

# Tutorial de NetBeans IDE Acessando banco de dados com Struts Data Source

## *Configurações e Software*

*Antes de começar a escrita do código, você tem que certificar-se que tem todo o software necessário e que seu projeto está ajustado corretamente.*

### *Instalando o software*

*Antes que você comece, você necessita instalar o seguinte software em seu computador:*

- NetBeans IDE 5.x ([download](#)).
- Versão padrão 1.4.2 ([download](#)) ou 5.0 Java (JDK™) ([download](#))
- Neste tutorial, você usa uma base de dados do Derby de exemplo que vem com o Sun Java System Application Server 8.2
- Para começar origens de dados dos suportes trabalhar corretamente, você necessita fazer exame de uma etapa adicional. Você necessita commons-pooling.jar, commons-dbcp.jar, e common-collections.jar.

1. Ir à página do [Apache Commons Download](#)
2. Download commons-pooling.jar, commons-dbcp.jar, e common-collections.jar
3. Adicionar os JAR's no diretório common/lib do Tomcat

# 1 - Trabalhando com Struts Data Source : Um Cenário simples

Primeiramente, você define um Struts Data Source . Em seguida, você cria um Struts Action para acessar o Data Source . Finalmente, você cria uma página JSP para renderizar os resultados.

Criando o Data Source

Adicionar o Data Source no arquivo `struts-config.xml`. Abaixo o código a adicionar no arquivo :

```
<data-sources>
  <data-source type="org.apache.commons.dbcp.BasicDataSource"
key="empTable">
    <set-property property="driverClassName"
value="org.apache.derby.jdbc.ClientDriver" />
    <set-property property="url"
value="jdbc:derby://localhost:1527/sample" />
    <set-property property="username" value="app" />
    <set-property property="password" value="app" />
    <set-property property="validationQuery" value="SELECT * FROM
CUSTOMER" />
  </data-source>
</data-sources>
```

Nota: O atributo `KEY` é muito importante. Você irá utilizar na classe Struts Action conforme a linha abaixo:

```
dataSource = (DataSource) servlet.getServletContext ().getAttribute
("empTable");
```

Alcançando a origem dos dados dos suportes de uma classe da ação dos suportes

1. Na janela dos projetos, o clique com o botão direito em `com.myapp.struts`, escolha novo > classe de Java, e nomeie-o Row . A classe de `Row.java` representará os registro da tabela. . No editor da fonte, adicionar dois atributos, `name` e `city` , como mostrado abaixo:

```
package com.myapp.struts;

public class row {

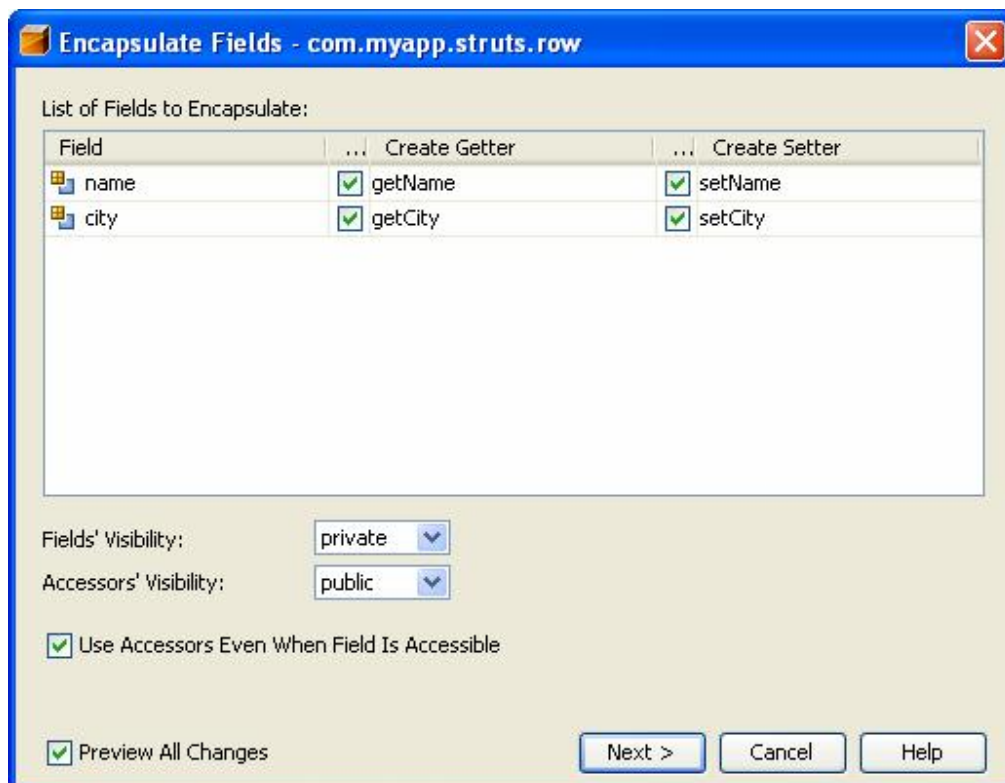
    String name;
    String city;
```

```

    /** Creates a new instance of row */
    public row() {
    }
}

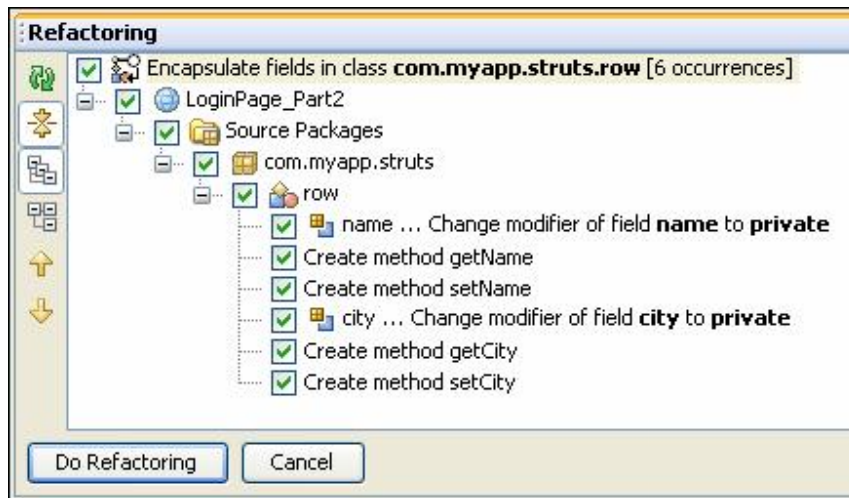
```

2. Click com o botão direito em qualquer lugar no corpo da classe e escolha Refactor > Encapsulate Fields.



Manter os valores e o clique em Next.

No fundo do IDE, a janela de Refactoring propõe as mudanças:



Manter os valores de defeito e clique em Do Refactoring.

3. Preencher o construtor como mostrado abaixo (o código destacado abaixo é a única parte que foi mudada):

```
package com.myapp.struts;

public class row {

    private String name;
    private String city;

    /** Creates a new instance of Row */
    public row(String name, String ci ty) {
        thi s. name = name;
        thi s. ci ty = ci ty;
    }

    public String getName() {
        return name;
    }

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

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

}
```

4. Na janela dos projetos, o clique com o botão direito em `com.myapp.struts`, escolha `New > File/Folder`. Na categoria `Web`, selecione `Struts Action`. Nomear a classe `DataSourceConnectionAction.java` da `Action Struts`.
5. No editor da fonte, preencher a classe de `DataSourceConnectionAction.java` como mostrado abaixo. Onde relevante, os comentários in-line foram adicionados ao código abaixo, para explicar sua finalidade:

```
package com.myapp.struts;

import javax.servlet.http.*;
import org.apache.struts.action.*;
import java.sql.*;
import java.util.ArrayList;
import javax.sql.*;

public class DataSourceConnectionAction extends Action {
    private DataSource dataSource;
    public ArrayList customerList = new ArrayList();
    private final static String SUCCESS = "success";

    public ActionForward execute(ActionMapping mapping, ActionForm
form,
                                HttpServletRequest request, HttpServletResponse response)
                                throws Exception {

        HttpSession session = request.getSession();
        /** Aqui o metodo de conexão ao Data Source é chamado */
        customerList = getCustomers() ;
        /** Aqui adicionado o customerList no scope, para utilizar no
jsp */
        if(customerList != null){
            session.setAttribute("allMyCustomers", customerList);
        }
        return (mapping.findForward(SUCCESS));
    }

    private ArrayList getCustomers(){
        Connection conn = null;
        Statement stmt = null;
        PreparedStatement prpStmt = null;
        ResultSet rs = null;
        StringBuffer resultString ;

        try{
            /** Aqui recuperamos o datasource atraves da chave
'empTable' mapeada em struts-config.xml: */
            dataSource =
(DataSource)servlet.getContext().getAttribute("empTable");

            conn = dataSource.getConnection();
            String sqlQuery = "SELECT * FROM CUSTOMER";
            prpStmt = conn.prepareStatement(sqlQuery);
            rs = prpStmt.executeQuery();
```

```

        /** Aqui adicionamos o campo da posição 4 do resultSet ao
        atributo name e o campo da posição 7 ao atributo city */
        while (rs.next()) {
            customerList.add(new row(rs.getString(4),
rs.getString(7)));
        }
        rs.close();

    } catch ( SQLException e ) {
        System.err.println("SQL Exception occurred while accessing
the table" );
        e.printStackTrace();
        return null;

    } catch ( Exception e ) {
        e.printStackTrace();
        return null;
    }

    return customerList;
}
}

```

6. Na janela dos projetos, expandir o Configuration Files e modificar a seção dos mappings no arquivo struts-config.xml para registrar a nova Action. Abaixo, as partes destacadas de código são as únicas partes de traçar da ação que foram modificadas:

```

<action-mappings>

    <action input="/loginForm.jsp" name="LoginActionForm" path="/login"
scope="request"
        type="com.myapp.struts.DataSourceConnectionAction">

        <forward name="success" path="/mydatasourceSuccessful.jsp"
redirect="true"/>
        <forward name="cancel" path="/loginCancel.html"/>
    </action>

    <action forward="/loginOut.jsp" path="/logout"/>

    <action path="/Welcome" forward="/welcomeStruts.jsp"/>

</action-mappings>

```

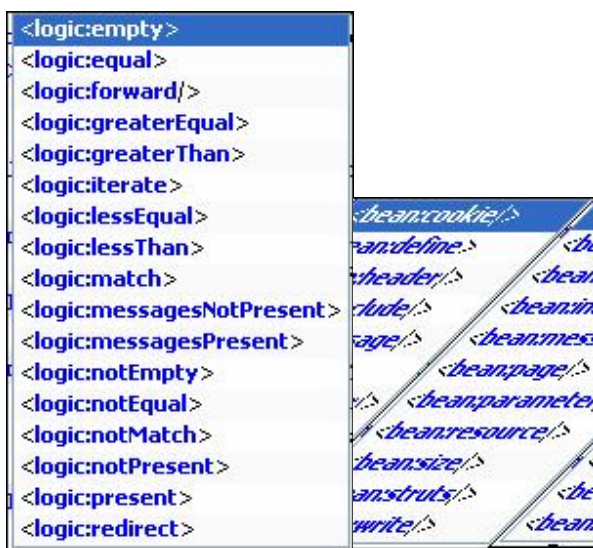
## Usando Tag do Struts para renderizar as informações

O Struts fornecem a tag library `logic` para paginas JSP. O Tag da `logic` permite que você inclua a lógica de exposição em uma página de JSP sem adicionar o código Java como `scriptlets`. Isto faz sua página de JSP mais legível e permite um programador não-Java trabalhar na vista de sua aplicação. O Struts fornecem também o `bean` tag library para recuperar e atribuir dados.

1. Na pagina Web Pages , criar uma nova JSP com o nome `mydatasourceSuccessful.jsp`, para exibir os dados recuperados.
2. Adicionar as seguintes taglib no alto da página:

```
<%@ taglib uri="http://jakarta.apache.org/struts/tags-logic"
prefix="logic" %>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-bean"
prefix="bean" %>
```

Agora, quando você pressiona o Ctrl-Espaço em uma arquivo JSP, todos as tag de `logic` tags e `bean` tags é indicados:



3. Adicionar o texto "Successful data connection!" entre as tags `<title>` e `<h1>`, como mostrado abaixo:

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
  <title>Successful data connection!</title>
</head>
```

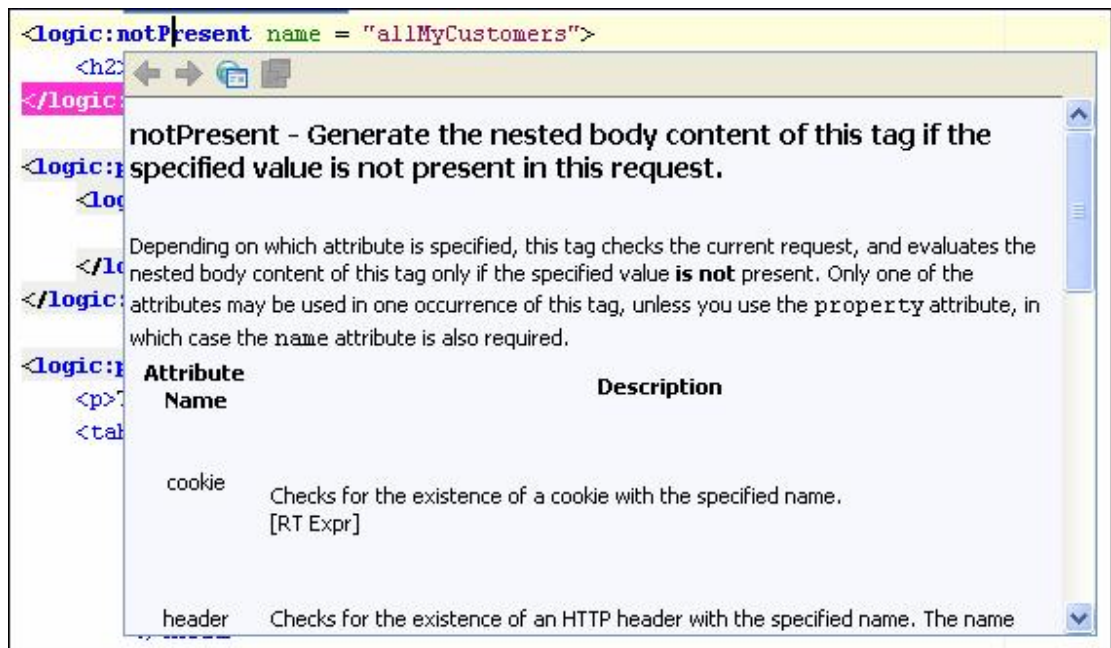
```
<body>
```

```
    <h1>Successful data connection!</h1>
```

4. Abaixo da tag <h1>, adicione <logic:notPresent> tag, como mostrado abaixo:

```
    <logic:notPresent name = "allMyCustomers">
        <h2>Data source not in scope!</h2>
    </logic:notPresent>
```

Pressionar o Ctrl-Espaço dentro do Tag para ler o Javadoc relacionado:

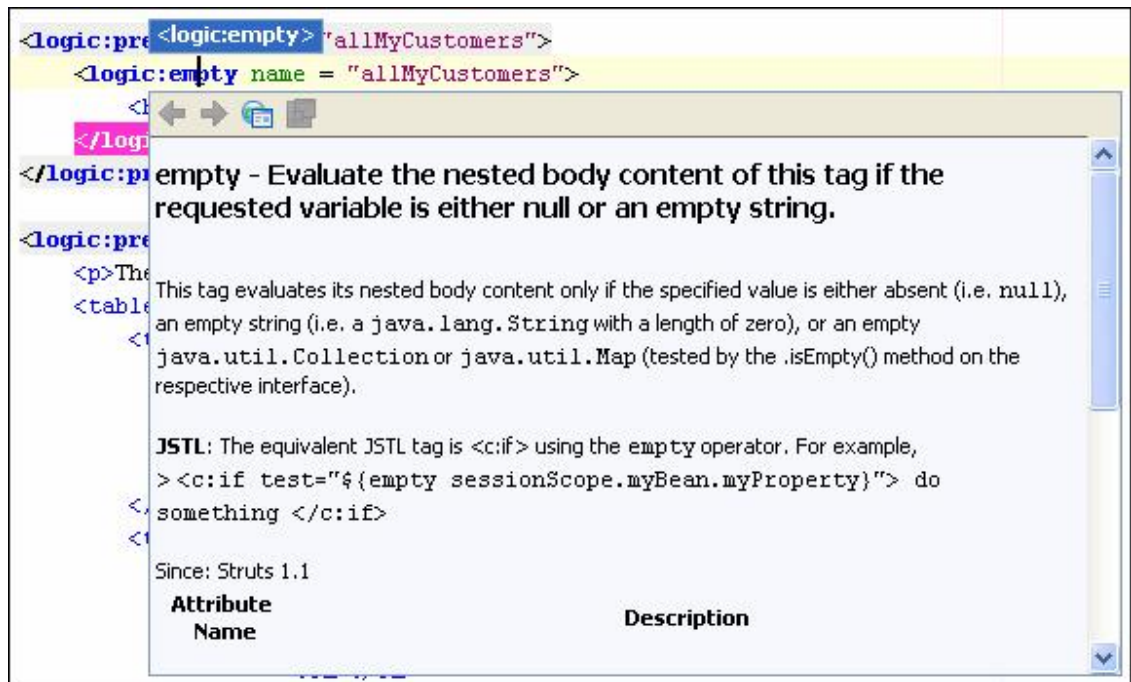


5. Diretamente abaixo da tag <logic:notPresent>, adicione a tag do Struts <logic:present>, with a tag do Struts <logic:empty>, como mostrado abaixo:

```
    <logic:present name = "allMyCustomers">
        <logic:empty name = "allMyCustomers">
            <h2>Data source in scope but no data found!</h2>
        </logic:empty>
    </logic:present>
```

Pressionar o Ctrl-Espaço dentro do Tag para ler o Javadoc relacionado:





6. Diretamente abaixo das Tag precedentes, adicionar a tag do struts `<logic:present>`. Esta vez, entretanto, você está indo dizer ao Struts o que fazer quando a origem dos dados dos suportes não está vazia. Arrastar do HTML Palette uma tabela para a página de JSP. Utilize a tag do Struts `<logic:iterate>` para iterar os dados e usar a tag do Struts `<bean:write>` para renderizar os resultados, como mostrado abaixo:

```

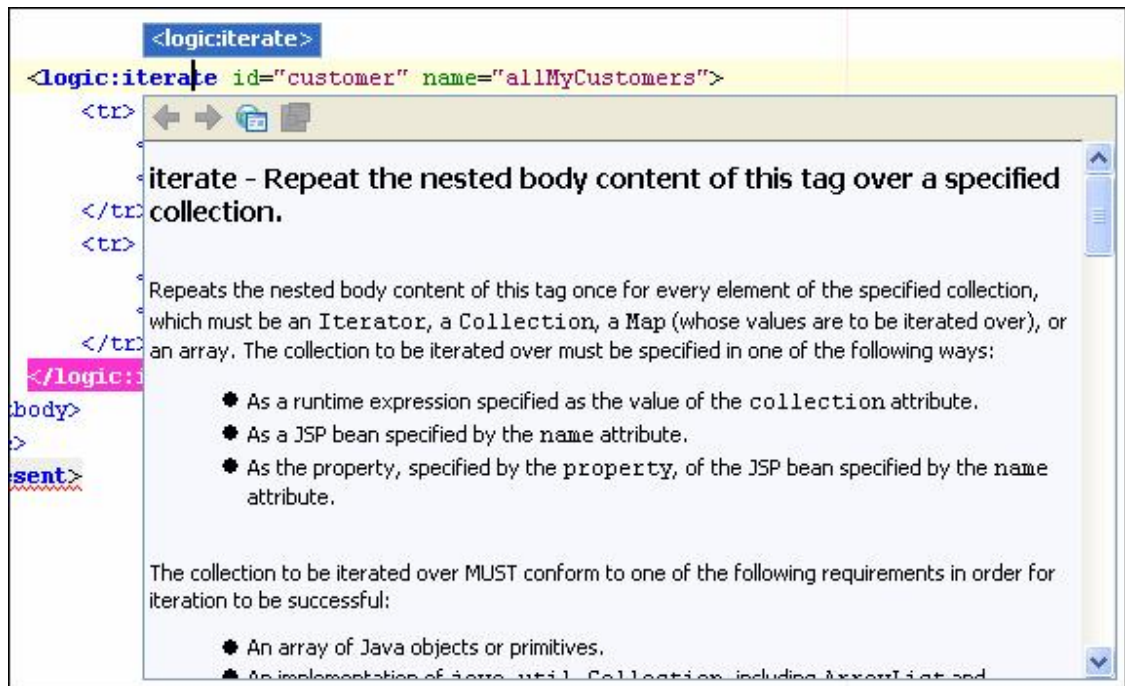
<logic:present name = "allMyCustomers">
  <p>These are our users:</p>
  <table border="1">
    <thead>
      <tr>
        <th>Name</th>
        <th>City</th>
      </tr>
    </thead>
    <tbody>
      <logic:iterate id="customer" name="allMyCustomers">
        <tr>
          <td></td>
          <td></td>
        </tr>
        <tr>
          <td><bean:write name="customer"
property="name" /></td>
          <td><bean:write name="customer"
property="city" /></td>
        </tr>
      </logic:iterate>
    </tbody>
  </table>

```

`</logic:present>`

`<body>`

Pressionar o Ctrl-Espaço dentro dos Tag para ler o Javadoc relacionado:



**<logic:iterate>**

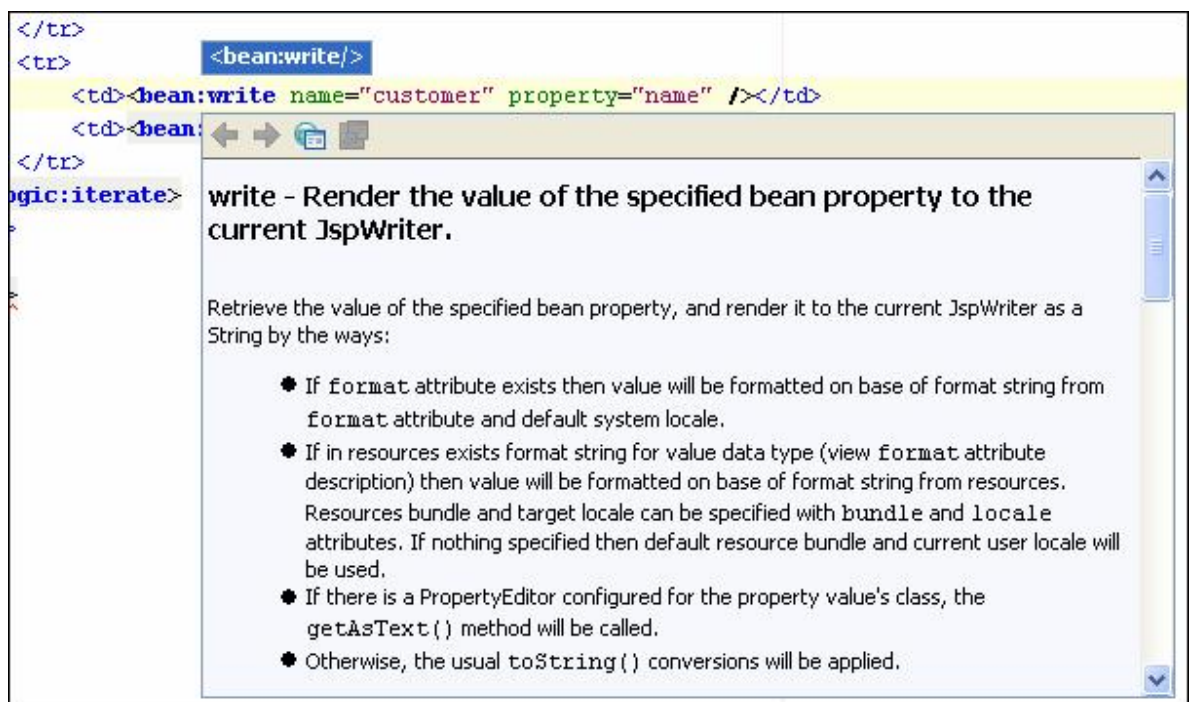
**iterate - Repeat the nested body content of this tag over a specified collection.**

Repeats the nested body content of this tag once for every element of the specified collection, which must be an Iterator, a Collection, a Map (whose values are to be iterated over), or an array. The collection to be iterated over must be specified in one of the following ways:

- As a runtime expression specified as the value of the collection attribute.
- As a JSP bean specified by the name attribute.
- As the property, specified by the property, of the JSP bean specified by the name attribute.

The collection to be iterated over MUST conform to one of the following requirements in order for iteration to be successful:

- An array of Java objects or primitives.
- An implementation of java.util.Collection, including ArrayList and



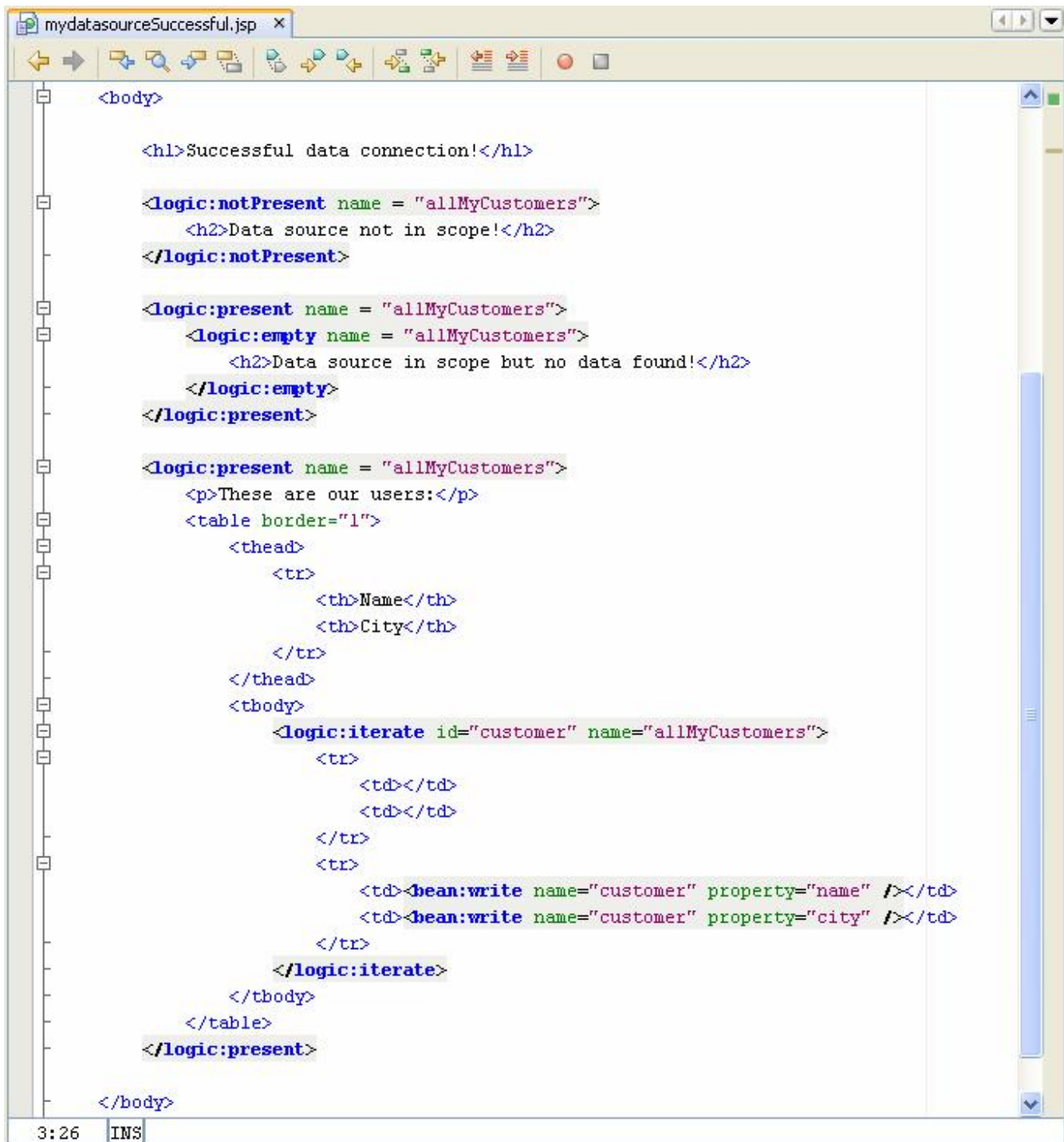
**<bean:write/>**

**write - Render the value of the specified bean property to the current JspWriter.**

Retrieve the value of the specified bean property, and render it to the current JspWriter as a String by the ways:

- If format attribute exists then value will be formatted on base of format string from format attribute and default system locale.
- If in resources exists format string for value data type (view format attribute description) then value will be formatted on base of format string from resources. Resources bundle and target locale can be specified with bundle and locale attributes. If nothing specified then default resource bundle and current user locale will be used.
- If there is a PropertyEditor configured for the property value's class, the getAsText() method will be called.
- Otherwise, the usual toString() conversions will be applied.

7. Certificar-se de que o corpo da página de JSP olhe agora como segue:



```
<body>

    <h1>Successful data connection!</h1>

    <logic:notPresent name = "allMyCustomers">
        <h2>Data source not in scope!</h2>
    </logic:notPresent>

    <logic:present name = "allMyCustomers">
        <logic:empty name = "allMyCustomers">
            <h2>Data source in scope but no data found!</h2>
        </logic:empty>
    </logic:present>

    <logic:present name = "allMyCustomers">
        <p>These are our users:</p>
        <table border="1">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>City</th>
                </tr>
            </thead>
            <tbody>
                <logic:iterate id="customer" name="allMyCustomers">
                    <tr>
                        <td></td>
                        <td></td>
                    </tr>
                    <tr>
                        <td><bean:write name="customer" property="name" /></td>
                        <td><bean:write name="customer" property="city" /></td>
                    </tr>
                </logic:iterate>
            </tbody>
        </table>
    </logic:present>

</body>
```

3:26 INS

## Deploy e Executando

A IDE usa o ANT para realizar o deploy e a execução da aplicação WEB. A IDE gerou o certificado da configuração quando você criou a aplicação, baseando o nas opções que você incorporou ao wizard novo do projeto e à caixa de diálogo das propriedades do projeto do projeto. Click no projeto e selecione Build and run

## 2 - Usando DAO para Encapsular o código de acesso da base de dados

Atualmente, o código que alcança a base de dados é contido dentro da classe Action do Struts. Mesmo que esta seja uma solução trabalhando, não é ótima. Todo o código de acesso da base de dados deve ser encapsulado atrás das classes do API do negócio, usando o padrão do objeto do acesso dos dados (DAO). Para detalhes deste padrão, ver [Core J2EE Patterns - Data Access Object](#).

Realizar o refactoring da aplicação de modo que o código de acesso dos dados seja encapsulado, você necessita fazer o seguinte:

- Criar uma fabrica de DAO.
- Criar uma interface que define os metodos necessarios.
- Criar o DAO implementando a interface e adicionando o acesso a base de dados .
- Adicionar o acesso ao DAO na Action do Struts.
- .

As instruções detalhadas para cada uma das etapas acima são fornecidas abaixo.

1. Para criar a fábrica, criar uma classe nova conforme o código abaixo :

```
package com.myapp.struts;

import java.sql.Connection;

public class DAOFactory {

    /** Creates a new instance of DAOFactory */
    public DAOFactory() {
    }

    public static UserDao createUserDAO(Connection connection){
        return new DerbyUserDAO(connection);
    }
}
```

2. Para criar a interface, criar uma interface nova conforme o código abaixo:

```

package com.myapp.struts;

import java.util.ArrayList;

public interface UserDAO {
    public ArrayList getCustomers();
}

```

3. Para criar o DAO, criar uma classe nova que implemente interface conforme o código abaixo:

```

package com.myapp.struts;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import javax.sql.DataSource;

public class DerbyUserDAO implements UserDAO {

    private Connection connection;
    private DataSource dataSource;

    /** Creates a new instance of DerbyUserDAO */
    public DerbyUserDAO(Connection connection) {
        this.connection = connection;
    }

    public ArrayList getCustomers(){
        ArrayList customerList = new ArrayList();
        try{

            String sqlQuery = "SELECT * FROM CUSTOMER";
            PreparedStatement prpStmt =
connection.prepareStatement(sqlQuery);
            ResultSet rs = prpStmt.executeQuery();

            /** Here we put field 4 (the name) and field 7 (the city)
in the customerList: */
            while (rs.next()) {
                customerList.add(new row(rs.getString(4),
rs.getString(7)));
            }
            rs.close();

        } catch ( SQLException e ) {
            System.err.println("SQL Exception occured while accessing
the table" );
            e.printStackTrace();

```

```

        return null;

    } catch ( Exception e ) {
        e.printStackTrace();
        return null;
    }

    return customerList;
}
}

```

4. Em DataSourceConnectionAction.java, você pode agora remover o método `getCustomers ()`, porque agora será fornecido pelo DAO.

Em seguida, você necessita reescrever esta linha:

```
customerList = getCustomers ();
```

Substituir a linha com este snippet do código:

```

dataSource =
(DataSource)servlet.getServletContext().getAttribute("empTable");
Connection conn = dataSource.getConnection();
UserDAO dao = DAOFactory.createUserDAO(conn);
customerList = dao.getCustomers();

```

A classe inteira de DataSourceConnectionAction.java deve agora ser como o código abaixo :

```

package com.myapp.struts;

import javax.servlet.http.*;
import org.apache.struts.action.*;
import java.sql.*;
import java.util.ArrayList;
import javax.sql.*;

public class DataSourceConnectionAction extends Action {
    private DataSource dataSource;
    public ArrayList customerList = new ArrayList();
    private final static String SUCCESS = "success";

    public ActionForward execute(ActionMapping mapping, ActionForm
form,
        HttpServletRequest request, HttpServletResponse response)
        throws Exception {

        HttpSession session = request.getSession();

```

```

        /** Here the method that connects to the datasource is called:
*/
        dataSource =
(dataSource)servlet.getContext().getAttribute("empTable");
        Connection conn = dataSource.getConnection();
        UserDao dao = DAOFactory.createUserDAO(conn);
        customerList = dao.getCustomers();

        /** Here we put the customerList in scope, so that we can use
it in the JSP page: */
        if(customerList != null){
            session.setAttribute("allMyCustomers", customerList);
        }
        return (mapping.findForward(SUCCESS));
    }
}

```

5. Build e run . O resultado deve ser o mesmo que antes.