**\*\*\*\*\*Advance Spring\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Spring-Hibernate\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*Table Script\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

CREATE TABLE stock(STOCK\_ID NUMBER(10) PRIMARY KEY,

STOCK\_CODE VARCHAR2(20) UNIQUE,

STOCK\_NAME VARCHAR2(20) UNIQUE);

**\*\*\*\*\*\*\*\*\*\*\*StockDao.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hib3.Dao;

import training.spring.hib3.entity.Stock;

public interface StockDao {

void save(Stock stock);

void update(Stock stock);

void delete(Stock stock);

Stock findByStockCode(String stockCode);

}

**\*\*\*\*\*\*\*\*\*\*\*\*StockDaoImpl.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hib3.Dao;

import java.util.List;

import org.hibernate.SessionFactory;

import org.hibernate.Session;

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

import org.springframework.orm.hibernate3.support.HibernateDaoSupport;

import org.springframework.stereotype.Repository;

import training.spring.hib3.entity.Stock;

@Repository("stockDao")

public class StockDaoImpl implements StockDao {

private SessionFactory sessionFactory;

@Autowired

public StockDaoImpl(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

System.out.println("this.sessionFactory : " + this.sessionFactory);

}

private Session currentSession() {

return sessionFactory.getCurrentSession();

}

public void save(Stock stock) {

System.out.println("in save");

currentSession().save(stock);

}

public void update(Stock stock) {

currentSession().update(stock);

}

public void delete(Stock stock) {

currentSession().delete(stock);

}

public Stock findByStockCode(String stockCode) {

Query qry=currentSession().createQuery("from Stock where stockCode=?");

qry.setString(0,stockCode);

List list=qry.list();

**return** (Stock) list.get(0); }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Stock.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hib3.entity;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import static javax.persistence.GenerationType.IDENTITY;

import javax.persistence.Id;

import javax.persistence.Table;

import javax.persistence.UniqueConstraint;

@Entity

@Table(name = "stock")

public class Stock implements java.io.Serializable {

private Integer stockId;

private String stockCode;

private String stockName;

public Stock() { }

public Stock(String stockCode, String stockName) {

this.stockCode = stockCode;

this.stockName = stockName;

}

@Id

@GeneratedValue

@Column(name = "STOCK\_ID", unique = true, nullable = false)

public Integer getStockId() {return this.stockId; }

public void setStockId(Integer stockId) {this.stockId = stockId; }

@Column(name = "STOCK\_CODE", unique = true, nullable = false, length = 10)

public String getStockCode() {return this.stockCode; }

public void setStockCode(String stockCode) {this.stockCode = stockCode; }

@Column(name = "STOCK\_NAME", unique = true, nullable = false, length = 20)

public String getStockName() {return this.stockName;}

public void setStockName(String stockName) {this.stockName = stockName;}

@Override

public String toString() {

return "Stock [stockCode=" + stockCode + ", stockId=" + stockId

+ ", stockName=" + stockName + "]";

}}

**\*\*\*\*\*\*\*\*\*\*StockService.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hib3.service;

import training.spring.hib3.entity.Stock;

public interface StockService {

void save(Stock stock);

void update(Stock stock);

void delete(Stock stock);

Stock findByStockCode(String stockCode);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*StockServiceImpl.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hib3.service;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import training.spring.hib3.Dao.StockDao;

import training.spring.hib3.entity.Stock;

@Service("stockService")

@Transactional

public class StockServiceImpl implements StockService{

@Autowired

StockDao stockDao;

public void setStockDao(StockDao stockDao) {this.stockDao = stockDao; }

public void save(Stock stock){ stockDao.save(stock); }

public void update(Stock stock){stockDao.update(stock); }

public void delete(Stock stock){stockDao.delete(stock); }

public Stock findByStockCode(String stockCode){

return stockDao.findByStockCode(stockCode);

}}

**\*\*\*\*\*\*\*\*\*\*In Src Folder StockConfig1.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?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:tx="http://www.springframework.org/schema/tx"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx.xsd">

<context:component-scan base-package="training.spring.hib3" />

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />

<property name="url" value="jdbc:oracle:thin:@localhost:1521:XE" />

<property name="username" value="system" />

<property name="password" value="root" />

</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

<property name="dataSource">

<ref bean="dataSource" />

</property>

<property name="annotatedClasses" value="training.spring.hib3.entity.Stock" />

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.OracleDialect</prop>

<prop key="hibernate.hbm2ddl.auto">update</prop>

<!-- uncomment this for first time run-->

<prop key="hibernate.show\_sql">true</prop>

</props>

</property>

</bean>

<bean id="txMgr"

class="org.springframework.orm.hibernate3.HibernateTransactionManager">

<property name="sessionFactory" ref="sessionFactory" />

</bean>

<tx:annotation-driven transaction-manager="txMgr" />

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*StockAppClient.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hib3;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import training.spring.hib3.entity.Stock;

import training.spring.hib3.service.StockService;

public class StockClientApp

{

public static void main( String[] args )

{

ApplicationContext appContext =

new ClassPathXmlApplicationContext("stock-config1.xml");

StockService stockService = (StockService)appContext.getBean("stockService");

/\*\* insert \*\*/

Stock stock = new Stock();

stock.setStockCode("8343");

stock.setStockName("Jweahller");

stockService.save(stock);

Stock stock1 = new Stock();

stock1.setStockCode("876224");

stock1.setStockName("Elssjjectro");

stockService.save(stock1);

/\*\* select \*\*/

Stock stock2 = stockService.findByStockCode("7668");

System.out.println(stock2);

/\*\* update \*\*/

stock2.setStockName("HAIO-1");

stockService.update(stock2);

/\*\* delete \*\*/

//stockService.delete(stock2);

System.out.println("Done");

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*UserDao.Java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hibernateTemplate.Dao;

import java.util.List;

import training.spring.hibernateTemplate.entity.User;

public interface UserDao {

public void saveUser(User user);

public List<User> getAllUser(User user);

public User selectUserById(int userId);

public void deleteUser(User user);

}

**\*\*\*\*\*\*\*UserDaoImpl.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hibernateTemplate.Dao;

import java.util.List;

import org.hibernate.SessionFactory;

import org.hibernate.classic.Session;

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

import org.springframework.orm.hibernate3.HibernateTemplate;

import org.springframework.stereotype.Repository;

import training.spring.hibernateTemplate.entity.User;

@Repository("userDao")

public class UserDaoImpl implements UserDao {

private SessionFactory sessionFactory;

private HibernateTemplate hibernateTemplate;

@Autowired

public UserDaoImpl(SessionFactory sessionFactory) {

this.sessionFactory = sessionFactory;

hibernateTemplate = new HibernateTemplate(sessionFactory);

}

public void saveUser(User user) {

hibernateTemplate.saveOrUpdate(user);

}

public void deleteUser(User user) {

hibernateTemplate.delete(user);

}

@SuppressWarnings("unchecked")

public List<User> getAllUser(User user) {

return (List<User>) hibernateTemplate.find("from "

+ User.class.getName());

}

public User selectUserById(int userId) {

return hibernateTemplate.get(User.class, userId);

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*User.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hibernateTemplate.entity;

import java.io.Serializable;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.Id;

import javax.persistence.Table;

//import org.hibernate.annotations.GenericGenerator;

@Entity

@Table(name = "usertable")

public class User implements Serializable {

private static final long serialVersionUID = 8496087166198616020L;

private int userId;

private String userName;

private Integer age;

private Boolean registered;

@Id

@GeneratedValue

@Column(name = "userId")

public int getUserId() { return userId; }

public void setUserId(int userId) { this.userId = userId; }

@Column(name = "userName", nullable=false)

public String getUserName() { return userName; }

public void setUserName(String userName) { this.userName = userName; }

@Column(name = "age") public Integer getAge() { return age; }

public void setAge(Integer age) { this.age = age; }

@Column(name = "registered", nullable=false)

public Boolean getRegistered() { return registered; }

public void setRegistered(Boolean registered) { this.registered = registered; }

}

**\*\*\*\*\*\*\*\*\*\*In Src folder JdbcConfig.properties\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

jdbc.driver.className=oracle.jdbc.driver.OracleDriver

jdbc.url=jdbc:oracle:thin:@localhost:1521:XE

jdbc.username=system

jdbc.password=root

jdbc.hibernate.dialect=org.hibernate.dialect.OracleDialect

**\*\*\*\*\*\*\*\*\*\*\*UserClientApp.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package training.spring.hibernateTemplate;

import java.util.Iterator;

import java.util.List;

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

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import training.spring.hibernateTemplate.Dao.UserDaoImpl;

import training.spring.hibernateTemplate.entity.User;

public class UserClientApp {

static UserDaoImpl userDao;

public static void main(String[] args) throws Exception {

ApplicationContext appContext = new ClassPathXmlApplicationContext(

"user-config.xml");

userDao = (UserDaoImpl) appContext.getBean("userDao");

/\* insert \*/

System.out.println("----------- Saving User object -------------");

saveUser();

/\* retrieve all \*/

System.out.println("----------- Retrieving all users -------------");

retrieveAllUsers();

/\* retrieve \*/

System.out.println("----------- Retrieving user with id 1 -------------");

retrieveUser(1);

/\* update \*/

System.out.println("----------- Modifying user with id 1 -------------");

updateUser(1);

/\* delete \*/

System.out.println("----------- deleting user with id 2 -------------");

deleteUser(2);

System.out.println("----------- Retrieving all users -------------");

retrieveAllUsers();

}

public static void saveUser() {

User user1 = new User();

user1.setAge(23);

user1.setUserName("Aditya");

user1.setRegistered(true);

userDao.saveUser(user1);

User user2 = new User();

user2.setAge(18);

user2.setUserName("Soha");

user2.setRegistered(false);

userDao.saveUser(user2);

}

public static void retrieveAllUsers() {

List<User> userList = userDao.getAllUser(new User());

Iterator<User> it = userList.iterator();

while (it.hasNext()) {

User user = (User) it.next();

System.out.println("UserID : " + user.getUserId()+ " User Name : " + user.getUserName()+ " User Age : " + user.getAge());

} }

public static void retrieveUser(int userid) {

User user = userDao.selectUserById(userid);

System.out.println("User Name : " + user.getUserName()

+ " User Age : " + user.getAge());

}

public static void updateUser(int userid) {

User user = userDao.selectUserById(userid);

user.setUserName("Kumar");

userDao.saveUser(user);

User userChanged = userDao.selectUserById(userid);

System.out.println("User Name : " + userChanged.getUserName()

+ " User Age : " + userChanged.getUserName());

}

public static void deleteUser(int userid) {

User userToDelete = userDao.selectUserById(userid);

userDao.deleteUser(userToDelete);

}

}

**\*\*\*\*\*\*\*\*\*\*\*In Src Folder UserConfig.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?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/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="training.spring.hibernateTemplate" />

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />

<property name="url" value="jdbc:oracle:thin:@localhost:1521:XE" />

<property name="username" value="system" />

<property name="password" value="root" />

</bean>

<bean id="sessionFactory" class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">

<property name="dataSource">

<ref bean="dataSource" />

</property>

<property name="annotatedClasses"

value="training.spring.hibernateTemplate.entity.User" />

<property name="hibernateProperties">

<props>

<prop key="hibernate.dialect">org.hibernate.dialect.OracleDialect</prop>

<prop key="hibernate.hbm2ddl.auto">create</prop>

<!-- uncomment this for first time run-->

<prop key="hibernate.show\_sql">false</prop>

</props>

</property>

</bean>

<bean id="hibernateTransactionManager"

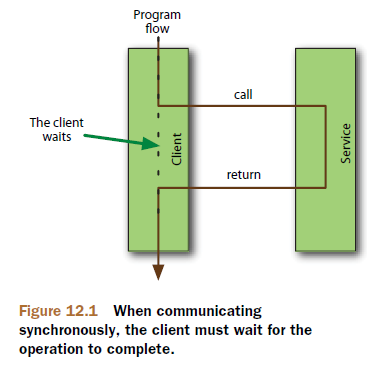
class="org.springframework.orm.hibernate3.HibernateTransactionManager">

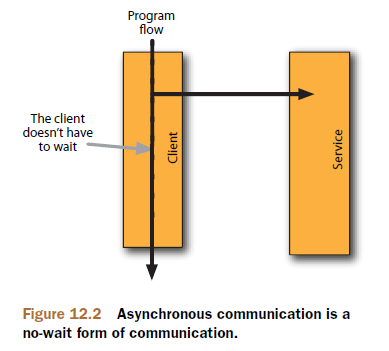
<property name="sessionFactory" ref="sessionFactory" />

</bean></beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Jms Notes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* Remoting options such as RMI and Hessian/Burlap are synchronous.
* when the client invokes a remote method, the client must wait for the method to complete before moving on. Even if the remote method doesn’t return anything back to the client, the client will be put on hold until the service is done.
* JMS, on the other hand, provides asynchronous communication between applications. When messages are sent asynchronously, as shown in figure 12.2, the client doesn’t have to wait for the service to process the message or even for the message to be delivered. The cli-ent sends its message and then moves along with the assumption that the ser-vice will eventually receive and process the message.





**Architecting JMS**

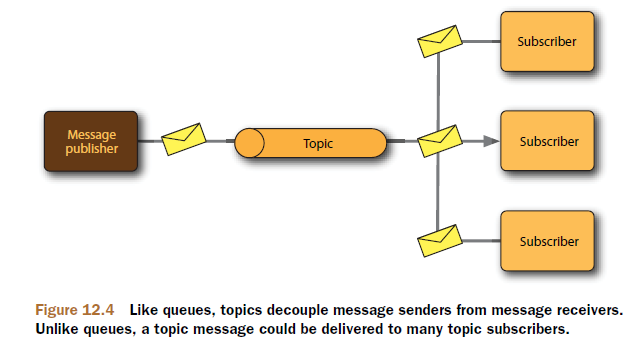
* The key to the postal service is indirection. When Grandma’s birthday comes around, it’d be inconvenient if we had to deliver a card directly to her. Depending on where she lives, we’d have to set aside anywhere from a few hours to a few days to deliver a birthday card. Fortunately, the postal service will deliver the card to her while we go about our lives.
* Similarly, indirection is the key to JMS. When one application sends information to another through JMS, there’s no direct link between the two applications. Instead, the sending application places the message in the hands of a service that will ensure deliv-ery to the receiving application.
* There are two main actors in JMS: message brokers and destinations.
* When an application sends a message, it hands it off to a message broker. A mes-sage broker is JMS’s analog of the post office. The message broker will ensure that the message is delivered to the specified destination, leaving the sender free to go about other business.
* When you send a letter through the mail, it’s important to address it so that the postal service knows where it should be delivered. Likewise, in JMS, messages are addressed with a destination. Destinations are like mailboxes where the messages are placed until someone comes to pick them up.
* But unlike mail addresses, which may indicate a specific person or street address, destinations are less specific. Destinations are only concerned about where the message will be picked up—not who will pick them up. In this way, destinations are like sending a letter addressed, “To current resident.”
* In JMS, there are two types of destination: queues and topics. Each of these is asso-ciated with a specific messaging model, either point-to-point (for queues) or publish-subscribe (for topics).

**POINT-TO-POINT MESSAGING**

* In the point-to-point model, each message has exactly one sender and one receiver, as illustrated in figure 12.3. When the message broker is given a message, it places the message in a queue. When a receiver comes along and asks for the next message in the queue, the message is pulled from the queue and delivered to the receiver. Because the message is removed from the queue as it’s delivered, it’s guaranteed that the message will be delivered to only one receiver.
* Although each message in a message queue is delivered to only one receiver, this doesn’t imply that only one receiver is pulling messages from the queue. In fact, it’s likely that several receivers are processing messages from the queue. But they’ll each be given their own messages to process.
* This is analogous to waiting in line at the bank. As you wait, you may notice that multiple tellers are available to help you with your financial transaction. After each customer is helped and a teller is freed up, she will call for the next person in line. When it’s your turn at the front of the line, you’ll be called to the counter and helped by one teller. The other tellers will help other banking customers.
* Another observation to be made at the bank is that when you get in line, you prob-ably won’t know which teller will eventually help you. You could count how many peo-ple are in line, match that up with the number of available tellers, note which teller is fastest, and then come up with a guess as to which teller will call you to their window. But chances are you’ll be wrong and end up at a different teller’s window.
* Likewise, in JMS, if multiple receivers are listening to a queue, there’s no way of knowing which one will actually process a specific message. This uncertainty is a good thing because it enables an application to scale up message processing by simply add-ing another listener to the queue.

**PUBLISH-SUBSCRIBE MESSAGING**

* In the publish-subscribe messaging model, messages are sent to a topic. As with queues, many receivers may be listening to a topic. But unlike queues, where a mes-sage is delivered to exactly one receiver, all subscribers to a topic will receive a copy of the message,



* As you may have guessed from its name, the publish-subscribe message model is much like the model of a magazine publisher and its subscribers. The magazine (a message) is published, sent to the postal service, and then all subscribers receive their own copy.
* The magazine analogy breaks down when you realize that in JMS, the publisher has no idea of who its subscribers are. The publisher only knows that its message will be pub-lished to a particular topic—not who’s listening to that topic. This also implies that the publisher has no idea of how the message will be processed.

**Setting up a message broker in Spring**

* ActiveMQ is a great open source message broker and a wonderful option for asynchro-nous messaging with JMS. As I’m writing this, the current version of ActiveMQ is 5.4.2.
* To get started with ActiveMQ, you’ll need to download the binary distribution from http://activemq.apache.org. Once you’ve downloaded ActiveMQ, unzip it to your local hard drive. In the lib directory of the unzipped distribution, you’ll find activemq-core-5.4.2.jar. This is the JAR file you’ll need to add to the application’s classpath to be able to use ActiveMQ’s API.
* Under the bin directory, you’ll find several subdirectories for various operating sys-tems. Within those, you’ll find scripts that you can use to start ActiveMQ. For example, to start ActiveMQ on Mac OS X, run activemq start from the bin/macosx directory. Within moments ActiveMQ will be ready and waiting to broker your messages.

**Creating a connection factory**

* Throughout this chapter, we’re going to see different ways that Spring can be used to both send and receive messages through JMS. In all cases, we’ll need a JMS connection factory to be able to send messages through the message broker.
* Since we’re using ActiveMQ as our message broker, we’ll have to configure the JMS connection factory so that it knows how to connect to ActiveMQ.
* ActiveMQConnectionFactory is the JMS connection factory that comes with ActiveMQ, and it’s configured in Spring like this:

<bean id="connectionFactory" class="org.apache.activemq.spring.ActiveMQConnectionFactory">

<property name="brokerURL" value="tcp://localhost:61616"/>

</bean>

* Optionally, since we know that we’re dealing with ActiveMQ, we can use ActiveMQ’s own Spring configuration namespace (available with all versions of ActiveMQ since version 4.1) to declare the connection factory.
* First, be sure to declare the amq namespace in the Spring configuration XML file:

**Declaring an ActiveMQ message destination**

* In addition to a connection factory, we’ll need a destination for the messages to be passed along to.
* The destination can be either a queue or a topic, depending on the needs of the application.
* Regardless of whether you’re using a queue or a topic, you must configure the des-tination bean in Spring using a message broker–specific implementation class.
* For example, the following <bean> declaration declares an ActiveMQ queue:

<bean id="queue" class="org.apache.activemq.command.ActiveMQQueue"> <constructor-arg value="spitter.queue"/>

</bean>

* Similarly, the following <bean> declares a topic for ActiveMQ:

<bean id="topic" class="org.apache.activemq.command.ActiveMQTopic"> <constructor-arg value="spitter.topic"/> </bean>

* In either case, the <constructor-arg> specifies the name of the queue, as it’s known to the message broker—spitter.topic in this case.
* As with the connection factory, the ActiveMQ namespace offers an alternative way to declare queues and topics.
* For queues, we could also use the <amq:queue> element:

<amq:queue id="queue" physicalName="spitter.queue" />

Or, if it’s a JMS topic that’s in order, use the <amq:topic>:

<amq:topic id="topic" physicalName="spitter.topic" />

Either way, the physicalName attribute sets the name of the message channel.

* At this point we’ve seen how to declare the essential components of working with JMS, whether you’re sending or receiving messages

**Using Spring’s JMS template**

* As you’ve seen, JMS gives Java developers a standard API for interacting with message brokers and for sending and receiving messages.
* Furthermore, virtually every message broker implementation out there supports JMS.
* So there’s no reason to learn a propri-etary messaging API for every message broker you deal with.
* But though JMS offers a universal interface to all message brokers, its convenience comes at a cost.
* Sending and receiving messages with JMS isn’t a simple matter of lick-ing a stamp and placing it on an envelope. As you’ll see, JMS demands that you also (figuratively) fuel up the mail carrier’s truck.

**Working with JMS templates**

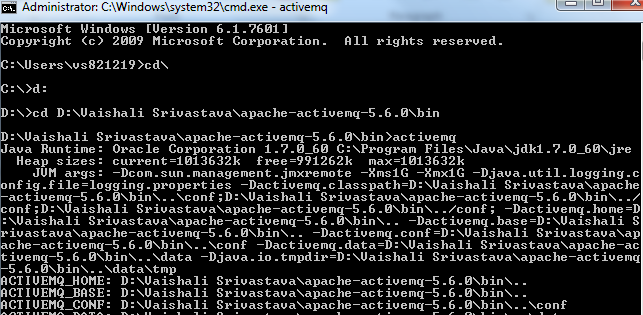
* JmsTemplate is Spring’s answer to verbose and repetitive JMS code.
* JmsTemplate takes care of creating a connection, obtaining a session, and ultimately sending or receiving messages.
* This leaves you to focus your development efforts on constructing the mes-sage to send or processing the messages that are received.
* What’s more, JmsTemplate can handle any clumsy JMSException that may be thrown along the way.

**WIRING A JMS TEMPLATE**

* To use JmsTemplate, we’ll need to declare it as a bean in the Spring configuration file. The following XML should do the trick:
* <bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"> <property name="connectionFactory" ref="connectionFactory" /> </bean>
* Because JmsTemplate needs to know how to get connections to the message broker, we must set the connectionFactory property with a reference to the bean that imple-ments JMS’s ConnectionFactory interface

**\*\*\*Spring JMS Demos\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

Extract Apache Active MQ in Any Folder



**Type following url in Browser\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<http://localhost:8161/admin/>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*JmsMessageConsumer.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jms;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.MessageListener;

import javax.jms.TextMessage;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.stereotype.Component;

@Component("messageListener")

public class JMSMessageConsumer implements MessageListener {

public JMSMessageConsumer() {

super();

System.out.println("in msg consumer");

}

private static final Logger logger = LoggerFactory

.getLogger(JMSMessageConsumer.class);

public void onMessage(Message message) {

System.out.println("in onMessage()");

try {

int messageCount = message

.getIntProperty(JMSMessageProducer.MESSAGE\_COUNT);

if (message instanceof TextMessage) {

TextMessage tm = (TextMessage) message;

String msg = tm.getText();

System.out.println("Processed message : " + msg

+ " " + messageCount);

logger.info("Processed message : ", msg, messageCount);

}

} catch (JMSException e) {

logger.error(e.getMessage(), e);

}}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*JmsMessageProducer.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jms;

import javax.annotation.PostConstruct;

import javax.jms.JMSException;

import javax.jms.Message;

import javax.jms.Session;

import javax.jms.TextMessage;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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

import org.springframework.jms.core.JmsTemplate;

import org.springframework.jms.core.MessageCreator;

import org.springframework.stereotype.Component;

@Component("messageProducer")

public class JMSMessageProducer {

public JMSMessageProducer()

{

super();

System.out.println("in producer constructor");

}

private static final Logger logger = LoggerFactory.getLogger(JMSMessageProducer.class);

@Autowired

private JmsTemplate template = null;

private int messageCount = 5;

static String MESSAGE\_COUNT="Message\_Count";

@PostConstruct

public void sendMessages() throws JMSException

{

System.out.println("in sendMessages()");

for (int i = 0; i < messageCount; i++) {

final int index = i;

final String text = "Message number is " + i + ".";

template.send(new MessageCreator()

{

public Message createMessage(Session session) throws JMSException

{

TextMessage message = session.createTextMessage(text);

message.setIntProperty(MESSAGE\_COUNT, index);

System.out.println("Sending message: " + text);

//System.out.println("message: " + message);

logger.info("Sending message: " + text);

return message;

}

});

} }}

**\*\*\*\*\*\*\*\*\*\*\*\*\*RumMessageproducer.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jms;

import javax.jms.JMSException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

/\*\*

\* This class drives the example from the producer side. It loads the Spring

\* {@link ApplicationContext} and sends messages. The entire configuration for

\* this app is held in <tt>src/main/resources/jms-context.xml</tt>.

\*/

public class RunMsgProducer {

private static final Logger logger = LoggerFactory

.getLogger(RunMsgProducer.class);

public static void main(String[] args) throws JMSException {

ApplicationContext context = new ClassPathXmlApplicationContext(

"message-context.xml", RunMsgProducer.class);

JMSMessageProducer producer = (JMSMessageProducer) context

.getBean("messageProducer");

try {

producer.sendMessages();

} catch (Exception e) {

e.printStackTrace();

}}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Message-Context.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?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:p="http://www.springframework.org/schema/p"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:jms="http://www.springframework.org/schema/jms"

xmlns:amq="http://activemq.apache.org/schema/core"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/jms

http://www.springframework.org/schema/jms/spring-jms.xsd

http://activemq.apache.org/schema/core

http://activemq.apache.org/schema/core/activemq-core.xsd">

<context:component-scan base-package="com.igatepatni.jms" />

<!-- For any message broker configuring connection factory use this tag. -->

<bean id="connectionFactory" class="org.apache.activemq.ActiveMQConnectionFactory"

p:brokerURL="tcp://localhost:61616" />

<!-- for only active mq as message broker use following tag

<amq:connectionFactory id="connectionFactory" brokerURL="tcp://localhost:61616"/> -->

<bean id="destination" class="org.apache.activemq.command.ActiveMQQueue">

<constructor-arg value="com.igatepatni.myqueue" />

</bean>

<bean id="jmsTemplate" class="org.springframework.jms.core.JmsTemplate"

p:connectionFactory-ref="connectionFactory"

p:defaultDestination-ref="destination" />

<!--<bean id="messageProducer" class="com.igatepatni.jms.JMSMessageProducer"

p:jmsTemplate-ref="jmsTemplate" />

<bean id="messageListener" class="com.igatepatni.jms.JMSMessageConsumer" />

-->

<!-- This tag configures the Message consumer -->

<jms:listener-container concurrency="5-10">

<jms:listener destination="com.igatepatni.myqueue" ref="messageListener"/>

</jms:listener-container>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*log4j.properties\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

#loggging file details for Log4jProperties

#Sat Oct 22 11:09:26 IST 2011

#log4j.rootCategory=INFO,fileAppender

#log4j.appender.fileAppender=org.apache.log4j.FileAppender

#log4j.appender.fileAppender.File=d:\\spring

#log4j.appender.fileAppender.layout.ConversionPattern=%-4d %-C : %M - %m%n

#log4j.appender.fileAppender.layout=org.apache.log4j.PatternLayout

log4j.rootLogger=debug, myAppender

log4j.appender.myAppender=org.apache.log4j.FileAppender

log4j.appender.myAppender.File=message.log

log4j.appender.myAppender.layout=org.apache.log4j.SimpleLayout

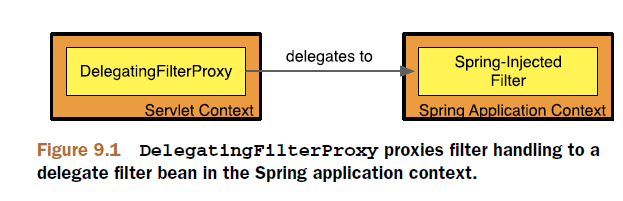
**\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Security Notes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Introducing Spring Security**

Spring Security is a security framework that provides declarative security for your Spring-based applications. Spring Security provides a comprehensive security solu-tion, handling authentication and authorization at both the web request level and at the method invocation level. Based on the Spring Framework, Spring Security takes full advantage of dependency injection (DI) and aspect-oriented techniques.

**Securing web requests**

* To get started with web security in Spring, we must set up the servlet filters that provide the various security features.

****

**Configuring minimal web security**

* The <http> element automatically sets up a FilterChainProxy (which is delegated to by the DelegatingFilterProxy we configured in web.xml) and all of the filter beans in the chain.
* In addition to those filter beans, we also get a few more freebies by setting the auto-config attribute to true.
* Autoconfiguration gives our application a free login page, support for HTTP Basic authentication, and support for logging out.
* In fact, set-ting auto-config to true is equivalent to explicitly asking for those features like this:

<http>

<form-login />

<http-basic />

<logout />

<intercept-url pattern="/\*\*" access="ROLE\_SPITTER" />

</http>

**LOGGING OUT**

* The <logout> element sets up a Spring Security filter that will invalidate a user ses-sion. When used as is, the filter set up by <logout> is mapped to /j\_spring\_security\_ logout.
* But so that this doesn’t collide with how we’ve set up DispatcherServlet, we need to override the filter’s URL much as we did for the login form.
* To do that, we need to set the logout-url attribute: <logout logout-url="/static/j\_spring\_security\_logout"/>

**Intercepting requests**

* IThe <intercept-url> element is the first line of defense in the request-level secu-rity game.
* Its pattern attribute is given a URL pattern that will be matched against incoming requests. If any requests match the pattern, then that <intercept-url>’s security rules will be applied.

<intercept-url pattern="/\*\*" access="ROLE\_SPITTER" />

* we’ve set the pattern attribute to /\*\*, indicating that we want all requests, regardless of the URL, to require ROLE\_SPITTER access.
* The /\*\* has a broad reach, but you can be more specific. Suppose that some special areas of the Spitter application are restricted to admin-istrative users. For that, we can insert the following <intercept-url>
* <intercept-url pattern="/admin/\*\*" access="ROLE\_ADMIN"

**Authenticating users**

* Every application’s a little different. That truth is evident in how every application stores user information. Sometimes it’s kept in a relational database. Other times it might be in an LDAP-enabled directory.
* Some applications rely on a decentralized user identity system. And some may employ more than one strategy. Fortunately, Spring Security is flexible and can handle almost any authentication strategy you need. Spring Security is prepared to cover many common authentication scenarios, including authenticating users against  In-memory (Spring-configured) user repositories
* JDBC-based user repositories
* LDAP-based user repositories
* OpenID decentralized user identity systems
* Central Authentication System (CAS)
* X.509 certificates
* JAAS-based providers

**Configuring an in-memory user repository**

* One of the easiest authentication options available is to declare the user details directly in the Spring configuration. This is done by creating a user service using the <user-service> element from Spring Security’s XML namespace:

<user-service id="userService">

<user name="habuma" password="letmein" authorities="ROLE\_SPITTER,ROLE\_ADMIN"/>

<user name="twoqubed" password="longhorns" authorities="ROLE\_SPITTER"/>

<user name="admin" password="admin" authorities="ROLE\_ADMIN"/>

</user-service>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring-Web Security Demo\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

[http://localhost:8081/SpringWebSecurityProject/\*.doc](http://localhost:8081/SpringWebSecurityProject/*.doc)

<http://localhost:8081/SpringWebSecurityProject/welcome>

<http://localhost:8081/SpringWebSecurityProject/admin>

<http://localhost:8081/SpringWebSecurityProject/trainer/vaishali>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*project/WEB-INF/mvc-dispatcher-servlet.xml\*\*\*\*\*\*\*\*\*\***

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="com.igatepatni.security" />

<bean

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="prefix">

<value>/pages/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*/project/web-in/spring-security.xml\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/security"

xmlns:beans="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.1.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security-3.0.xsd">

<!-- use this attribute for hasRoleMethod

<http auto-config="true" use-expressions="true" >

-->

<http auto-config="true" >

<intercept-url pattern="/welcome\*" access="ROLE\_USER" />

<intercept-url pattern="/admin\*" access="ROLE\_ADMIN" />

<intercept-url pattern="/\*.doc" access="ROLE\_ADMIN" />

<!--

<intercept-url pattern="/trainer/\*\*" access="hasRole('TRAINER\_ROLE')" /> -->

<logout logout-url="/static/j\_spring\_security\_logout"/>

</http>

<authentication-manager>

<authentication-provider>

<user-service>

<user name="uma" password="123456" authorities="ROLE\_USER" />

<user name="admin" password="admin" authorities="ROLE\_ADMIN,ROLE\_USER" />

<user name="vaishali" password="vaishali123" authorities="TRAINER\_ROLE" />

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

**\*\*\*\*\*\*\*\*\*\*\*\*web.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp\_ID" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee

http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>Spring MVC Application</display-name>

<!-- Spring MVC -->

<servlet>

<servlet-name>mvc-dispatcher</servlet-name>

<servlet-class>

org.springframework.web.servlet.DispatcherServlet

</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>mvc-dispatcher</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<listener>

<listener-class>

org.springframework.web.context.ContextLoaderListener

</listener-class>

</listener>

<context-param>

<param-name>contextConfigLocation</param-name>

<param-value>

/WEB-INF/mvc-dispatcher-servlet.xml,

/WEB-INF/spring-security.xml

</param-value>

</context-param>

<!-- Spring Security -->

<filter>

<filter-name>springSecurityFilterChain</filter-name>

<filter-class>

org.springframework.web.filter.DelegatingFilterProxy

</filter-class>

</filter>

<filter-mapping>

<filter-name>springSecurityFilterChain</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<welcome-file-list>

<welcome-file>index.jsp</welcome-file>

</welcome-file-list>

</web-app>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*HelloController.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.security;

import org.springframework.stereotype.Controller;

import org.springframework.ui.ModelMap;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

@Controller

//@RequestMapping("/welcome")

public class HelloController {

@RequestMapping(value="/welcome", method = RequestMethod.GET)

public String printWelcome(ModelMap model) {

model.addAttribute("message", "Spring Security Hello World");

return "hello";

}

@RequestMapping(value="/admin", method = RequestMethod.GET)

public String showAdminPage(ModelMap model) {

model.addAttribute("message", "Spring Security Admin page - Welcome");

return "adminpage";

}

@RequestMapping(value = "/", method = RequestMethod.GET)

public String defaultPage(ModelMap map) {

System.out.println("In default home redirect to Page....................");

return "redirect:/home";

}

@RequestMapping(value = "/home", method = RequestMethod.GET)

public String displayHomePage(ModelMap map) {

System.out.println("In home Page....................");

return "home";

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Webcontent/pages/adminpages.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<h1>Message : ${message}</h1>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Webcontent/pages/hello.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Hello World</title>

</head>

<body>

<h1>Message : ${message}</h1>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Webcontent/pages/home.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

Home page

</body></html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Webcontent/pages/index\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

Welcome

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Method Security\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Securing methods**

* security is an aspect-oriented concept. And Spring AOP is the basis for method-level security in Spring Security.
* All of the AOP involved in securing methods is packed into a single element:

<global-method-security>. Here’s a com-mon way of using <global-method-security>

<global-method-security secured-annotations="enabled" />

* This sets up Spring Security for securing methods that are annotated with Spring Security’s own @Secured annotation.
* This is just one of four ways that Spring Security supports method-level security

Methods annotated with @Secured

Methods annotated with JSR-250’s @RolesAllowed

Methods annotated with Spring’s pre- and post-invocation annotations

Methods matching one or more explicitly declared pointcuts

**Securing methods with @Secured**

When <global-method-security> is configured with its secured-annotations attri-bute set to enabled, a pointcut is created such that the Spring Security aspects will wrap bean methods that are annotated with @Secured. For example:

Secured( { "ROLE\_SECRET\_AGENT" })

public String getSecuredData()

{

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Method Security\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\* SecuredObject .java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.springsecurity.annotation;

import org.springframework.security.access.annotation.Secured;

public class SecuredObject

{

@Secured( { "ROLE\_SECRET\_AGENT" })

public String getSecuredData()

{

return "Top-Secret Data";

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SpringSecurityTest.java\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.springsecurity.annotation;

import static org.junit.Assert.\*;

import org.junit.After;

import org.junit.Test;

import org.junit.runner.RunWith;

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

import org.springframework.context.ApplicationContext;

import org.springframework.security.access.AccessDeniedException;

import org.springframework.security.authentication.AuthenticationCredentialsNotFoundException;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.test.context.ContextConfiguration;

import org.springframework.test.context.junit4.SpringJUnit4ClassRunner;

@RunWith(SpringJUnit4ClassRunner.class)

@ContextConfiguration("SpringSecurityTest-context.xml")

public class SpringSecurityTest

{

@Autowired

private ApplicationContext context;

@Test(expected = AuthenticationCredentialsNotFoundException.class)

public void shouldRejectAccessForUnauthenticatedUser()

{

SecuredObject secured = (SecuredObject) context.getBean("securedBean");

secured.getSecuredData();

}

@Test(expected = AccessDeniedException.class)

public void shouldRejectAccessForUnauthorizedUser()

{

authenticateUser("intruder", "password");

SecuredObject secured = (SecuredObject) context.getBean("securedBean");

secured.getSecuredData();

}

@Test

public void shouldAllowAccessForAuthorizedUser()

{

authenticateUser("training", "secret");

SecuredObject secured = (SecuredObject) context.getBean("securedBean");

assertEquals("Top-Secret Data", secured.getSecuredData());

}

private void authenticateUser(String username, String password)

{

UsernamePasswordAuthenticationToken login = new UsernamePasswordAuthenticationToken(

username, password);

AuthenticationManager authManager = (AuthenticationManager) context

.getBean("authenticationManager");

SecurityContextHolder.getContext().setAuthentication(

authManager.authenticate(login));

}

@After

public void tearDown()

{

SecurityContextHolder.getContext().setAuthentication(null);

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*src/Security-Context.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<beans:beans xmlns="http://www.springframework.org/schema/security"

xmlns:beans="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/security

http://www.springframework.org/schema/security/spring-security-3.0.xsd">

<beans:bean id="securedBean" class="com.igatepatni.springsecurity.annotation.SecuredObject" />

<global-method-security secured-annotations="enabled" />

<authentication-manager alias="authenticationManager">

<authentication-provider>

<user-service>

<user password="secret" name="training" authorities="ROLE\_SECRET\_AGENT"/>

<user password="password" name="intruder" authorities="ROLE\_ANONYMOUS"/>

</user-service>

</authentication-provider>

</authentication-manager>

</beans:beans>

**\*\*\*\*\*\*Spring OXM Using Castor Framework\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Employee.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.castor;

public class Employee {

private String id;

private String name;

private double salary;

public Employee() {}

public Employee(String id, String name, double salary) {

this.id = id;

this.name = name;

this.salary = salary;

}

public String getId() {return id; }

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

public String getName() { return name; }

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

public double getSalary() { return salary; }

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

}

**\*\*\*\*\*\*\*\*\*Employees.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.castor;

import java.util.List;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

public class Employees {

List<Employee> employees;

public List<Employee> getEmployees() {return employees; }

public void setEmployees(List<Employee> employees) {

this.employees = employees;

}

}

**\*\*\*\*\*\*\*\*\*\*\*OXMCastorTest.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.castor;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.util.ArrayList;

import javax.xml.transform.stream.StreamResult;

import javax.xml.transform.stream.StreamSource;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.oxm.castor.CastorMarshaller;

public class OxmCastorTest

{

public static void main(String[] args) throws Exception {

ApplicationContext context = new ClassPathXmlApplicationContext("castor-context.xml");

CastorMarshaller marshaller = (CastorMarshaller)context.getBean("castorMarshaller");

testObjectToXml(marshaller);

// testXmlToObject(marshaller);

}

private static void testObjectToXml(CastorMarshaller marshaller) throws Exception{

Employees emps=new Employees();

Employee empObject = new Employee("12345", "Shrjjilata", 2000.00);

Employee empObject2 = new Employee("222","Vaishali", 1000.00);

Employee empObject3 = new Employee("444", "Anju", 2000.00);

ArrayList empList=new ArrayList();

empList.add(empObject);

empList.add(empObject2);

empList.add(empObject3);

emps.setEmployees(empList);

FileOutputStream outputStream = new FileOutputStream(new File("castor-employee.xml"));

StreamResult xmlFileResult = new StreamResult(outputStream);

marshaller.marshal(emps, xmlFileResult);

}

/\*private static void testXmlToObject(CastorMarshaller marshaller) throws Exception{

FileInputStream inputStream = new FileInputStream(new File("mobile.xml"));

StreamSource xmlFileSource = new StreamSource(inputStream);

marshaller.setTargetClass(Mobile.class);

Mobile mobileObject = (Mobile)marshaller.unmarshal(xmlFileSource);

System.out.println("Name is " + mobileObject.getName());

System.out.println("Model is " + mobileObject.getModel());

System.out.println("Price is " + mobileObject.getPrice());

}\*/

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Castor-Context.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<beans default-autowire="no"

xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id = "castorMarshaller"

class = "org.springframework.oxm.castor.CastorMarshaller">

<property name="mappingLocation"

value="classpath:castor-mapping.xml" />

</bean>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Castor-Mapping.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0"?>

<mapping>

<description>Description of the mapping</description>

<class name="com.igatepatni.castor.Employees">

<map-to xml="Employees"/>

<field name="employees" type="com.igatepatni.castor.Employee" collection="arraylist">

<bind-xml name="emp" />

</field>

</class>

<class name="com.igatepatni.castor.Employee">

<map-to xml="Employee"/>

<field name="id" type="string">

<bind-xml name="empid" node="attribute"/>

</field>

<field name="name" type="string">

<bind-xml name="empname" node="element"/>

</field>

<field name="salary" type="double">

<bind-xml name="salary" node="element"/>

</field>

</class>

</mapping>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Cator-employee.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<Employees>

<emp empid="12345">

<empname>Shrjjilata</empname>

<salary>2000.0</salary>

</emp>

<emp empid="222">

<empname>Vaishali</empname>

<salary>1000.0</salary>

</emp>

<emp empid="444">

<empname>Anju</empname>

<salary>2000.0</salary>

</emp>

</Employees>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*SpringOXM Using Jaxb\*\*\*\*\*\*\*\*\*\*\***

**Department.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jaxb;

import javax.xml.bind.annotation.XmlAttribute;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "department")

public class Department {

private String id;

private String name;

@XmlAttribute(name = "deptId")

public String getId() { return id; }

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

@XmlElement(name = "deptName")

public String getName() { return name; }

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

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Employee.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jaxb;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "employee")

public class Employee {

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + ", department="

+ department + "]";

}

private String id;

private String name;

private Department department;

public Employee() {}

public Employee(String id, String name){

this.id = id;

this.name = name;

}

@XmlElement(name = "empId")

public String getId() { return id; }

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

@XmlElement(name = "empName") public String getName() { return name; }

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

@XmlElement(name = "department")

public Department getDepartment() {return department; }

public void setDepartment(Department department) {this.department = department;}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Employees.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jaxb;

import java.util.List;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "employees")

public class Employees {

List<Employee> employees;

@XmlElement(name = "employee")

public List<Employee> getEmployees() { return employees; }

public void setEmployees(List<Employee> employees) {this.employees = employees;}

}

**\*\*\*\*\*\*\*\*\*\*\*OXMJaxbTest.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jaxb;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

import javax.xml.bind.Marshaller;

import javax.xml.transform.stream.StreamResult;

import javax.xml.transform.stream.StreamSource;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.oxm.jaxb.Jaxb2Marshaller;

public class OxmJaxbTest {

public static void main(String[] args) throws Exception {

ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");

Jaxb2Marshaller jaxbMarshaller = (Jaxb2Marshaller)context.getBean("jaxbMarshaller");

Map<String, Boolean> marshallerProperties = new HashMap<String, Boolean>();

marshallerProperties.put(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

jaxbMarshaller.setMarshallerProperties(marshallerProperties);

marshalObject(jaxbMarshaller);

unmarshallXML(jaxbMarshaller);

}

private static void marshalObject(Jaxb2Marshaller jaxbMarshaller) throws Exception{

Department department = new Department();

department.setId("IT");

department.setName("IT Department");

Employee employee = new Employee("12345", "Vaishali");

employee.setDepartment(department);

Department department2 = new Department();

department2.setId("Act");

department2.setName("Acout Department");

Employee employee2 = new Employee("1111", "Priti");

employee2.setDepartment(department2);

Employees emps=new Employees();

ArrayList empList=new ArrayList();

empList.add(employee);

empList.add(employee2);

emps.setEmployees(empList);

FileOutputStream outputStream = new FileOutputStream(new File("employee.xml"));

StreamResult result = new StreamResult(outputStream);

jaxbMarshaller.marshal(emps, result);

}

private static void unmarshallXML(Jaxb2Marshaller jaxbMarshaller) throws Exception{

FileInputStream inputStream = new FileInputStream(new File("employee.xml"));

StreamSource source = new StreamSource(inputStream);

Employees employees = (Employees)jaxbMarshaller.unmarshal(source);

List<Employee> employeeList=employees.getEmployees();

System.out.println("Emp Id is " + employeeList.get(0).getId());

//System.out.println("Emp Name is " + employee.getName());

//Department department = employee.getDepartment();

//System.out.println("Dept Id is " + department.getId());

//System.out.println("Dept Name is " + department.getName());

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*OxmJaxbTest1.java\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.jaxb;

package com.igatepatni.jaxb;

import java.io.File;

import java.io.FileInputStream;

import java.io.FileOutputStream;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import javax.xml.bind.Marshaller;

import javax.xml.transform.stream.StreamResult;

import javax.xml.transform.stream.StreamSource;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.oxm.jaxb.Jaxb2Marshaller;

public class OxmJaxbTest1 {

public static void main(String[] args) throws Exception {

ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");

Jaxb2Marshaller jaxbMarshaller = (Jaxb2Marshaller)context.getBean("jaxbMarshaller");

Map<String, Boolean> marshallerProperties = new HashMap<String, Boolean>();

marshallerProperties.put(Marshaller.JAXB\_FORMATTED\_OUTPUT, true);

jaxbMarshaller.setMarshallerProperties(marshallerProperties);

//marshalObject(jaxbMarshaller);

unmarshallXML(jaxbMarshaller);

}

private static void marshalObject(Jaxb2Marshaller jaxbMarshaller) throws Exception{

Department department = new Department();

department.setId("IT");

department.setName("IT Department");

Employee employee = new Employee("12345", "Test Employee");

//employee.setDepartment(department);

FileOutputStream outputStream = new FileOutputStream(new

File("employee.xml"));

StreamResult result = new StreamResult(outputStream);

jaxbMarshaller.marshal(employee, result);

}

private static void unmarshallXML(Jaxb2Marshaller jaxbMarshaller) throws Exception{

FileInputStream inputStream = new FileInputStream(new File("employee.xml"));

StreamSource source = new StreamSource(inputStream);

//Employees emps = (Employees)jaxbMarshaller.unmarshal(source);

Employees emps = (Employees)jaxbMarshaller.unmarshal(source);

System.out.println(emps.getEmployees().get(0).getName());

/\*List l = emps.getEmployees();

System.out.println(l);

Iterator it = l.iterator();

while(it.hasNext()){

Employee employee = (Employee)it.next();

System.out.println("Emp Id is " + employee.getId());

System.out.println("Emp Name is " + employee.getName());

}\*/

//Department department = employee.getDepartment();

//System.out.println("Dept Id is " + department.getId());

//System.out.println("Dept Name is " + department.getName()); }}

**\*\*\*\*\*\*\*\*\*\*\*context.xml in src folder\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<beans default-autowire="no"

xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id = "jaxbMarshaller"

class = "org.springframework.oxm.jaxb.Jaxb2Marshaller">

<property name = "classesToBeBound">

<array>

<value>com.igatepatni.jaxb.Employees</value>

<value>com.igatepatni.jaxb.Employee</value>

<value>com.igatepatni.jaxb.Department</value>

</array>

</property>

</bean>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*employee.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>

<employees>

<employee>

<department deptId="IT">

<deptName>IT Department</deptName>

</department>

<empId>12345</empId>

<empName>Vaishali</empName>

</employee>

<employee>

<department deptId="Act">

<deptName>Acout Department</deptName>

</department>

<empId>1111</empId>

<empName>Priti</empName>

</employee>

</employees>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring OXM Using XMLBean\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*Code Generator.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.xmlbeans;

import org.apache.xmlbeans.impl.tool.SchemaCompiler;

public class CodeGenerator {

public static void main(String[] args) {

String[] params = new String[]{"scomp", "message.xsd"};

SchemaCompiler.main(params);

System.out.println("This is done.......");

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*OXMXMLBeanTest.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.xmlbeans;

import java.io.File;

import java.io.FileOutputStream;

import javax.xml.transform.stream.StreamResult;

import org.example.message.Message;

import org.example.message.MessageType;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.oxm.xmlbeans.XmlBeansMarshaller;

public class OxmXmlBeansTest {

public static void main(String[] args) throws Exception{

ApplicationContext context = new ClassPathXmlApplicationContext("xmlbeans-context.xml");

XmlBeansMarshaller marshaller = (XmlBeansMarshaller)context.getBean("xmlBeansMarshaller");

testObjectToXml(marshaller);

}

private static void testObjectToXml(XmlBeansMarshaller marshaller) throws Exception{

Message message = Message.Factory.newInstance();

message.setData("Test message");

message.setSenderId("test@test.com");

message.setSize(12345);

MessageType messageType = MessageType.Factory.newInstance();

messageType.setValue("VIDEO");

message.setMessageType(messageType);

FileOutputStream outputStream = new FileOutputStream(new File("message.xml"));

StreamResult xmlResult = new StreamResult(outputStream);

marshaller.marshal(message, xmlResult);

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*xml-beans-context.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<beans default-autowire="no"

xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id = "xmlBeansMarshaller"

class = "org.springframework.oxm.xmlbeans.XmlBeansMarshaller">

</bean>

</beans>

**\*\*\*\*\*\*\*\*\*Message.xsd in project folder\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<schema xmlns="http://www.w3.org/2001/XMLSchema"

targetNamespace="http://www.example.org/message"

xmlns:tns="http://www.example.org/message"

elementFormDefault="qualified">

<complexType name="message">

<sequence>

<element name="data" type="string" minOccurs="1" maxOccurs="1">

</element>

<element name="senderId" type="string" minOccurs="1" maxOccurs="1">

</element>

<element name="size" type="int" minOccurs="1" maxOccurs="1">

</element>

<element name="messageType" type="tns:messageType" minOccurs="1" maxOccurs="1"></element>

</sequence>

</complexType>

<complexType name="messageType">

<sequence>

<element name="value" type="string"></element>

</sequence>

</complexType>

</schema>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Scheduler Project\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*App.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler.trigger;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args ) throws Exception

{

new ClassPathXmlApplicationContext("Spring-Quartz.xml");

}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*AppTest.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler.trigger; import junit.framework.Test;

import junit.framework.TestCase;

import junit.framework.TestSuite;

/\*\* \* Unit test for simple App. \*/

public class AppTest

extends TestCase

{

**/\*\* \* Create the test case \***

**\* @param** testName name of the test case

\*/

public AppTest( String testName )

{ super( testName ); }

/\*\*

\* @return the suite of tests being tested

\*/

public static Test suite()

{

return new TestSuite( AppTest.class );

}

/\*\*

\* Rigourous Test :-)

\*/

public void testApp()

{ assertTrue( true ); }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RunMeJob.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler.trigger;

import org.quartz.JobExecutionContext;

import org.quartz.JobExecutionException;

import org.springframework.scheduling.quartz.QuartzJobBean;

public class RunMeJob extends QuartzJobBean

{

private RunMeTask runMeTask;

public void setRunMeTask(RunMeTask runMeTask) {

this.runMeTask = runMeTask;

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RunMeTask.java\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler.trigger;

public class RunMeTask

{

public void printMe() {

System.out.println("Run Me ~");

}

protected void executeInternal(JobExecutionContext context)

throws JobExecutionException {

runMeTask.printMe();

}}}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring-Quarts.xml\*\*\*\*\*\*\*\*\*\*\*\*\***

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="runMeTask" class="com.igatepatni.scheduler.trigger.RunMeTask" />

<bean name="runMeJob"

class="org.springframework.scheduling.quartz.JobDetailBean">

<property name="jobClass" value="com.igatepatni.scheduler.trigger.RunMeJob" />

<property name="jobDataAsMap">

<map>

<entry key="runMeTask" value-ref="runMeTask" />

</map>

</property>

</bean>

<!--

<bean id="runMeJob"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">

<property name="targetObject" ref="runMeTask" />

<property name="targetMethod" value="printMe" />

</bean>

-->

<!-- Simple Trigger -->

<bean id="simpleTrigger"

class="org.springframework.scheduling.quartz.SimpleTriggerBean">

<property name="jobDetail" ref="runMeJob" />

<property name="repeatInterval" value="5000" />

<property name="startDelay" value="1000" />

</bean>

<!-- Cron Trigger -->

<bean id="cronTrigger"

class="org.springframework.scheduling.quartz.CronTriggerBean">

<property name="jobDetail" ref="runMeJob" />

<property name="cronExpression" value="0/5 \* \* \* \* ?" />

</bean>

<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="jobDetails">

<list>

<ref bean="runMeJob" />

</list>

</property>

<property name="triggers">

<list>

<ref bean="simpleTrigger" />

</list>

</property>

</bean>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Shedular Notes\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Introduction**

* The Spring Framework provides abstractions for asynchronous execution and scheduling of tasks with the TaskExecutor and TaskScheduler interfaces, respectively.
* Spring also features implementations of those interfaces that support thread pools or delegation to CommonJ within an application server environment.
* Ultimately the use of these implementations behind the common interfaces abstracts away the differences between Java SE 5, Java SE 6 and Java EE environments.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring-schedular.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="taskExecutor"

class="org.springframework.scheduling.concurrent.ThreadPoolTaskExecutor">

<property name="corePoolSize" value="5" />

<property name="maxPoolSize" value="10" />

<property name="queueCapacity" value="25" />

</bean>

<bean id="taskExecutorExample" class="com.igatepatni.scheduler.TaskExecutorExample">

<constructor-arg ref="taskExecutor" />

</bean>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\* TaskExecutorExample.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.core.task.TaskExecutor;

import org.springframework.oxm.castor.CastorMarshaller;

public class TaskExecutorExample

{

private class MessagePrinterTask implements Runnable

{

private String message;

public MessagePrinterTask(String message)

{

this.message = message;

}

public void run()

{

System.out.println("Run........."+message);

}

}

private TaskExecutor taskExecutor;

public TaskExecutorExample(TaskExecutor taskExecutor)

{

this.taskExecutor = taskExecutor;

}

public void printMessages()

{

for (int i = 0; i < 5; i++)

{

taskExecutor.execute(new MessagePrinterTask("Message............" + i));

}

}

public static void main(String args[])

{

ApplicationContext context = new ClassPathXmlApplicationContext("spring-scheduler.xml");

TaskExecutorExample tee = (TaskExecutorExample)context.getBean("taskExecutorExample");

tee.printMessages();

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Using Quartz Shedular\*\*\*\*\*\*\*\*\*\*\*\*\***

## Using the Quartz Scheduler

Quartz uses Trigger, Job and JobDetail objects to realize scheduling of all kinds of jobs. For the basic concepts behind Quartz, have a look at [http://quartz-scheduler.org](http://quartz-scheduler.org/). For convenience purposes, Spring offers a couple of classes that simplify the usage of Quartz within Spring-based applications.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring-Quartz.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

<bean id="runMeTask" class="com.igatepatni.scheduler.trigger.RunMeTask" />

<bean name="runMeJob"

class="org.springframework.scheduling.quartz.JobDetailBean">

<property name="jobClass" value="com.igatepatni.scheduler.trigger.RunMeJob" />

<property name="jobDataAsMap">

<map>

<entry key="runMeTask" value-ref="runMeTask" />

</map>

</property>

</bean>

<!--

<bean id="runMeJob"

class="org.springframework.scheduling.quartz.MethodInvokingJobDetailFactoryBean">

<property name="targetObject" ref="runMeTask" />

<property name="targetMethod" value="printMe" />

</bean>

-->

<!-- Simple Trigger -->

<bean id="simpleTrigger"

class="org.springframework.scheduling.quartz.SimpleTriggerBean">

<property name="jobDetail" ref="runMeJob" />

<property name="repeatInterval" value="5000" />

<property name="startDelay" value="1000" />

</bean>

<!-- Cron Trigger -->

<bean id="cronTrigger"

class="org.springframework.scheduling.quartz.CronTriggerBean">

<property name="jobDetail" ref="runMeJob" />

<property name="cronExpression" value="0/5 \* \* \* \* ?" />

</bean>

<bean class="org.springframework.scheduling.quartz.SchedulerFactoryBean">

<property name="jobDetails">

<list>

<ref bean="runMeJob" />

</list>

</property>

<property name="triggers">

<list>

<ref bean="simpleTrigger" />

</list>

</property>

</bean>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

* Quartz JobDetail objects contain all information needed to run a job.
* Spring provides a JobDetailFactoryBean which provides bean-style properties for XML configuration purposes.
* The job detail configuration has all information it needs to run the job (RunMeJob).
* The timeout is specified in the job data map.
* The job data map is available through the JobExecutionContext (passed to you at execution time), but the JobDetail also gets its properties from the job data mapped to properties of the job instance. So in this case, if the ExampleJob contains a bean property named timeout, the JobDetail will have it applied automatically:

### Wiring up jobs using triggers and the SchedulerFactoryBean

* We’ve created job details and jobs. We’ve also reviewed the convenience bean that allows you to invoke a method on a specific object.
* Of course, we still need to schedule the jobs themselves. This is done using triggers and a SchedulerFactoryBean.
* Several triggers are available within Quartz and Spring offers two Quartz FactoryBean implementations with convenient defaults: CronTriggerFactoryBean and SimpleTriggerFactoryBean.
* Triggers need to be scheduled. Spring offers a SchedulerFactoryBean that exposes triggers to be set as properties. SchedulerFactoryBean schedules the actual jobs with those triggers.

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RunMeJob.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler.trigger;

import org.quartz.JobExecutionContext;

import org.quartz.JobExecutionException;

import org.springframework.scheduling.quartz.QuartzJobBean;

public class RunMeJob extends QuartzJobBean

{

private RunMeTask runMeTask;

public void setRunMeTask(RunMeTask runMeTask)

{

this.runMeTask = runMeTask;

}

protected void executeInternal(JobExecutionContext context) throws JobExecutionException {

runMeTask.printMe();

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*RunMeTask.java\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler.trigger;

public class RunMeTask

{

public void printMe()

{

System.out.println("Run Me ~");

}

}

**\*\*\*\*\*\*\*\*\*\*App.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.scheduler.trigger;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App

{

public static void main( String[] args ) throws Exception

{

new ClassPathXmlApplicationContext("Spring-Quartz.xml");

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Spring JPA Notes \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Spring and the Java Persistence API**

* From its beginning, the EJB specification has included the concept of entity beans.
* In EJB, entity beans are a type of EJB that describes business objects that are persisted in a relational database.
* Entity beans have undergone several tweaks over the years, includ-ing bean-managed persistence (BMP) entity beans and container-managed persistence (CMP) entity beans
* . Entity beans both enjoyed the rise and suffered the fall of EJB’s popularity.
* In recent years, developers have traded in their heavyweight EJBs for simpler POJO-based development.
* This presented a challenge to the Java Community Process to shape the new EJB specification around POJOs.
* JPA is a POJO-based persistence mechanism that draws ideas from both Hibernate and Java Data Objects (JDO), and mixes Java 5 annotations in for good measure.
* With the Spring 2.0 release came the premiere of Spring integration with JPA. The irony is that many blame (or credit) Spring with the demise of EJB.
* But now that Spring provides support for JPA, many developers are recommending JPA for persistence in Spring-based applications.
* In fact, some say that Spring-JPA is the dream team for POJO development.
* The first step toward using JPA with Spring is to configure an entity manager factory as a bean in the Spring application context.

**Configuring an entity manager factory**

* In a nutshell, JPA-based applications use an implementation of EntityManager- Factory to get an instance of an EntityManager.
* The JPA specification defines two kinds of entity managers:

**Application-managed**—Entity managers are created when an application directly requests one from an entity manager factory.

* With application-managed entity managers, the application is responsible for opening or closing entity managers and involving the entity manager in transactions.
* This type of entity manager is most appropriate for use in standalone applications that don’t run within a Java EE container.

**Container-managed**—Entity managers are created and managed by a Java EE con-tainer.

* The application doesn’t interact with the entity manager factory at all. Instead, entity managers are obtained directly through injection or from JNDI.
* The container is responsible for configuring the entity manager factories.
* This type of entity manager is most appropriate for use by a Java EE container that wants to maintain some control over JPA configuration beyond what’s specified in persistence.xml.
* Both kinds of entity manager implement the same EntityManager interface.
* The key difference isn’t in the EntityManager itself, but rather in how the EntityManager is cre-ated and managed.
* Application-managed EntityManagers are created by an Entity- ManagerFactory obtained by calling the createEntityManagerFactory() method of the PersistenceProvider.
* Meanwhile, container-managed EntityManagerFactorys are obtained through PersistenceProvider’s createContainerEntityManager- Factory() method.
* So what does this all mean for Spring developers wanting to use JPA?
* Not much. Regardless of which variety of EntityManagerFactory you want to use, Spring will take responsibility for managing EntityManagers for you. If using an application- entity manager, Spring plays the role of an application and transparently deals with the EntityManager on your behalf.
* In the container-managed scenario, Spring plays the role of the container.
* Each flavor of entity manager factory is produced by a corresponding Spring fac-tory bean:
* LocalEntityManagerFactoryBean produces an application-managed Entity- ManagerFactory.
* LocalContainerEntityManagerFactoryBean produces a container-managed EntityManagerFactory.

**CONFIGURING APPLICATION-MANAGED JPA:**

<persistence xmlns="http://java.sun.com/xml/ns/persistence" version="1.0"> <persistence-unit name="spitterPU"> <class>com.habuma.spitter.domain.Spitter</class> <class>com.habuma.spitter.domain.Spittle</class> <properties> <property name="toplink.jdbc.driver" value="org.hsqldb.jdbcDriver" /> <property name="toplink.jdbc.url" value= "jdbc:hsqldb:hsql://localhost/spitter/spitter" /> <property name="toplink.jdbc.user" value="sa" /> <property name="toplink.jdbc.password" value="" /> </properties> </persistence-unit> </persistence>

* Because so much configuration goes into a persistence.xml file, little configuration is required (or even possible) in Spring. The following <bean> declares a LocalEntity- ManagerFactoryBean in Spring:

<bean id="emf" class="org.springframework.orm.jpa.LocalEntityManagerFactoryBean"> <property name="persistenceUnitName" value="spitterPU" /> </bean>

**CONFIGURING CONTAINER-MANAGED JPA**

<bean id="emf" class= "org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="dataSource" ref="dataSource" />

<property name="jpaVendorAdapter" ref="jpaVendorAdapter" />

</bean>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*SpringJPA Demos \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Contact table is automatically created\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ContactDao.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.test.dao;

import javax.persistence.EntityManager;

import javax.persistence.PersistenceContext;

import org.springframework.stereotype.Component;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

import com.test.entities.ContactEO;

@Repository

public class ContactDAO {

@PersistenceContext

private EntityManager entityManagerFactory;

@Transactional(readOnly = false)

public void addContact(ContactEO contactEO){

this.entityManagerFactory.persist(contactEO);

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ContactEO.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8**

package com.test.entities;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Table;

import javax.persistence.Transient;

import javax.persistence.\*;

import com.test.vo.Contact;

import com.test.vo.ContactVO;

@Entity

@Table(name="contact")

public class ContactEO implements Contact{

@Transient

Contact contact;

@Transient

public Contact getContact() { return contact;}

public void setContact(Contact contact) { this.contact = contact; }

public ContactEO(){contact = new ContactVO(); }

public ContactEO(Contact contact){ this.contact = contact; }

@Column(name="FIRSTNAME")

public String getFirstName() { return contact.getFirstName(); }

public void setFirstName(String firstName) { contact.setFirstName(firstName); }

@Column(name="LASTNAME")

public String getLastName() { return contact.getLastName(); }

public void setLastName(String lastName) {

contact.setLastName(lastName);

}

@Column(name="EMAIL")

public String getEmail() {

return contact.getEmail();

}

public void setEmail(String email) {

contact.setEmail(email);

}

@Id

@GeneratedValue

@Column(name="ID")

public long getId() { return contact.getId(); }

public void setId(long id) { contact.setId(id); }

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ContactService.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.test.service;

import org.springframework.stereotype.Component;

import com.test.vo.Contact;

@Component

public interface ContactService {

public void addContactInformation(Contact contact);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ContactServiceImpl.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.test.service;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

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

import org.springframework.dao.DataAccessException;

import com.test.dao.ContactDAO;

import com.test.entities.ContactEO;

import com.test.vo.Contact;

public class ContactServiceImpl implements ContactService{

@Autowired

ContactDAO contactDAO;

//@Transactional(readOnly = false)

public void addContactInformation(Contact contact){

try {

// construct the Entity bean

ContactEO contactEO = new ContactEO(contact);

// save it

contactDAO.addContact(contactEO);

} catch (DataAccessException e) {

System.out.println("error occured\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" + e.getMessage());

}

catch(Exception e){

System.out.println("\*\*\*\*\*\*\*\*\*\*in gneric exception "+e.getMessage());

}

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Contact.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*8**

package com.test.vo;

public interface Contact {

public String getFirstName();

public void setFirstName(String firstName);

public String getLastName();

public void setLastName(String lastName);

public String getEmail();

public void setEmail(String email);

public long getId();

public void setId(long id);

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ContactVO.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.test.vo;

public class ContactVO implements Contact{

private String firstName;

private String lastName;

private String email;

private long id;

public String getFirstName() { return firstName; }

public void setFirstName(String firstName) { this.firstName = firstName; }

public String getLastName() { return lastName; }

public void setLastName(String lastName) { this.lastName = lastName; }

public String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

public long getId() { return id; }

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

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Src/META-INF/persistence.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<persistence xmlns="http://java.sun.com/xml/ns/persistence"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence\_1\_0.xsd"

version="1.0">

<persistence-unit name="ContactPU" transaction-type="RESOURCE\_LOCAL">

<provider>org.hibernate.ejb.HibernatePersistence</provider>

<class>com.test.entities.ContactEO</class>

<properties>

<property name="hibernate.connection.url" value="jdbc:oracle:thin:@127.0.0.1:1521:XE" />

<property name="hibernate.dialect" value="org.hibernate.dialect.OracleDialect" />

<property name="hibernate.connection.driver\_class" value="oracle.jdbc.driver.OracleDriver" />

<property name="hibernate.connection.password" value="root" />

<property name="hibernate.connection.username" value="system" />

<property name="hibernate.hbm2ddl.auto" value="create" />

<property name="hibernate.show\_sql" value="true" />

<property name="hibernate.format\_sql" value="true" />

</properties>

</persistence-unit>

</persistence>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*src/applicationContext.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?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:p="http://www.springframework.org/schema/p" xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context" xmlns:jee="http://www.springframework.org/schema/jee"

xmlns:tx="http://www.springframework.org/schema/tx"

xsi:schemaLocation="

http://www.springframework.org/schema/aop http://www.springframework.org/schema/aop/spring-aop-2.5.xsd

http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans-2.5.xsd

http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-2.5.xsd

http://www.springframework.org/schema/jee http://www.springframework.org/schema/jee/spring-jee-2.5.xsd

http://www.springframework.org/schema/tx http://www.springframework.org/schema/tx/spring-tx-2.5.xsd">

<!-- Connection Pool -->

<bean id="dataSource"

class="org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name="driverClassName" value="oracle.jdbc.driver.OracleDriver" />

<property name="url" value="jdbc:oracle:thin:@127.0.0.1:1521:XE" />

<property name="username" value="system" />

<property name="password" value="root" />

</bean>

<!-- JPA EntityManagerFactory configure container manged JPA-->

<!-- <bean id="entityManagerFactory" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"

p:dataSource-ref="dataSource">

<property name="jpaVendorAdapter">

<bean class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">

<property name="showSql" value="true"/>

</bean>

</property>

<property name="jpaProperties">

<props>

<prop key="hibernate.hbm2ddl.auto">create</prop>

<prop key="hibernate.dialect">org.hibernate.dialect.OracleDialect</prop>

</props>

</property>

</bean>-->

<bean id="entityManagerFactory"

class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">

<property name="persistenceUnitName" value="ContactPU"/>

</bean>

<!-- Transaction manager for a single JPA EntityManagerFactory (alternative to JTA) -->

<bean id="transactionManager" class="org.springframework.orm.jpa.JpaTransactionManager"

p:entityManagerFactory-ref="entityManagerFactory"/>

<!-- Activates various annotations to be detected in bean classes for eg @Autowired-->

<context:annotation-config/>

<!-- enable the configuration of transactional behavior based on annotations -->

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

<!--<bean class="org.springframework.dao.annotation.PersistenceExceptionTranslationPostProcessor"/> -->

<context:component-scan base-package="com.test.dao"/>

<bean id="contactService" class="com.test.service.ContactServiceImpl"/>

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*TestContact.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.test.test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.test.service.ContactService;

import com.test.vo.Contact;

import com.test.vo.ContactVO;

public class TestContact {

public static void main(String args[]){

ContactService contactService ;

ApplicationContext appContext =

new ClassPathXmlApplicationContext("applicationContext.xml");

contactService = (ContactService) appContext.getBean("contactService");

Contact contact = new ContactVO();

contact.setFirstName("Vaishali");

contact.setLastName("Srivastava");

contact.setEmail("vai@xxx.com");

System.out.println("here\*\*\*\*\*\*\*\*\*\*\*");

contactService.addContactInformation(contact);

System.out.println("record added");

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*Spring Rest Service\*\*\*\*\*\*\*\*\*\*\*\*\***

**CHARACTERISTICS OF A RESTFUL URL**

* In contrast to their RESTless cousins, RESTful URLs fully acknowledge that HTTP is all about resources.
* For example, figure 11.2 shows how we might restructure the REST-less URL to be more resource-oriented.
* Although outside of the scope of this book, the semantic web takes advantage of the identifying nature of URLs in creating a linked web of resources.
* **Figure 11.1 A RESTless URL is action-oriented and doesn’t identify or locate a resource. *Giving Spring some REST*** One thing that’s not clear about this URL is what it does. That’s because the URL doesn’t *do* any-thing. Rather, it identifies a resource.
* Specifi-cally, it locates the resource that represents a Spittle object. What will be done with that resource is a separate matter—one for HTTP methods to decide (which we’ll look at in sec-tion 11.2.3).
* This URL not only locates a resource, but it also uniquely identifies that resource—it serves equally well as a URI as it does as a URL. Instead of using a query parameter to identify the resource, the entire base URL identifies the resource. In fact, the new URL has no query parameters at all.
* Although query parameters are still a legitimate way to send information to the server, they’re intended to provide guidance to the server in producing the resource. They shouldn’t be used to help identify a resource. One final observation should be made about RESTful URLs: they tend to be hierar-chical.
* As you read them from left to right, you move from a broad concept to some-thing more precise. In our example, the URL has several levels, any of which could identify a resource:  *http://localhost:8080* identifies a domain and port.
* Although our application won’t associate a resource with this URL, there’s no reason why it couldn’t.
* *http://localhost:8080/Spitter* identifies the application’s servlet context. This URL is more specific in that it has identified an application running on the server
* *http://localhost:8080/Spitter/spittles* identifies a resource that represents a list of Spittle objects within the Spitter application.
* *http://localhost:8080/Spitter/spittles/123* is the most precise URL, identifying a spe-cific Spittle resource. What makes the RESTful URL interesting is that its path is parameterized.
* Whereas the RESTless URL took its input from query parameters, the RESTful URL’s input is part of the URL’s path. To be able to handle requests for that kind of URL, we’ll need a way to write a controller’s handler method so that it can take input from the URL’s path.
* **EMBEDDING PARAMETERS IN URLS** To enable parameterized URL paths, Spring 3 introduced a new @PathVariable anno-tation. To see how this works, look at SpittleController, a new Spring MVC control-ler

**\*\*\*\*\*\*\*\*\* EmployeeController.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.rest.controller;

import java.util.List;

import javax.xml.ws.RequestWrapper;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.ModelAttribute;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RequestMethod;

import org.springframework.web.bind.annotation.ResponseBody;

import org.springframework.web.servlet.ModelAndView;

import com.igatepatni.rest.domain.Employee;

import com.igatepatni.rest.domain.EmployeeDS;

import com.igatepatni.rest.domain.EmployeeList;

@Controller

@RequestMapping("/employee")

public class EmployeeController {

private EmployeeDS employeeDS;

public void setEmployeeDS(EmployeeDS ds) {

this.employeeDS = ds;

}

private static final String VIEW\_NAME = "employees";

@RequestMapping(value = "/updateEmp", method = RequestMethod.GET)

public String showForm(Model model) {

model.addAttribute(new Employee());

return "updatepage";

}

@RequestMapping(value = "/addEmp", method = RequestMethod.GET)

public String addEmp(Model model) {

model.addAttribute(new Employee());

return "addpage";

}

@RequestMapping(value = "/deleteEmp", method = RequestMethod.GET)

public String deleteEmp(Model model) {

model.addAttribute(new Employee());

return "deletepage";

}

@RequestMapping(method = RequestMethod.GET, value = "/{id}")

public ModelAndView getEmployee(@PathVariable String id) {

System.out.println("in get in jsp response " + id);

Employee e = employeeDS.getEmployee(Integer.parseInt(id));

return new ModelAndView("getpage", "employee", e);

}

/\*@RequestMapping(method = RequestMethod.GET, value = "/{id}")

@ResponseBody

public String getEmployeeforClient(@PathVariable String id) {

System.out.println("in get using json response " + id);

Employee e = employeeDS.getEmployee(Integer.parseInt(id));

return e.toString();

}

\*/

@RequestMapping(method = RequestMethod.PUT, value = "/{id}")

public String updateEmployee(@ModelAttribute Employee employee) {

System.out.println("in updateEmp() " + employee);

employeeDS.update(employee);

return "resultpage";

}

@RequestMapping(method = RequestMethod.POST)

public ModelAndView addEmployee(@ModelAttribute Employee employee) {

System.out.println("in add emp : " + employee);

employeeDS.add(employee);

List<Employee> employees = employeeDS.getAll();

EmployeeList list = new EmployeeList(employees);

return new ModelAndView("employees", "employees", list);

}

@RequestMapping(method = RequestMethod.DELETE, value = "/{id}")

public ModelAndView removeEmployee(@PathVariable String id,@ModelAttribute Employee employee) {

System.out.println("in delete " + id);

System.out.println("in delete1 " + employee.getId());

employeeDS.remove(employee.getId());

List<Employee> employees = employeeDS.getAll();

EmployeeList list = new EmployeeList(employees);

return new ModelAndView(VIEW\_NAME, "employees", list);

}

@RequestMapping(method = RequestMethod.GET, value = "/employees")

public ModelAndView getEmployees() {

List<Employee> employees = employeeDS.getAll();

EmployeeList list = new EmployeeList(employees);

return new ModelAndView("employees", "employees", list);

}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Employee.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.rest.domain;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="employee")

public class Employee {

private int id;

private String name;

private String email;

public Employee() {}

@Override

public String toString() {

return "Employee [email=" + email + ", id=" + id + ", name=" + name

+ "]";

}

public Employee(int id, String name, String email) {

this.id = id;

this.name = name;

this.email = email;

}

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 String getEmail() { return email; }

public void setEmail(String email) { this.email = email; }

}

**\*\*\*\*\*\*\*\*\*\*\*\*EmployeeDS.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.rest.domain;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

public class EmployeeDS {

private static Map<Integer, Employee> allEmployees;

static {

allEmployees = new HashMap<Integer, Employee>();

Employee e1 = new Employee(12345, "Rahul Dravid", "rahul@ipl.com");

Employee e2 = new Employee(17889, "Virat Kohli", "viratk@ipl.com");

allEmployees.put(e1.getId(), e1);

allEmployees.put(e2.getId(), e2);

System.out.println("in static constr :" + allEmployees);

}

public void add(Employee e) {

allEmployees.put(e.getId(), e);

}

public Employee getEmployee(int id) {

System.out.println("in DS : " + id );

System.out.println(allEmployees.get(id));

return allEmployees.get(id);

}

public List<Employee> getAll() {

List<Employee> employees = new ArrayList<Employee>();

for( Iterator<Employee> it = allEmployees.values().iterator(); it.hasNext(); ) {

Employee e = it.next();

employees.add(e);

}

return employees;

}

public void remove(int id) {allEmployees.remove(id);}

public void update(Employee e) {allEmployees.put(e.getId(), e); }

}

**\*\*\*\*\*\*\*\*\*\*\*\*EmployeeList.java\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

package com.igatepatni.rest.domain;

import java.util.List;

import javax.xml.bind.annotation.XmlElement;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name="employees")

public class EmployeeList {

private int count;

private List<Employee> employees;

public EmployeeList() {}

public EmployeeList(List<Employee> employees) {

this.employees = employees;

this.count = employees.size();

}

public int getCount() { return count; }

public void setCount(int count) {this.count = count; }

@XmlElement(name="employee")

public List<Employee> getEmployees() {return employees; }

public void setEmployees(List<Employee> employees) {this.employees = employees;}

}

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*addpage.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<h1>Create New Employee</h1>

<c:url var="saveUrl" value="/employee" />

<form:form modelAttribute="employee" method="POST" action="${saveUrl}">

<table>

<tr>

<td><form:label path="id">Id:</form:label></td>

<td><form:input path="id" /></td>

</tr>

<tr>

<td><form:label path="name">Name:</form:label></td>

<td><form:input path="name"/></td>

</tr>

<tr>

<td><form:label path="email">Email</form:label></td>

<td><form:input path="email"/></td>

</tr>

</table>

<input type="submit" value="Save" />

</form:form>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*deletepage.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<h1>Delete Employee</h1>

<c:url var="saveUrl" value="/employee/${employee.id}" />

<form:form modelAttribute="employee" method="delete" action="${saveUrl}">

<table>

<tr>

<td><form:label path="id">Id:</form:label></td>

<td><form:input path="id" /></td>

</tr>

</table>

<input type="submit" value="Delete" />

</form:form>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*employees.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Employees</title>

</head>

<body>

<table border=1>

<thead><tr>

<th>ID</th>

<th>Name</th>

<th>Email</th>

</tr></thead>

<c:forEach var="employee" items="${employees.employees}">

<tr>

<td>${employee.id}</td>

<td>${employee.name}</td>

<td>${employee.email}</td>

</tr>

</c:forEach>

</table>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*getpage.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<h1>Get Employee</h1>

<c:if test="${**empty** employee}">

No records found!

</c:if>

<c:if test="${!**empty** employee}">

<table style="border: 1px solid #333">

<tr>

<td style="width: 150px">Id</td>

<td>${employee.id}</td>

</tr>

<tr>

<td>Employee Name </td>

<td>${employee.name}</td>

</tr>

<tr>

<td>Email-ID</td>

<td>${employee.email}</td>

</tr>

</table>

</c:if>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*RestfulClient.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

<h1>Employee Editing page:</h1>

<a href="http://localhost:8081/SpringRestDemo/employee/12345"> Get </a>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*resultpage.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<h1>Result</h1>

Request has been sent!

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*updatepage.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

<%@ page language="java" contentType="text/html; charset=UTF-8"

pageEncoding="UTF-8"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Insert title here</title>

</head>

<body>

<h1>Edit Employee</h1>

<c:url var="saveUrl" value="/employee/${employee.id}" />

<form:form modelAttribute="employee" method="PUT" action="${saveUrl}">

<table>

<tr>

<td><form:label path="id">Id:</form:label></td>

<td><form:input path="id"/></td>

</tr>

<tr>

<td><form:label path="name">Name:</form:label></td>

<td><form:input path="name"/></td>

</tr>

<tr>

<td><form:label path="email">Email</form:label></td>

<td><form:input path="email"/></td>

</tr>

</table>

<input type="submit" value="Save" />

</form:form>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*index.jsp\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"

pageEncoding="ISO-8859-1"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">

<title>Insert title here</title>

</head>

<body>

Welcome to ReST

<a href="http://localhost:8081/SpringRestDemo/employee/addEmp">Add Emp</a><br/>

<a href="http://localhost:8081/SpringRestDemo/employee/deleteEmp">Delete Emp</a><br/>

<a href="http://localhost:8081/SpringRestDemo/employee/employees">List All Emp</a><br/>

<a href="http://localhost:8081/SpringRestDemo/employee/updateEmp">Update Emp</a><br/>

<a href="RestfulClient.jsp">RestfulClient</a><br/>

</body>

</html>

**\*\*\*\*\*\*\*\*\*\*\*\*\*restServlet.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?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/context"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context-3.0.xsd">

<context:component-scan base-package="com.igatepatni.rest.controller" />

<!--

To enable @RequestMapping process on type level and method level

-->

<bean

class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />

<bean

class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />

<bean id="viewResolver"

class="org.springframework.web.servlet.view.InternalResourceViewResolver">

<property name="viewClass">

<value>org.springframework.web.servlet.view.JstlView</value>

</property>

<property name="prefix">

<value>/jsp/</value>

</property>

<property name="suffix">

<value>.jsp</value>

</property>

</bean>

<!--<bean

class="org.springframework.web.servlet.view.ContentNegotiatingViewResolver">

<property name="mediaTypes">

<map>

<entry key="xml" value="application/xml" />

<entry key="html" value="text/html" />

</map>

</property>

<property name="viewResolvers">

<list>

<bean class="org.springframework.web.servlet.view.BeanNameViewResolver" />

<bean id="viewResolver"

class="org.springframework.web.servlet.view.UrlBasedViewResolver">

<property name="viewClass"

value="org.springframework.web.servlet.view.JstlView" />

<property name="prefix" value="/jsp/" />

<property name="suffix" value=".jsp" />

</bean>

</list>

</property>

</bean>

--><!--

bean id="viewResolver"

class="org.springframework.web.servlet.view.BeanNameViewResolver" /

-->

<bean id="employeeController" class="com.igatepatni.rest.controller.EmployeeController">

<property name="employeeDS" ref="employeeDS" />

</bean>

<bean id="employeeDS" class="com.igatepatni.rest.domain.EmployeeDS" />

</beans>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*web.xml\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

<?xml version="1.0" encoding="UTF-8"?>

<web-app id="WebApp\_ID" version="2.4"

xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app\_2\_4.xsd">

<display-name>SpringRestDemo</display-name>

<filter>

<filter-name>httpMethodFilter</filter-name>

<filter-class>org.springframework.web.filter.HiddenHttpMethodFilter</filter-class>

</filter>

<filter-mapping>

<filter-name>httpMethodFilter</filter-name>

<url-pattern>/\*</url-pattern>

</filter-mapping>

<servlet>

<servlet-name>rest</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

<load-on-startup>1</load-on-startup>

</servlet>

<servlet-mapping>

<servlet-name>rest</servlet-name>

<url-pattern>/</url-pattern>

</servlet-mapping>

<jsp-config>

<taglib>

<taglib-uri>http://java.sun.com/jsp/jstl/core</taglib-uri>

<taglib-location>/WEB-INF/c.tld</taglib-location>

</taglib>

</jsp-config>

<welcome-file-list>

<welcome-file>index.html</welcome-file>

</welcome-file-list>

</web-app>

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Thank You\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***