# 9 Expression Language

Questo capitolo introduce l’Expression Language (EL), che fornisce un importante meccanismo che agevola la presentazione (pagine web) per comunicare con la logica dell’applicazione (managed beans). L’EL è usato in molte tecnologie JavaEE, come nella JavaServer Faces, JSP, Context and Dependency Injection (CDI). L’EL può essere inoltre usato in ambienti a sé. Questo capitolo mostra solo l’uso dell’EL nei Java EE containers.

## 9.1 Overview of the EL

L’EL agevola l’autore delle pagine nell’usare espressioni semplici per accedere dinamicamente ai dati da componenti JavaBeans. Per esempio, l’attributo test del seguente tag condizionale è fornito con un’espressione EL che compara 0 con il numero di elementi nel session-scoped bean chiamato cart.

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

...

</c:if>

Per riassumere, l’EL fornisce un modo per usare semplici espressioni per eseguire I seguenti compiti:

* Leggere dinamicamente i dati dell’applicazione memorizzati nei componenti JavaBeans, varie strutture di dati, e oggetti impliciti
* Scrivere dinamicamente dati, come gli input dell’utente nei form, nei componenti JavaBeans
* Invocare metodi statici e pubblici arbitrari
* Eseguire dinamicamente operazioni aritmetiche, booleane e con le stringhe
* Costruire dinamicamente oggetti in collezione e eseguire operazioni sulle collezioni

In una pagina JavaServer Faces, l’espressione EL può essere usata o in testo statico, o nell’attributo di un custom tag o standard action.

Infine, l’EL fornisce un API per risolvere le espressioni così che i custom resolvers che possono gestire le espressioni non ancora supportate dall’EL possono essere implementate.

## 9.2 Immediate and Deferred Evaluation Syntax

L’EL supporta sia la valutazione immediate che la valutazione rinviata delle espressioni. La **valutazione immediata (immediate)** significa che l’espressione è valutata e il risultato restituito appena è caricata la pagina. La **valutazione rinviata (deferred)** significa che la tecnologia che usa l’EL può usare la propria macchina per valutare l’espressione dopo un po’ di tempo, durante il ciclo di vita della pagina, quando è appropriato farlo.

Le espressioni valutate immediatamente usano la sintassi ${}. Le espressioni in cui la valutazione è rinviata usano la sintassi #{}.

A causa del ciclo di vita a più fasi, la tecnologia JavaServer Faces usa maggiormente le espressioni a valutazione rinviata. Durante il ciclo di vita, gli eventi dei componenti sono gestiti, i dati convalidati, e altri compiti eseguiti in un ordine particolare. Perciò, un’implementazione JavaServer Faces deve rinviare la valutazione delle espressioni fino al punto appropriato del ciclo di vita.

Le altre tecnologie che usano l’EL potrebbero avere ragioni differenti per utilizzare le espressioni rinviate.

### 9.2.1 Immediate Evaluation

Tutte le espressioni che usano la sintassi ${} sono valutate immediatamente. Queste espressioni possono apparire come parte di un testo di template (static) o come valore di un tag che può accettare espressioni di runtime.

Il seguente esempio mostra un tag il cui attributo value si riferisce a un espressione a valutazione immediata che aggiorna la quantità di libri recuperata dal baking bean chiamato catalog:

<h:outputText value=”${catalog.bookQuantity}” />

L’implementazione JavaServer Faces valuta l’espressione ${catalog.bookQuantity}, la converte, e passa il valore restituito al tag che la gestisce. Il valore è aggiornato sulla pagina.

### 9.2.2 Deferred Evaluation

Le espressioni a valutazione rimandata prendono la forma #{expr} e possono essere valutate in altre fasi del ciclo di vita della pagina come definito da qualsiasi tecnologia stia utilizzando l’espressione. Nel caso della tecnologia JavaServer Faces, il suo controller può valutare l’espressione in diverse fasi del ciclo di vita, in base a come l’espressione è utilizzata nella pagina.

Il seguente esempio mostra un tag JavaServer Faces h:inputText, che rappresenta un componente di campo in cui un utente inserisce un valore. L’attributo value del tag h:inputText si riferisce a un’espressione a valutazione rinviata che punta alla proprietà name del bean customer:

<h:inputText id=”name” value=”#{customer.name}” />

Per una richiesta iniziale della pagina contenente questo tag, l’implementazione JavaServer Faces valuta l’espressione #{customer.name} durante la fase di rendering-risposta del ciclo di vita. Durante questa fase, l’espressione accede solo al valore del name dal bean customer, come in una valutazione immediata.

Per una richiesta postback, l’implementazione JavaServer Faces valuta l’espressione in diverse fasi del ciclo di vita, durante cui il valore è recuperato dalla richiesta, validato, e propagato al customer bean.

Come mostrato in questo esempio, le espressioni a valutazione rinviata possono essere

* Espressioni di valore che possono essere usate per leggere e scrivere dati
* Espressioni di metodo

## 9.3 Value and Method Expressions

L’EL definisce due tipi di espressioni: espressioni di valore e espressioni di metodo. Le **espressioni di valore** possono essere valutate per produrre un valore, e le **espressioni di metodo** sono usate come riferimento ad un metodo.

### 9.3.1 Value Expression

Le espressioni di valore possono essere ulteriormente categorizzate in espressioni **rvalue** e **lvalue**. Un’espressione lvalue può specificare un target, come un oggetto, una proprietà di un bean, o un elemento di una collezione, a cui può essere assegnato un valore. Un espressione rvalue non può specificare tale target.

Tutte le espressioni che sono valutate immediatamente usano i delimitatori ${}, e sebbene l’espressione può essere un’espressione lvalue, non avverrà alcuna assegnazione. Le espressioni la cui valutazione può essere rimandata usa i delimitatori #{} e può agire sia come espressione rvalue che come lvalue; se l’espressione è lvalue, può essere assegnata ad un nuovo valore. Considera le due seguenti espressioni di valore:

${customer.name}

#{customer.name}

Il primo usa la sintassi di valutazione immediata, mentre il successivo usa la sintassi di valutazione rinviata. La prima espressione accede alla proprietà name, ne prende il valore, e lo passa al tag che lo gestisce. Con la seconda espressione, il tag che la gestisce può rinviare la valutazione dell’espressione ad un secondo momento nel ciclo di vita della pagina se la tecnologia che usa il tag lo permette.

Nel caso della tecnologia JavaServer Faces, l’espressione del tag successivo è valutata immediatamente durante la richiesta iniziale della pagina. Durante una richiesta di postback, questa espressione può essere usata per impostare il valore della proprietà name con l’input dell’utente.

#### 9.3.1.1 Referencing Objects

Un identificatore a livello alto (come customer nell’espressione customer.name) può far riferimento ai seguenti oggetti:

* Parametri Lambda
* Variabili EL
* Managed beans
* Oggetti impliciti
* Classi di campi e metodi statici

Per far riferimento a questi oggetti, scrivi un’espressione usando una variabile che è il nome dell’oggetto. La seguente espressione si riferisce ad un managed bean chiamato customer:

${customer}

Puoi usare un custom EL resolver per modificare il modo in cui le variabili sono risolte. Per esempio, puoi fornire un EL resolver che intercetta gli oggetti con il nome customer, cosicché ${costumer} restituisca invece un valore nell’EL resolver. (La tecnologia JavaServer Faces usa un EL resolver per gestire i managed beans.)

Una costante enum è un caso speciale di campo statico, e puoi far riferimento a tale costante direttamente. Per esempio, considera questa classe enum:

public enum Suit {hearts, spades, diamonds, clubs}

Nella seguente espressione, in cui mySuit è un’istanza di Suit, puoi confrontare suit.hearts con l’istanza:

${mySuit == suit.hearts}

#### 9.3.1.2 Referencing Object Properties or Collection Elements

Per fare riferimento a proprietà di un bean, campi o metodi static di una classe, o oggetti di una collezione, puoi usare la notazione . o []. La stessa sintassi può essere usata per gli attributi di un oggetto implicito, poiché gli attributi sono allocati in una mappa.

Per far riferimento alla proprietà name di un customer bean, usa o l’espressione ${customer.name} o l’espressione ${customer[“name”]}. Qui, la parte nelle parentesi quadre è una String letterale, il nome della proprietà a cui ci si riferisce. La sintassi [] è più generale rispetto alla sintassi . , poiché la parte tra parentesi quadre può essere una qualsiasi espressione Stringa, non solo letterale.

Puoi usare apici singoli o doppi (virgolette) per la String. Puoi inoltre combinare le notazioni [] e . , come mostrato di seguito:

${customer.address[“street”]}

Puoi far riferimento ad un campo o un metodo statico usando la sintassi *classname.field*, come nel seguente esempio:

Boolean.FALSE

Il *classname* è il nome della classe senza il nome del package. Di default, tutti i packages java.lang sono importati. Puoi importare altri packages, classi e campi statici necessari.

Se stai accedendo ad un oggetto in un array o in una lista, devi usare la notazione [] e specificare un indice nell’array o nella lista. L’indice è un’espressione che può essere convertita in int. L’esempio seguente si riferisce al secondo degli ordini del customer, considerando customer.orders come una List:

${customer.orders[1]}

Se stai accedendo a un oggetto in una Map, devi specificare la key per la Map. Se la key è una String letterale, può essere usato il punto ( . ). Considerando customer.orders come una Map con una chiave String, i seguenti esempi si riferiscono ad un oggetto con la key “socks”:

${customer.orders[“socks”]}

${customer.orders.socks}

#### 9.3.1.3 Referencing Literals

L’EL definisce i seguenti letterali:

* **Boolean**: true e false
* **Integer**: come in Java
* **Floating-point**: come in Java
* **String**: Con apici singoli o doppi; “ è evaso con \”, ‘ è evaso con \’, e \ è evaso con \\
* **Null**: null

Ecco alcuni esempi:

* ${"literal"}
* ${true}
* ${57}

#### 9.3.1.4 Parametrized Method Calls

L’EL offre supporto per le chiamate di metodi parametrizzati.

Sia . che [] possono essere usati per invocare le chiamate dei metodi con parametri, come mostrato nella sintassi della seguente espressione:

* *expr-a*[*expr-b*] (*parameters*)
* *expr-a*.*identifier-b* (*parameters*)

Nella sintassi della prima espressione, *expr-a* è valutata per rappresentare un oggetto bean. L’espressione *expr-b* è valutata e costretta (cast) in una stringa che rappresenta un metodo nel bean rappresentato da *expr-a*. Nella sintassi della seconda espressione, *expr-a* è valutata per rappresentare un oggetto bean, e l’*identifier-b* è una stringa che rappresenta un metodo nell’oggetto bean. I *parameters* tra parentesi sono gli argomenti per l’invocazione del metodo. I parametri possono essere zero o piò valori dell’espressione, separati da virgole.

I parametri sono supportati sia per le espressioni di valore sia le espressioni di metodo. Nel seguente esempio, che è un tag modificato dall’applicazione guessnumber, un numero random è fornito come argomento piuttosto che da un input dell’utente per la chiamata del metodo:

<h:inputText value=”#{userNumberBean.userNumber(‘5’)}”>

L’esempio precedente usa un’espressione di valore.

Considera il seguente esempio di un tag componente JavaServer Faces che usa un’espressione di metodo:

<h:commandButton action=”{trader.buy}” value=”buy” />

L’espressione EL trader.buy chiama il metodo buy del bean trader. Puoi modificare il tag per passargli un parametro. Ecco il tag modificato con il parametro passato:

<h:commandButton action=”#{trader.buy(‘SOMESTOCK’)}” value=”buy” />

Nell’esempio precedente, si passa la stringa ‘SOMESTOCK’ (un simbolico stock) come parametro al metodo buy.

#### 9.3.1.5 Where Value Expression Can Be Used

Le espressioni di valore che usano i delimitatori ${}possono essere usati

* in testo statico
* In qualsiasi tag standard o personalizzato che accetta un’espressione

Il valore di un’espressione in testo statico è compilato e inserito nell’output corrente. Ecco un esempio di un’espressione integrata in testo statico:

<some:tag>

some text ${expr} some text

<some:tag>

Un attributo di un tag può essere impostato nel seguente modo:

* Con un costrutto di un’espressione singola:

<some:tag value=”${expre}”/>

<another:tag value=”#{expr}”/>

Queste espressioni sono valutate e il risultato convertito nel tipo atteso dall’attributo

* Con una o più espressioni separate o circondate da testo:

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

<another:tag value=”some#{expr}#{expr}text#{expr}”/>

Questi tipi di espressione, chiamati **espressioni composte**, sono valutate da sinistra a destra. Ogni espressione integrata nell’espressione composta è convertita in una String e concatenata con qualsiasi testo intromesso. La String risultante è poi convertita nel tipo atteso dall’attributo.

* Con solo testo:

<some:tag value=”sometext”/>

Il valore String dell’attributo è convertito nel tipo atteso dall’attributo.

Puoi usare l’operatore di concatenazione di stringhe += per creare una singola espressione da ciò che sarebbe altrimenti un’espressione composta. Per esempio, puoi cambiare l’espressione composta

<some:tag value=”sometext ${expr} moretext”/>

In

<some:tag value=”${sometext += expr += moretext}”/>

Tutte le espressioni usate per impostare i valori dell’attributo sono valutate nel contest di un tipo atteso. Se il risultato della valutazione dell’espressione non combacia esattamente con il tipo atteso, sarà eseguita una conversione di tipo. Per esempio, l’espressione ${1.2E4} fornito come valore di un attributo float risulterà nella seguente conversione:

Float.valueOf(“1.2E4”).floatValue()

### 9.3.2 Method Expressions

Un’altra caratteristica dell’EL è il suo supporto di espressioni di metodo rinviato. Un’espressione di metodo è usata per far riferimento ad un metodo pubblico di un bean e ha la stessa sintassi di un’espressione lvalue.

Nella tecnologia JavaServer Faces, un component tag rappresenta un componente su una pagina. Il component tag usa espressioni di metodo per specificare i metodi che possono essere invocati per eseguire determinati processi per il componente. Questi metodi sono necessari per gestire gli eventi che il componente genera e per validare i dati del componente, come mostrato nell’esempio:

<h:form>

<h:inputText

id=”name”

value=”#{customer.name}”

validator=”#{customer.validateName}”/>

<h:commandButton

id=”submit”

action=”#{customer.submit}” />

</h:form>

Il tag h:inputText mostra un campo. L’attributo validator di questo tag h:inputText di riferisce a un metodo, chiamato validateName, nel bean, chiamato customer.

Poiché un metodo può essere invocato durante diverse fasi del ciclo di vita, le espressioni di metodo devono sempre usare la sintassi di valutazione rinviata.

Come le espressione lvalue, le espressioni di metodo possono usare gli operatori . e []. Per esempio, #{object.method} è equivalente a #{object[“method”]}. Il letterale nelle [] è convertito in una String ed è usato per trovare il nome del metodo che combacia.

Le espressioni di metodo possono essere usate solo in attributi di tag e solo nel seguente modo:

* Con un costruttore con una singola espressione, dove *bean* si riferisce ad un componente JavaBeans e *method* si riferisce ad un metodo del componente JavaBeans:

<some:tag value=”#{*bean.method}”/>*

L’espressione è valutata in un’espressione di metodo, che viene passata al gestore del tag. Il metodo rappresentato dall’espressione di metodo può poi essere invocato successivamente.

* Con solo testo:

<some:tag value=”sometext”/>

Le espressioni di metodo supportano i letterali soprattutto per supportare gli attributi action nella tecnologia JavaServer Faces. Quando il metodo riferito da questa espressione di metodo è invocato, il metodo restituisce la String letterale, che è poi convertita nel tipo restituito atteso, come definito dal tag descriptor del tag library.

### 9.3.3 Lambda Expressions

Una lambda expression è un’espressione di valore con parametri. La sintassi è simile a quella delle lambda expression del linguaggio Java, eccetto che nel EL il corpo della lambda expression è un’espressione EL.

Una lambda expression usa l’operatore freccia ( -> ). Gli identificatori a sinistra dell’operatore sono chiamati identificatori lambda. Il corpo, a destra dell’operatore, dev’essere un’espressione EL. I parametri lambda sono racchiusi tra parentesi; le parentesi possono essere omesse solo se c’è un solo parametro. Ecco qualche esempio:

x -> x+1

(x, y) -> x + y

() -> 64

Una lambda expression si comporta come una funzione. Può essere invocata immediatamente. Per esempio, la seguente invocazione valuta a 7:

((x, y) -> x + y)(3, 4)

Puoi usare una lambda expression in congiunzione con gli operatori = e ;. Per esempio, il seguente codice assegna alla precedente lambda expression una variabile e poi la invoca. Il risultato è ancora 7:

v = (x, y) -> x + y; v(3, 4)

Una lambda expression può essere inoltre passata come argomento a un metodo e essere invocato in un metodo. Può essere inoltre annidata in un’altra lambda expression.

## 9.4 Operations on Collection Objects

L’EL supporta le operazioni sugli oggetti collezione: sets, liste e maps. Ciò permette la creazione dinamica di oggetti collezione, su cui si può poi operare usando flussi e pipelines.

Per esempio puoi costruire un set come:

{1,2,3}

Puoi costruire una lista come segue; una lista può contenere vari tipi di elementi:

[1,2,3]

[1, “two”, [three,four]]

Puoi costriure una map usando i due punti per definire l’accesso:

{“one”:1, “two”:2, “three”:3}

Si opera sugli oggetti collezione usando le chiamate del metodo al flusso di elementi derivato dalla collezione. Alcune operazioni restituiscono un altro flusso, che permette operazioni aggiuntive. Perciò, puoi collegare queste operazioni in una pipeline.

Un flusso pipeline consiste in:

* Una fonte (on oggetto Stream)
* Un qualsiasi numero di operazioni intermedie che restituiscono un flusso (per esempio, filter e map)
* Un’operazione finale che non restituisce uno stream (per esempio, toList())

Il metodo stream ottiene uno Stream da un java.util.Collection o un array Java. Le operazioni di flusso non modificano l’oggetto collezione originale.

Per esempio, potresti generare una lista di titoli di libri di storia in questo modo:

books.stream().filter(b->b.category == ‘history’)

.map(b->b.title)

.toList()

Il seguente esempio più semplice restituisce una versione ordinate della lista originale:

[1,3,5,2].stream().sorted().toList()

I flussi e le operazioni sul flusso sono documentati della Java SE 8 API documentation, available at <http://docs.oracle.com/javase/8/docs/api/>. I seguenti sottoinsieme di operazioni è supportato dall’EL:

allMatch

anyMatch

average

count

distinct

filter

findFirst

flatMap

forEach

iterator

limit

map

max

min

noneMatch

peek

reduce

sorted

substream

sum

toArray

toList

Guarda la specifica EL a <http://www.jcp.org/en/jsr/detail?id=341> per i dettagli sulle operazioni.

## 9.5 Operators

In aggiunta a . e [], l’EL fornisce i seguenti operatori, usabili solo nelle espressioni rvalue.

* **Aritmetici**: +, - (binario), \*, / e div, % e mod, - (unario).
* **Concatenazione di Stringhe**: +=.
* **Logici**: and, &&, or, ||, not, !.
* **Relazionali**: ==, eq, !=, ne, <, lt, >, gt, <=, ge, >=, le. Le comparazioni possono essere fatte con altri valori o con Boolean, stringhe, integer, o floating-point letterali.
* **Empty**: l’operatore empty è un’operazione prefissata che può essere usata per determinare se un valore è null o empty .
* **Condizionali**: A ? B : C. Valuta B o C, in base al risultato della valutazione di A.
* **Lambda Expression**: ->.
* **Assegnazione**: =.
* **Punto e Virgola**: ;.

La precedenza degli operatori, dall’alto verso il basso, da sinistra verso destra:

* [] .
* () (usato per cambiare la precendenza)
* - (unario) not ! empty
* \* / div % mod
* + - (binario)
* +=
* <> <= >= lt gt le ge
* == != eq ne
* && and
* || or
* ? :
* ->
* =
* ;

## 9.6 Reserved Words

Le seguenti parole sono riservate per l’EL e non dovrebbero essere usate come identificatori:

and

or

not

eq

ne

lt

gt

le

ge

true

false

null

instanceof

empty

div

mod

## 9.7 Examples of EL Expressions

