© 2008 coreservlets.com

Spring JDBC Part 1

Originals of Slides and Source Code for Examples:
http://courses.coreservlets.com/Course-Materials/spring.html

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

© 2008 coreservlets.com

For live Spring & Hibernate training, see courses at http://courses.coreservlets.com/.

Taught by the experts that brought you this tutorial.
Available at public venues, or customized versions can be held on-site at your organization.

- Courses developed and taught by Marty Hall
  - Java 5, Java 6, intermediate/beginning servlets/JSP, advanced servlets/JSP, Struts, JSF, Ajax, GWT, custom mix of topics
- Courses developed and taught by coreservlets.com experts (edited by Marty)
  - Spring, Hibernate/JPA, EJB3, Ruby/Rails

Contact hall@coreservlets.com for details

Topics in This Section

- Introduction to Spring JDBC
- Spring JDBC development
- Spring IoC integration

© 2008 coreservlets.com

Introduction

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Motivation

```
public void save(Customer customer) {
    Connection conn = null;
    boolean autoCommit = false;
    try{
      conn = this.dataSource.getConnection();
      autoCommit = conn.getAutoCommit();
      conn.setAutoCommit(false);
      if(!update(customer, conn)){
        insert(customer, conn);
      }
      conn.commit();
    }
    catch(SQLException e){
      if(conn != null){
        try{
          conn.rollback();
        }
        catch(SQLException suppressed){}
      }
      throw new CustomerPersistenceException("Error:
       SQL."
        + " Error saving customer: " + customer,e);
    }
    catch(RuntimeException e){
      if(conn != null){
        try{
          conn.rollback();
        }
        catch(SQLException suppressed){}
      }
      throw new CustomerPersistenceException("Error:
       SQL."
        + " Error saving customer: " + customer,e);
    }
    finally{
      if(conn != null){
        try{
          conn.setAutoCommit(autoCommit);
          conn.close();
          conn = null;
        }
        catch(SQLException suppressed){}
      }
    }
  }
```

```
private boolean update(Customer customer, Connection
      conn) {
    PreparedStatement stmt = null;
    try{
      stmt = conn.prepareStatement(
        "update customer set name = ? where id = ?");
      stmt.setString(1, customer.getName());
      stmt.setString(2, customer.getId());
      return stmt.executeUpdate() > 0;
    }
    catch(SQLException e){
      throw new CustomerPersistenceException("Error:
       SQL."
        + " Failed to update customer.",e);
    }
    finally{
      if(stmt != null){
        try{
          stmt.close();
          stmt = null;
        }
        catch(Exception suppressed){}
      }
    }
  }
```

```
private boolean insert(Customer customer, Connection
      conn) {
    PreparedStatement stmt = null;
    try{
      stmt = conn.prepareStatement(
        "insert into customer (id, name) values (?,
       ?)");
      stmt.setString(1, customer.getId());
      stmt.setString(2, customer.getName());
      return stmt.executeUpdate() > 0;
    }
    catch(SQLException e){
      throw new CustomerPersistenceException("Error:
       SQL."
        + " Failed to insert customer.",e);
    }
    finally{
      if(stmt != null){
        try{
          stmt.close();
          stmt = null;
        }
        catch(Exception suppressed){}
      }
    }
  }
```

Spring JDBC Solution

```
public void save(Customer customer) {
  Map<String, Object> parameterMap =
    new HashMap<String, Object>();
  parameterMap.put("customerId", customer.getId());
  parameterMap.put("customerName", customer.getName());

  boolean updated = simpleJdbc.update(
    "update customer set name = :customerName"
    + " where id = :customerId", parameters) > 0;

  if(updated){
    return;
  }

  simpleJdbc.update(
    "insert into customer (id, name)"
    + " values (:customerId, :customerName)", parameters);
}
```

Spring JDBC

- **Standalone JDBC software**
  - No dependencies on a running Spring IoC container
- **Templated software**
  - Replaces tedious Java SQL APIs
  - Mitigates JDBC resource mismanagement risks
- **No configuration management overhead**
  - No XML
  - No annotations
- **Pure code solution**
  - Explicit settings
  - Verbose
  - No class or domain modeling constraints
    - Interface-driven domain models
    - Complex constructors
    - Separates class and relational cardinality
- **Outperforms O/R mapping solutions**

Spring JDBC Templates

- **Fine-grained templates**
  - `JdbcTemplate`
  - `NamedParameterJdbcTemplate`
- **Coarse-grained templates**
  - `SimpleJdbcTemplate`
  - `SimpleJdbcInsert`
  - `SimpleJdbcCall`
- **SQL objects**
  - `SqlUpdate`
  - `MappingSqlQuery`

© 2008 coreservlets.com

Spring IoC Process Review

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Spring IoC Process

- **Develop POJO library**
  - Define the interfaces
  - Create the implementations
- **Register Spring JARs**
  - spring-core.jar
  - spring-context.jar
  - spring-beans.jar
  - commons-logging.jar
- **Create the bean definitions file**
  - Default to the file name `applicationContext.xml`
  - Place the file in an accessible location
- **Register beans**
  - Register the spring-beans XML schema into the bean definitions file Assign bean identifiers using the `id` attribute
  - Specify the bean creation method; e.g, the `class` attribute for direct constructor invocation

Spring IoC Process Continued

- **Integrate configuration**
  - Register the **`spring-context`** XML schema into the bean definitions file
  - Declare a **`property-placeholder`** element with the configuration file path assigned to the **`location`** attribute
- **Define bean interdependencies**
  - Select a DI method; e.g., constructor, property setter, lookup-method, etc…
  - Specify the injection value; e.g., collaborators, values, resources, etc…
- **Initialize container**
  - Select an ApplicationContext implementation
    - The integration method will depend on the target environment
  - Specify the location of the bean definitions file(s)
- **Access and use beans from the Spring IoC container**
  - For example, via the BeanFactory API
    - **`BeanFactory#getBean(beanName:String):Object`**
    - **`BeanFactory#getBean(beanName:String, requiredType:Class):Object`**

Develop POJO Library

```
package coreservlets;

public class Customer {

  private String id;
  private String name;

  public Customer(String id, String name){
    this.id = id;
    this.name = name;
  }
  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
}
```

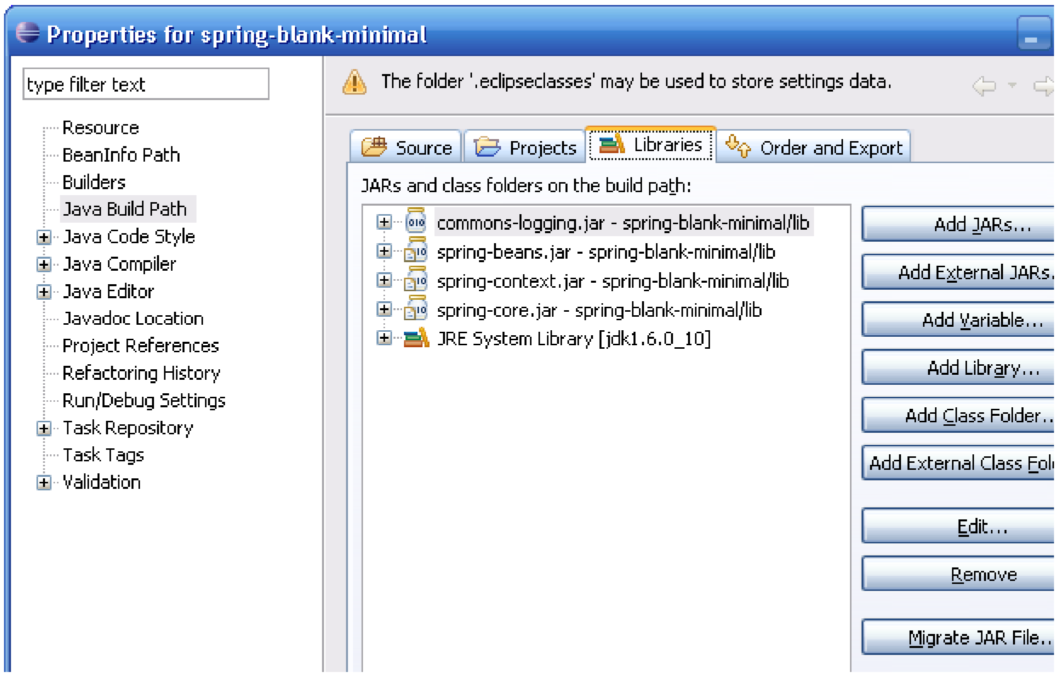
Develop POJO Library

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}
```

Develop POJO Library

```
public class CustomerQueryImpl implements CustomerQuery {

  private List<Customer> customers;

  public CustomerQueryImpl(List<Customer>customers) {
    this.customers = customers;
  }

  public Customer getCustomerByName(String name) {
    for(Customer c : customers){
      if(c.getName().equals(name)){
        return c;
      }
    }
    return null;
  }
}
```

Register Spring JARs



The folder '.eclipseclasses' may be used to store settings data.

JARs and class folders on the build path:

```
commons-logging.jar - spring-blank-minimal/lib
spring-beans.jar - spring-blank-minimal/lib
spring-context.jar - spring-blank-minimal/lib
spring-core.jar - spring-blank-minimal/lib
JRE System Library [jdk1.6.0_10]
```

Register Spring JARs Eclipse .classpath

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry kind="src" path="src"/>
  <classpathentry kind="lib" path="lib/spring-core.jar" />
  <classpathentry kind="lib" path="lib/spring-beans.jar" />
  <classpathentry kind="lib" path="lib/spring-context.jar" />
  <classpathentry kind="lib" path="lib/commons-logging.jar"/>
  <classpathentry kind="con"
    path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="output" path=".eclipseclasses"/>
</classpath>
```

Create the bean definitions file

- **Default to `applicationContext.xml`**
- **Place the file in an accessible location**
  - Classpath, filesystem or web module path

Register Beans

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

</beans>
```

Register Beans

```
<?xml version="1.0" encoding="UTF-8"?>
<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="customerQuery"
         class="coreservlets.mockup.CustomerQueryImpl" />

</beans>
```

Define Bean Interdependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<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="customerQuery"
         class="coreservlets.mockup.CustomerQueryImpl">
     <constructor-arg>
       <list>
         <bean class="coreservlets.Customer">
           <property name="id" value="jjoe" />
           <property name="name" value="Java Joe" />
         </bean>
       </list>
     </constructor-arg>
   </bean>

</beans>
```

Initialize Container

```
import org.springframework.context.support.*;

public class Main {

  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/applicationContext.xml");

  }
}
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/applicationContext.xml");

    CustomerQuery query =
     (CustomerQuery) beanFactory.getBean("customerQuery");

    Customer customer =
      query.getCustomerByName("Java Joe");

    System.out.println(customer);
  }
}
```

Standard output

```
Customer id=jjoe, name=Java Joe
```

© 2008 coreservlets.com

Spring JDBC Process

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Spring JDBC Process

- **Define persistence interfaces**
- **Register Spring JDBC JARs**
  - Compilation dependencies
    - spring-jdbc.jar
    - spring-tx.jar
  - Runtime dependencies
    - spring-core.jar
    - spring-context.jar
    - spring-beans.jar
    - commons-logging.jar

Spring JDBC Process Continued

- **Create persistence implementations**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Initialize and execute the persistence objects**
  - Instantiate the persistence objects
    - Inject a DataSource object for connectivity

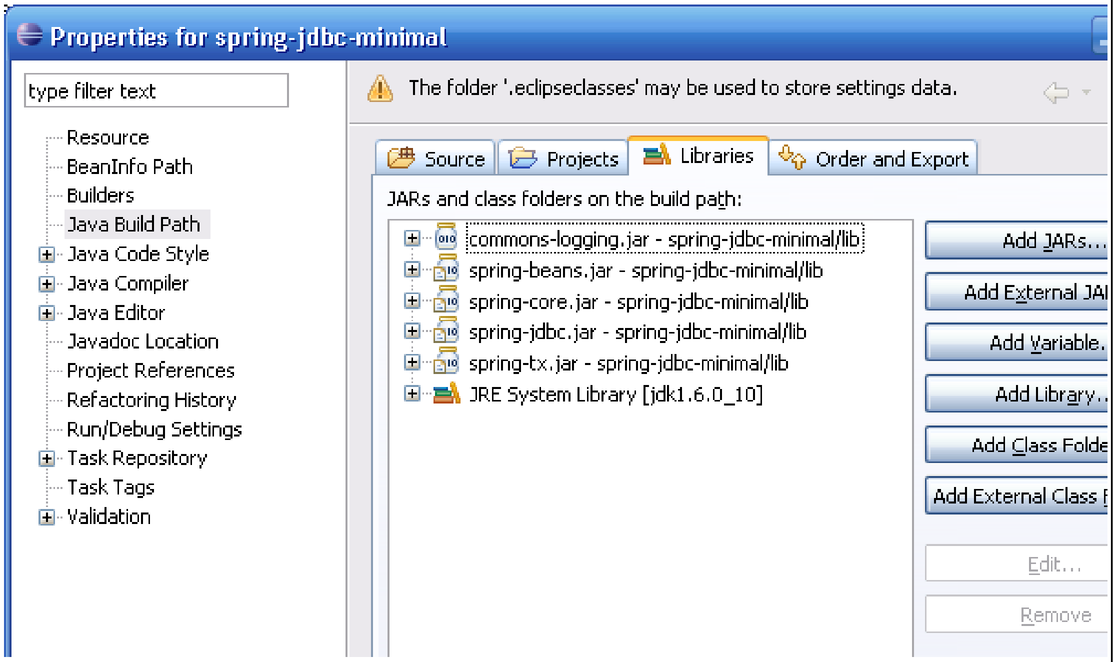
Develop Persistence Interfaces

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}
```

Develop Persistence Interfaces

```
package coreservlets;

public class Customer {

  private String id;

  private String name;

  public Customer(String id, String name){
    this.id = id;
    this.name = name;
  }
  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
}
```

Register Spring JDBC JARs



The folder '.eclipseclasses' may be used to store settings data.

JARs and class folders on the build path:

- commons-logging.jar - spring-jdbc-minimal/lib
- spring-beans.jar - spring-jdbc-minimal/lib
- spring-core.jar - spring-jdbc-minimal/lib
- spring-jdbc.jar - spring-jdbc-minimal/lib
- spring-tx.jar - spring-jdbc-minimal/lib
- JRE System Library [jdk1.6.0_10]

Register Spring JDBC JARs Eclipse .classpath

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry kind="src" path="src"/> <classpathentry
   kind="lib" path="lib/spring-core.jar" />
  <classpathentry kind="lib" path="lib/spring-beans.jar" />
  <classpathentry kind="lib" path="lib/spring-jdbc.jar" />
  <classpathentry kind="lib" path="lib/spring-tx.jar" />
  <classpathentry kind="lib" path="lib/commons-logging.jar"/>
<classpathentry kind="con"
    path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="output" path=".eclipseclasses"/>
</classpath>
```

Implement Persistence Class

```
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.sql.DataSource;

import org.springframework.dao.*;
import org.springframework.jdbc.core.simple.*;

public class SpringJdbcCustomerQuery
implements CustomerQuery {

  private SimpleJdbcTemplate jdbc;

  public SpringJdbcCustomerQuery(DataSource dataSource) {
    this.jdbc = new SimpleJdbcTemplate(dataSource);
  }
  ...
}
```

Implement Persistence Class Continued

```
public class SpringJdbcCustomerQuery
implements CustomerQuery {
  ...
  public Customer getCustomerByName(String customerName) {
    try{
      return this.jdbc.queryForObject(
        "select id, name from customer where name = ?"
        , customerRowMapper
        , customerName);
    }
    catch(EmptyResultDataAccessException e){
      return null;
    }
  }
  ...
}
```

Initialize and Execute Persistence Objects

```
public class Main {
  public static void main(String[] args) throws Exception {
    DataSource dataSource =
      new EmbeddedDerbyDataSource(
        "target/ngcdb", "/setup.sql");
    CustomerQuery query =
      new SpringJdbcCustomerQuery(dataSource);
    Customer customer =
      query.getCustomerByName("Java Joe");
    System.out.println(customer);
  }
}
```

Standard output

```
Customer id=jjoe, name=Java Joe
```

© 2008 coreservlets.com

Spring JDBC and Spring IoC Process

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Spring JDBC and Spring IoC Process

- **Register Spring JARs**
  - spring-core.jar
  - spring-context.jar
  - spring-beans.jar
  - spring-jdbc.jar
  - spring-tx.jar
  - commons-logging.jar
- **Develop persistence interfaces**
- **Create the persistence implementations**
  - Defer connectivity responsibilities
    - Design class for `DataSource` dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected `DataSource`
- **Create the bean definitions file**
  - Default to the file name `applicationContext.xml`
  - Place the file in an accessible location

Spring JDBC and Spring IoC Process Continued

- **Register beans**
  - Register the spring-beans XML schema into the bean definitions file Assign bean identifiers using the **`id`** attribute
    - **Register a DataSource bean**
    - **Register persistence implementation beans**
  - Specify the bean creation method; e.g, the **`class`** attribute for direct constructor invocation
- **Integrate configuration**
  - Register the **`spring-context`** XML schema into the bean definitions file
  - Declare a **`property-placeholder`** element with the configuration file path assigned to the **`location`** attribute
    - **Map DataSource configuration into DataSource**
- **Define bean interdependencies**
  - Select a DI method; e.g., constructor, property setter, lookup-method, etc…
  - Specify the injection value; e.g., collaborators, values, resources, etc…
    - **Register the DataSource with persistence implementations**

Spring JDBC and Spring IoC Process Continued

- **Initialize container**
  - Select an ApplicationContext implementation
    - The integration method will depend on the target environment
  - Specify the location of the bean definitions file(s)
- **Access and use beans from the Spring IoC container**
  - For example, via the BeanFactory API
    - **`BeanFactory#getBean(beanName:String):Object`**
    - **`BeanFactory#getBean(beanName:String, requiredType:Class):Object`**

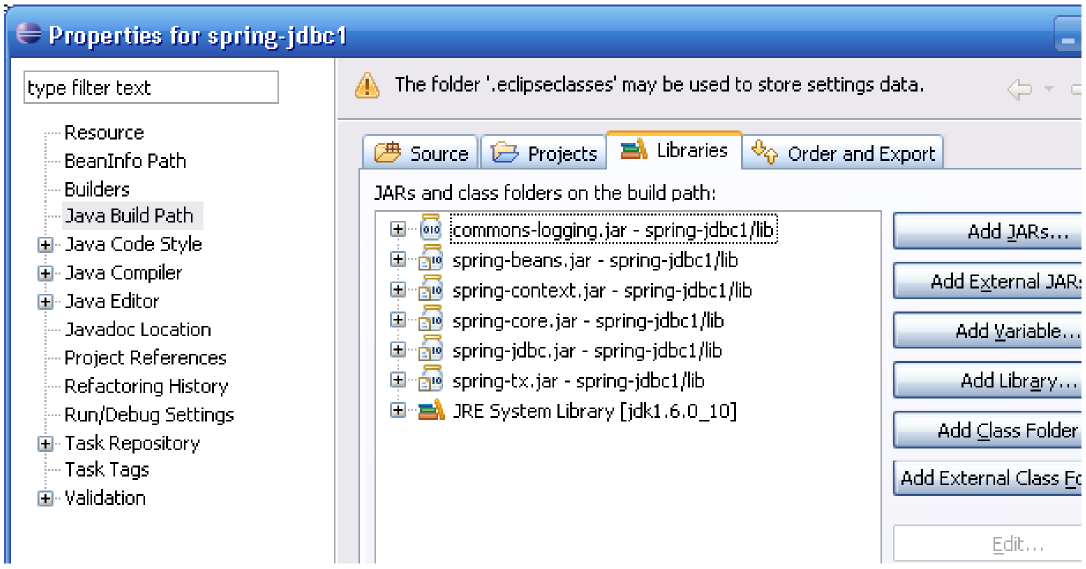
Develop Persistence Interfaces

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}
```

Develop Persistence Interfaces

```
package coreservlets;

public class Customer {

  private String id;

  private String name;

  public Customer(String id, String name){
    this.id = id;
    this.name = name;
  }
  public String getId() {
    return id;
  }
  public String getName() {
    return name;
  }
}
```

Register Spring JDBC JARs



The folder '.eclipseclasses' may be used to store settings data.

JARs and class folders on the build path:

- commons-logging.jar - spring-jdbc1/lib
- spring-beans.jar - spring-jdbc1/lib
- spring-context.jar - spring-jdbc1/lib
- spring-core.jar - spring-jdbc1/lib
- spring-jdbc.jar - spring-jdbc1/lib
- spring-tx.jar - spring-jdbc1/lib
- JRE System Library [jdk1.6.0_10]

Register Spring JDBC JARs Eclipse .classpath

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
  <classpathentry kind="src" path="src"/>
  <classpathentry kind="lib" path="lib/spring-core.jar" />
  <classpathentry kind="lib" path="lib/spring-beans.jar" />
  <classpathentry kind="lib" path="lib/spring-context.jar" />
  <classpathentry kind="lib" path="lib/spring-jdbc.jar" />
  <classpathentry kind="lib" path="lib/spring-tx.jar" />
  <classpathentry kind="lib" path="lib/commons-logging.jar"/>
  <classpathentry kind="con"
    path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
  <classpathentry kind="output" path=".eclipseclasses"/>
</classpath>
```

Implement Persistence Class

```
import java.sql.ResultSet;
import java.sql.SQLException;

import javax.sql.DataSource;

import org.springframework.dao.*;
import org.springframework.jdbc.core.simple.*;

public class SpringJdbcCustomerQuery
implements CustomerQuery {

  private SimpleJdbcTemplate jdbc;

  public SpringJdbcCustomerQuery(DataSource dataSource) {
    jdbc = new SimpleJdbcTemplate(dataSource);
  }
  ...
}
```

Implement Persistence Class Continued

```
public class SpringJdbcCustomerQuery
implements CustomerQuery {
  ...
  private ParameterizedRowMapper<Customer> customerRowMapper =
    new ParameterizedRowMapper<Customer>(){
      public Customer mapRow(ResultSet rslt, int rowNum)
          throws SQLException {
        return new Customer(rslt.getString("id"),
                            rslt.getString("name"));
      }
    };
  ...
}
```

Implement Persistence Class Continued

```
public class SpringJdbcCustomerQuery
implements CustomerQuery {
  ...
  public Customer getCustomerByName(String customerName) {
    try{
      return jdbc.queryForObject(
        "select id, name from customer where name = ?"
        , customerRowMapper
        , customerName);
    }
    catch(EmptyResultDataAccessException e){
      return null;
    }
  }
  ...
}
```

Create the bean definitions file

- **Default to `applicationContext.xml`**
- **Place the file in an accessible location**
  - Classpath, filesystem or web module path

Register Beans

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

</beans>
```

Register Beans

```
<?xml version="1.0" encoding="UTF-8"?>
<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="customerQuery"
        class="coreservlets.SpringJdbcCustomerQuery" />

  <bean id="dataSource"
        class="coreservlets.util.EmbeddedDerbyDataSource" />

</beans>
```

Integrate Configuration

- Create the properties file
  - `dataSource.properties`
- Place the file in an accessible location
  - classpath root

Integrate Configuration

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

   <bean id="customerQuery"
         class="coreservlets.SpringJdbcCustomerQuery" />

   <bean id="dataSource"
         class="coreservlets.util.EmbeddedDerbyDataSource" />

</beans>
```

Integrate Configuration

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

  <context:property-placeholder
    location="classpath:/dataSource.properties" />

  <bean id="customerQuery"
        class="coreservlets.SpringJdbcCustomerQuery" />

  <bean id="dataSource"
        class="coreservlets.util.EmbeddedDerbyDataSource" />
</beans>
```

Define Bean Interdependencies

```
<?xml version="1.0" encoding="UTF-8"?>
<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="customerQuery" class="coreservlets.SpringJdbcCustomerQuery">
    <constructor-arg ref="dataSource" />
  </bean>

  <bean id="dataSource" class="coreservlets.util.EmbeddedDerbyDataSource">
    <constructor-arg value="${derby.db.name}" />
    <constructor-arg>
      <list>
        <value>${derby.db.setup}</value>
      </list>
    </constructor-arg>
  </bean>
</beans>
```

Initialize Container

```
import org.springframework.context.support.*;

public class Main {

  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/applicationContext.xml");

  }
}
```

Access and Use Beans

```
import org.springframework.context.support.*;
public class Main {
  public static void main(String[]args) {

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(
        "/applicationContext.xml");

    CustomerQuery query =
     (CustomerQuery) beanFactory.getBean("customerQuery");

    Customer customer =
      query.getCustomerByName("Java Joe");

    System.out.println(customer);
  }
}
```

Standard output

```
Customer id=jjoe, name=Java Joe
```

© 2008 coreservlets.com

Wrap-up

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.

Spring JDBC and Spring IoC Process

- **Spring JARs**
  - spring-jdbc.jar, spring-tx.jar, spring-core.jar, spring-context.jar, spring-beans.jar, commons-logging.jar
- **Develop persistence implementations**
  - Initialize JDBC template(s) with a **`DataSource`**
  - Defer connectivity responsibilities. Use constructor or property setter DI integrating with the **`DataSource`**
- **Create `applicationContext.xml`**
  - Save in an accessible location
- **Register and wire beans**
  - Register **`spring-beans`** XML schema
  - Create persistence and **`DataSource`** beans
  - Wire **`DataSource`** bean into persistence beans using constructor or property setter DI

Spring JDBC and Spring IoC Process Continued

- **Integrate configuration**
  - Create and save the properties file in an accessible location
  - Register **`spring-context`** XML schema
  - Create a **`property-placeholder`** declaration with a **`location`** attribute
- **Initialize container**
  - Instantiate a BeanFactory
    - e.g., **`ClassPathXmlApplicationContext`**
- **Access and use beans from the Spring IoC container**
  - Pass the bean name to **`BeanFactory#getBean(beanName:String):Object`**

© 2008 coreservlets.com

Questions?

Customized Java EE Training: http://courses.coreservlets.com/
Servlets, JSP, Struts, JSF/MyFaces/Facelets, Ajax, GWT, Spring, Hibernate/JPA, Java 5 & 6.
Developed and taught by well-known author and developer. At public venues or onsite at *your* location.