Spring JDBC

Data Source with connection pooling

```
<bean id="dataSource" class="
org.springframework.jdbc.datasource.DriverManagerDataSource”>
        <property name="driverClassName" value="org.hsqldb.jdbcDriver"/>
        <property name="url" value="jdbc:jsqldb:hsql://localhost/myapp/mydb"/>
        <property name="username" value="root"/>
        <property name="password" value="password"/>
        <property name="intialSize" value="5"/>
        <property name="maxActive" value="10"/>
</bean>
```

Data Source by JNDI

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
        <property name="jndiName" value="/jdbc/mydbDataSource"/>
        <property name="resourceRef" value="true"/>
</bean>
```

Accessing JdbcTemplate

```
<bean id="jdbcTemplate" class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
</bean>

<bean id="empDao" class="com.myapp.examples.dao.EmployeeDao">
        <property name="jdbcTemplate" ref="jdbcTemplate"/>
</bean>
```

Accessing 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="oracle"/>
</bean>

<bean id="mysessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">

<property name="dataSource" ref="dataSource"></property>
<property name="mappingResources">
<list>
        <value>employee.hbm.xml</value>
</list>
</property>
```

```
<property name="hibernateProperties">
<props>
<prop key="hibernate.dialect">org.hibernate.dialect.Oracle9Dialect</prop>
<prop key="hibernate.hbm2ddl.auto">update</prop>
<prop key="hibernate.show_sql">true</prop>
</props>
</property>
</bean>

<bean id="hibernateTemplate"
class="org.springframework.orm.hibernate3.HibernateTemplate">

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

<bean id="empDao" class="com.javatpoint.EmployeeDao">
        <property name="template" ref="hibernateTemplate"/>
</bean>
```

```
<bean id="hibernateSessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
<property name="dataSource" ref="myDataSource"/>
<!-- if we have hibernate config file then we add below code instead of above -->
<!-- <property name="configLocation" ><value>hibernate.cfg.xml</value></property> -->
<property name="annotatedClasses">
        <list>
                <value>org.springexamples.bean.Employee</value>
        </list>
</property>
<property name="hibernateProperties">
        <props>
        <prop key="hibernate.dialect">org.hibernate.dialect.MySQLDialect</prop>
        <prop key="hibernate.show_sql">true</prop>
        </props>
</property>
</bean>
```

Insert, update delete and select operation using JdbcTemplate:

```
public class MyEmployeeDaoImpl implements MyEmployeeDao {

        private JdbcTemplate empJdbcTemplate;

        public JdbcTemplate getEmpJdbcTemplate() {

                return empJdbcTemplate;

        }

        public void setEmpJdbcTemplate(JdbcTemplate empJdbcTemplate) {

                this.empJdbcTemplate = empJdbcTemplate;

        }

        @Override
```

```
public void addEmployee(MyEmployee employee) {
        String sql="INSERT INTO myemployee (empname,birthdate,joindate,idtype,idno,email,mobileno)VALUES (?,?,?,?,?,?,?)";
Object[] args=new Object [] {
                        employee.getEmpname(),
                        employee.getBirthDate(),
                        employee.getJoinDate(),
                        employee.getIdType(),
                        employee.getIdno(),
                        employee.getEmail(),
                        employee.getMobileno()
        };
        int result=empJdbcTemplate.update(sql, args);
        if(result !=0){
                System.out.println("Employee Record Saved Successfully!!");
        }else{
                System.out.println("Err:Record Saving Failure!!");
        }
}
@Override
public void updateEmployee(MyEmployee employee) {
        String sql="update myemployee set email=?, mobileno=? where empid=?";
        Object[] args=new Object[]{
                        employee.getEmail(),
                        employee.getMobileno(),
                        employee.getEmpid()
        };
        int result=empJdbcTemplate.update(sql, args);
        if(result !=0){
                System.out.println("Employee Record updated Successfully!!");
        }else{
                System.out.println("Err:Record Updating Failure!!");
        }
}
@Override
public void deleteEmployee(MyEmployee employee) {
```

```
		String sql="delete from myemployee where empid=?";
		Object[] args=new Object[]{
			employee.getEmpid()
		};
		int result=empJdbcTemplate.update(sql, args);
		if(result !=0){
			System.out.println("Employee Record deleted Successfully!!");
		}else{
			System.out.println("Err:Record deleting Failure!!");
		}
	}
	@Override
	public List<MyEmployee> fetchEmployee() {
		String sql="select * from myemployee";
		List<MyEmployee> employeeList=empJdbcTemplate.query(sql,new RowMapper<MyEmployee>(){
			@Override
			public MyEmployee mapRow(ResultSet rs, int rowNum)throws SQLException {
				MyEmployee employee=new MyEmployee();
				employee.setEmpid(rs.getInt("empid"));
				employee.setEmpname(rs.getString("empname"));
				employee.setBirthDate(rs.getString("birthdate"));
				employee.setJoinDate(rs.getString("joindate"));
				employee.setIdType(rs.getString("idtype"));
				employee.setIdno(rs.getString("idno"));
				employee.setEmail(rs.getString("email"));
				employee.setMobileno(rs.getString("mobileno"));
				return employee;
			}
		});
		return employeeList;
	}
	@Override
	public Map<String,Object> getEmployeeUsingStoredProcById(MyEmployee employee) {

		SqlParameter InParam_empid=new SqlParameter(Types.BIGINT);
```

```
                List<SqlParameter> parameters=new ArrayList<SqlParameter>();
                parameters.add(InParam_empid);
                Map<String,Object> empRec=empJdbcTemplate.call(new CallableStatementCreator() {
                        @Override
                        public CallableStatement createCallableStatement(Connection conn)
                                        throws SQLException {
                                CallableStatement callStmnt=conn.prepareCall("{call employeeById(?)}");
                                callStmnt.setLong(1, employee.getEmpid());
                                return callStmnt;
                        }
                },parameters);
                return empRec;
        }
        @Override
        public MyEmployee getEmployeeById(MyEmployee employee) {
                String sql="select * from myemployee where empid=?";
        MyEmployee emp=empJdbcTemplate.queryForObject(sql, new Object[]{employee.getEmpid()}, new MyEmployeeMapper());
                return emp;
        }
}
class MyEmployeeMapper implements RowMapper<MyEmployee>{
        @Override
        public MyEmployee mapRow(ResultSet rs, int rowNum) throws SQLException {
                MyEmployee employee=new MyEmployee();
                employee.setEmpid(rs.getInt("empid"));
                employee.setEmpname(rs.getString("empname"));
                employee.setBirthDate(rs.getString("birthdate"));
                employee.setJoinDate(rs.getString("joindate"));
                employee.setIdType(rs.getString("idtype"));
                employee.setIdno(rs.getString("idno"));
                employee.setEmail(rs.getString("email"));
                employee.setMobileno(rs.getString("mobileno"));
                return employee;
        }
}
```

NamedParameterJdbcTemplate and SimpleJdbcCall Example:

Spring provides another way to insert data by named parameter. In such way, we use names instead of? (Question mark). So it is better to remember the data for the column.

A SimpleJdbcCall is a multi-threaded, reusable object representing a call to a stored procedure or a stored function.

```
private NamedParameterJdbcTemplate namedParameterjdbcTemplate;

private SimpleJdbcCall simpleJdbcCall;


public void setDataSource(DataSource dataSource){
        this.namedParameterjdbcTemplate=new NamedParameterJdbcTemplate(dataSource);
        this.simpleJdbcCall=new SimpleJdbcCall(dataSource);
}
public void updateEmployee(MyEmployee employee) {
                String sql="update myemployee set empname=:ename,birthdate=:bdate,
                        joindate=:jdate,idtype=:idtype,idno=:idno,email=:email,mobileno=:mobile
                        where empid=:eid";
                MapSqlParameterSource namedParameters = new MapSqlParameterSource();
                namedParameters.addValue("ename", employee.getEmpname());
                namedParameters.addValue("bdate", employee.getBirthDate());
                namedParameters.addValue("jdate", employee.getJoinDate());
                namedParameters.addValue("idtype", employee.getIdType());
                namedParameters.addValue("idno", employee.getIdno());
                namedParameters.addValue("email", employee.getEmail());
                namedParameters.addValue("mobile", employee.getMobileno());
                namedParameters.addValue("eid", employee.getEmpid());
                int result=namedParameterjdbcTemplate.update(sql, namedParameters);
                if(result > -1){
                        System.out.println("Updated Successfully!!");
                }else{
```

```
                System.out.println("Updation Failed!!");
            }
        }
```

SimpleJdbcCall Example

```
public Map<String, Object> getEmployeeUsingStoredProcById(MyEmployee employee) {
        simpleJdbcCall.withProcedureName("getEmployeeDetailById");
        MapSqlParameterSource in = new MapSqlParameterSource();
        in.addValue("id", employee.getEmpid());
        Map result=simpleJdbcCall.execute(in);
        return result;
}
```

HibernateTemplate Example

```
import org.springframework.orm.hibernate3.HibernateTemplate;

public class PageDaoImpl implements PageDao {
        private HibernateTemplate hibernateTemplate;

        public void setHibernateTemplate(HibernateTemplate
hibernateTemplate) {
                this.hibernateTemplate = hibernateTemplate;
        }

        public void persist(){
         User u1= new User(1,"Ankita");
         hibernateTemplate.save(u1);

         User u2= new User(2,"Renu");
         hibernateTemplate.save(u2);
    }
        public void save(User user) {
                hibernateTemplate.saveOrUpdate(user);
        }
        public List<User> findAll() {
                return hibernateTemplate.find("from User");
        }
        public void remove(User user) {
            hibernateTemplate.delete(user);
        }
        public User findById(String id) {
            return (User) hibernateTemplate.load(User.class, id);
        }
}
```

The **JdbcTemplate** class provided by Spring Framework is an abstraction wrapper around JDBC. It does several things out of the box as described in the Spring Framework Reference:

- Opens connection.
- Prepares and executes statements.
- Set up the loop to iterate through the results.
- Process any exception.
- Handle transactions.
- Close the connection, statement, and resultset.

RowMapper vs. ResultSetExtractor

ResultSetExtractor is supposed to extract the whole ResultSet (possibly multiple rows), while RowMapper is feeded with row at a time.

```
public List<Employee> getAllEmployees(){
 return template.query("select * from employee",new
ResultSetExtractor<List<Employee>>(){
    @Override
     public List<Employee> extractData(ResultSet rs) throws SQLException,
            DataAccessException {

        List<Employee> list=new ArrayList<Employee>();
        while(rs.next()){
        Employee e=new Employee();
        e.setId(rs.getInt(1));
        e.setName(rs.getString(2));
        e.setSalary(rs.getInt(3));
        list.add(e);
        }
        return list;
        }
    });
  }
}
```

Thank You!!