeServiceImpl implements EmployeeService{

    @Autowired
    private EmployeeDao dao;

    public void saveEmployee(Employee employee) {
        dao.saveEmployee(employee);
    }

    public List<Employee> findAllEmployees() {
        return dao.findAllEmployees();
    }

    public void deleteEmployeeBySsn(String ssn) {
        dao.deleteEmployeeBySsn(ssn);
    }

    public Employee findBySsn(String ssn) {
        return dao.findBySsn(ssn);
    }

    public void updateEmployee(Employee employee){
```

```
        dao.updateEmployee(employee);
    }
}
```

Most interesting part above is `@Transactional` which starts a transaction on each method start, and commits it on each method exit (or rollback if method was failed due to an error). Note that since the transaction are on method scope, and inside method we are using DAO, DAO method will be executed within same transaction.

Step 6: Create Domain Entity Class(POJO)

Let's create the actual Employee Entity itself whose instances we will be playing with in database.

`com.websystique.spring.model.Employee`

```
package com.websystique.spring.model;

import java.math.BigDecimal;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;

import org.hibernate.annotations.Type;
import org.joda.time.LocalDate;

@Entity
@Table(name="EMPLOYEE")
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    @Column(name = "NAME", nullable = false)
    private String name;

    @Column(name = "JOINING_DATE", nullable = false)
    @Type(type="org.jadira.usertype.dateandtime.joda.PersistentLoca
    private LocalDate joiningDate;

    @Column(name = "SALARY", nullable = false)
    private BigDecimal salary;

    @Column(name = "SSN", unique=true, nullable = false)
    private String ssn;

    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }
```

```
    public void setName(String name) {
        this.name = name;
    }

    public LocalDate getJoiningDate() {
        return joiningDate;
    }

    public void setJoiningDate(LocalDate joiningDate) {
        this.joiningDate = joiningDate;
    }

    public BigDecimal getSalary() {
        return salary;
    }

    public void setSalary(BigDecimal salary) {
        this.salary = salary;
    }

    public String getSsn() {
        return ssn;
    }

    public void setSsn(String ssn) {
        this.ssn = ssn;
    }

    @Override
    public int hashCode() {
        final int prime = 31;
        int result = 1;
        result = prime * result + id;
        result = prime * result + ((ssn == null) ? 0 : ssn.hashCode
        return result;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (!(obj instanceof Employee))
            return false;
        Employee other = (Employee) obj;
        if (id != other.id)
            return false;
        if (ssn == null) {
            if (other.ssn != null)
                return false;
        } else if (!ssn.equals(other.ssn))
            return false;
        return true;
    }

    @Override
    public String toString() {
        return "Employee [id=" + id + ", name=" + name + ", joining
                + joiningDate + ", salary=" + salary + ", ssn=" + s
    }



}
```

This is a standard Entity class annotated with JPA annotations `@Entity`, `@Table`, `@Column` along with hibernate specific annotation `@Type`

which we are using to provide mapping between database date type and Joda-Time `LocalDate`

Step 7: Create Schema in database

```
CREATE TABLE EMPLOYEE(
    id INT NOT NULL auto_increment,
    name VARCHAR(50) NOT NULL,
    joining_date DATE NOT NULL,
    salary DOUBLE NOT NULL,
    ssn VARCHAR(30) NOT NULL UNIQUE,
    PRIMARY KEY (id)
);
```

Please visit MySQL installation on Local PC in case you are finding difficulties in setting up MySQL locally.

Step 8: Create Main to run as Java Application

```
package com.websystique.spring;

import java.math.BigDecimal;
import java.util.List;

import org.joda.time.LocalDate;
import org.springframework.context.annotation.AnnotationConfigAppli
import org.springframework.context.support.AbstractApplicationConte

import com.websystique.spring.configuration.AppConfig;
import com.websystique.spring.model.Employee;
import com.websystique.spring.service.EmployeeService;

public class AppMain {

    public static void main(String args[]) {
        AbstractApplicationContext context = new AnnotationConfigAp

        EmployeeService service = (EmployeeService) context.getBear

        /*
         * Create Employee1
         */
        Employee employee1 = new Employee();
        employee1.setName("Han Yenn");
        employee1.setJoiningDate(new LocalDate(2010, 10, 10));
        employee1.setSalary(new BigDecimal(10000));
        employee1.setSsn("ssn00000001");

        /*
         * Create Employee2
         */
        Employee employee2 = new Employee();
        employee2.setName("Dan Thomas");
        employee2.setJoiningDate(new LocalDate(2012, 11, 11));
        employee2.setSalary(new BigDecimal(20000));
        employee2.setSsn("ssn00000002");

        /*
         * Persist both Employees
         */
        service.saveEmployee(employee1);
        service.saveEmployee(employee2);
```

```
        /*
         * Get all employees list from database
         */
        List<Employee> employees = service.findAllEmployees();
        for (Employee emp : employees) {
            System.out.println(emp);
        }

        /*
         * delete an employee
         */
        service.deleteEmployeeBySsn("ssn00000002");

        /*
         * update an employee
         */

        Employee employee = service.findBySsn("ssn00000001");
        employee.setSalary(new BigDecimal(50000));
        service.updateEmployee(employee);

        /*
         * Get all employees list from database
         */
        List<Employee> employeeList = service.findAllEmployees();
        for (Employee emp : employeeList) {
            System.out.println(emp);
        }

        context.close();
    }
}
```

Note : In case you want to drop AppConfig altogether, in above main, you just have to replace

```
AbstractApplicationContext context = new AnnotationConfigApplicatic
```

with

```
AnnotationConfigApplicationContext  context = new AnnotationConfigA
context.scan("com.websystique.spring");
context.refresh();
```

Rest of code remains same. Run above program, you will see following output

```
Employee [id=1, name=Han Yenn, joiningDate=2010-10-10, salary=10000
Employee [id=2, name=Dan Thomas, joiningDate=2012-11-11, salary=200
Employee [id=1, name=Han Yenn, joiningDate=2010-10-10, salary=50000
```

That's it.

Download Source Code

Download Now!

References

- Spring framework
- Hibernate

websystiqueadmin

If you like tutorials on this site, why not take a step further and connect me on Facebook , Google Plus & Twitter as well? I would love to hear your thoughts on these articles, it will help me improve further our learning process.

If you appreciate the effort I have put in this learning site, help me improve the visibility of this site towards global audience by sharing and linking this site from within and beyond your network. You & your friends can always link my site from your site on www.websystique.com, and share the learning.

After all, we are here to learn together, aren't we?

Related Posts:

1. **Spring 4 MVC+Hibernate 4+MySQL+Maven integration example using annotations**
2. **Spring 4 MVC+Hibernate 4+MySQL+Maven integration + Testing example using annotations**
3. **Spring Auto-detection autowire & Component-scanning Example With Annotations**
4. **Spring Beans Auto-wiring Example using XML Configuration**

spring. permalink.

← Spring @Profile Example

Spring Dependency Injection Example with Constructor and Property Setter (XML) →

Guilherme Marquesini Reis Ribe • 15 days ago

Hi websys.

I'm trying to use this code in a old webserver that don't use Spring MVC, its using @WebServlet of Javax. I just want to include the spring data to manage my transactions instead of I do this mannualy.

I made all the configurations correctly. I'm trying to call my UserService but I'm getting a NullPointerException when I call a method of this class, like:
user = userService.createUser(user);

I believe that the Spring Data didnt starts with my application context. I didnt use the :

```
AnnotationConfigApplicationContext context = new
AnnotationConfigApplicationContext();
context.scan("com.websystique.");
context.refresh();
```

see more

• Reply • Share ›

websystique Mod ➚ Guilherme Marquesini Reis Ribe
• 13 days ago

Hi Guilherme, Did you try to simply declare contextLoaderListner in web.xml and then provide a seperate XML file [using contextConfigLocation] in which specifying context:component-scan with path to your packages?It should be sufficient.

• Reply • Share ›

Guilherme Marquesini Reis Ribe ➚ websystique
• 12 days ago

Hi websys. Thanks for the reply. I solved this without Spring, opening and closing each session on each DAO. Is a old approach of course. I will

recommend to migrate to Spring on the next releases.
Thanks.

• Reply • Share ›

Luxmanrao Potadar • 5 months ago

hi,
I am working on your examples. When I created your examples with STS and execute them I am getting error as:Exception in thread "main" org.springframework.beans.factory.NoSuchBeanDefinitionExceptior No bean named "..." but when I download your code and execute it works fine. I am not getting what is causing this issue. Even I tried creating sample maven projects and executed the code I get same error.

• Reply • Share ›

websystique Mod ➚ Luxmanrao Potadar • 5 months ago

Hi Laxmanrao,

Could you please post the full exception here, if possible? I want to know which Bean it is complaining about.

• Reply • Share ›

deepak • 6 months ago

my data source look like
jdbc.driverClassName = oracle.jdbc.driver.OracleDriver
jdbc.url = jdbc:oracle:thin:@localhost:1521:xe
jdbc.username = hr
jdbc.password = hr
hibernate.dialect = org.hibernate.dialect.Oracle10gDialect
hibernate.show_sql = false
hibernate.format_sql = false

• Reply • Share ›

websystique Mod ➚ deepak • 6 months ago

Hey Deepak,
If this file is in classpath, it will be found. I see that you are getting following error

Caused by: java.lang.IllegalStateException: required key [jdbc:oracle:thin:@localhost:1521:xe] not found at org.springframework.core.env.AbstractPropertyResolver.ge

Can you check if your database url is OK? And you referring to this using a KEY defined in properties file dataSource.setUrl(environment.getRequiredProperty("jdbc.u

This error gives me impression that you tried to use something like dataSource.setUrl(environment.getRequiredProperty("jdbc:o

• Reply • Share ›

deepak ➚ websystique • 5 months ago

now i changed dataSource.setUrl(environment.getRequiredProperty it saying java.lang.IllegalStateException: required key [hibernate.show_sql] not found.
Instantiation of bean failed-hibernateProperties and sessionFactory.

• Reply • Share ›

websystique Mod ➚ deepak • 5 months ago

Hi Deepak, Do you have more than one properties file in your project? Are you sure you are using only annotation config(and not additional XML file with propertyplaceholderconfig defined in it)? It won't matter much but where exactly did you placed the properties file? Did you try to run the project as it is from download, to see if you succeed to run it? Which version of Spring are you using? Example is so simple & straight-forward , i don't see why it won't work for you.

• Reply • Share ›

deepak ➚ deepak • 5 months ago

Hi,
If u have a skype id..plz share with me on my mail id.so that i could share my screen and show you what happening.

• Reply • Share ›

deepak • 6 months ago

looks like spring container is unable to find my property file where i defined my data source

• Reply • Share ›

deepak • 6 months ago

I wrote my code this way..
@Configuration
@EnableTransactionManagement
@ComponentScan({"com.spring4hibernate.configuration"})
@PropertySource(value = {"classpath:application.properti..."})
public class HibernateConfiguration {
===compiler===
Aug 17, 2015 3:35:41 PM
org.springframework.context.annotation.AnnotationConfigApplicatior

prepareRefresh
INFO: Refreshing
org.springframework.context.annotation.AnnotationConfigApplicatior
startup date [Mon Aug 17 15:35:41 IST 2015]; root of context
hierarchy
Aug 17, 2015 3:35:42 PM
org.springframework.jdbc.datasource.DriverManagerDataSource
setDriverClassName
INFO: Loaded JDBC driver: oracle.jdbc.driver.OracleDriver

see more

• Reply • Share ›

deepak • 6 months ago

@PropertySource(value = {"classpath:application.properti..."})..Do i need to specify my classpath to resource folder explicitly...need help!!

• Reply • Share ›

websystique Mod ➚ deepak • 6 months ago

No you don't need to do anything special for resource folder.It is in classpath

• Reply • Share ›

Guilherme Marquesini Reis Ribe • 7 months ago

Thank you sir. It's working . Good example

• Reply • Share ›

websystique Mod ➚ Guilherme Marquesini Reis Ribe • 6 months ago

Hey Guilherme,

Glad it helped. Many Thanks for your feedback.

• Reply • Share ›

Ajeet Mohan • 7 months ago

I am running in Eclipse, getting exception on executing mail method, No bean named 'employeeService' is defined, I have added all the jars required... plz advice.

• Reply • Share ›

websystique Mod ➚ Ajeet Mohan • 7 months ago

Hey Ajeet, make sure to annotate your service implementation class with Spring annotation [@Service("employeeService")]

• Reply • Share ›

Nick Yashaev ➚ websystique • 7 months ago

doesnt work for me too ...it is annotated , and still nothing

10:19:29 2015 ,04 אוג PM
org.springframework.context.annotation.AnnotationC
prepareRefresh
INFO: Refreshing
org.springframework.context.annotation.AnnotationC
startup date [Tue Aug 04 22:19:29 IDT 2015]; root of
context hierarchy
Exception in thread "main"
org.springframework.beans.factory.NoSuchBeanDefi
No bean named 'userService' is defined
at
org.springframework.beans.factory.support.DefaultLi
at
org.springframework.beans.factory.support.AbstractB
at
org.springframework.beans.factory.support.AbstractB
at
org.springframework.beans.factory.support.AbstractB
at
org.springframework.context.support.AbstractApplica
at
com.websystique.spring.AppMain.main(AppMain.java

here is the trace...

• Reply • Share ›

websystique Mod ➚ Nick Yashaev
• 7 months ago

Hi Nick, Are you sure you are not mixing up this project with other projects you might be working/trying on? We don't have any userService in this project. We have only EmployeeSerivce.

Look at above AppMain class, basically this line :

EmployeeService service = (EmployeeService) context.getBean("employeeService");

Download the project and try again please. Let me know if any issue.

• Reply • Share ›

Nick Yashaev ➚ websystique
• 7 months ago

i have done the same thing with the same annotations , just my class is called user and not employee.... and im having the same issue the first commenter had...

the issue is that i dont understand this even if it supposed to work , HOW ?
there is no "EmployeeManager" Bean defined at anyplace , how can we get that bean ?

Reply • Share ›

websystique Mod ➚ Nick Yashaev • 7 months ago

There is no EmployeeManager, but there is an EmployeeServiceImpl defined above with "employeeService"

```
@service("employeeService")
@Transactional
public class EmployeeServiceImpl
implements EmployeeService
```

And this employeeService is then looked up in AppMain

```
EmployeeService service =
(EmployeeService)
context.getBean("employeeService");
```

Reply • Share ›

Manjunath M • 10 months ago

Does all the set up will be same for linx as well.

Reply • Share ›

websystique Mod ➚ Manjunath M • 9 months ago

Hi Manjunath,

Yes it should remain same.

Reply • Share ›

Mythul • a year ago

Great guide. Love it.

Reply • Share ›

Evan Hu • a year ago

Thank you sir. It's working very well in my local pc.

Reply • Share ›

Ramu • 8 hours ago

hi sir i tried to use your code in my project i ma getting this exception please help me

Exception in thread "main"
org.springframework.transaction.CannotCreateTransactionExceptio
Could not open Hibernate Session for transaction; nested exception is org.hibernate.exception.SQLGrammarException: Could not open

connection

at
org.springframework.orm.hibernate4.HibernateTransactionManager.

at
org.springframework.transaction.support.AbstractPlatformTransacti

at
org.springframework.transaction.interceptor.TransactionAspectSupp

at

---

see more

• Reply • Share ›

otuwa • 6 months ago

Hi,

I followed your tutorial. It was really helpfull. After running it once, i ran the application again. But then error occured saying that duplicate entry and cannot execute. Then I tried adding "hibernate.hbm2ddl.auto=create" and it didn't work even. Please help me out

• Reply • Share ›

otuwa ➚ otuwa • 6 months ago

ERROR: Duplicate entry 'ssn00000001' for key 'ssn'

Exception in thread "main"
org.hibernate.exception.ConstraintViolationException: could not execute statement

at
org.hibernate.exception.internal.SQLExceptionTypeDelegate

at
org.hibernate.exception.internal.StandardSQLExceptionConv

at
org.hibernate.engine.jdbc.spi.SqlExceptionHelper.convert(Sc