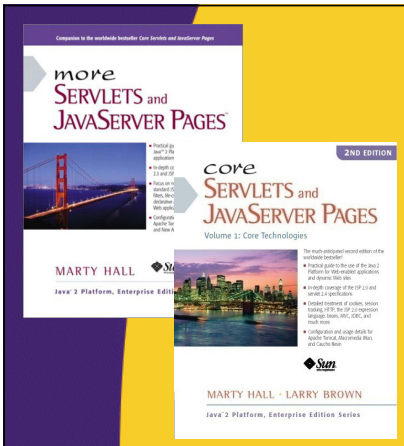
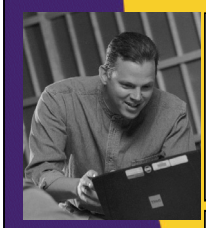
Spring JDBC
Part 2

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



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 APIs
- Result transformations
- Parameter mapping
- Updating tables

Handling JDBC Query Results

Spring JDBC ResultSet Support

- **Transforms information from JDBC `ResultSet` objects into domain types**
  - Hard-coded mapping information
    - Database changes results in code changes
  - No reflection overhead
  - No configuration overhead
- **Spring API offers numerous mapping options**
  - Row to object mapping
  - Typed Row to object mapping
  - Result to object collection mapping

Row to Object Mapping

- **Implemented as a Spring callback interface**
  - **`RowMapper`**
- **Method signature is simple and intuitive**
  - Single method requirement
    - **`mapRow(resultSet:ResultSet,rowNumber:RowNum)#Object`**
  - Only handles rows
  - Stateless callback
    - Supports stateless implementations such as anonymous class implementations
- **Method signature throws `SQLException`**
  - Limited capability for domain exceptions
- **Design is simple but limited**
  - Limited utility without JDBC template
  - No generics support
  - Only supported by **`JdbcTemplate`**

Row to Object Mapping Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Implement callback**
  - Create static **`RowMapper`** reference
  - Implement callback and **`mapRow`** method
- **Integrate callback into business method**
  - Integrate with a JDBC template call

Row to Object Mapping Process Continued

- **Create `applicationContext.xml`**
- **Register beans**
  - **`DAO`** and **`DataSource`** beans
- **Inject dependencies**
  - Specify the **`DataSource`** bean as a **`DAO`** bean dependency
- **Initialize the container**
- **Access and use the DAO bean**

DAO Interface

```
package coreservlets;

import java.util.List;

public interface CustomerListQuery {

  public List<Customer> getCustomers();

}
```


Setup DAO

```
import org.springframework.jdbc.core.*;

public class RowMapperCustomerListQuery
implements CustomerListQuery {

  private JdbcTemplate jdbc;

  public RowMapperCustomerListQuery(DataSource dataSource) {
    jdbc = new JdbcTemplate(dataSource);
  }
  ...
}
```

Implement Callback

```
import org.springframework.jdbc.core.*;

public class RowMapperCustomerListQuery
implements CustomerListQuery {
  ...
  private static final RowMapper customerRowMapper =
      new RowMapper (){

    public Object mapRow(ResultSet rslt, int rowNum)
        throws SQLException {

      return new Customer(rslt.getString("id"),
                          rslt.getString("name"));

    }
  };
}
```


Integrate Callback

```
import org.springframework.jdbc.core.*;

public class RowMapperCustomerListQuery
implements CustomerListQuery {
  ...
  public List<Customer> getCustomers() {
    return jdbc.query(
              "select id, name from customer",
              customerRowMapper);
  }
  ...
}
```

DAO Bean

```
<?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="customerListQuery"
        class="coreservlets.RowMapperCustomerListQuery">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```


DAO Execution

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

public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});
    CustomerListQuery query = (CustomerListQuery)
      beanFactory.getBean("customerListQuery");
    List<Customer>customers = query.getCustomers();
    for(Customer customer : customers) {
      System.out.println(customer);
    }
  }
}
```

Standard output

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

Typed Row to Object Mapping

- **Implemented as a Spring callback interface with generics**
  - **`ParameterizedRowMapper <T>`**
- **Method signature is simple and intuitive**
  - Single method requirement
    - **`mapRow(resultSet : ResultSet, rowNumber : RowNum)#T`**
  - Only handles rows
  - Stateless callback
    - Supports stateless implementations such as anonymous class implementations
- **Method signature throws `SQLException`**
  - Limited capability for domain exceptions

Typed Row to Object Mapping Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Implement callback**
  - Create static **`ParameterizedRowMapper`** reference
  - Implement callback and **`mapRow`** method
- **Implement business method**
  - Integrate callback into JDBC template call

Typed Row to Object Mapping Process Continued

- Create `applicationContext.xml`
- Register beans
  - `DAO` and `DataSource` beans
- Inject dependencies
  - Specify the `DataSource` bean as a `DAO` bean dependency
- Initialize the container
- Access and use the DAO bean

DAO Interface

```
public interface CustomerListQuery {

  public List<Customer> getCustomers();

}
```

Setup DAO

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

public class ParameterizedRowMapperCustomerListQuery
implements CustomerListQuery {

  private SimpleJdbcTemplate simpleJdbc;

  public ParameterizedRowMapperCustomerListQuery(
      DataSource dataSource) {
    simpleJdbc = new SimpleJdbcTemplate(dataSource);
  }
  ...
}
```


Implement Callback

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

public class ParameterizedRowMapperCustomerListQuery
implements CustomerListQuery {
  ...
  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"));

    }
  };
}
```

Integrate Callback

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

public class ParameterizedRowMapperCustomerListQuery
implements CustomerListQuery {
  ...
  public List<Customer> getCustomers() {

    return this.simpleJdbc.<Customer>query(
      "select id, name from customer",
      customerRowMapper);
  }
  ...
}
```


DAO Bean

```
<?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="customerListQuery"
    class="coreservlets.ParameterizedRowMapperCustomerListQuery">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```

DAO Execution

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

public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});
    CustomerListQuery query = (CustomerListQuery)
      beanFactory.getBean("customerListQuery");
    List<Customer>customers = query.getCustomers();
    for(Customer customer : customers) {
      System.out.println(customer);
    }
  }
}
```

Standard output

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


ResultSet to Object Collection Mapping

- **Implemented as a Spring callback interface**
  - `ResultSetExtractor`
- **Method signature is simple and intuitive**
  - Single method requirement
    - `extractData(resultSet : ResultSet, rowNumber : RowNum)#T`
  - Handles full `ResultSet`
  - Stateless callback
    - Supports stateless implementations such as anonymous class implementations
- **Method signature throws SQLException**
  - Limited capability for domain exceptions
- **Design is simple but limited**
  - Limited utility without JDBC template
  - No generics support
  - Only supported by JdbcTemplate

ResultSet to Object Collection Mapping Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Implement callback**
  - Create static **`ResultSetExtractor`** reference
  - Implement callback and **`extractData`** method
- **Implement business method**
  - Integrate callback into JDBC template call

ResultSet to Object Collection Mapping Process Continued

- **Create `applicationContext.xml`**
- **Register beans**
  - **`DAO`** and **`DataSource`** beans
- **Inject dependencies**
  - Specify the **`DataSource`** bean as a **`DAO`** bean dependency
- **Initialize the container**
- **Access and use the DAO bean**

DAO Interface

```
public interface CustomerListQuery {

  public List<Customer> getCustomers();

}
```


Setup DAO

```
import org.springframework.jdbc.core.*;

public class ResultSetExtractorCustomerListQuery
implements CustomerListQuery {

  private JdbcTemplate jdbc;

  public ResultSetExtractorCustomerListQuery(
      DataSource dataSource) {
    jdbc = new JdbcTemplate(dataSource);
  }
  ...
}
```

Implement Callback

```
...
  private static final ResultSetExtractor customerListExtractor =
        new ResultSetExtractor () {

    public Object extractData(ResultSet resultSet)
        throws SQLException {

      List<Customer>list = new ArrayList<Customer>();
      while(resultSet.next()){
        Customer customer = new Customer(
                                resultSet.getString("id"),
                                resultSet.getString("name"));
        list.add(customer);
      }
      return list;
    }
  };
...
```


Integrate Callback

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

public class ResultSetExtractorCustomerListQuery
implements CustomerListQuery {
  ...
  public List<Customer> getCustomers() {
    return (List) this.jdbc.query(
      "select id, name from customer"
      , customerListExtractor);
  }
  ...
}
```

DAO Bean

```
<?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="customerListQuery"
    class="coreservlets.ResultSetExtractorCustomerListQuery">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```


DAO Execution

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

public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});
    CustomerListQuery query = (CustomerListQuery)
      beanFactory.getBean("customerListQuery");
    List<Customer>customers = query.getCustomers();
    for(Customer customer : customers) {
      System.out.println(customer);
    }
  }
}
```

Standard output

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

Passing JDBC Parameters

Spring JDBC Parameter Support

- **Passes Java values to JDBC statement variables**
  - Converts from Java values to JDBC data types
    - Type conversion implied by Java argument type
    - Optionally, explicitly specified via SQL parameter type arrays
  - Assigns values to query variable placeholders
    - Mapping implied by position or named parameters
  - No configuration overhead
    - No XML
    - No annotations
  - Hard-coded settings
    - Database changes results in code changes
- **Spring API offers several mapping option**
  - Simple objects
  - Parameter map objects
  - Spring parameter objects

Simple Object Parameters

- **Passes parameter values as plain `java.lang.Object`(s)**
- **Type conversion is implied or explicit**
  - Based on Java argument type
  - Additional arguments can be supplied specifying the type
    - **`JdbcTemplate`** feature only
- **Value assignment is implied**
  - Based on argument position
- **JDBC template support**
  - **`JdbcTemplate#query`**
  - **`SimpleJdbcTemplate#query`**

Simple Object Parameter Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Implement callback**
- **Implement business method**
  - Integrate callback into JDBC template call
  - Integrate object parameters into JDBC template call
    - Create variable placeholders in SQL
    - Map object arguments according to placeholders

Simle Object Parameter Process Continued

- Create `applicationContext.xml`
- Register beans
  - `DAO` and `DataSource` beans
- Inject dependencies
  - Specify the `DataSource` bean as a `DAO` bean dependency
- Initialize the container
- Access and use the DAO bean

DAO Interface

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}
```

Setup DAO

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

public class ObjectParameterCustomerQuery
implements CustomerQuery {

  private SimpleJdbcTemplate simpleJdbc;

  public ObjectParameterCustomerQuery(DataSource dataSource) {
    simpleJdbc = new SimpleJdbcTemplate(dataSource);
  }
  ...
}
```


Implement Callback

```
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"));

    }
  };
```

Integrate Object Parameters

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

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


DAO Bean

```
<?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.ObjectParameterCustomerQuery">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```

DAO Execution

```
import org.springframework.beans.factory.*;
import org.springframework.context.support.*;
public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});

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

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

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

Standard output

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


Parameter Map Objects

- **Passes parameter names and values as maps**
  - Enabled by parameter naming support
  - Map keys are parameter names
  - Map values are parameter values
- **Type conversion is implied**
  - Based on Java argument type
- **Value assignment is explict**
  - Mapped to named parameters
- **JDBC template support**
  - **`NamedParameterJdbcTemplate#query`**
  - **`SimpleJdbcTemplate#query`**

Parameter Map Object Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Implement callback**
- **Implement business method**
  - Integrate callback into JDBC template call
  - Integrate object parameters into JDBC template call
    - Create named variable placeholders in SQL command string
      - Replace conventional variable placeholders (**`?`**) with a variable name prefixed with a colon (**`:namedParameter`**)
    - Convert parameters into a map object
    - Integrate map into the JDBC template call

Parameter Map Object Process Continued

- **Create `applicationContext.xml`**
- **Register beans**
  - **`DAO`** and **`DataSource`** beans
- **Inject dependencies**
  - Specify the **`DataSource`** bean as a **`DAO`** bean dependency
- **Initialize the container**
- **Access and use the DAO bean**

DAO Interface

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}
```


Setup DAO

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

public class MapParameterCustomerQuery
implements CustomerQuery {

  private SimpleJdbcTemplate simpleJdbc;

  public MapParameterCustomerQuery(DataSource dataSource) {
    simpleJdbc = new SimpleJdbcTemplate(dataSource);
  }
  ...
}
```

Implement Callback

```
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"));

    }
  };
```


Convert Parameters to Map Object

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

public class MapParameterCustomerQuery
implements CustomerQuery {
  ...
  private Map<String, Object> parameterize(String customerName) {

    Map<String, Object>parameterMap =
      new HashMap<String, Object>();

    parameterMap.put("customerName", customerName);

    return parameterMap;
  }

}
```

Integrate Parameter Map Object

```
import org.springframework.jdbc.core.simple.*;
public class MapParameterCustomerQuery
implements CustomerQuery {
  ...
  public Customer getCustomerByName(String customerName) {
    try{
      Map<String, Object>parameterMap =
        parameterize(customerName);
      return simpleJdbc.queryForObject(
        "select id, name from customer"
        + " where name = :customerName",
        customerRowMapper,
        parameterMap);
    }
    catch(EmptyResultDataAccessException e){
      return null;
    }
  }
}
```


DAO Bean

```
<?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.MapParameterCustomerQuery">
    <constructor-arg ref="dataSource" />
  </bean>
</beans>
```

DAO Execution

```
import org.springframework.beans.factory.*;
import org.springframework.context.support.*;
public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});

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

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

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

Standard output

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


Spring Parameter Object

- **Passes parameter names, values and/or types using a custom Spring type**
  - Enabled by parameter naming support
  - Associates a parameter name with a value
  - Associates a parameter name with a type
- **Type conversion is implied or explicit**
  - Based on Java argument value type
  - Based on explicit type setting
- **Value assignment is explict**
  - Mapped to named parameters
- **JDBC template support**
  - **`NamedParameterJdbcTemplate#query`**
  - **`SimpleJdbcTemplate#query`**

Spring Parameter Object Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Implement callback**
- **Implement business method**
  - Integrate callback into JDBC template call
  - Integrate object parameters into JDBC template call
    - Create named variable placeholders in SQL
    - Convert parameters into a **`SqlParameterSource`** object
    - Integrate Spring type into a JDBC template call

Spring Parameter Object Process

- **Create `applicationContext.xml`**
- **Register beans**
  - **`DAO`** and **`DataSource`** beans
- **Inject dependencies**
  - Specify the **`DataSource`** bean as a **`DAO`** bean dependency
- **Initialize the container**
- **Access and use the DAO bean**

DAO Interface

```
public interface CustomerQuery {

  public Customer getCustomerByName(String name);

}
```


Setup DAO

```
import org.springframework.jdbc.core.namedparam.*;
import org.springframework.jdbc.core.simple.*;

public class SqlParameterSourceCustomerQuery
implements CustomerQuery {

  private SimpleJdbcTemplate simpleJdbc;

  public SqlParameterSourceCustomerQuery
                              (DataSource dataSource) {
    simpleJdbc = new SimpleJdbcTemplate(dataSource);
  }  ...
}
```

Implement Callback

```
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"));

    }
  };
```


Convert Parameters to Spring Parameter Object

```
import org.springframework.jdbc.core.namedparam.*;
import org.springframework.jdbc.core.simple.*;

public class SqlParameterSourceCustomerQuery
implements CustomerQuery {
  ...
  private SqlParameterSource parameterize(String customerName) {

    MapSqlParameterSource parameterMap =
       new MapSqlParameterSource();

    parameterMap.addValue("customerName",
                          customerName,
                          Types.VARCHAR);

    return parameterMap;
  }
}
```

Integrate Spring Parameter Object

```
import org.springframework.jdbc.core.simple.*;
public class MapParameterCustomerQuery
implements CustomerQuery {
  ...
  public Customer getCustomerByName(String customerName) {
    try{
      SqlParameterSource parameterMap =
        parameterize(customerName);
      return this.jdbc.queryForObject(
        "select id, name from customer"
        + " where name = :customerName",
        customerRowMapper,
        parameterMap);
    }
    catch(EmptyResultDataAccessException e){
      return null;
    }
  }
}
```


DAO Bean

```
<?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.SqlParameterSourceCustomerQuery">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```

DAO Execution

```
import org.springframework.beans.factory.*;
import org.springframework.context.support.*;
public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});

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

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

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

Standard output

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


Spring Query Objects

Spring Query Object

- **JDBC query template**
  - Stores the SQL command
  - Implementor defines result set to domain type transformation
  - Reusable and thread-safe
- **Parameter passing**
  - Supports objects and maps
  - Types are set explicitly
- **Standalone implementation**
  - Depends on a DataSource and a SQL statement
  - Separate JDBC template is unnecessary

Spring Query Object Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for `DataSource` dependency injection
- **Do not initialize `JdbcTemplate`, `SimpleJdbcTemplate`, or implement a callback**
- **Implement Spring query object**
  - Extend `MappingSqlQuery`
  - Specify SQL and variable types
  - Implement `mapRow` method
  - Initialize and cache the query object from the DAO constructor body

Spring Query Object Process Continued

- **Implement business method**
  - Integrate Spring query object
- **Create `applicationContext.xml`**
- **Register beans**
  - **`DAO`** and **`DataSource`** beans
- **Inject dependencies**
  - Specify the **`DataSource`** bean as a **`DAO`** bean dependency
- **Initialize the container**
- **Access and use the DAO bean**

Setup DAO

```
import org.springframework.jdbc.core.*;
import org.springframework.jdbc.object.*;

public class QueryObjectCustomerQuery
implements CustomerQuery {

  public QueryObjectCustomerQuery(DataSource dataSource) {
  }
  ...
}
```

Implement Spring Query Object

```
class CustomerMappingSqlQuery extends MappingSqlQuery{

  public CustomerMappingSqlQuery(DataSource dataSource) {
    super(dataSource,
      "select id, name from customer where name = ?");
    super.setParameters(
        new SqlParameter[]{new SqlParameter(VARCHAR)});
  }

  protected Object mapRow(ResultSet rslt, int rowNumber)
      throws SQLException {
    return new Customer(rslt.getString("id"),
                        rslt.getString("name"));
  }
}
```


Initialize and Cache Spring Query Object

```
import org.springframework.jdbc.core.*;
import org.springframework.jdbc.object.*;

public class QueryObjectCustomerQuery
implements CustomerQuery {

  private MappingSqlQuery customerMappingSqlQuery;

  public QueryObjectCustomerQuery(DataSource dataSource) {
    customerMappingSqlQuery =
      new CustomerMappingSqlQuery(dataSource);
  }
  ...
}
```

Integrate Spring Query Object

```
public class QueryObjectCustomerQuery
implements CustomerQuery {

  private MappingSqlQuery query;

  public QueryObjectCustomerQuery(DataSource dataSource) {
    query = new CustomerMappingSqlQuery(dataSource);
  }

  public Customer getCustomerByName(String customerName) {
    try{
      return (Customer) query.findObject(customerName);
    }
    catch(EmptyResultDataAccessException e){
      return null;
    }
  }
}
```


DAO Bean

```
<?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.QueryObjectCustomerQuery">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```

DAO Execution

```
import org.springframework.beans.factory.*;
import org.springframework.context.support.*;
public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});

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

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

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

Standard output

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


Modifying the Database

Spring JDBC Table Updates

- **Database update interfaces exposed as JDBC template methods**
  - **`JdbcTemplate#update`**
  - **`SimpleJdbcTemplate#update`**
- **Multiple parameter mapping options**
  - Simple object arrays
    - Optional SQL type settings
  - Parameter map structure
  - Spring parameter object
- **No `ResultSet` transformations**
- **Added responsibility of checking the number of modified rows**

Spring JDBC Table Update Process

- **Setup DAO**
  - Defer connectivity responsibilities
    - Design class for **`DataSource`** dependency injection
  - Use Spring JDBC APIs
    - Initialize Spring JDBC template(s) with the injected **`DataSource`**
- **Implement parameter mapping mechanism**
  - Simple object, map, or Spring parameter object
- **Implement business method**
  - Implement JDBC template call
    - Create SQL command with variable placeholders
    - Handle parameters using parameter mapping mechanism
    - Verify result based on the affected rowcount

Spring JDBC Table Update Process

- Create `applicationContext.xml`
- Register beans
  - `DAO` and `DataSource` beans
- Inject dependencies
  - Specify the `DataSource` bean as a `DAO` bean dependency
- Initialize the container
- Access and use the DAO bean

DAO Interface

```
public interface CustomerUpdate {

  public void save(Customer customer);

  public void deleteById (String customerId);

}
```

Setup DAO

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

public class SpringCustomerUpdate implements CustomerUpdate {

  private SimpleJdbcTemplate simpleJdbc;

  public SpringCustomerUpdate(DataSource dataSource) {
    simpleJdbc = new SimpleJdbcTemplate(dataSource);
  }
  ...
}
```


Implement Parameter Mapping

```
  private Map<String, Object> parameterize(Customer cust) {

    Map<String, Object> parameterMap =
      new HashMap<String, Object>();

    parameterMap.put("customerId", cust.getId());
    parameterMap.put("customerName", cust.getName());

    return parameterMap;

  }
```

Implement Save

```
public void save(Customer customer) {
  Map<String, Object>parameters = parameterize(customer);

  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);
}
```


Implement Delete

```
public void deleteById (String customerId) {

  simpleJdbc.update(
    "delete from customer where id = ?", customerId);

}
```

DAO Bean

```
<?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="customerUpdate"
        class="coreservlets.SpringCustomerUpdate">
    <constructor-arg ref="dataSource" />
  </bean>

  <bean id="customerListQuery"
    class="coreservlets.ParameterizedRowMapperCustomerListQuery">
    <constructor-arg ref="dataSource" />
  </bean>

</beans>
```


Instantiate Container

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

public class Main {

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

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});
  }
}
```

Acquire Beans

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

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

    BeanFactory beanFactory =
      new ClassPathXmlApplicationContext(new String[]{
        "/applicationContext.xml",
        "/dataSourceContext.xml"});

   CustomerUpdate customerUpdate = (CustomerUpdate)
     beanFactory.getBean("customerUpdate");

   CustomerListQuery customerQuery = (CustomerListQuery)
     beanFactory.getBean("customerListQuery");
  }
}
```


Setup Customer Object

```
public class Main {
  public static void main(String[] args) throws Exception {
    BeanFactory beanFactory = ...;
    CustomerUpdate customerUpdate = ...;
    CustomerListQuery customerQuery = ...;

    Customer customer = new Customer();
    customer.setId("jspring");
    customer.setName("Joe Spring");
  }
}
```

Save Customer Object

```
public class Main {
  public static void main(String[] args) throws Exception {
    ...
    Customer customer = new Customer();
    customer.setId("jspring");
    customer.setName("Joe Spring");

    customerUpdate.save(customer);
    System.out.println("After initial save : " +
      customerQuery.getCustomers());
  }
}
```

Standard output

```
After initial save : [Customer id=jspring, name=Joe Spring]
```


Update Customer Object

```
public class Main {
  public static void main(String[] args) throws Exception {
    ...
    Customer customer = new Customer();
    customer.setId("jspring");
    customer.setName("Joe Spring");

    customerUpdate.save(customer);
    System.out.println("After initial save : " +
      customerQuery.getCustomers());

    customer.setName("Joseph Spring");
    customerUpdate.save(customer);
    System.out.println("After update       : " +
      customerQuery.getCustomers());
  }
}
```

Standard output

```
After initial save : [Customer id=jspring, name=Joe Spring]
After update       : [Customer id=jspring, name=Joseph Spring]
```

Delete Customer Object

```
public class Main {
  public static void main(String[] args) throws Exception {
    ...
    customerUpdate.save(customer);
    System.out.println("After initial save : " +
      customerQuery.getCustomers());

    customer.setName("Joseph Spring");
    customerUpdate.save(customer);
    System.out.println("After update       : " +
      customerQuery.getCustomers());

    customerUpdate.deleteById(customer.getId());
    System.out.println("After delete       : " +
      customerQuery.getCustomers());
  }
}
```

Standard output

```
After initial save : [Customer id=jspring, name=Joe Spring]
After update       : [Customer id=jspring, name=Joseph Spring]
After update       : []
```


Wrap-up

Summary

- **Steps for Spring JDBC**
  - Initialize **`JdbcTemplate`** and/or **`SimpleJdbcTemplate`**
  - Define SQL statement
    - Traditional placeholder (**`?`**) or named parameters (**`:namedParameter`**)
  - Implement parameter mapping
    - Simple object (**`java.lang.Object`**), map (**`java.util.Map`**) or Spring parameter object (**`SqlParameterSource`**)
    - Overloaded JDBC template methods or Spring parameter object to explicitly set SQL types
- **Process `ResultSet` objects for queries**
  - Row to collection - **`RowMapper`** with **`mapRow`** method
  - Row to typed collection - **`ParameterizedRowMapper<T>`** with **`mapRow#T`** method
  - **`ResultSet`** to collection - **`ResultSetExtractor`** with **`extractData`** method
- **Inspect the modified row count for table updates**
  - Insert, update and delete statements via template **`update`** methods

Summary Continued

- **Create `applicationContext.xml`**
- **Register beans**
  - **`DAO`** and **`DataSource`** beans
- **Inject dependencies**
  - Specify **`DataSource`** bean as a **`DAO`** bean dependency
- **Initialize the container**
- **Access and use the `DAO` beans**

Questions?