BAB 6

JSP Lanjutan

6.1 Pendahuluan

Dalam pembahasan JSP sebelumnya, Kita telah mempelajari syntax dasar : bagaimana cara menggunakan scriptlets, expressions, JavaBeans, dan JSP action yang sudah dikenal untuk menyediakan data dinamis beserta content yang statis; bagaimana menggunakan directive untuk menampilkan perintah pengolahan halaman yang spesifik kepada container. Kita juga telah menguraikan peran JSP di dalam suatu aplikasi web untuk bertindak sebagai layer presentasi dan lain-lain.

Pokok pembahasan lainnya yang dijelaskan pada bab sebelumnya adalah kemampuan JSP untuk menghasilkan content yang dinamis selama penyimpanan elemen pemrograman dibuat menjadi minimalis. Hal ini untuk memenuhi kepada personil layer presentasi yang dipersembahkan kepada siapapun yang memiliki sedikit pengalaman dalam pemrograman dengan Java atau bahasa pemrograman lainnya. Perubahan ke bentuk berbasis text sangat banyak mengurangi baris dari perancang kode dibandingkan dengan servlet : kemudian content file JSP menjadi sangat mirip dengan standard HTML yang digunakan untuk handling. Proses refactor logika dari JSP ke dalam servlet dan menunjukkan hasil sebagai JavaBeans yang selanjutnya mengurangi jumlah tersebut. Meskipun demikian dalam posisi ini, kode Java (melalui scriptlet) merupakan elemen yang tidak dapat dihindari untuk JSP. Ada kemampuan yang sederhana yang tidak dapat ditunjukkan oleh elemen JSP yang dibahas sejauh ini, seperti iterasi list dan logika bersarang(pernyataan if-else).

Pada bab ini, Kita akan membahas dua elemen JSP yang akan mengurangi pemakaian scriptlet dan expression dalam JSP menjadi nol. Mereka memberikan sebuah method alternatif dalam menunjukkan kemampuan dimana sebelumnya diserahkan kepada scriptlet dan expression dalam sebuah bentuk yang lebih mudah dan lebih mudah diterima untuk personil dengan latar belakang pemrograman yang masih pemula.

6.2 Expression dalam JSP

Satu dari dua elemen JSP yang akan Kita pelajari dalam bab ini adalah JSP Expression Language(EL).Expression Language ini diperkenalkan dengan spesifikasi JSP 2.0 dan disediakan sebuah syntax sederhana dan jelas untuk menulis expression yang melakukan logika sederhana atau mengakses lingkup nilai.

Kegunaan JSP EL lebih baik dijelaskan dalam sebuah contoh sederhana. Menggunakan method yang dibahas dalam bab sebelumnya, jika Kita ingin mengakses property dari sebuah JavaBean, Kita juga bisa menggunakan :

```
<% String name = user.getLastName(); %>
```

atau

```
<jsp:getProperty name="user" property="lastname"/>
```

Menggunakan method yang pertama, pengembang diarahkan ke konsep pemrograman Java : untuk mengakses properti dari sebuah bean, pengembang diharapkan telah membuat method getter dari sebuah bean. Pengembang juga perlu memahami property dari tipe Java. Method yang kedua lebih cenderung ke pemrograman yang netral : akses ke property sebuah bean dikerjakan oleh tag-tag yang secara konsep hampir sama dengan tag-tag HTML. Bagaimanapun juga, method mengakses property bean ini panjang dan susah. Dan juga, method ini hanya dapat digunakan untuk output langsung kepada user.

Menggunakan EL, property bean dapat diakses sebagai :

```
${user.lastName}
```

Konsep diatas adalah pemrograman netral : pengembang tidak harus mengetahui jenis Java atau syntax. Konsep tersebut juga pendek dan langsung ke titik permasalahan : konsep hanya berisi variabel yang terjangkau dan property untuk diakses, dengan penambahan sedikit character. Dan juga, dapat digunakan bersama untuk menampilkan output langsung ke user, atau untuk membongkar nilai untuk tag yang sesuai untuk diproses.

6.2.1 Syntax EL Literals

EL didesain untuk bersifat sederhana dalam syntaxnya. Pada dasarnya, konsep EL bisa merupakan literal atau expression yang ditulis diantara ${ dan }.

EL mendukung literal seperti berikut :

- Boolean
- Long
- Float
- String
- Null

Literal EL diperlakukan secara sama sebagaimana di dalam Java. Sebagai contoh, nilai boolean dapat bernilai true atau false, nilai String membutuhkan pembatas di dalam tanda double petik ("") atau tanda petik (''), null menggambarkan nilai yang kosong.

Literal dengan sendirinya mempunyai sedikit nilai fungsional didalam sebuah bahasa expression. EL juga menggambarkan standard operator yang dapat digunakan pada literal. Sebuah set terapat operator aritmatika dasar (+,-,*,/,%),operator logikal (&&,||,!) dan operator pembanding(==,!=.<.<=,>,>=).

EL juga menggambarkan kata-kata kunci yang menyerupai fungsi dari beberapa operator-operator : and (&&), eq (==), gt(>), ge(>=), or (||), ne (!=), le (<=), lt (<), div (/), and mod (%). Dan juga menggambarkan kata-kata kunci lainnya :

- True – berhubungan dengan nilai literal boolean
- false – berhubungan dengan nilai literal boolean
- instanceof - serupa dengan kata kunci Java. Menentukan apakah sebuah objek adalah anak dari objek yang diberikan jenisnya.
- Null – serupa dengan kata kunci Java. Menentukan sebuah nilai null
- empty – dapat digunakan dalam berbagai cara. Ketika diterapkan pada literal array atau instans sebuah objek Collection, menentukan apakah berisi sebuah nilai atau tidak.

Berikut adalah contoh expression EL, diberikan apa yang telah Kami kerjakan selama ini :

```
${'JEDI'}            - menghasilkan String 'JEDI'
${5 + 37}            - menghasilkan  42
${ (10 % 5) == 2}    - menghasilkan  true
${empty ''}          - menghasilkan  true
```

6.2.2 Mengakses variabel Scope dan Properties

Selama mampu untuk mengevaluasi expression literal sederhana sangatlah berguna, EL menunjukkan banyak kegunaannya ketika mengakses dan mengolah variabel da properties dalam scope yang berbeda.

Mengakses variabel begitu sederhana dengan EL : mereka hanyalah mengacu pada nama, properties Bean, method, dan array. Semua dapat diakses dari penamaan variabel menggunakan notasi ".". Hal ini digambarkan oleh contoh pertama Kami :

```
${user.lastName}
```

Hal ini mengakses sebuah JavaBean dimana dapat disesuaikan dengan nama user dan menerima nilai property lastName. Dengan catatan bahwa scope dari bean tidak berarti. EL melakukan pencarian untuk Kita, mengecek di dalam halaman, permintaan, session dan scope aplikasi (dengan perintah) untuk sebuah bean dengan nama yang spesifik. Jika EL disediakan nama bean yang tidak ada di dalam scope manapun, maka nilai null akan dikembalikan.

6.2.3 Object Implisit EL

Sementara itu pencarian otomatis scope membuat hal tersebut lebih mudah untuk pengembang dalam menulis kode, menetapkan scope variabel dengan tegas mempermudah untuk memelihara oleh pengembang yang akan datang. EL menyediakan untuk kita dengan beberapa object implicit yang mewakili sebuah Map object didalam scope yang berbeda

- pageScope
- requestScope
- sessionScope

- applicationScope

Sebagai contoh, jika object user kita sebelumnya diletakkan didalam scope session :

```
${sessionScope.user.lastName}
```

Terkecuali di atas, EL juga menggambarkan object implicit berikut:

- Param – mewakili sebagai sebuah permintaan nama dan nilai parameter Map.Sebagai contoh meng-invoke ${param.loginName} sama seperti untuk memanggil request.getParameter(“loginName”). Nilai yang disimpan disini adalah nilai String single.
- ParamValues – object implicit ini adalah sebuah Map dimana nama parameter Map yang bertipe array String perwakilan nilai untuk sebuah nama. Meng-invoke ${paramValues.choices} adalah sama dengan memanggil request.getParameterValues(“choices”).
- Header – sebuah Map mewakili header yang terdapat dari sebuah permintaan yang diberikan. Contentnya serupa dengan yang diterima pada saat memanggil method getHeader dari sebuah object ServletRequest.
- HeaderValues – sebuah Map mewakili header yang terdapat dari sebuah permintaan. Contentnya serupa dengan yang diterima pada saat memanggil method getHeader dari sebuah object ServletRequest.
- Cookies – Map dari single cookies terdapat di dalam sebuah permintaan. Serupa dengan meng-invoke method getCookies dari sebuah object HttpServletRequest.

6.2.4 Notasi []

Selain notasi “.”, EL juga menyediakan notasi “[]” dalam mengakses variabel anggota, method, dan array. Dalam banyak hal, kedua notasi adalah sama : ${user.lastName} sama seperti ${user[lastName]}.

6.2.5 JSTL

JSP Expression Language yang telah kita uraikan diatas telah menyediakan untuk Kita dengan sebuah method sederhana dan nyaman dalam mengakses scope properties dan menunjukkan expression sederhana. Bagaimanapun juga, dengan sendirinya itu tidak membebaskan kita dari penggunaan scriptlet di dalam halaman JSP Kita. Untuk melakukannya, Kita kembali lagi kepada tag-tag yang sesuai, khususnya dengan Java Standard Tag Library.

6.2.6 Tag-Tag custom

Salah satu poin terkuat dari spesifikasi JSP adalah karena mempertimbangkan seberapa besar proses penyesuaian dan stabilitas melalui penggunaan tag-tag custom. Telah ada tag-tag khusus yang digambarkan dalam spesifikasi yang melaksanakan tugas-tugas khusus : standard JSP action <jsp:useBean>, <jsp:getProperty>, dan <jsp:setProperty> adalah sebuah contoh yang sempurna. Tag menyediakan sebuah cara untuk mengungkapkan kemampuan “ditulis ulang” di dalam halaman JSP menggunakan sebuah tampilan yang sangat sederhana ketika menyembunyikan implementasi yang internal. Pengembang dapat membuat tag-tag Mereka sendiri selama hal ini dirasa sangat menguntungkan diri Mereka.

Dengan jenis kemampuan “ditulis ulang” ini (tag-tag custom dapat digunakan di banyak aplikasi web dengan ketentuan bahwa implementasi JAR dan deskriptor telah dipaketkan dengan aplikasi web tersebut), hal tersebut akan menjadi biasa bahwa satu set tag custom akan dikembangkan oleh programmer dan digunakan untuk berbagai janis arsitektur framework atau container JSP yang sesuai. Ada sejumlah library tag yang tersedia, dan tak terelakkan bagi mereka untuk memiliki beberapa macam overlap di dalam fungsi yang mereka sediakan.

Java telah mengenal akan hal ini dan dalam kerja sama dengan komunitas pemrograman telah menyediakan sebuah tag library standard yang memilih bidang-bidang overlap yang disebut Java Standard Tag Library atau JSTL

6.2.7 Memasukkan JSTL dalam aplikasi Kita

Untuk memasukkan fungsi JSTL dalam aplikasi Kita, unpack file zip yang didapat dari download, dan copy-kan file standard.jar dan jstl.jar yang terdapat pada direktori /lib ke dalam direktori /lib dari aplikasi Kita.

Ada beberapa set tag library yang dipaketkan dengan JSTL. Berikut, Kita akan membahas dua : Core tag library dan SQL tag library.

6.2.8 Core

set tag Core menyediakan fungsi yang sangat berguna untuk setiap proyek web. Set Core dapat dibagi menjadi sub kategori sebagai berikut :

- General purpose tags
- Iteration
- Conditionals
- URL manipulation

CATATAN : untuk setiap halaman JSP yang akan menggunakan fungsi yang terdapat dalam Core tag library, harus menambahkan directive pada halaman tersebut :

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

Ini menunjukkan bahwa tag-tag didalam core library menggunakan prefix c

6.2.9 General Purpose Tags

General purpose tag di dalam set Core mengerjakan pekerjaan yang sederhana, meskipun salah satu dari mereka tidak relevan sejak dikeluarkannya spesifikasi JSP 2.0. Tag-tag yang terdapat dalam set general purpose adalah : out, set, remove, dan catch.

<c:out>
Tag <c:out> menerima sebuah EL expression, mengevaluasi hasilnya dan kemudian menampilkan hasil secara langsung kepada obyek Writer yang sesuai dengan halaman output. Tag ini berfungsi seperti standard action <jsp:getProperty>, dengan perkecualian bahwa sebuah JavaBean tidak dibutuhkan untuk mengakses fungsi. Bagaimanapun juga, dengan dikeluarkan spesifikasi JSP 2.0, tag ini menjadi agak basi : EL expression dapat dievaluasi langsung menjadi stream output dalam setiap bagian halaman JSP tanpa menggunakan tag-tag dengan JSP 2.0

<c:set>
Tag ini bekerja seperti fungsi action <jsp:setProperty> dimana dapat mengatur nilai di dalam target JavaBean. Dan juga dapat mengatur variabel di dalam scope tertentu sehingga dapat digunakan setelahnya oleh JSP atau disuatu tempat di dalam aplikasi.

Action ini memiliki atribut :

- Value – nilai yang akan diatur kedalam target bean.
- Var – nama dari sebuah variabel yang akan diatur didalam scope tertentu
- scope – menjelaskan variabel scope yang dijelaskan oleh atribut var. Nilainya bisa jadi seperti berikut ini : page, request, session, dan aplikasi.
- Target – nama dari JavaBEan dimana property akan diatur
- property – nam dari property dalam JavaBean yang akan menerima nilai.

Seperti yang dijelaskan sebelumnya, ada dua penggunaan utama untuk tag ini. Untuk mengatur nilai dalam sebuah target JavaBean, tag hanya menggunakan atribut value, target, dan property

```
<c:set target="user" property="name" value="JEDI"/>
```

Untuk mengatur sebuah variabel kedalam sebuah scope tertentu untuk penggunaan yang akan datang, tag <c:set> hanya menggunakan atribut value, var, dan scope, meskipun demikian atribut scope adalah tambahan saja. Ketika atribut scope dihilangkan, otomatis defaultnya menggunakan scope halaman.

```
<c:set var="myString" value="This is a test String"/>
```

Dalam tujuan tentang pembatasan JSP untuk pada presentasi, meskipun demikian kita seharusnya membatasi penggunaan tag ini. Mengatur properties bean atau mengatur variabel ke dalam scope yang berbeda membuktikan bahwa fungsi dapat di refactor disuatu tempat dan tidak baik dengan domain lapisan presentasi.

<c:remove>
tag ini menyediakan cara untuk menghapus variabel dari scope tertentu. Tag ini memiliki dua atribut :

- Scope – scope variabel yang akan dihapus
- var – nama variabel yang akan dihapus dari scope tertentu

Seperti tag <c:set>, menggunakan tag ini seharusnya dibatasi. Membebaskan variabel dari scope yang beragam adalah bukan wilayah objek didalam lapisan presentasi; jenis fungsi ini dapat dijabarkan disuatu tempat di dalam aplikasi.

<c:catch>
Tag <c:catch> menyediakan kemampuan penanganan error di dalam sebuah area dalam JSP. Tag ini mudah digunakan : letakkan isi JSP yang memungkinkan melempar error dalam tubuh tag <c:catch>.

Tag ini hanya memiliki satu atribut :

- Var – menjelaskan nama yang akan digunakan untuk menggambarkan thrown exception.

6.2.10 Iterasi

Tag iterasi JSTL menyediakan alternatif untuk menempelkan do-while, while dan for didalam halaman JSP Kita sebagai scriptlet. JSTL menyediakan dua tag : <forEach> dan <forTokens>.

<c:forEach>
Ternyata semakin banyak digunakan iterasi tag saat ini. Tag ini mengijinkan iterasi melaui array primitif, instans dari java.util.collection, java.util.iterator, java.util.map, dan java.util.enumeration. Bekerja dengan melalui setiap elemen dalam target dan menggambarkan nilai yang ada dari iterasi ke kode JSP yang berisi di dalam body tag.

Tag ini memiliki atribut :

- Var – menjelaskan nama variabel yang akan digunakan untuk menggambarkan nilai yang ada dari iterasi ke body tag. Jenisnya tergantung pada pada collection yang menjadi iterasi diatas. Nilai dari tag ini tidak bisa satu expression dimana akan dievaluasi pada saat runtime.
- Items – menjelaskan collection menjadi iterasi di atas. Ini dapat ditetapkan sebagai sebuah EL expression.

- VarStatus – (tambahan). Menjelaskan nama variabel yang akan diakses oleh body loop untuk mendapatkan informasi pada status loop terbaru. Nilai ini tidak dapat dievaluasi pada saat runtime.
- Begin – (tambahan). Sebuah nilai integer yang menjelaskan indeks yang akan digunakan sebagai titik awal iterasi. Jika nilai ini tidak disediakan, indeks iterasi akan dimulai pada nilai 0, sebagai permulaan collection. Bisa merupakan suatu expression runtime.
- End – (tambahan). Sebuah nilai integer yang menjelaskan indeks untuk menentukan titik akhir dari iterasi. Jika tidak ditetapkan nilainya, iterasi akan berlanjut hingga elemen terakhir tercapai.
- Step – Sebuah nilai integer yang menjelaskan ukuran langkah untuk digunakan ketika iterasi berjalan. Sebagai contoh, mengatur nilai ini dengan angka dua berarti tidak semua elemen akan diakses. Hal ini berarti pada elemen 3 akan diakses setelah setiap 2 elemen terlebih dahulu.

Suatu penggunaan yang umum dari tag ini adalah untuk mengulang hasil di atas ditunjukkan di suatu tempat dalam sebuah aplikasi (biasanya sebuah servlet). Mari kita ambil sebuah contoh dari skenario berikut : Kita memiliki sebuah modul aplikasi yang menerima dari detail database user yang termasuk child dari kategori pencarian. Umumnya, Kita ingin menampilkan logika akses database dari sebuah servlet dan menyampaikan data kepada sebuah JSP untuk presentasi.

Dibawah ini adalah kode snippet dari sebuah servlet yang akan menangani akses.

```
   ...
   // me-load parameter ke dalam Map
   Map parameters = loadUserParameters();
   UserDataService service = new UserDataService();

   // Menampilkan pencarian database dan menyimpan hasilnya di dalam sebuah Collection
   Collection results = service.searchUsers(parameters);

   // Menyimpan hasil untuk akses berikutnya
   request.setAttribute("searchResults", results);

   // Melanjutkan permintaan ke JSP untuk ditampilkan
   request.getRequestDispatcher("/results.jsp").forward(request, response);
...
```

Dan ini adalah sebuah halaman JSP yang sederhana yang bertanggung jawab untuk menampilkan hasil. Mari Kita asumsikan bahwa isi collection adalah instans dari object User dimana memiliki nama dan alamat yang diatur sebagai properties String yang dapat diakses via publik method get.

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

<H1>The following users met your search criteria : </H1> <br/>

<c:forEach var="user" items="${requestScope.searchResults}">
   <li> ${user.name} - ${user.address}
</c:forEach>
```

Dalam halaman JSP, Kita menggunakan tag forEach untuk mengulangi hasil pencarian diatas. Kita dapat melakukan ini dengan menunjuk tag forEach kepada instans Collection yang disimpan dalam scope permintaan menggunakan EL. Kemudian Kita menunjukkan setiap elemen dalam Collection menggunakan variabel user yang telah Kita definisikan di dalam atribut var, dan menggunakan EL untuk menampilkan nilai.

Bandingkan kepada apa yang akan terlihat seperti tanpa JSTL :

```
<H1>The following users met your search criteria : </H1> <br/>

<%
     Collection results = (Collection) request.getAttribute("searchResults");
     Iterator iter = results.iterator();

     while (iter.hasNext()) {
         User user = (User) iter.next();
%>
    <li> <%= user.getName() %> - <%= user.getAddress() %>
<%
      }
%>
```

Jelas sekali bahwa versi JSTL jauh lebih dapat dimengerti, khususnya kepada desainer situs tanpa pengetahuan Java.

<c:forTokens>
Iterasi tag yang disediakan oleh JSTL adalah tag <forTokens>. Tag ini berbentuk String dan merubah mereka menjadi token berdasar pada delimiter yang diberikan. Semua atribut tag forEach dibagi bersama oleh tag ini.

Terkecuali mereka, atribut yang berikut ditambahkan :

- Delims – menjelaskan delimiter untuk digunakan ketika merubah target dari String menjadi tokens.

Atribut item sekarang telah memiliki sebuah tujuan baru untuk tag ini. Atribut tersebut menggambarkan String yang akan di tokenized.

Sebuah contoh diberikan dibawah ini :

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

<c:forTokens items="item1,item2,item3,item4" delims="," var="item">
   <li> ${item}
</c:forTokens>
```

6.2.11 Kondisi

Satu set tag dibawah kategori ini meniru fungsi yang disediakan oleh standard pernyataan if, else-if,... yang dapat ditemukan dalam standard Java. Pemakaian dari tag pernyataan standard Java diatas tertuju kepada kode pembersih; dapat berpindah dari scripting server-side ke output normal HTML dan kemudian sesekali membuat dimungkinkan sebuah kondisi dengan scriptlet yang susah untuk dimengerti dan diatur.

Ada dua set utama conditionals : tag <c:if> meniru sebuah pernyataan if Java sederehana; dan tag <c:choose> berhubungan selalu deng tag <c:when> dan <c:otherwise>, dimana kesemuanya meniru pernyataan fungsi switch.

<c:if>
Tag <c:if> mengijinkan eksekusi isi yang terdapat di dalam body-nya jika nilai expression yang dievaluasi dalam atribut test bernilai true. Jika expression dievaluasi menjadi false, maka body tidak akan pernah dieksekusi.

Contoh :

```
<c:if test="${empty sessionScope.user}">
    You are not currently logged in! Please correct before continuing any further.
</c:if>
```

<c:choose>, <c:when>, <c:otherwise>

Ketiga tag bekerja bersama untuk menyediakan fungsi switch dalam aplikasi Kita. Pada dasarnya, satu atau lebih ketika tag diletakkan dalam body dari tag choose. Tag choose mengevaluasi setiap atribut test dari tag when dan mengeksekusi body dari tag when dimana mengevaluasi menjadi benar. Jika tidak ada tag when yang berhasil cocok, choose memanggil tag otherwise, jika dimasukkan.

Contoh :

```
<c:choose>
    <c:when test="${userGrade >95}">
          Outstanding!
     </c:when>
     <c:when test="${userGrade > 80}">
           Good!
     </c:when>
     <c:when test="${userGrade < 60}">
          Failed!
     </c:when>
     <c:otherwise>
          Congratulations ...
     </c:otherwise>
</c:choose>
```