BAB 3

Pembahasan Servlet Lanjutan

Pada pembahasan sebelumnya, kita telah mengamati tentang bagaimana servlet dapat digunakan oleh pengembang Java untuk memenuhi permintaan client dan menghasilkan respon secara dinamis. Kita juga telah melihat bagaimana untuk mendistribusikan setiap servlet; dengan cara membentuknya menjadi package-package dalam bentuk file WAR dan memindahkannya ke servlet container.

Pada pembahasan ini, meliputi beberapa topik tentang servlet lanjutan :

- Pengalihan respon
- Peran objects
- Sessions dan session tracking
- Filters

3.1. Pengalihan Respon

Ada beberapa kasus ketika kita ingin servlet kita hanya menjalankan beberapa proses inisialisasi dan meninggalkan isi turunan dari beberapa entity asli yang lain.

Mari membuat skenario dimana kita mengambil beberapa nilai dari user dan menyajikannya dengan beberapa tampilan yang mungkin berdasarkan pada nilai tersebut. Mari mengatakan bahwa kita mempunyai sebuah tempatnya, setelah memproses login user, akan menampilkannya dengan halaman yang berbeda tergantung pada peran user pada sistem tersebut. Menempatkan isi kode turunan untuk semua halaman ke satu servlet mungkin membuat servlet kita terlalu tidak teratur. Pada kasus ini, akan menjadi lebih baik untuk servlet untuk memindahkan output turunan.

Ada dua method yang dapat dipakai oleh pengembang untuk melakukan pemindahan ini. Satu adalah menyelesaikan penggunaan dari sebuah object RequestDispatcher, yang lain adalah dengan menggunakan method sendRedirect() yang dapat ditemukan dalam object HttpServletResponse.

3.1.1 RequestDispatcher

Kita mencapai sebuah instance dari object RequestDispatcher dengan menjalankan method selanjutnya yang mana dapat ditemukan dalam object HttpServletRequest :

public RequestDispatcher getRequestDispatcher(String path)

Parameter string yang diambil dalam method ini adalah lokasi dari HTML, JSP, atau servlet yang ingin kita hubungkan dengan request. Sekali kita mempunyai sebuah pengontrol pada sebuah instance dari object RequestDispatcher , Kita memiliki pilihan untuk menjalankan satu dari dua method :

- public void include(ServletRequest req, ServletResponse res)
- public void forward(ServletRequest req, ServletResponse res)

Kedua method diatas mengambil isi yang dihasilkan oleh lokasi khusus dan membuatnya sebuah bagian dari respon servlet ke user. Perbedaan utama diantara mereka adalah : perintah forward membuat target entity dengan satu-satunya tanggung jawab untuk mengeluarkan respon, sementara perintah include hanya menggabungkan isi dari target. Menggunakan perintah include, kita dapat menambahkan isi lain ke respon, mungkin memasukkan even target yang lain.

Dibawah ini adalah beberapa contoh kode memanfaatkan kedua perintah yaitu include dan forward.

```
public DispatchServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

   response.setContentType("text/html");

   PrintWriter out = response.getWriter();
   out.println("Error occurred during login request processing");
   RequestDispatcher rd = request.getRequestDispatcher("/login.html");
   rd.include(request, response);
   }
}
```

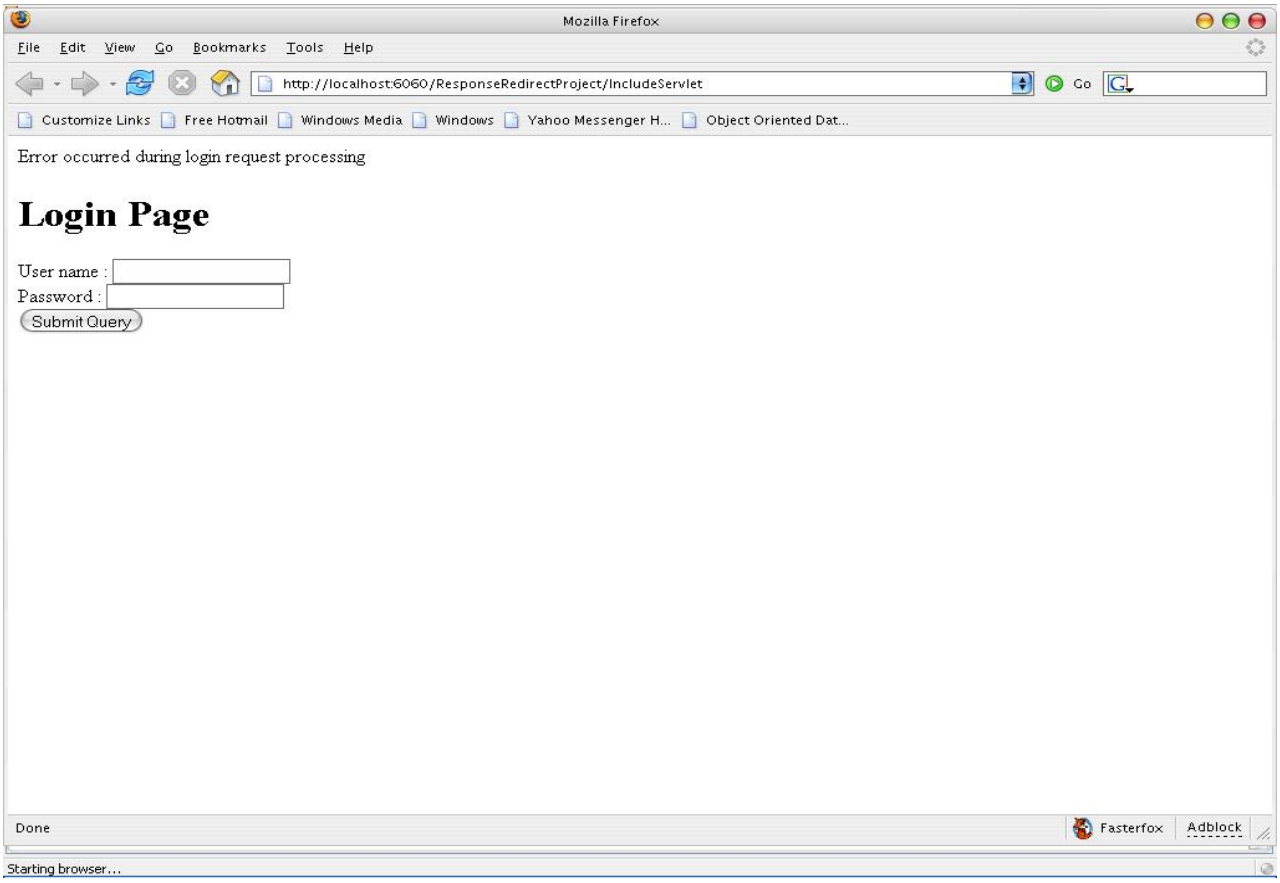
```
public DispatchServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws
ServletException, IOException {

   response.setContentType("text/html");

   PrintWriter out = response.getWriter();
   out.println("Error occurred during login request processing");
   RequestDispatcher rd = request.getRequestDispatcher("/login.html");
   rd.forward(request, response);
   }
}
```

```
<H1>Login Page</H1>
<form action="DispatchServlet">
User name : <input type="text" name="loginName"><br/>
Password : <input type="password" name="password"><br/>
<input type="submit"/>
</form>
```

Kesamaan secara virtual, dengan hanya membedakan method yang dijalankan pada permintaan object dispatcher. Ini akan menyajikan penanda perbedaan diantara keduanya. Dibawah ini adalah keluaran program tersebut.



Kode yang diberikan sama secara virtual, tetapi kita memiliki dua output yang berbeda. Menggunakan method include, pesan String yang kita keluarkan ke user lebih dahulu untuk memanggil method yang ditampilkan. Ini bukan kasus untuk output dari method forward : pesan yang kita tambahkan pada respon lebih dahulu untuk memanggil method bukan merupakan bagian dari output.

Dengan method forward, semua isi dalam response buffer dibersihkan lebih dahulu untuk memanggil method, Setelah respon tertentu di commit; Tidak ada isi yang dapat ditambahkan lebih lanjut. Dengan method include, semua isi ditempatkan dalam response buffer yang tersimpan sebelum dan sesudah memanggil method.

Menggunakan method include, sangat mungkin untuk servlet kita untuk membawakan sebagai keseluruhan output dari beberapa sumber yang berbeda. Yang juga memungkinkan untuk menambahkan pesan ke isi yang sifatnya static dan sebaliknya. Perhatikan contoh berikut ini.

```
public LoginServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
      RequestDispatcher rd = null;

      String loginName = request.getParameter("loginName");
      String password = request.getParameter("password");

      User user = null;

      UserService service = new UserService();

      user = service.authenticateUser(loginName, password);

      if (user == null) {
          response.setContentType("text/html");
          PrintWriter out = response.getWriter();
          out.println("User does not exist with given login and/or password");

          rd = request.getRequestDispatcher("/login.html");
          rd.include(request, response);
          out.close();
      } else {
          HttpSession session = request.getSession();
          session.setAttribute(ApplicationConstants.USER_OBJECT, user);

          rd = request.getRequestDispatcher("/header.html");
          rd.include(request, response);

         rd = request.getRequestDispatcher("/mainContent.html");
         rd.include(request, response);

         rd = request.getRequestDispatcher("/footer.html");
         rd.include(request, response);
      }
}
```

Disini, kita menggunakan method include untuk menampilkan beberapa halaman HTML yang berbeda(header.html, mainContent.html, footer.html) Untuk user sebagai satu yang mewakili keseluruhan user. Juga, sebuah pesan error akan ditampilkan jika user gagal dalam melakukan validasi disisi server.

sendRedirect

Di sisi lain dari pengalihan output ke suatu entity diluar servlet adalah method sendRedirect. Ini akan memiliki tanda method sebagai berikut :

```
public void sendRedirect(String relativePath)
```

Dan dapat ditemukan dalam object HttpServletResponse. Parameter yang diambil adalah sebuah String mewakili path untuk target yang kita inginkan untuk pengalihan user. Memanggil method ini secara efektif memerintahkan browser untuk mengirim dalam HTTP request yang lain ke tujuan tertentu.

Ini membuat target satu-satunya entity yang bertanggungjawab untuk meng generate isi, sama dengan sebuah cara untuk hasil dengan menggunakan method forward dalam object RequestDispatcher yang telah didiskusikan sebelumnya. Bagaimanapun, permintaan dikirim kembali untuk menampilkan beberapa perbedaan secara praktek dibandingkan dengan method forward :

– alamat URL pada browser bar mencerminkan target yang ditentukan. Ini membuat penggunaan method sendRedirect tidak digunakan jika kita tidak mengingatkan tentang kesadaran user terhadap fungsi pengalihan.

– Sejak ini menjadi sebuah request baru secara efektif, data tersimpan dalam object request yang sebelumnya tidak tersimpan. Disediakan parameter-parameter untuk user, jika ada banyak, harus di submit kembali jika kita ingin target kita sadar terhadap prosedur tersebut. Biasanya data tersimpan dalam object request harus juga dipelihara dengan beberapa cara, jika tidak data-data tersebut akan hilang.

Disarankan ketika method include digunakan untuk memungkinkan banyak output source, method forward digunakan jika pengalihan ke sebuah komponen yang menghasilkan isi yang dinamis (servlets atau JSPs yang akan kita diskusikan kemudian), dan method sendRedirect digunakan ketika mengalihkan ke isi yang static.

3.2 PEMBATASAN OBJECT-OBJECT

Spesifikasi servlet memungkinkan kita untuk melakukan empat bidang dimana untuk menempatkan data, agar komponen-komponen kita dapat digunakan bersama. Untuk meningkatkan spesifikasinya :

- Halaman-halaman menyangkut suatu bidang didefinisikan menjadi bidang meliputi sebuah halaman JSP tunggal. Variabel-variabel dideklarasikan dalam halaman yang sifatnya dapat dilihat hanya dalam halaman tersebut dan dan tidak akan nampak sekalipun jika servis halaman berakhir. Object yang tergabung dengan bidang ini adalah object PageContext.

- Pembatasan Request - Request didefinisikan menjadi bidang yang meliputi sebuah client dengan request tunggal. Data yang tersimpan dalam bidang ini sifatnya dapat dilihat oleh semua komponen web mengatur request dari client. Setelah request client telah selesai – Kemudian client menerima sebuah respon HTTP dan menutup koneksi ke server – semua data tersimpan dalam bidang ini tidak lagi terlihat.

Object yang tergabung dengan bidang ini adalah object HttpServletRequest. Instance-instance dari object ini siap dan dapat digunakan untuk servlet, sama seperti yang mereka berikan sebagai parameter untuk method service yang dipanggil oleh container diatas request client.

- Session – bidang ini meliputi sebuah “session” tunggal dengan client. Session ini dimulai ketika client pertama dimulai dengan aplikasi web dan diakhiri sekali ketika user log out dari sistem atau server didefinisikan dengan nilai timeout telah tercapai. Data dalam bidang ini dapat dilihat oleh semua komponen web client membuat dan memakainya selama interval tersebut. Data dari satu session user, bagaimanapun, tidak terlihat dari dalam session user yang lain.

  Object yang tergabung dengan bidang ini adalah object HttpSession. Sebuah instance dari ini bisa didapatkan kembali dengan menggunakan method getSession() dalam object HttpServletRequest.

- Aplikasi – bidang ini meliputi semua aplikasi. Data yang tersimpan dalambidang ini terlihat oleh semua komponen tanpa memperhatikan request user atau session client yang mengatur dan mengakhiri sampai aplikasi dihentikan.

  Object yang tergabung dengan bidang ini adalah object ServletContext. Seperti yang telah dibahas sebelumnya, ini bisa didapatkan kembali dengan memanggil method getServletContext() dari object yang valid ServletConfig.

3.2.1 Menyimpan dan mendapatkan kembali data dari suatu bidang

Semua dari object bidang seperti yang disebutkan diatas terdiri dari perintah method untuk bisa mendapatkan kembali dan menyimpan data didalamnya. Untuk menyimpan data gunakan method

public void setAttribute(String key, Object value);

parameter String yang diambil method ini akan menyimpan object dibawah nilai tersebut. Untuk mendapatkan data kemudian, lewati key yang sama seperti parameter.

public Object getAttribute(String key);

Jika tidak ada object yang bisa mendapatkan kembali dari key yang diberikan, niali null dikembalikan oleh method. Juga, sejak return type adalah Object, itu akan membuat pengembang untuk memilih Object ke class yang sesuai dan menampilan pengujian type.

Untuk menghapus sebuah attribut dari object bidang secara sederhana panggil method `removeAttribute()` dan lalui kunci attribut seperti parameternya.

3.2.2 Contoh skenario

Mari kita mengikuti skenario berikut ini : kita ingin aplikasi kita dapat menampilkan rincian tentang seseorang diberikan personalIDnya. PersonalID ini akan disediakan oleh user. Untuk memajukan kemampuan maintenance, kita ingin memisahkan komponen yang akan mendapatkan kembali rincian individu dan komponen yang akan menampilkan informasi kepada user. Dua komponen ini kemudian untuk berkomunikasi menggunakan penyimpanan data dan fasilitas pengembalian yang dapat dilakukan ketika terjadi request.

```
public PersonalDataRetrievalServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {

      String personalID = request.getParameter("personalID");

     DataService service = new DataService();

     Person person = service.retrievePersonalDetails(personalID);

     request.setAttribute(ApplicationConstants.PERSON, person);

    RequestDispatcher dispatcher = request.getRequestDispatcher("/DisplayServlet");
    dispatcher.forward(request, response);
   }
}
```

```
public DisplayServlet extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response) throws
      ServletException, IOException {

   Person person = (Person) request.getAttribute(ApplicationConstants.PERSON);

   StringBuffer buffer = new StringBuffer();
   buffer.append("<HTML><TITLE>Personal Info</TITLE>");
   buffer.append("<BODY><H1>Details : </H1><br/>");
   buffer.append("Name : ");
   buffer.append(person.getName());
   buffer.append("<br> Address : ");
   buffer.append(person.getAddress());
   buffer.append("</BODY>");

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println(buffer.toString());
   out.close();
  }
}
```

Kode untuk object DataService dan object individu tidak terdaftar disini, implementasinya tidak penting untuk contoh kita. Apa yang ditampilkan pada contoh diatas adalah bagaimana untuk menyimpan object data dari satu servlet dan mengambil data yang sama untuk servlet yang lain yang telah mengakses ke bidang yag sama.

Perhatikan contoh berikut : pertama, servlet yang kedua bisa mendapatkan data kembali dari request karena ini masih bagian mengatur servlet method getAttribute akan memberikan nilai kembalian null, sejak sendRedirect akan membuat browser untuk request yang lain. Ini mengakhiri secara efektif batas waktu dari object request dalam menyimpan data.

Kedua, disarankan bahwa key yang digunakan untuk menyimpan dan mendapatkan data dibuat dapat dipakai untuk aplikasi sebagai konstanta, Seperti contoh diatas. Ini meyakinkan bahwa pasti key yang sama digunakan untuk penyimpanan dan pengambilan kembali data, mengurangi kemungkinan jumlah kemungkinan error yang bisa muncul ketika proses pengembangan dilakukan.

3.3 SESSION TRACKING DAN PENGATURAN

Kita harus memanggil kembali HTTP yang didesain sebagai bagian yang terhubung, protocol yang tidak memiliki state. Ketika browser mengirim request ke server, koneksi dibuka, mengirimnya melalui HTTP request, menggunakan respon HTTP, dan kemudian menutup koneksi. Sejak setiap request browser terkirim secara efektif dalam koneksi berbeda tiap waktunya, dan server tidak berhubungan dan memperbaiki secara langsung, secara tidak sadar client mengaksesnya, masalahnya ditunjukkan melalui aplikasi web : bagaimana memperbaiki langkah untuk bagian transaksi web user?

Solusi untuk masalah ini adalah server dalam memperbaiki konsep “user session”. Sementara dalam sebuah session, server akan dapat mengenal client diantara banyak request. Secara tidak sadar,kini server dapat memperbaiki state untuk client.

Untuk setiap session agar tetap ada, browser client harus dapat mengirim data ke sisi server dari standard HTTP request-nya. Data ini memungkinkan server untuk mengenali user yang mana yang melakukan reques, dan memperbaiki sesuai dengan statenya. Ada tiga tipe solusi untuk masalah ini : cookies, penulisan kembali alamat URL, dan form fields yang tersembunyi.

3.3.1 Cookies

Cookies adalah struktur data kecil yang digunakan oleh web server untuk mengirim data ke browser client. Data ini disimpan oleh browser, dan dalam beberapa kondisi browser mengembalikan data tersebut kembali ke web server.

Menggunakan cookies, servlets dapat menyimpan "session ids" yang dapat digunakan untuk mengenali user sebagai bagian yang ikut serta dalam bagian session. Setelah ID diproses, disimpan dalam object cookie dan mengirim kembali ke browser client untuk penyimpanan. Object cookie ini kemudian dapat ditangkap kembali dalam object request setiap waktu untuk mengetahui jika user ada dalam session.

Dibawah ini adalah contoh dari bagaimana untuk mengunakan cookies untuk session tracking dalam servlets :

```
...
String sessionID = generateSessionID();

HashMap map = new HashMap();

HashMap containerMap = retrieveSessionMaps();

containerMap.put(sessionID, map);

Cookie sessionCookie = new Cookie("JSESSIONID", sessionID);

response.addCookie(sessionCookie);

..
```

Untuk bisa mendapatkan gambaran yang benar mengenai data session, servlet kemudian mendapatkan cookie yang berisi session ID, dan menggunakannya sebagai key, mencapai HashMap yang sesuai.

Sementara cookies merupakan solusi yang baik untuk session tracking, penggunaannya memerlukan pengembang untuk mengatur banyak detail :

- Membangkitkan session id yang spesifik untuk setiap user.
- Mendapatkan cookie yang sesuai dari browser yang berisi session ID.
- Mengatur waktu yang terpakai sesuai dengan kebutuhan cookie.

Sesuai dengan isi detail diatas, masalah dengan penggunaan cookie dari beberapa user yang browsernya tidak support dengan cookie berkenan dengan keperluan keamanan. Pendekatan alternatif mutlak diperlukan.

3.3.2 Penulisan kembali alamat URL

Pada pendekatan ini, browser client menambahkan sebuah session ID yang spesifik pada akhir setiap request yang dibuat untuk server. Session ID ini dapat dibaca, dan informasi user yang sesuai bisa didapatkan.

Ini adalah solusi lain yang cukup baik, dan satu dari pekerjaan tersebut jika user me non aktifkan pengunaan cookies. Bagaimanapun, ini mengenalkan masalah yang dihadapi untuk meyakinkan bahwa setiap URL yang digunakan oleh client pada bagian atas halaman site memiliki session ID yang ditambahkan didalamnya.

3.3.3 Form fields tersembunyi

Pada pendekatan ini, form field tersembunyi dikenalkan dalam bentuk HTML, dengan nilai yang diset untuk bagian session ID. Bagaimanapun, method ini sangat terbatas berhubungan dengan fakta bahwa ini hanya dapat digunakan ketika ada form di halaman yang digunakan client.

3.3.4 Session Tracking dalam Servlets

Spesifikasi servlet menyediakan sebuah API tingkat tinggi untuk menyediakan access untuk session tracking : API HttpSession. Menggunakan API ini, pengembang tidak lagi perlu khawatir tentang banyak detail yang disebutkan seperti yang telah disebut diatas : pengenalan session ID , manipulasi detail cookie, dan informasi mengenai detail diringkas oleh pengembang. Juga, pengembang menyediakan lokasi yang sesuai untuk menyimpan data untuk session user. Penggunaan API HttpSession yang benar juga memungkinkan aplikasi Anda untuk secara otomatis beralih ke metod URL-rewriting jika dideteksi adanya support cookie di browser client di non aktifkan.

3.3.5 Mendapatkan sebuah instance dari object HttpSession

Object HttpSession mewakili data session yang tergabung untuk diberikan request client dapat dicapai dengan memanggil method `getSession()` dalam object `HttpServletRequest` . Container kemudian bertanggung jawab untuk membaca data dari client (salah satu dari cookies atau dari URL-rewriting), dan membuat instance dari object HttpSession.

Dengan melewatkan sebuah nilai boolean ke method getSession()(misal, getSession(true)) kita dapat membatasi ke server jika object HttpSession yang baru harus dibuat secara otomatis dalam kasus user tidak berperan langsung dalam beberapa session.

3.3.6 Menyimpan dan mendapatkan data dalam sebuah session

Dengan HttpSession API, pengembang tidak perlu mengatur object secara eksplisit untuk menyimpan data yang diperlukan untuk diperbaiki dalam user session. Semua yang diperlukan adalah untuk memanggil salah satu dari kedua method berikut :

- public void setAttribute(String key, Object value)
- public Object getAttribute(String key)

3.3.7 Menghapus data yang tersimpan session

Untuk menghapus data yang berada pada batasan session, panggil method `removeAttribute()`, dan lewatkan sebagai parameter key String yang tergabung dengan data.

3.3.8 Terminasi session

Sessions secara otomatis diterminasi setelah didefinisikan awal interval terbesarnya. Interval ini dapat ditemukan dan dimanipulasi dalam application's deployment descriptor.

deployment descriptor untuk contoh FirstServlet dicetak kembali disini untuk memanggil kembali :

```
...
  </servlet-mapping>
  <session-config>
    <session-timeout>
          30
    </session-timeout>
  </session-config>
  <welcome-file-list>
...
```

Nilai 30 dalam elemen session-timeout mengatakan ke server untuk menunggu suatu periode inactive akhir selama 30 menit sebelum melakukan terminasi session. Pada point ini semua object ditempatkan dalam batasan session akan dihapus dari batasan, dan object HttpSession menjadi invalid.

Pengembang dapat melkukan terminasi session melalui program dengan memanggil method `invalidate()`.

3.3.9 Melakukan URL-Rewriting

Secara default, HttpSession API membuat dan mengunakan cookies untuk track sessions. Bagaimanapun, kita harus mengembangkan aplikasi web kita hingga dapat bekerja sama baiknya pada browser yang tidak mendukung cookies dengan menambahkan support untuk URL rewriting.

Kita dapat menambahkan support URL rewriting dengan menggunakan method encodeURL() yang dapat ditemukan dalam object HttpServletResponse. Method ini mengambil dalam sebuah String yang mewakili sebuah path atau alamat URL sebagai parameternya. Kemudian dia dapat mengenali jika cookie support diaktifkan pada browser target. Jika dia diaktifkan, dia akan mengembalikan memberikan String sepertinya.jika cookies di non aktifkan, dia mengaktifkan URL rewriting dengan menambahkan session ID ke URL yang diberikan.

Berikut ini adalah contoh tentang bagaimana untuk menggunakan method encodeURL.

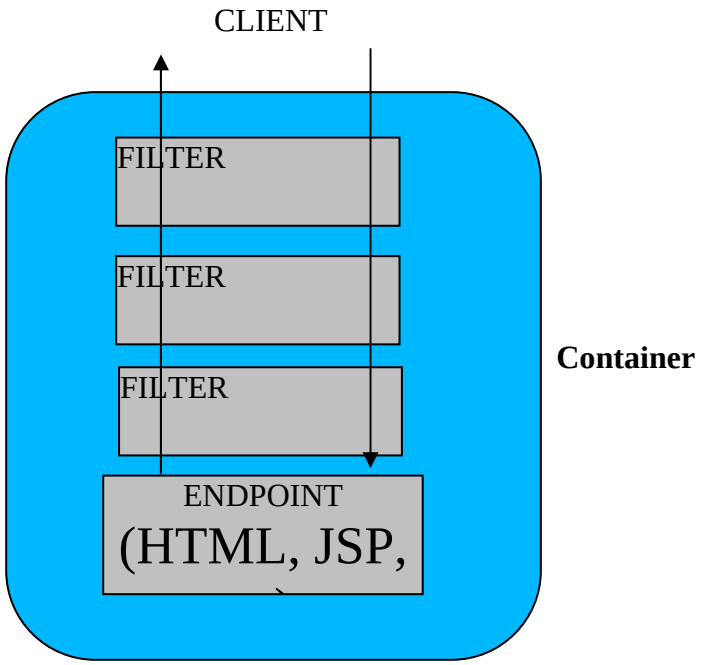
```
...
String encodedURL = response.encodeURL("/welcome.jsp");
out.println("<A HREF='" + encodedURL + "'>Click here to continue</A>");
...
```

Untuk menyediakan fungsi URL rewriting pada aplikasi kita kemudian , kita harus memindahkan semua URL melewatkannya sebagai link ke user dengan suatu pengkodean kembali dengan menjalankan method encodeURL. Path dilewatkan melalui method sendRedirect yang telah kita diskusikan sebelumnya. Juga harus dikodekan kembali, sekarang menggunakan hasil dari method encodeRedirectURL().

3.4 FILTERS

Filters adalah komponen web lanjutan yang dikenal sejak spesifikasi Serlvet 2.3. Pada dasarnya, komponen mereka yang berdiri diantara request client dan bagian sumber – beberapa telah mencoba untuk mendapat target sumber untuk menjalankan filter. Isi Sumber dapat menjadi static atau dynamic source (HTML, JSP, GIF, ...)

Filters bekerja dengan menangkap request client dan tetap ada sebagai bagian dari rantai; berikut ini, pendefinisian dari filter yang requestnya dilewatkan berjalan sebelum benar-benar tiba di sumber target. Ketika request melewati filter, filter menjalankan proses tersebut dan kemudian dapat memutuskan apakah request dapat dilewatkan pada proses filter selanjutnya pada urutan tersebut (jika filter tersebut adalah filter terakhir pada urutan, request dilewatkan pada sumber target), atau untuk melintasi rantai siklus sepenuhnya, menolak acces user ke sumber.



Gambar diatas adalah representasi grafis tentang bagaimana request melewatkan beberapa tahap filter sebelum benar-benar tiba pada titik akhir target.

Filter adalah komponen yang dapat dipakai sejak ia menawarkan kemudahan untuk pengembang yaitu cara mudah untuk menambahkan proses sebelum sumber didalam aplikasi web diakses. Ini adalah macam-macam kegunaan yang mungkin dengan menggunakan servlet dan pengalihan respon. Ini lebih sulit untuk implementasi berlanjut, dan mensyaratkan setiap servlet pada tingkat yang lebih tinggi sadar akan lokasi servlet selanjutnya dalam urutan. Filter tidak memiliki pembatasan; ini pembangun servlet yang mengatur urutan komponen yang dipanggil sebelumnya request pernah mencapai titik akhir, dan bukan pengembangnya.

3.4.1 Membuat sebuah Filter

Untuk membuat sebuah filter, pengembang harus membuat sebuah class yang mengimplementasikan interface javax.servlet.Filter. Interface ini mendefinisikan method-method berikut ini :

- void init(FilterConfig config) melempar ServletException – method ini dipanggil oleh servlet container awalnya ini akan mengirim filter ke memory. Inisialisasi kode harus ditempatkan disini, termasuk kode yang mengumpulkan inisialisasi parameter yang terletak dalam web.xml sepanjang penggunaan object FilterConfig yang diberikan.

- void destroy – Method ini dipanggil oleh container ketika filter diambil dari memory. Ini selalu dilakukan ketika aplikasi dimatikan. Kode yang menghentikan beberapa sumber dibuat dengan filter yang harus dikerjakan disini.

- void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) melempar IOException, ServletException – method ini berisi semua kegunaan dari class filter. Ini dipanggil oleh servlet container ketika server mengenali bahwa filter diharuskan untuk digunakan untuk menangkap respon dari sebagian user.

  Parameter-parameter yang terlewatkan pada method ini adalah instances dari object ServletRequest, ServletResponse, dan FilterChain. Jika Filter ini berpartisipasi dalam sebuah lingkungan web (seperti kasus yang biasa terjadi) pengembang dapat memilih object permintaan dan respon untuk instances dari HttpServletRequest dan HttpServletResponse, masing-masing, jadi mereka bisa mendapatkan informasi HTTP-tertentu.

  Seperti servlet, container akan menciptakan hanya satu instance dari sebuah object Filter dan menggunakan multi-threading untuk memungkinkan pengaturan beberapa request client secara bersama-sama. Ini berarti bahwa method ini HARUS didefinsikan menjadi thread-safe.

3.4.2 Rantai Filter

Rantai filter memungkinkan filter ditampilkan pada urutan yang benar untuk sebagian sumber, dan diwakili oleh suatu object FilterChain. Kemampuan suatu filter untuk menjadi satu silklus berbentuk rantai tersendiri. Sebaliknya Filter memiliki kegunaan yang sama seperti Servlet.

Rantai filter memungkinkan pembersihan secara terpisah diantara tingkat pemrosesan yang berbeda : misalkan, sebagai contoh, fungsi baru yang perlu diimplementasikan sebelum request diproses, ini dapat dibuat dengan sederhana sebagai filter. Pertama-tama filter

dikonfigurasi, Filter hanya ditambahkan ke pelayanan komponen terhadap request sebelum titik akhir dapat dicapai, ia tidak menyerahkan sepenuhnya beberapa proses yang dilakukan sebelumnya kecuali jika secara explisit ditulis untuk melakukan hal yang sama dapat dilakukan oleh pengembang.

Urutan rantai filter dikenali oleh lokasi filter dalam deployment descriptor dan mengikuti pola ascending yang mengharuskan penggunaan elemen<filter-mapping> yang mewakili proses mapping dari setiap filter (selanjutnya akan dibahas lebih dalam mengenai konfigurasi filter).

Seperti yang telah dibahas sebelumnya, object FilterChain mewakili urutan dari filter yang akan dipanggil sebelum pada akhirnya mencapai titik akhir. Satu-satunya akses pengembang melalui urutan ini adalah method doFilter dalam object FilterChain : ini memanggil filter selanjutnya dalam urutan, atau sumber target jika suatu filter terletak pada urutan terakhir. Method ini hanya memerlukan object tertentu yaitu ServletRequest dan ServletResponse sebagai parameter. Lagi, sejak tidak ada yang membutuhkan filter programmer untuk memiliki kesadaran tentang filter yang mana akan menerima request selanjutnya, yang memungkinkan programmer untuk memfokuskan diri hanya pada suatu object tertentu saja.

Dibawah ini adalah sebuah filter yang menyediakan logging dasar untuk aktifitas pada aplikasi :

```
public LoggingFilter implements Filter {
   private FilterConfig config;

   public void init(FilterConfig config) {
     this.config = config;
   }

   public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain)
     throws ServletException, IOException {

        ServletContext context = config.getServletContext();

        String logEntry = request.getServerName() + ":" + request.getServerPort();
        logEntry += "/" + request.getContextPath() + "/" + request.getPathInfo();
        logEntry += "--> accessed by the user on " + new java.util.Date();

        context.log(logEntry)
        chain.doFilter(request, response);
   }
}
```

3.4.3 Konfigurasi Filter

Konfigurasi Filter adalah sangat sama seperti yang diperlukan pada servlets. Ada bagian yang diperlukan untuk mendefinisikan setiap filter untuk digunakan dalam aplikasi, sama seperti bagian untuk mendefinisikan pola url-yang akan didengar oleh filter untuk ditangkap dan diproses lebih lanjut.

Sebuah contoh dari konfigurasi filter diberikan dibawah ini :

```
...
</context-param>
<filter>
   <filter-name>LoggingFilter</filter-name>
   <filter-class>jedi.filters.LoggingFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>LoggingFilter</filter-name>
   <url-pattern>/*</url-pattern>
</filter-mapping>
```

Diberikan file web.xml yang sangat khusus tentang pemesanan dari elemen-elemen tertentu, Paling baik untuk meyakinkan bahwa masukan elemen filter didefinisikan sebelum beberapa servlet didefinisikan, tapi setelah ada beberapa masukan context-param. Juga, semua masukan filter-mapping harus diletakkan setelah beberapa definisi filter.

Secara default, Filters tidak digunakan terhadap komponen-komponen web (servlets, JSP yang lain) inilah target dari pemanggilan include atau forward dari sebuah object RequestDispatcher. Mereka dipakai hanya untuk request yang dibuat secara langsung oleh client. Kebiasaan ini dapat dirubah biarpun dengan menambahkan satu atau lebih elemen pengiriman untuk memetakan filter.

```
...
</context-param>
<filter>
   <filter-name>LoggingFilter</filter-name>
   <filter-class>jedi.filters.LoggingFilter</filter-class>
</filter>

<filter-mapping>
   <filter-name>LoggingFilter</filter-name>
   <url-pattern>/*</url-pattern>
   <dispatch>REQUEST</dispatch>
   <dispatch>INCLUDE</dispatch>
</filter-mapping>
```

Elemen-elemen pengiriman memiliki satu dari nilai berikut : REQUEST, INCLUDE, FORWARD, dan ERROR. Ini menyatakan apakah filter akan digunakan hanya untuk request client, hanya includes, hanya request, atau hanya error, atau beberapa kombinasi dari keempatnya.