Лекция 19

JSTL

Назначение JSTL – упростить разработку (и вид) JSP исходя из того, что не все разработчики JSP владеют Java.

Кроме того, наличие на JSP кода перегружает страницу, делая её не удобной для редактирования.

JSTL предоставляет следующие возможности:

- Поддержка **Expression Language**. Позволяет разработчику писать простые выражения внутри атрибутов тега и предоставляет “прозрачный” доступ к переменным в различных областях видимости.
- Условные переходы и циклы, основанные на тегах, а не на скриптовом языке.
- Простое формирование **URL** к различным ресурсам.
- Интернационализацию **JSP**.
- Взаимодействие с базами данных.
- Обработку **XML**.
- Обработку строк (форматирование и разбор).

Expression language

JSTL вводит понятие **Expression Language (EL)** в **JSP.**

EL используется для упрощения доступа к данным, хранящимся в различных областях видимости (page, request, application) и вычисления простых выражений.

EL вызывается при помощи конструкции **“${}”.**

Начиная с версии спецификации JSP 2.0 / JSTL 1.1, EL является частью JSP и поддерживается безо всяких сторонних библиотек.

Операторы

Операторы, в **EL** поддерживают наиболее часто используемые манипулирования данными.

Типы операторов:

- Стандартные операторы отношения: **==** (или **eq**), **!=** (или **neq**), **<** (или **lt**), **>** (или **gt**), **<=** (или **le**), **>=** (или **ge**).
- Арифметические операторы: **+**, **-**, *, **/** (или **div**), **%** (или **mod**).
- Логические операторы: **&&** (или **and**), || (или **or**), ! (или **not**).
- Оператор **empty** – используется для проверки переменной на null или “пустое значение” (термин “пустое значение” зависит от типа проверяемого объекта. Например, нулевая длина для строки, или нулевой размер для коллекции).

Пример:

```
<c:if test=”${not empty user and user.name eq
                                        ‘guest’}>
  User is guest.
</c:if>
```

Автоматическое приведение типов

EL использует набор правил для автоматического приведения типов. Например, если оператор ожидает параметр типа **Integer**, то значение идентификатора будет приведено к типу **Integer** (если это возможно).

Тэги JSTL

Библиотека тэгов JSTL состоит из четырёх групп тэгов: основные тэги - **core**, тэги форматирования - **fmt**, тэги для работы с SQL – **sql**, тэги для обработки XML – **xml**.

JSTL core

Библиотека **core** содержит в себе наиболее часто используемые теги.

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

- для обычной **JSP**.

```
<jsp:root version=”1.2”
   xmlns:c=”http://java.sun.com/jstl/core”>
   ...
</jsp:root>
```

- для **XML** формата **JSP**.

Тэги общего назначения

<c:out /> - вычисляет и выводит значение выражения.

Пример:

You have

<c:out value="${sessionScope.user.itemCount}"/> items.

По умолчанию, <c:out> конвертирует символы <, >, ‘, “, & в их коды (например, < конвертируется в &lt). Преобразование может быть отменено, если указать false в атрибуте escapeXml.

Можно также в тэге <c:out> указывать значение по умолчанию для случаев, где значение вычисляемого выражения равно null.

Пример:

```
<c:out value="${customer.address.city}"
                                    default="unknown"/>
```

Синтаксис:

Без тела:

```
<c:out value=”value” [escapeXml=”{true|false}”]
 [default=”defaultValue”] />
```

С телом:

```
<c:out value=”value” [escapeXml=”{true|false}”]>
 default value
</c:out>
```

<c:set /> - устанавливает переменную в указанную область видимости.

Пример:

<c:set var=”foo” value=”elexprvalue”/>

<c:set /> может также быть использован для изменений свойств объектов JavaBeans или добавлять элементы в объект java.util.Map.

Синтаксис 1: Установка переменной в указанную область видимости, используя значение аттрибута

<c:set value=”value” var=”varName” [scope=”{page|request|session|application}”]/>

Пример

```
<c:set var="name“   scope="session" >
        ${param.name}
</c:set>
```

Синтаксис 2: Установка переменной в указанную область видимости, используя тело тега:

```
<c:set var=”varName”
    [scope=”{page|request|session|application}”]>
  body content
</c:set>
```

Пример:

```
<c:set var="name"  scope="session"
                        value="${param.name}"/>
```

Синтаксис 3: Установка свойства объекта, используя значение атрибута

<c:set value=”value”

target=”target” property=”propertyName”/>

Здесь target – имя переменной, чье свойство должно быть изменено; property – свойство, которое должно быть изменено; value – значение, которое должно получить свойство.

Синтаксис 4: Установка свойства объекта, в теле тега

<c:set target=”target” property=”propertyName”>

body content

</c:set>

<c:remove /> - удаляет переменную из указанной области видимости.

Пример:

```
<c:remove var="cachedResult"
                                    scope="application"/>
```

Синтаксис:

```
<c:remove var=”varName”
   [scope=”{page|request|session|application}”]/>
```

<c:catch /> - перехватывает обработку исключения.

Пример:

```
<c:catch var=”exception”>
...
</c:catch>
<c:if test=”${exception != null}”>
    Error
</c:if>
```

Синтаксис:

```
<c:catch [var=”varName”]>
……..
</c:catch>
```

Пример:

```
<%@ taglib uri="http://java.sun.com/jstl/core"
                                                prefix="c" %>
<html>
  <head>
      <title>Catching the Exception</title>
   </head>
 <body>
   <strong>I can catch the exception:</strong><br>
   <c:catch var ="catchException">
    The exception will be thrown inside the catch:<br>
    <%= 20 %> <br>
    <% int x = 5/0;%>
    <%= 10 %>
   </c:catch>
```

```
<c:if test = "${catchException!=null}">
 The exception is : ${catchException} <br><br>
 There is an exception: ${catchException.message}
  <br>
</c:if>
</body>
</html>
```

В браузере получим:

I can catch the exception:

The exception will be thrown inside the catch:20
The exception is : java.lang.ArithmeticException: / by zero
There is an exception: / by zero

Тэги условного перехода

<c:if /> - тело тэга вычисляется только в том случае если значение выражения true.

Пример:

```
<c:if test="${user.visitCount == 1}">
This is your first visit. Welcome to the site!
</c:if>
```

Синтаксис:

Без тела:

```
<c:if test=”testCondition” var=”varName”
   [scope=”{page|request|session|application}”]/>
```

С телом:

```
<c:if test=”testCondition” [var=”varName”]
   [scope=”{page|request|session|application}”]>
 body content
</c:if>
```

Рассмотрим пример.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
  <head>
    <title> Tag Example</title>
  </head>
  <body>
    <c:set var="salary" scope="session" value="${2000*2}"/>
    <c:if test="${salary > 2000}" var="varName">
        <p>My salary is: <c:out value="${salary}"/><p>
    </c:if>
      <c:out value="${varName}"/>
  </body>
</html>
```

В браузере получим

My salary is: 4000

true

<c:choose /> (<c:when />, <c:otherwise />) – то же что и **<c:if />** с поддержкой нескольких условий, и действия, производимого по умолчанию.

Пример:

```
<c:choose>
  <c:when test="${user.category == 'trial'}”>
   ...
  </c:when>

  <c:when test="${user.category == 'member'}”>
   ...
  </c:when>
  <c:when test="${user.category == 'vip'}”>
   ...
  </c:when>
  <c:otherwise>
   ...
  </c:otherwise>
  </c:choose>
```

Выражение if/then/else может быть легко записано как:

```
<c:choose>
  <c:when test="${count == 0}”>
    No records matched your selection.
  </c:when>
  <c:otherwise>
    <c:out value="${count}"/> records matched
                                    your selection.
  </c:otherwise>
</c:choose>
```

Синтаксис:

<c:choose>

body content (<when> and <otherwise> subtags)

</c:choose>

<c:when test=”testCondition”>

body content

</c:when>

<c:otherwise>

conditional block

</c:otherwise>

Итераторы

<c:forEach /> - выполняет тело тега для каждого элемента коллекции.

Пример:

```
<table>
  <c:forEach var=”customer” items=”${customers}”>
     <tr><td> <c:out value=”${customer}”/> </td></tr>
  </c:forEach>
</table>
```

Пример:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jstl/core" prefix="c" %>
<html>
<body>
<jsp:useBean id="books" scope="session" class="aaa.BookList" />
    <ul>
      <c:forEach var="book" items="${books}">
        <li>
         <c:out value="${book.title}"/>
        </li>
       </c:forEach>
     </ul>
</body>
</html>
```

Класс BookList имеет вид:

```
package aaa;
import java.util.*;

public class BookList extends AbstractCollection<Book>{
  public List<Book> book_list;
  public BookList(){
        book_list = new ArrayList<Book>(2);
        book_list.add(new Book("BookTitle_01","book_01.html"));
        book_list.add(new Book("BookTitle_02","book_02.html"));
   }
   public Iterator<Book> iterator() { return book_list.iterator();}
   public int size(){ return 2; }
}
```

Класс Book имеет вид:

```
package aaa;
public class Book {
    public String title;
    public String url;
    public Book(String new_title, String new_url){
        title = new_title;
        url = new_url;
    }
public String gettitle(){return title;}
public String geturl(){return url;}
}
```

<c:forTokens /> - выполняет тело тега для каждого токена в строке.

Синтаксис:

```
<c:forTokens items="stringOfTokens"
        delims="delimiters" [var="varName"]
        [varStatus="varStatusName"]
        [begin="begin"] [end="end"]
        [step="step"]>
  body content
</c:forEach>
```

Пример:

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core"%>
<c:set var="names" value="Joe:Petter;Ryan|John" scope="page"/>
<html>
 <head>
    <title>forTokens action</title>
 </head>
 <body>
 <c:forTokens items="${pageScope.names}"  delims=":;|"
                                    var="currentName" varStatus="status" >
   Family member #<c:out value="${status.count}" /> is
   <c:out value="${currentName}" /> <br />
 </c:forTokens>
</body>
</html>
```

Тэги обработки URL

<c:redirect /> - перенаправляет запрос на указанный URL.

Пример:

<c:redirect url="http://acme.com/register"/>

<c:import/> - добавляет на **JSP** содержимое указанного WEB-ресурса.

Примеры:

<c:import
url=”http://acme.com/exec/customers?country=Japan”/>

<c:import url=”/copyright.html”/>

<c:import url=”/logo.html” context=”/master”/>

<c:url /> - формирует адрес с учётом контекста приложения (request.getContextPath()).

<c:param /> - добавляет параметр к запросу, сформированному при помощи **<c:url />.**

Пример:

```
<c:url value="http://acme.com/exec/register"
                                        var="myUrl">
  <c:param name="name" value="${param.name}"/>
  <c:param name="country"
                                value="${param.country}"/>
</c:url>
<a href=’<c:out value="${myUrl}"/>’ >Register </a>
```

JSTL fmt

Содержит тэги форматирования и интернационализации.

<%@taglib uri=”http://java.sun.com/jstl/fmt”
prefix=”fmt” %>

- для обычной **JSP**.

<jsp:root version=”1.2”
xmlns:c=”http://java.sun.com/jstl/fmt”>
...
</jsp:root>

- для **XML** формата **JSP**.

Тэги интернационализации

<fmt:setLocale /> - устанавливает объект Locale, используемый на странице.

Пример:

<fmt:setLocale value=”en_US” />

<fmt:setBundle />, <fmt:bundle /> - устанавливают объект ResourceBundle, используемый на странице.

В зависимости от установленной локали, выбирается ResourceBundle, соответствующий указанному языку, стране и региону.

Действие <fmt: setBundle> служит для создания локализированного контекста.

Если атрибут var не определен, тогда для сохранения контекста используется конфигурационная переменная javax.servlet.jsp.jstl.fmt.localizationContext

Пример:

```
<fmt:setBundle basename="resource_bundle">
    <fmt:message key="title" />
    <fmt:message key="welcome" />
    <fmt:bundle basename="resource_bundle_ru" >
    <fmt:message key="title" />
    <fmt:message key="welcome" />
</fmt:bundle>
```

Файл resource_bundle.properties имеет вид:

```
title=My Company
welcome=Welcome!
```

Файл resource_bundle_ru.properties имеет вид:

```
title=Моя компания
welcome=Привет!
```

В браузере получим

Моя компания Привет! Моя компания Привет!

<fmt:message /> - выводил локализованное сообщение.

<fmt:param> - используется для задания параметра в теге **<fmt:message>**

Синтаксис:

Значение параметра задано в атрибуте “value”

<fmt:param value=”messageParameter”/>

Пример:

```
<fmt:message key="welcome">
        <fmt:param value="${userNameString}"/>
</fmt:message>
```

Персонализация приветствия

Тэги форматирования

<fmt:formatNumber />, <fmt:formatDate /> - форматирует числа / даты с учётом установленной локали (либо указанного шаблона).

Пример:

```
<fmt:formatNumber value="9876543.21"
                                    type="currency"/>
```

Данный код напечатает следующее:

SFr. 9’876’543.21 – если регион fr_CH

$9,876,543.21 – если регион en_US

Атрибут pattern задает шаблон для вывода.

С его помощью можно указывать количество разрядов после запятой.

Например, код

<fmt:formatNumber value="12.3" pattern=".000"/>

выведет следующее: 12.300.

Знак 0- означает “обязательную” цифру.

Знак #- означает цифру, если эта цифра 0, то она не отображается

Шаблон #,#00.0#- значит, минимум две цифры перед точкой, если перед точкой более 3 цифр, то три нужно отделить запятой. После точки как минимум одна цифра, а максимум 2.

Например, код

<fmt:formatNumber value="123456.7891"

pattern="#,#00.0#"/>

напечатает следующее: 123,456.79.

<fmt:timeZone />, <fmt:setTimeZone /> - устанавливает временную зону используемую для форматирования.

Синтаксис:

```
<fmt:timeZone value=”timeZone”>
        body content
</fmt:timeZone>
<fmt:setTimeZone value=”timeZone”
            [var=”varName”]
       [scope=”{page|request|session|application}”]/>
```

атрибут value, может иметь тип либо String либо java.util.TimeZone

Если атрибут value не определен, тогда временной зоной по умолчанию является GMT (Greenwich Mean Time - гринвичское время).

var- указывает переменную в которую будет сохранена новая временная зона

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core“ prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt“  prefix="fmt" %>
<html>
  <head>
     <title>JSTL fmt:setTimeZone Tag</title>
  </head>
<body>
 <c:set var="now" value="<%=new java.util.Date()%>" />
  <p>Date in Current Zone:
  <fmt:formatDate value="${now}" type="both" timeStyle="long"
                                                dateStyle="long" /></p>
    <p>Change Time Zone to GMT-8</p>
  <fmt:setTimeZone value="GMT-8" />
  <p>Date in Changed Zone:
     <fmt:formatDate value="${now}" type="both"  timeStyle="long"
                                                  dateStyle="long" />
  </p>
 </body>
</html>
```

<fmt:parseNumber /> - переводит строковое представление числа в объект подклассов Number.

Синтаксис:

```
<fmt:parseNumber value=”numericValue”
    [type=”{number|currency|percent}”]
    [pattern=”customPattern”]
    [parseLocale=”parseLocale”]
    [integerOnly=”{true|false}”]
    [var=”varName”]
  [scope=”{page|request|session|application}”]/>
```

```
<html>
 <head>
    <title>JSTL fmt:parseNumber Tag</title> </head>
  <body>
    <h3>Number Parsing:</h3>
    <c:set var="balance" value="1250003.350" />
    <fmt:parseNumber var="i" type="number"
                                          value="${balance}" />
   <p>Parsed Number (1) : <c:out value="${i}" /></p>
    <fmt:parseNumber var="i" integerOnly="true"
                        type="number" value="${balance}" />
   <p>Parsed Number (2) : <c:out value="${i}" /></p>
 </body>
</html>
```

JSTL sql

Используется для выполнения запросов **SQL** и обработки результатов запроса на **JSP.**

<%@taglib uri=”http://java.sun.com/jstl/sql”
prefix=”sql” %>

- для обычной **JSP**.

<jsp:root version=”1.2”
xmlns:c=”http://java.sun.com/jstl/sql”>
...
</jsp:root>

- для **XML** формата **JSP**.

Доступ к физическим источникам данных предоставляется через объект javax.sql.DataSource.

Пример:

```
<sql:query var="customers" dataSource="${dataSource}">
  SELECT * FROM customers
  WHERE country = ’China’
  ORDER BY lastname
</sql:query>
<table>
<c:forEach var="row" items="${customers.rows}">
<tr>
 <td><c:out value="${row.lastName}"/></td>
 <td><c:out value="${row.firstName}"/></td>
 <td><c:out value="${row.address}"/></td>
</tr>
</c:forEach>
</table>
```

<sql:update> вносит изменения в базу данных.

Для поддержания целостности базы данных несколько операций могут быть объединены в транзакцию путем включения <sql:update> в тег <sql:transaction>.

Пример:

```
<sql:transaction dataSource="${dataSource}">
<sql:update>
     UPDATE account
     SET Balance = Balance - ?
     WHERE accountNo = ?
    <sql:param value="${transferAmount}"/>
    <sql:param value="${accountFrom}"/>
</sql:update>
<sql:update>
     UPDATE account
     SET Balance = Balance + ?
     WHERE accountNo = ?
    <sql:param value="${transferAmount}"/>
    <sql:param value="${accountTo}"/>
</sql:update>
</sql:transaction>
```

JSTL xml

Используется для обработки данных XML на JSP.

<%@taglib uri=”http://java.sun.com/jstl/xml”
prefix=”x” %>

- для обычной JSP.

<jsp:root version=”1.2”
xmlns:c=”http://java.sun.com/jstl/xml”>
…
</jsp:root>

- для XML формата JSP.

Теги JSTL для работы с XML базируются на XPath.

Теги <x:parse> и <x:transform> позволяют трансформировать XML документы в объекты String и Reader.

Доступ к ресурсу через URL осуществляется с помощью тега <c:import>.

Пример:

```
<c:import url="http://acme.com/productInfo"
                                        var="xml">
   <c:param name="productName"
                                value="${product.name}"/>
</c:import>
<x:parse xml="${xml}" var="doc"/>
```

Рассмотрим пример. Пропарсим xml файл

```
<?xml version="1.0" encoding="UTF-8"?>
<org>
   <company>
      <emp>
         <name>Mahendra Singh</name>
          <designation>Sofware developer</designation>
          <age>25</age>
      </emp>
      <emp>
        <name>Anismita Singh</name>
         <designation>Sofware developer</designation>
         <age>23</age>
     </emp>
   </company>
```

```
<emp>
   <name>Ravi Kant</name>
   <designation>Sr. Sofware
                                developer</designation>
   <age>27</age>
</emp>
<emp>
        <name>Suman</name>
        <designation>Sr. Graphics
                                 Designer</designation>
        <age>25</age>
</emp>
</org>
```

Jsp страница будет иметь вид

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

<html>
  <head>
   <title>Example of x:parse tag</title>
  </head>
  <body>
   <c:import var="importFile" url="employee.xml"/>

   <x:parse var="doc" doc="${importFile}"/>
   <table border=1>
     <tr>
       <th>Given Xml Document</th>
       <th>Some operations on this file</th>
     </tr>
```

```
<tr>
    <td valign="top">
      <pre>
        <c:out value="${importFile}"/>
      </pre>
    </td>
    <td>
      <table border=1>
        <tr>
          <th>Expression</th>
          <th>Result</th>
        </tr>
        <tr>
          <td>$doc//*</td>
          <td><pre> <x:out select="$doc//*"/> </pre></td>
        </tr>
```

```
<tr>
    <td> $doc/org/* </td>
     <td>
        <pre>
             <x:out  select="$doc/org/*"/>
        </pre>
      </td>
  </tr>
  <tr>
        <td> $doc/org/company/* </td>
        <td>
           <pre>
               <x:out select="$doc/org/company/*"/>
           </pre>
        </td>
   </tr>
```

```
<tr>
    <td> $doc/org/company/emp </td>
    <td>
      <pre>
       <x:out select="$doc/org/company/emp/*"/>
      </pre>
    </td>
</tr>
<tr>
    <td> $doc/org/company[last()] </td>
    <td>
      <pre>
       <x:out select="$doc/org/company/emp[last()]"/>
      </pre>
    </td>
</tr>
</table>
```

```
  </td>
 </tr>
 </table>
</body>
</html>
```

Рассмотрим подробнее <x:out>.

Данный тег имеет следующий вид

<x:out select=XPath выражение>

Таким образом для разбора xml файла используются XPath выражения.

Язык XML Path (XPath) является набором синтаксических и семантических правил для ссылок на части XML-документов.

XPath предназначен для использования другими спецификациями, такими как XSL Transformations (XSLT) и XML Pointer Language (XPointer).

Рассмотрим некоторые XPath выражения

Есть два типа маршрутов местоположения: относительный и абсолютный.

Относительный маршрут местоположения - это последовательность местоположений, разделенных /.

Например:

list/item[currentPrice<20.0]

Этот маршрут местоположения состоит из двух шагов установки местоположения: первый, list, выбирает набор узлов относительно контекстного узла; второй, item[currentPrice<20.0], выбирает набор узлов в подмножестве, идентифицированном первым шагом, и т.д., если есть еще узлы.

Абсолютный маршрут местоположения состоит из /, за которым, возможно, следует относительный маршрут местоположения, здесь / ссылается на корневой узел.

Абсолютный маршрут местоположения является относительным маршрутом местоположения, вычисляемым в контексте корневого узла, например:

/list/item[currentPrice<20.0]

Рассмотрим наиболее широко используемые символы:

- @ используется для ссылки на атрибуты. Например, маршрут местоположения @currency идентифицирует атрибут currency. list/item[@private] идентифицирует элементы item с атрибутом private
- * используется для ссылки на все элементы, которые являются дочерними для узла контекста.
- @* используется для ссылки на все атрибуты узла контекста.
- [] могут также использоваться для ссылки на определенный элемент в упорядоченной последовательности. Например, list/item[2] ссылается на второй элемент item.

- // используется для ссылки на все дочерние элементы узла контекста. Например, //item ссылается на все элементы item, а //list/item ссылается на все элементы item, которые имеют родителем list
- . используется для ссылки на сам узел контекста. Например, . выбирает узел контекста, а .//item ссылается на все элементы item, которые являются дочерними для узла контекста.
- .. используется для ссылки на родительский узел узла контекста. Например, ../item ссылается на первый элемент item

XPath также допускает математические выражения и имеет библиотеку функций.

Таким образом в браузере получим следующее:

```
            <?xml version="1.0" encoding="UTF-8"?>
rg>
  <company>
       <emp>
           <name>Mahendra Singh</name>
            <designation>Sofware developer</designation>
            <age>25</age>
       </emp>
       <emp>
          <name>Anismita Singh</name>
           <designation>Sofware developer</designation>
           <age>23</age>
      </emp>
  </company>
mp>
 <name>Ravi Kant</name>        <designation>Sr. Sofware
                                      developer</designation>
 <age>27</age>
emp>
mp>
      <name>Suman</name>
      <designation>Sr. Graphics
                                     Designer</designation>
      <age>25</age>
emp>
org>
```

Expression	Result
$doc//*	Mahendra Singh Sofware developer 25 Anismita Singh Sofware developer 23 Ravi Kant Sr. Sofware developer 27 Suman Sr. Graphics Designer 25

JSTL functions

Теги из этой библиотеки выполняют роль, подобную смыслу библиотеки fmt, только не для чисел и дат, а для строк.

Теги библиотеки используют префикс fn и во

многом копируют известные методы класса String. Подключение библиотеки осуществляется директивой taglib:

<%@ taglib prefix="fn"

uri="http://java.sun.com/jsp/jstl/functions" %>

Список тегов:

${fn:length(аргумент)}- подсчитывает число элементов в коллекции или длину строки;

${fn:toUpperCase(String str)}** и **${fn:toLowerCase(String str)}-

изменяет регистр строки;

${fn:substring(String str, int from, int to)} - извлекает подстроку;

${fn:substringAfter(String str, String after)} и

${fn:substringBefore(String str, String before)}- извлекает подстроку до или после указанной во втором аргументе;

${fn:trim(String str)}- обрезает все пробелы по краям строки;

${fn:replace(String str, String str1, String str2)}- заменяет все вхождения строки str1 на строку str2;

${fn:split(String str, String delim)} - разбивает строку на подстроки, используя delim, как разделитель;

${fn:join(массив, String delim)} - соединяет массив в строку, вставляя между элементами подстроку delim;

${fn:escapeXml(String xmlString)} - сохраняет в

обрабатывемой строке теги;

${fn:indexOf(String str, String searchString)} - возвращает индекс первого вхождения строки searchString;

<c:if test="${fn:startsWith(String str, String part)}"> - возвращает истину, если строка начинается с подстроки part

<c:if test ="${fn:endsWith(String str, String part)}"> - возвращает истину, если строка заканчивается на подстроку part;

<c:if test ="${fn:contains(String name, String searchString)}"> - возвращает истину, если строка содержит подстроку searchString;

<c:if test="${fn:containsIgnoreCase(String name, String searchString))}"> - возвращает истину, если строка содержит подстроку searchString, без учета регистра