EL-элементы

EL-элементы были придуманы как замена скриптовым выражениям: они имеют то же назначение, но записываются не на языке Java, а на специальном EL-языке. Последний гораздо более ограничен по возможностям, но их достаточно для выполнения определенных задач. EL-язык более выразителен, чем Java, и позволяет записывать выражения в более компактной форме. Кроме краткой записи преимущество EL-элементов перед обычными выражениями еще и в том, что их синтаксис не противоречит спецификации XML и не нужно менять синтаксис EL-элементов при переходе от стандартного синтаксиса JSP к XML-синтаксису.

Есть два вида EL-элементов: с немедленным и отложенным выполнением. Вид этих элементов, соответственно:

`${текст_EL-выражения}` и `#{текст_EL-выражения}`.

Примеры:

```
${requestScope.username}
${username}, ${product.model}
#{client.name}
#{operations.addClient}
```

Важное отличие EL-элементов с точки зрения их вычисления в том, что они не преобразуются в соответствующий Java-код на первом этапе трансляции. Текст EL-элементов передается как параметр EL-вычислителю, который интерпретирует их на этапе выполнения. Это означает, что ошибки в обычных выражениях могут быть обнаружены на втором этапе (при трансляции компилятором Java). Ошибки в EL-элементах могут быть обнаружены только на этапе обработки запросов. EL-вычислитель реализуется специфическим для сервера приложений классом, на первом этапе трансляции EL-элементы преобразуются в обращения к этому вычислителю для интерпретации EL-элементов. EL-элементы могут использоваться также для формирования значений атрибутов действий на этапе обработки запросов.

Два вида EL-элементов появились в результате унификации языков EL-элементов, использовавшихся в спецификациях JSP 2.0 и JSF 1.1. Эти виды элементов обладают общим синтаксисом, но различной семантикой выполнения.

Значение EL-элемента с немедленным выполнением вычисляется сразу, как только элемент встречается на JSP-странице. Значение же EL-элемента с отложенным выполнением вычисляется тогда, когда это необходимо. Например, при обработке запроса к JSF отложенный EL-элемент может вычисляться на шаге «Применить значения из запроса» и/или «Отобразить ответ». С точки зрения автора JSP-страницы, указываемый в немедленном EL-элементе объект должен существовать до отображения страницы, а указываемый в отложенном EL-элементе – создается при первом обращении к нему.

Кроме того, EL-элемент с отложенным выполнением может использоваться в качестве левой стороны оператора присваивания (т.н. lvalue). Например, следующее действие на странице JSF приведет к тому, что при повторном запросе к странице свойству `property` управляемого бина `bean` будет присвоено введенное пользователем значение:

```
<h:inputText value="#{bean.property}"/>
```

Еще одно отличие: значением EL-элемента с отложенным выполнением может быть не свойство объекта, а его метод (т.н. method expression). В JSF это используется для связи различных обработчиков с компонентами интерфейса пользователя:

```
<h:commandButton value="Refresh" actionListener="#{bean.refresh}"/>
```

Синтаксис EL-выражений

Язык EL впервые был определен в спецификации JSTL 1.0 и изначально не являлся частью спецификации JSP. Начиная с версии JSP 2.0 EL включен в состав JSP.Язык EL

был разработан с учетом языков XPath и ECMAScript (стандартизованная консорциумом ECMA версия языка JavaScript).

EL-выражения конструируются из имен объектов (встроенных и пользовательских), арифметических и логических операций, операций сравнения, вызовов функций и операций обращения к хэш-таблицам (Map), спискам (List), массивам и свойствам JavaBeans.

Литералы

Булевские литералы задаются значениями `true` и `false`. Числовые значения (целочисленные и дробные) задаются как в языке Java. Строковые литералы заключаются в двойные или одинарные кавычки, причем распознаются esc-последовательности `\"`, `\'`, `\\`. Предусмотрено также значение `null`.

Операции [] и .

В языке EL операции `[]` и `.` унифицированы, то есть `вырА.идБ` эквивалентно `вырА["идБ"]`. Выражение `вырА[вырБ]` вычисляется следующим образом:

1) вычисляем `вырА`, получаем значение `значА`, если оно равно `null`, то значение всего выражения тоже `null`;
2) вычисляем `вырБ`, получаем `значБ`, если оно равно `null`, то значение всего выражения тоже `null`;
3) если `значА` – объект типа `java.util.Map`, то результат всего выражения – значение, извлекаемое методом `get()` по ключу, в качестве которого используется `значБ`. Если такого ключа не существует, то значение всего выражения `null`;
4) если `значА` – объект типа `java.util.List`, то результат всего выражения – это элемент списка, который извлекается методом `get()`, а в качестве индекса элемента используется `значБ`, которое приводится к типу `int` по определенным правилам. Если метод `get()` выбрасывает исключение `IndexOutOfBoundsException`, то значение всего выражения `null`. Если выбрасывается какое-либо другое исключение или `значБ` не может быть преобразовано к типу `int`, то возникает ошибка;
5) если `значА` – массив, то значение этого выражения – это элемент массива, который извлекается методом `Array.get()`, в качестве индекса используется `значБ`, которое преобразуется к типу `int`. Если метод `Array.get()` выбрасывает исключение `ArrayIndexOutOfBoundsException`, то значение всего выражения `null`. Если выбрасывается какое-либо другое исключение или `значБ` не может быть преобразовано к типу `int`, то возникает ошибка;
6) во всех остальных случаях `значА` предполагается объектом JavaBean, `значБ` преобразуется в строку и используется как имя свойства этого JavaBean. Значение всего выражения – это результат вызова get-метода для получения значения этого свойства. Если метод отсутствует или выбрасывает исключение, то возникает ошибка.

При вычислении отложенных EL-выражений использование операций [] и . имеет следующую особенность. Если требуется присвоить значение объекту, на который указывает EL-выражение (то есть EL-выражение является lvalue), а на шаге 1 или 2 получено значение `null`, то возникает исключение `PropertyNotFoundException`.

Арифметические операции

Арифметические операции могут оперировать значениями типа `java.lang.Long` и `java.math.BigInteger` (целочисленные значения), а также `java.lang.Double` и

`java.math.BigDecimal` (дробные значения). Предусмотрено пять операций: `+`, `–`, `*`, `/` или `div`, `%` или `mod` (остаток от деления). Есть также операция унарный минус. Любая арифметическая операция возвращает ноль, если оба операнда равны `null`. Один из операндов может иметь тип `String` и содержать строковое представление числа.

Операции сравнения

В языке EL определены такие же операции сравнения, как в языке Java, но для них дополнительно введены буквенные обозначения:

```
== – eq
!= - ne
<  - lt
<= - le
>  - gt
>= - ge
```

Операции сравнения могут оперировать значениями любых типов, при этом могут выполняться неявные преобразования. Для булевских операндов предусмотрены операции `==` и `!=`. Для строк выполняется лексикографическое сравнение.

Логические операции

В языке EL определены такие же логические операции, как в языке Java, но дополнительно вводятся буквенные обозначения:

```
&&   –   and
||   –   or
!    –   not
```

Операции могут применяться к значениям любых типов, при этом они преобразуются к значениям типа `boolean`.

`empty` – унарная префиксная операция для проверки значения на пустоту. Возвращает `true`, если аргумент имеет значение `null`, либо это пустая строка, пустой массив, пустой хэш или пустая коллекция, иначе – `false`.

`? :` – условная операция, аналогичная условной операции в языке Java, при этом результат первого выражения преобразуется к типу `boolean`.

Приоритет операций

1. `[ ]  .`
2. `( )`
3. – (унарный) `not ! empty`
4. `*  div  /  mod  %`
5. `+  –`
6. `lt  <  gt  >  le  <=  ge  >=`
7. `eq  ==  ne  !=`
8. `and  &&`
9. `or  ||`
10. `?:`

Именованные переменные

При вычислении EL-выражений имена переменных преобразуются в объекты с помощью "резольверов" (объектов типа `ELResolver`). "Резольверы" выстраиваются в цепочку, и их порядок определяет последовательность поиска объектов и алгоритм вычисления операции [].

Выстраиваемая по умолчанию цепочка "резольверов" предусматривает наличие встроенных объектов, доступных по предопределенным именам, остальные имена считаются атрибутами контекста JSP-страницы, запроса, сессии или контекста веб-приложения. Поиск выполняется методом `findAttribute()` объекта типа `PageContext` последовательно в областях видимости `page`, `request`, `session`, `application`. Первый найденный атрибут используется для получения значения переменной. Если атрибут ни в одной из областей видимости не найден, то значение переменной считается равным `null`. Для переменных с предопределенными именами всегда используются встроенные объекты, даже если в одной из областей видимости существует атрибут с таким же именем.

Встроенные объекты

Для немедленных EL-выражений, используемых в JSP-страницах, доступны следующие встроенные объекты:

- `pageContext` – контекст JSP-страницы, то есть объект типа `javax.servlet.jsp.PageContext`.
- `pageScope` – объект типа `java.util.Map`, который содержит значения атрибутов контекста JSP-страницы, ключами являются имена атрибутов.
- `requestScope` – объект типа `java.util.Map`, который содержит значения атрибутов запроса, ключами являются имена атрибутов.
- `sessionScope` – объект типа `java.util.Map`, который содержит значения атрибутов сессии, ключами являются имена атрибутов.
- `applicationScope` – объект типа `java.util.Map`, который содержит значения атрибутов контекста веб-приложения, ключами являются имена атрибутов.
- `param` – объект типа `java.util.Map`, содержащий значения параметров запроса, которые были извлечены методом `ServletRequest.getParameter(String name)`. Для параметров, которые имеют несколько значений, здесь будет доступно только первое значение. Все значения представляют собой объекты типа `String`.
- `paramValues` – объект типа `java.util.Map`, содержащий значения параметров запроса, которые были извлечены методом `ServletRequest.getParameterValues(String name)`. Значения всех параметров представляются в виде массивов строк (`String[]`). Для параметров, которые имеют несколько значений, будут доступны все значения.
- `header` – объект типа `java.util.Map`, содержащий значения заголовков запроса, которые были извлечены методом `HttpServletRequest.getHeader(String name)`. Для заголовков, имеющих несколько значений, здесь будет присутствовать только первое из них. Все значения представляют собой объекты типа `String`, ключами являются имена заголовков.
- `headerValues` – объект типа `java.util.Map`, содержащий значения заголовков запроса, которые были извлечены методом `HttpServletRequest.getHeaderValues(String name)`. Значения всех заголовков представляются в виде массивов строк (`String[]`), ключами являются имена заголовков. Для заголовков, имеющих несколько значений, будут доступны все значения.
- `cookie` – объект типа `java.util.Map`, содержащий все куки запроса, извлеченные методом `HttpServletRequest.getCookies()`. Ключами являются имена куки, а значения представляются в виде объектов типа `javax.servlet.http.Cookie`. Если в запросе есть несколько куки с одинаковым именем, здесь будет присутствовать только первое куки, найденное в массиве, а порядок куки в массиве, возвращаемом методом `getCookie()`, не определен.

- `initParam` – объект типа `java.util.Map`, содержащий значения параметров инициализации контекста веб-приложения, извлеченные методом `ServletContext.getInitParameter(String name)`.

В отложенных EL-выражениях, обрабатываемых каркасом JSF, доступны следующие встроенные объекты:

- `facesContext` – объект типа `javax.faces.context.FacesContext`, представляющий контекст запроса в рамках каркаса JSF.
- `view` – объект типа `javax.faces.component.UIViewRoot`, представляющий корень дерева компонентов в текущем JSF-представлении.
- `request` – объект типа `javax.servlet.http.HttpServletRequest`, представляющий целиком запрос.
- `requestScope` – аналогично немедленным EL-выражениям.
- `session` – объект типа `javax.servlet.http.HttpSession`, представляющий HTTP-сессию текущего пользователя.
- `sessionScope` – аналогично немедленным EL-выражениям.
- `application` – объект типа `javax.servlet.ServletContext`, представляющий веб-приложение целиком.
- `applicationScope` – аналогично немедленным EL-выражениям.
- `param, paramValues, header, headerValues, cookie, initParam` – аналогично немедленным EL-выражениям.

Имя объекта	Тип	Описание	$	# (JSF)
pageContext	PageContext	контекст JSP-страницы	+	
pageScope	Map	атрибуты JSP-страницы	+	
request	HttpServletRequest	запрос		+
requestScope	Map	атрибуты запроса	+	+
session	HttpSession	HTTP-сессия		+
sessionScope	Map	атрибуты сессии	+	+
application	ServletContext	веб-приложение		+
applicationScope	Map	атрибуты веб-приложения	+	+
param	Map	параметры запроса	+	+
paramValues	Map	параметры запроса со всеми значениями	+	+
header	Map	заголовки запроса	+	+
headerValues	Map	заголовки запроса со всеми значениями	+	+
cookie	Map	куки запроса	+	+
initParam	Map	параметры инициализации	+	+
facesContext	FacesContext	контекст запроса в каркасе JSF		+
view	UIViewRoot	корень дерева компонентов в текущем JSF-представлении		+

Функции

Та часть технологии JSP, которая позволяет расширять набор стандартных действий действиями, определяемыми разработчиком, позволяет также определить и функции, которые можно использовать в EL-выражениях. Эти функции реализуются на Java, а описываются в TLD, который должен быть подключен к JSP-странице с помощью директивы `taglib`. Функции реализуются статическими методами. В EL-выражениях функции вызываются по квалифицированному имени с использованием префикса,

введенного директивой `taglib` (то есть аналогично действиям, определенным программистом). Пример:

– описание функции в TLD:

```
<taglib>
      …
      <function>
            <!-- имя функции, по которому она будет вызываться в EL-выражениях
-->
            <name>nickname</name>
            <!-- полностью квалифицированное имя класса, содержащего
статический метод, реализующий функцию -->
            <function-class>
                  Mypackage.MyFunctions
            </function-class>
            <!-- заголовок статического метода, реализующего функцию -->
            <function-signature>
                  java.lang.String nickname(java.lang.String)
            </function-signature>
      </function>
</taglib>
```

– использование функции:

```
<%@taglib prefix="my" uri="http://my.com/functions" %>
<h1>Hello, dear ${my:nickname(user)}</h1>
```

Реализующий функцию класс должен быть публичным, реализующий функцию метод должен быть публичным и статическим. В заголовке метода, указываемом в TLD, типы возвращаемого значения и параметров должны быть указаны в полностью квалифицированной форме. В пределах одного TLD не должно быть двух функций с одинаковыми именами, иначе возникнет ошибка трансляции JSP-страницы.

Правила преобразования типов

Преобразование типов при вычислении EL-выражений отличается от тех правил, которые приняты в языке Java. Правила преобразования сформулированы в спецификации исключительно в терминах объектных типов. Если преобразуемое значение простого типа, то оно сначала преобразуется к значению соответствующего объектного типа (так называемый boxing). Если целевой тип простой, то полученный в результате вычисления объект преобразуется к значению соответствующего простого типа (unboxing).

1. Преобразование значения в строку.
   1) если значение равно `null`, то результат преобразования – пустая строка;
   2) для всех остальных случаев результат преобразования – это результат вызова метода объекта `toString()`
2. Преобразование значения в число.
   1) если значение равно `null` или пустая строка, то результат преобразования равен `0`;
   2) если значение имеет тип `Boolean`, то ошибка;
   3) если значение имеет тип `String`, то оно должно быть строковым представлением числа, по которому создается объект соответствующего числового типа;
   4) в остальных случаях, в зависимости от типа значения и целевого типа (тип значения должен быть числовым) выполняется извлечение значения простого типа подходящим методом (`byteValue()`, `shortValue()`, `intValue()` и т.д.) и создание по нему объекта соответствующего числового типа;
   5) если значение имеет тип `Character`, то выполняется создание нового объекта типа `Short` по числовому коду символа, который извлекается методом `charValue()`, и далее этот объект преобразуется к целевому типу по предыдущему правилу.
3. Преобразование значения в символ.

1) если значение равно `null` или пустая строка, то результат преобразования – символ с кодом `0`;
2) если значение имеет тип `Boolean`, то ошибка;
3) если значение имеет тип `String`, то результат – первый символ строки;
4) если значение имеет числовой тип, то оно преобразуется к типу `Short`, по которому создается объект типа `Character` (то есть символ с соответствующим кодом).

4. Преобразование значения к логическому типу.
   1) если значение равно `null` или пустая строка, то результат преобразования – `false`;
   2) если значение имеет тип `String`, то оно должно быть строковым представлением одной из логических констант (`true` или `false`), по которому создается объект типа `Boolean`.
5. Преобразование произвольного объекта к произвольному объектному типу.
   1) если значение равно `null`, то результат тоже равен `null`;
   2) если преобразуемое значение совместимо с целевым типом по присваиванию, то значение преобразуется по правилам языка Java;
   3) если значение имеет тип `String`, то результатом преобразования может быть значение `null` или объект целевого типа в зависимости от дополнительных условий.

Случаи, не рассмотренные в этих правилах, приводят к возникновению ошибки. Если методы или конструкторы, вызываемые при выполнении этих правил, выбрасывают исключения, то преобразование приводит к ошибке.

Примеры использования EL-выражений

Использование EL-выражения для вычисления значения атрибута во время выполнения JSP-страницы:

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

В этом случае выражение вычисляется и его значение приводится к допустимому типу значений атрибута с помощью описанных выше правил преобразования типов.

Значение атрибута также может содержать несколько EL-выражений:

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

В этом случае EL-выражения вычисляются слева направо, их значения преобразуются к строковому типу и складываются с промежуточным текстом, после чего результирующая строка приводится к допустимому типу значений атрибута.

EL-выражения могут использоваться непосредственно в тексте шаблона, в том числе внутри действий, как стандартных, так и определяемых пользователем:

```
<tr>
    <td>${client.lastName} ${client.firstName}</td>
    <td>${client.place}</td>
</tr>
<r:report>
    <r:title>${client.lastName} ${client.firstName} Orders</r:title>
</r:report>
```

Примеры использования встроенных объектов в EL-выражениях:

EL-выражение	Результат
`${pageContext.request.requestURI}`	URI запроса
`${sessionScope.profile}`	Значение атрибута сессии с названием `profile` (`null`, если атрибут не найден)
`${param.productId}`	Строковое значение параметра запроса `productId` или `null`, если параметр не задан