Expression Language

Um dos principais recursos da tecnologia JSP 2.0 é suportar a expression language (EL). A expression language torna possível simplificar o acesso a dados de aplicações armazenados em componentes JavaBeans. Por exemplo, a EL permite que um autor de página acessar um bean utilizando uma sintaxe simplificada tal como ${nome} para uma variável ou ${escola.util.contador} para uma propriedade de uma classe.

O atributo "test" da tag condicional abaixo possibilita comparar o número de itens num bean armazenado em uma sessão chamada "cart" com "0":

```
<c:if test="${sessionScope.cart.numberOfItems > 0}">

  ...
</c:if>
```

A expressão de avaliação é responsável por manipular expressões EL as quais são circundadas pelos caracteres ${ } e podem incluir literais, exemplo:

```
<c:if test="${bean1.a < 3}" >
  ...
</c:if>
```

Qualquer valor que não inicia com ${ é tratado como uma literal e é convertida para o tipo esperado utilizando o PropertyEditor:

```
<c:if test="true" >
...
</c:if>
```

Valores literais que contêm os caracteres ${ devem ser escapados conforme segue:

```
<mytags:exemple attr1="uma expressão é ${'${'}true}" />
```

Desativando a avaliação da EL

Em JSP 1.2 e anterior, strings do formato ${...} não tinham significado especial. Portanto, é possível que os caracteres ${ apareçam dentro de uma página criada anteriormente e que agora esteja sendo usada em um servidor que suporte JSP 2.0. Neste caso, você precisa desativar a linguagem de expressão nessa página com:

```
<%@ page isELIgnored ="false" %>
```

Usando Expressões

Expressões EL podem ser utilizadas:

- Em texto estático
- Em qualquer atributo de "tag" padrão ou customizada que aceite uma expressão

O valor de uma expressão em texto statico é computado e inserido na saída (output) atual. Se o texto statico aparecer no corpo de uma tag customizada, note que uma expressão não será avaliada se o corpo da tag customizada for declarada como sendo do tipo `tagdependent`.

Existem três formas de atribuir valor a um atributo de uma tag:

- Com uma expressão simplificada:

  ```
  <some:tag value="${expr}"/>
  ```

  A expressão é avaliada e o resultado é convertido para o tipo apropriado esperado pelo atributo.

- Com um ou mais expressões separadas ou envolvidas por texto:

  ```
  <some:tag value="some${expr}${expr}text${expr}"/>
  ```

  As expressions são avaliadas da esquerda para a direita. Cada expressão é convertida para uma `String` e concatenada com o texto circundante. A `String` resultante é então convertida para o tipo esperado pelo atributo.

- Com somente texto:

  ```
  <some:tag value="sometext"/>
  ```

  Neste caso, o valor do é convertido para o tipo esperado pelo atributo.

Expressões usadas para atribuir valores são avaliados no contesto de um tipo esperado. Se o resultado da expressão avaliada não combinar com o tipo esperado de forma exata, a conversão de tipo será realizada. Por exemplo, a expressão `${1.2E4}` informada como o valor de um atributo do tipo `float` irá resultar na seguinte conversão:

```
Float.valueOf("1.2E4").floatValue()
```

Veja a secção JSP2.8 da JSP 2.0 specification para consultar a lista completa das regras de conversão de tipo.

Variáveis

O container web avalia uma variável que aparece numa expressão procurando pelo seu valor de acordo com o comportamento do `PageContext.findAttribute(String)`. Por exemplo, quando avaliando a expressão `${product}`, o container procurará por `product` nos escopos page, request, session, e application e retornará seu valor. Se `product` não for encontrado, será retornado `null`. Uma variável que combinar com um dos objetos implícitos descrito no parágrafo Objetos Implícitos retornará este objeto implícito ao invés do valor da variável.

Propriedades das variáveis são acessadas usando o operador `.` e pode ser encadeada arbitrariamente.

A linguagem de expressões do JSP unifica o tratamento dos operadores `.` e `[]`. `expr-a.identifier-b` é equivalente a `expr-a["identifier-b"]`; a expressão `expr-b` é usada para construir uma literal cujo valor é o identificador, e o operador `[]` é usado com este valor.

Para avaliar `expr-a[expr-b]`, avalia `expr-a` em `value-a` e avalia `expr-b` em `value-b`. Se `value-a` ou `value-b` for null, será retornado `null`.

- Se `value-a` é um `Map`, retorna `value-a.get(value-b)`. Se `!value-a.containsKey(value-b)`, então retorna `null`.

- Se `value-a` é um `List` ou array, converte `value-b` para `int` e retorna `value-a.get(value-b)` ou `Array.get(value-a, value-b)`, o que for apropriado. Se a converção não puder ser realizada, um erro será retornado. Se a chamada `get` retornar um `IndexOutOfBoundsException`, `null` será retornado. Se a chamada `get` retornar outra exceção, um erro será retornado.
- Se `value-a` é um objeto JavaBeans , converte `value-b` para `String`. Se `value-b` é uma propriedade com permissão de leitura de `value-a`, então retorna o resultado d chamada `get`. Se o metodo `get` lançar uma exceção, um erro é retornado.

Objetos Implícitos

A linguagem de expressão do JSP define um conjunto de objetos implícitos:

- `pageContext`: O contexto para a página JSP. Fornece acesso á vários objetos incluindo:
  - `servletContext`: O contexto para o servlet da página JSP e qualquer componente web contido na mesma aplicação.
  - `session`: O objeto sessão para o cliente.
  - `request`: A requisição que invocou a execução da página JSP.
  - `response`: A resposta retornada pela página JSP.

Ainda, muitos objetos implícitos estão disponíveis que permitem fácil acesso aos seguintes objetos:

- `param`: Mapeia um nome de parâmetro de requisição para um valor
- `paramValues`: Mapeia um nome de parâmetro de requisição para um array de valores
- `header`: Mapeia um nome de cabeçalho da requisição para um valor
- `headerValues`: Mapeia um nome de cabeçalho da requisição para um array de valores
- `cookie`: Mapeia um nome de cookie para um cookie
- `initParam`: Mapeia um nome de parâmetro de inicialização de contexto para um valor

Finalmente, existem objetos que permitem acesso á varias variáveis de escopo.

- `pageScope`: Mapeia nomes de variáveis de escopo de página para seus valores
- `requestScope`: Mapeia nomes de variáveis de escopo de requisição para seus valores
- `sessionScope`: Mapeia nomes de variáveis de escopo de sessão para seus valores
- `applicationScope`: Mapeia nomes de variáveis de escopo de aplicação para seus valores

Quando uma expressão referencia um desses objetos pelo nome, o objeto apropriado é retornado ao invés do atributo correspondente. Por exemplo, `${pageContext}` retorna o objeto `PageContext`, mesmo se existir um atributo `pageContext` contendo algum outro valor.

Literais

A linguagem de expressão do JSP define as seguintes literais:

- Boolean: `true` e `false`
- Integer: como em Java
- Floating point: como em Java
- String: com apóstrofes ou aspas; `"` é escapado como `\"`, `'` é escapado como `\'`, e `\` é escapado como `\\`.
- Null: `null`

Operadores

Em adição aos operadores `.` e `[]` discutido em Variáveis, A linguagem de expressão do JSP oferece os seguintes operadores:

- Aritmético: `+`, `-` (binário), `*`, `/` e `div`, `%` e `mod`, - (unário)
- Logico: `and`, `&&`, `or`, `||`, `not`, `!`
- Relational: `==`, `eq`, `!=`, `ne`, `<`, `lt`, `>`, `gt`, `<=`, `ge`, `>=`, `le`. Comparações podem ser efetuadas contra outros valores, ou literais booleana, string, inteira, ou de ponto flutuante.
- Empty: O operador `empty` é um prefixo de operação que pode ser usada para determinar se um valor é `null` ou vazio.
- Condicional: `A ? B : C`. Avaliar `B` ou `C`, dependendo do resultado da avaliação de `A`.

A precedência de operadores de maior prioridade para o de menor, da esquerda para a direita comforme segue:

- `[] .`
- `()` - Utilizado para trocar a precedência de operadores.
- `-` (unário) `not ! empty`
- `* / div % mod`
- `+ -` (binário)
- `< > <= >= lt gt le ge`
- `== != eq ne`
- `&& and`
- `|| or`
- `? :`

Palavras Reservadas

A palavras a seguir são reservadas para a linguagem de expressão de JSP e não devem ser utilizados como identificadores.

```
and   eq   gt   true   instanceof
or    ne   le   false  empty
not   lt   ge   null   div   mod
```

Note que muitas destas palavras não estão na EL agora, mas poderão ser incorporadas em versões futuras, assim não devem ser utilizadas.

Exemplos

A tabela abaixo contém exemplos de expressões EL e o resultado de sua avaliação.

Exemplos de Expressões

Expressão EL	Resultado
`${1 > (4/2)}`	`false`
`${4.0 >= 3}`	`true`
`${100.0 == 100}`	`true`
`${(10*10) ne 100}`	`false`

`${'a' < 'b'}`	`true`
`${'hip' gt 'hit'}`	`false`
`${4 > 3}`	`true`
`${1.2E4 + 1.4}`	`12001.4`
`${3 div 4}`	`0.75`
`${10 mod 4}`	`2`
`${empty param.Add}`	True se o parâmetro de requisição chamado `Add` for null ou uma string vazia
`${pageContext.request.contextPath}`	O caminho do contexto (context path)
`${sessionScope.cart.numberOfItems}`	O valor da propriedade `numberOfItems` do atributo chamado `cart` salvo na sessão
`${param['mycom.productId']}`	O valor do parâmetro de requisição chamado `mycom.productId`
`${header["host"]}`	O host
`${departments[deptName]}`	O valor da entrada chamada `deptName` no mapa (Map) `departments`
`${requestScope['javax.servlet.forward.servlet_path']}`	O valor do atributo chamado `javax.servlet.forward.servlet_path` salvo na requisição (request)

Função

A linguagem de expressão de JSP permite a definição de funções que podem invocar uma expressão. Funções são definidas usando o mesmo mecanismo que as Tags Customizadas.

Usando Funções

Funções podem aparecer em textos estáticos e como valores para atributos em tags.

Para usar uma função numa página JSP , utilizamos a diretiva `taglib` para importar a tag library contendo a função. Após a invocação da função é prefixada conforme o prefixo definido na declaração da diretiva.

Por exemplo, a página de exemplo de data `index.jsp` importa a biblioteca `/functions` e invoca a função `equals` numa expressão:

```
<%@ taglib prefix="f" uri="/functions"%>
...
    <c:when
      test="${f:equals(selectedLocaleString,
        localeString)}" >
```

Definindo Funções

Para definir uma função devemos progrmas esta como um método publico e estático numa classe publica. A classe `mypkg.MyLocales` no exemplo `date` define uma função que testa a igualdade de duas `Strings` conforme segue:

```
package mypkg;
public class MyLocales {

  ...
  public static boolean equals( String l1, String l2 ) {
    return l1.equals(l2);
  }
}
```

Após é mapeado o nome da função que será utilizada na expressão EL para definir a classe e a assinatura da função numa TLD. Abaixo o arquivo `functions.tld` mapeia a função `equals` para a classe contendo a implementação da função `equals` e a assinatura desta função:

```
<function>
  <name>equals</name>
  <function-class>mypkg.MyLocales</function-class>
  <function-signature>boolean equals( java.lang.String,
    java.lang.String )</function-signature>
</function>
```

Uma biblioteca de tags (tag library) pode ter apenas um elemento `function` definido para o mesmo nome de elemento.

JSTL – Java Standard Tag Library

Antes do JSTL, era necessário declarar um Objeto, conhecer o tipo do Objeto, e conhecer alguma linguagem script para fazer uma simples manipulação. Isto resultava numa sintaxe complexa.

Agora, com o JSTL, nós temos acesso direto aos dados utilizando uma sintaxe muito mais simples. A EL é uma ótima opção para tratar desses tipos de tarefas.

Como um simples exemplo, poderemos partir disto:

```
<jsp:useBean id="customer" type="sample.Customer" scope="request"/> ...
Customer Name: <%=customer.getName()%>
...
<% if (customer.getState().equals("CO")){ %>
...
<%}%>
```

para:

```
Customer Name: ${ customer. name}
<c:if test="${customer. state == param. state}">
...
</c:if>
```

É possível fornecer valores default. Esses valores default são também do tipo correto. Valores default são uma forma útil de impedir o aparecimento de null-pointer exceptions em suas páginas. O exemplo a seguir mostra como atribuir um valor default:

```
<c:set var="city" value="${user.address.city}" default="N/A" />
```

Ações Principais (Core actions)

A EL é utilizada na Core tag library. A tag <c:out> emite expressões EL avaliadas para JspWriter corrente. Esta tag tem funcionalidade similar ao JSP <%= scripting exp %>. Utilizando as core actions terá a seguinte forma:

```
<c:out value="${customer.name}"  default="N/A" />
```

Também é possível definir e remover variáveis de escopo. O escopo default utilizado é Page. Por exemplo, podemos definir uma variável no escopo Page chamada customer utilizando <c:set var="customer" value=${customer}" /> e utilizar <c:remove var="customer" /> para remover do escopo.

Com o JSTL podemos utilizar uma tag para capturar (catch) java.lang.Throwable, por example, <c:catch var="myError" />. Esta tag possibilita a manipulação de erros numa página. Mas isto não significa uma substituição do mecanismo de página de erro do JSP. Usando a tag <c:catch> é possível ter um controle preciso na manipulação de erros; ele permite que os erros sejam manipulados pela página ao invés do redirecionamento para uma página de erro – nem todo o erro que é lançado tem necessidade de ser tratado por uma página de erro. O uso da tag <c:catch> oferece a possibilidade da construção de uma melhor interação com o usuário por causa que o fluxo da página se torna mais amigável.

Ações Condicionais

A utilização da EL em ações condicionais também pode ser um mecanismo útil para simplificar o código JSP, com a tag <c:if> ,é possível construir expressões condicionais simples. Por example, para acessar uma propriedade de Objetos:

```
<c:if test="${user.visitCount == 1}"
        Welcome back!

</c:if>
```

é possível notar que quando existe um if, com certeza existirá também um else. Condições mutuamente exclusivas são úteis somente quando um determinado número de ações alternativas estão disponíveis. É possível obter a funcionalidade do "if/then/else" através da utilização das tags <c:choose>, <c:when>, e <c:otherwise> .

Abaixo temos um exemplo. Se estivermos processando um resultado, poderemos utilizar estas tags para determinar qual e a melhor mensagem que deverá ser mostrada.

```
<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>
```

Ações de Iteração

Provavelmente os recursos mais úteis do JSTL são as ações de iteração. As ações que oferecem iterações são `<c:forEach>`, `<c:forToken>`, e `<x:forEach>`. Existem também tags XML que imitam as principais tags de interação.

Estas tags suportam todos os tipos de coleções do J2SE , incluindo `List`, `LinkedList`, `ArrayList`, `Vector`, `Stack`, e `Set`. Bem como objetos `java.util.Map` tais como `HashMap`, `Hashtable`, `Properties`, `Provider`, e `Attributes`. É posspivel efetuar iterações através de um array de Objetos ou tipos primitivos. Quando for usado tipos primitivos, o item corrente é encapado pelas classes wrapper do Java. Assim o item corrente num Array de `int`s será um Integer. Dois objetos são exportados para cada iteração, o item corrente e o status da iteração. Veja o exemplo a seguir:

```
<table>
<c:forEach var="product"
              items="${products}"
              varStatus="status">
<tr>
        <td><c:out value="${status.count}"/></td>
        <td><c:out value="${product.name}"/></td>
</tr>
</c:forEach>
</table>
```

Neste exemplo, a Collection e especificada pela EL como `products`. O item corrente esta contido na variável `product`. O status corrente da iteração é mantida no objeto `status`. Muito simples.

Ações para URL

Adicionalmente as ações de iteração, a Core library também oferece ações para tratar URLs. Isto inclui suporte para hiper-links, importação de recursos, e redirecionamentos. A tag `<c:url>` manipula a reescrita e codificação (encoding) de uma URL de forma automática. Se olharmos o exemplo da definição de uma URL com um parâmetro para utilizar este em um link, ele se pareceria como abaixo:

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

O uso do recurso de importação também é mais poderoso no o JSTL. Agora é possível especificar URLs absolutas, relativas, relacinados a um contexto externo, e recursos FTP. Veja alguns exemplos:

- URL com endereço absoluto:
  `<c:import url="http://sample.com/Welcome.html"/>`
- URL com endereço relativo (para o contexto corrente):
  `<c:import url="/copyright.html"/>`
- URL com endereço relativo a um contexto externo:
  `<c:import url="/myLogo.html" context="/common"/>`
- Recurso FTP:

  `<c:import url="ftp://ftp.sample.com/myFile"/>`

É possível incluir o conteúdo de um recurso inline ou exportado como tanto um objeto String ou Reader usando os atributos chamados `var` ou `varReader`. O objeto String é reusavel e tem seu conteúdo guardado em cache. A vantagem do uso de um objeto Reader é que o conteúdo pode ser acessado diretamente sem o uso de qualquer buffer. No entanto um objeto Reader necessita ser encadeado com um `<c:import>` para assegurar que o Reader será fechado apropriadamente e não será deixado aberto acidentalmente.Vide exemplo abaixo:

```
<c:import url=http://sample.com/customers varReader="customers">
        <mytag:process in="${customers}"/>
</c:import>
```

Formatação

Existem também uma forma para formatar números e datas. O uso de `<fmt:formatNumber>` ou `<fmt:parseNumber>` permite a formatação de números, valores monetários, e porcentagens de acordo com o Locale atual. Padrões de formatação também podem ser informados através do atributo pattern. Para demonstrar: `<fmt:formatNumber value="12.3" pattern=".00"/>` gera uma saída "12.30."

Data e hora são manipulados utilizando `<fmt:formatDate>`, `<fmt:timeZone>`, `<fmt:setTimeZone>`, e `<fmt:parseDate>`.