Enterprise

'Never a dull moment' in de wereld van Java. Op die regel is ook 2009 zeker geen uitzondering. Neem nu de overname van Sun Microsystems door Oracle. Eén van de ontwikkelingen, die ook als enige tijd zijn schaduw vooruit werpt en eind 2009 tot een hoogtepunt zou moeten komen, is JEE6, de nieuwste release van de Java Enterprise Edition. Java Magazine besteedt dit hele jaar aandacht aan de belangrijkste componenten in JEE6 - met in dit nummer de schijnwerper op de algemene JEE6 specificatie zelf en op Servlet 3.0.

Het jaar van JEE6 (4)

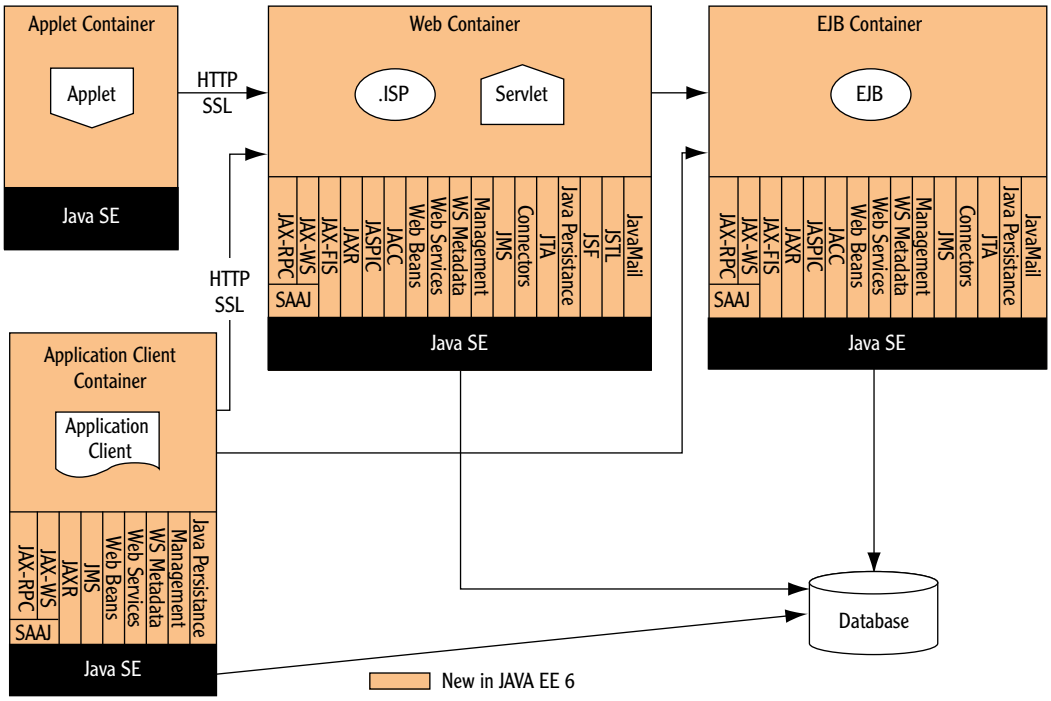
JEE6, Servlet 3.0 en JSP 2.2

Sinds enkele jaren worden alle grote onderdelen van het Java-platform vastgelegd in specificaties. De organizatie die de verzoeken voor deze specificaties, zogenaamde Java Specification Requests of JSRs, in behandeling neemt, heet de Java Community Process, JCP. De Java Enterprise Edition bevat een omvangrijk aantal specificaties, waaronder JSF 2.0. Vaak realiseert men zich niet dat ook voor de Java Enterprise Editions specificaties bestaan. In eerste instantie is JSR-313 opgesteld voor JEE6. Deze JSR is ingetrokken vanwege licentie-specifieke onderdelen in de specificatie. De nieuwe, en nog steeds in behandeling zijnde, specificatie voor JEE6 is JSR-315. De expert group voor JSR-315 is midden 2007 begonnen met het opstellen van de specificatie voor Java Enterprise Edition 6. Nu, ruim twee jaar later, is de specificatie zo goed als af. Tijd voor een overzicht!

Java Enterprise architectuur

Qua architectuur is er in JEE6 weinig veranderd ten opzichte van JEE5. Er is nog steeds een onder-

Wouter van Reeven
is JEE Technology Leader bij AMIS. Hij is te bereiken via email wouter.van.reeven@amis.nl.



Overzicht van de JEE6 architectuur. De afbeelding toont de logische relaties tussen de elementen. Dit is overigens niet een overzicht van hoe een hardwarematige architectuur opgezet zou moeten worden!

Om wildgroei van Profiles te voorkomen moet ieder Profile via een JSR goedgekeurd worden.

verdeling in zogenaamde Containers. Dit zijn JEE runtime omgevingen die bepaalde applicatiecomponenten beschikbaar maken aan clients. Merk op dat iedere container een aantal componenten beschikbaar moet stellen om als JEE Container aangemerkt te mogen worden. Deze componenten worden genoemd binnen iedere Container. De pijlen beschrijven verplichte toegang die een Container aan een client moet geven.

Bijvoorbeeld: een Application Client Container moet Java Message Service (JMS) APIs beschikbaar stellen aan Application Clients (en dus ook de andere genoemde componenten). Verder moet de Application Client Container toegang geven tot de Java EE (verplichte) Database via de (in dit geval) JDBC API.

Profielen

Een van de grote bezwaren tegen JEE was altijd de grootte en zwaarte ervan. JEE bestaat uit meer dan veertig specificaties die allemaal geïmplementeerd moeten zijn om als JEE-applicatieserver door het leven te mogen gaan. In heel veel gevallen is een groot deel, zo niet het grootste deel, van deze technologiën overbodig. Een applicatie die een aantal dynamische webpagina's heeft en gegevens opslaat in een database heeft misschien helemaal geen behoefte aan web services en JMS. Een SOA-applicatie, daarentegen, heeft mogelijk helemaal geen behoefte aan JSP of JSF pagina's. En toch moet een JEE-applicatieserver alle technologiën bevatten om een dynamische website of een SOA-applicatie te kunnen draaien.

Om dit dilemma op te lossen is het concept van profielen bedacht. Een Profile is een configuratie van het JEE-platform met een bepaalde soort applicatie in gedachten. Het komt er daarbij op neer dat er 'uitgeklede' of lichtere versies van JEE gedefinieerd worden die niet de volledige stack aan technologiën bevatten, maar desondanks wél als JEE bestempeld mogen worden.

Het is hierbij belangrijk je te realiseren dat alle JEE-profielen een set van gezamenlijke onderdelen bevatten. Denk hierbij aan naamgeving, resource injectie, packaging regels, beveiliging etcetera. Dit garandeert uniformiteit over alle JEE Profiles en applicaties heen. Verder zorgt dit ervoor dat onwikkelaars die bekend zijn met het ene Profile makkelijk over kunnen stappen naar een ander Profile. Maar afgezien van deze gezamenlijke onderdelen staat het ieder Profile vrij om een combinatie van JEE-technologiën te gebruiken. Uiteraard moeten de overkoepelende JEE-regels, zoals verplichte APIs bij bepaalde technologiën, wel in acht genomen worden. Als dit niet gedaan zou worden zou een Profile slechts een bundeling van APIs zijn en zou de onderlinge samenhang snel zoek raken. Denk bijvoorbeeld aan de Servlet en Java Transaction API (JTA) specificaties. Los van elkaar geven deze twee technologiën geen goede basis om portable applicaties te maken. Juist de combinatie van de twee en hun onderlinge verstandhouding, die vast gelegd is, maken deze technologiën zo waardevol.

Teneinde wildgroei onder Profiles te voorkómen heeft de JEE6 expert group besloten dat ieder Profile via een JSR goedgekeurd moet worden. Het zou toch wat zijn als vendor X ineens besluit dat een bepaalde verzameling aan technologiën een Profile zou zijn en daarmee de concurrenten Y en Z aan de kant kan zetten! Nee, vendor X moet via de JCP zijn Profile officieel erkend krijgen waarbij concurrenten Y en Z mee mogen beslissen over de inhoud van het Profile. De JEE6 expert group heeft hiervoor zelf het initiatief genomen door een specificatie voor een Web Profile in te dienen.

Het Web Profile

Het idee van het Web Profile is om ontwikkelaars van 'typische' webapplicaties tegemoet te komen. Het Web Profile bevat een minimum aan JEE6-onderdelen waarvan de verwachting is dat dit alles is wat je nodig hebt om een webapplicatie te kunnen maken. Iedere leverancier van een JEE6 Web Profile compatible applicatieserver staat het vrij om hier zelf andere technologiën aan toe te voegen. Deze minimale set is:

- Servlet 3.0;
- JavaServer Pages (JSP) 2.1 Maintenance Review 2;
- ExpressionLanguage (EL) 2.2;
- Debugging Support for Other Languages (JSR-45) 1.0;
- Standard Tag Library for JavaServer Pages (JSTL) 1.2;
- JavaServer Faces (JSF) 2.0;
- Common Annotations for Java Platform (JSR-250) 1.1;
- Enterprise JavaBeans (EJB) 3.1 Lite;
- Java Transaction API (JTA) 1.1;
- Java Persistence API (JPA) 2.0.

Hierbij is JSP 2.1 een vreemde eend in de bijt. De JEE6-specificatie zegt dat er geen nieuwe JSP-versie

komt. Het is de bedoeling dat alle ontwikkelaars overstappen op JSF en JSP wordt alleen ondersteund vanwege backward compatibility met JEE5 en daarvoor. Tóch is er een aanpassing gedaan aan JSP 2.1 die bekend is geworden op internet als JSP 2.2. De aanpassing is voornamelijk te vinden op het gebied van Expression Language alhoewel er ook minimale aanpassigen gedaan zijn op het gebied van JSP zelf er enkele onderdelen beter beschreven zijn en er fouten in de voorbeelden in de specificatie verbeterd zijn.

Applicatiedeployment

Een andere wijziging in JEE6 ten opzichte van eerdere versies is de manier waarop Enterprise Applicaties gedeployed kunnen worden. Voorheen mocht een .war-bestand alleen 'webgerelateerde' bestanden bevatten, zoals webpagina's, css, plaatjes, servlets en managed beans. Indien de webapplicatie gebruik maakte van Enterprise-componenten, zoals Session Beans en Entities, dan moesten deze in een apart .jar-bestand gezet worden en moesten de war en de jar gecombineerd worden in een .ear-bestand. Daarnaast was ooit een xml-bestand nodig dat aangaf welke Enterprise-componenten gebruikt mochten worden door de weblaag.

Het xml-bestand met een lijst van Enterprise-componenten was in JEE5 onder veel omstandigheden al overbodig geworden. Met JEE6 is ook de noodzaak voor een .ear-bestand komen te vervallen. Alle onderdelen van een JEE-applicatie mogen nu in een .war-bestand opgenomen worden. Het is daarbij aan de ontwikkelaar om te bepalen of Enterprise-componenten in een apart .jar-bestand opgenomen worden in het .war-bestand of dat de classes hierin net als bijvoorbeeld servlets en managed beans onder de classes directory opgenomen worden in het war bestand.

De manier waarop in Enterprise Applicaties in JEE6 gedeployed worden, is een wijziging met eerdere versie.

Servlet 3.0

Met JSR-315, Servlet 3.0, komt er een flinke uitbreiding op de manier waarop Servlets ontwikkeld kunnen worden. Zoals in alle 'moderne' JEE JSRs ligt de focus in JSR-315 op Ease Of Development. In het algemeen komt deze term neer op de introductie van annotaties en op het principe van Configuration By Exception. Een eenvoudige Servlet kan bijvoorbeeld als volgt gedeclareerd worden:

```
@WebServlet("/myurl")
public class TestServlet extends javax.servlet.http.
HttpServlet {
    ....
}
```

of iets uitgebreider:

```
@WebServlet(
    name="mytest",
    urlPatterns={"/myurl"},
    initParams={
        @WebInitParam(name="n1", value="v1"),
        @WebInitParam(name="n2", value="v2")
    }
)

public class TestServlet extends javax.servlet.http.
HttpServlet {
    ....
}
```

De @WebServlet annotatie neemt de rol over van de declaratie die voorheen in web.xml gedaan moest worden. Binnen deze annotatie kunnen parameters als de naam, waaronder de Servlet bekend is, de url, waarop de Servlet benaderd kan worden, en initiële waarden van Servlet parameters geconfigureerd worden. De class die met @WebServlet geannoteerd wordt, moet hierbij de HttpServlet class extenden. Ook moet de de URL waarop de servlet gemapped wordt, aangegeven worden met de value parameter (zoals in het eerste voorbeeld) of de urlPatterns parameter (zoals in het tweede voorbeeld). Verder geldt zoals bij alle andere annotaties dat configuratie die gedaan wordt in web.xml de configuratie op basis van annotaties overrrulet.

Op een vergelijkbare manier kunnen servlet filters gedefinieerd worden:

```
@WebFilter("/myurl")
public class MyFilter implements javax.servlet.Filter {
  public void doFilter(HttpServletRequest req,
HttpServletResponse res) {
        .....
  }
}
```

Hierbij geldt weer dat een vergelijkbare configuratie in web.xml deze annotatie overruled en dat de value parameter voor de url mapping verplicht is. En ook moet de class die met @WebFilter geannoteerd wordt de Filter interface implementeren.

Tenslotte kunnen ook listeners gedefinieerd worden met @WebListener. Er zijn verschillende Listener interfaces in de javax.servlet package en een class die geannoteerd is met @WebListener moet één van die interfaces implementeren, bijvoorbeeld

```
@WebListener
public class MyListener implements
ServletContextListener{
    public void contextInitialized(ServletContextEvent
sce) {
        ServletContext sc = sce.getServletContext();
       sc.addServlet("myServlet", "Sample servlet","foo.
bar.MyServlet", null, -1);
       sc.addServletMapping("myServlet", new String[] {"/
urlpattern/*" });
    }
}
```

Web Fragments

Een van de lastige zaken aan voorgaande Servlet-specificaties is dat alle servlet-gerelateerde informatie opgenomen moest zijn in het web.xml bestand.

Ook in het geval dat een servlet opgenomen is in een .jar-bestand, moest deze vernoemd zijn in web.xml. Om deze inflexibiliteit op te heffen is het concept van web module deployment descriptor fragments ofwel web fragments geïntroduceerd.

Een webfragment is een deel van (of een complete) web.xml dat relevant is voor de servlets en filters die zich in een .jar-bestand bevinden. Het enige dat een ontwikkelaar die een dergelijke jar beschikbaar stelt hoeft te doen, is een bestand met de naam web-fragment.xml opnemen in de META-INF directory van de jar. Een servlet container zal dan automatisch de in dat bestand genoemde servlets en filters laden en beschikbaar stellen. Een voorbeeld van een web-fragment.xml bestand is:

```
<web-fragment>
  <servlet>
    <servlet-name>welcome</servlet-name>
    <servlet-class>
      WelcomeServlet
    </servlet-class>
  </servlet>
  <listener>
    <listener-class>
      RequestListener
   </listener-class>
  </listener>
</web-fragment>
```

Samenstellen van de webconfiguratie

Bij het lezen van de bovenstaande paragraaf kun je je twee zaken afvragen. Ten eerste, hoe wordt nu bepaald in welke volgorde web.xml en web-fragment.xml wordt geparsed en geladen? Ten tweede, hoe kan voorkomen worden dat een web-fragment.xml uit veiligheidsoverwegingen geladen wordt?

Om met de eerste vraag te beginnen: de volgorde waarin web.xml en web-fragment.xml bestanden worden ingelezen is ongedefinieerd. Maar er zijn wel manieren om deze volgorde zelf te bepalen, namelijk met een <absolute-ordering> element in web.xml of een <ordering> element in web.xml waarmee relatieve ordering kan worden opgegeven. Stel dat we de volgende web.xml en twee web-fragment.xml bestanden hebben

```
web.xml
<web-app>
  <name>MyApp</name>
  ...
</web-app>

web-fragment1.xml
<web-fragment>
  <name>MyFragment1</name>
  <ordering><after>MyFragment2</after></ordering>
  ...
</web-fragment>

web-fragment2.xml
<web-fragment>
  <name>MyFragment2</name>
  ..
</web-fragment>
```

dan zal in dit geval de volgorde waarin de drie bestanden ingeladen en verwerkt worden zijn:

```
web.xml
web-fragment2.xml
web-fragment1.xml
```

Hiernaast zijn regels gedefinieerd voor de situatie waarin circulaire referenties optreden of wanneer verschillende web fragments servlets, filters of parameters definieren met dezelfde namen.
Dan de vraag betreffende de beveiliging van een web applicatie. Een malefiede ontwikkelaar zou in zijn jar bestand een servlet kunnen opnemen die een achterdeur openzet. Dit moet natuurlijk voorkomen kunnen worden. Dit kan gedaan worden door gebruik te maken van het <enabled> element in een web.xml bestand.

Asynchronous support

Met de opmars van SOA-applicaties is ook in de webwereld behoefte ontstaan aan asynchrone, non-blocking aanroepen van servlets en web applicaties. JSR-315 voorziet hierin door servlets in staat te stellen requests op de achtergrond te plaatsen en weer in leven te roepen als dat nodig is. Dit staat bekend als Comet-style programmeren. Op het moment dat een request op de achtergrond geplaatst wordt, wordt de thread die het request aan het afhandelen was weer vrijgegeven, zodat deze andere taken kan afhandelen. Zodra een resource, zoals een JMS queue of een database, met een antwoord komt, wordt het request weer in leven geroepen met een aanroep aan de resume method waarna een response aan de client gegeven wordt.

Overige aanpassingen

De specificatie voor JSR-315 is nog niet helemaal compleet, maar bevat nu al veel meer dan hierboven beschreven is. Een van de wijzigingen betreft de mogelijkheid om servlets, filters en listeners programmatisch te initializeren en te laten draaien. Verder zijn er discussies gaande in de JSR-315 expert group voor een uitgebreid security model dat door servlets afgedwongen kan worden. Denk hierbij aan servlets methods zoals authenticate, login en logout maar ook annotaties als @RolesAllowed, @DenyAll, @PermitAll en @TransportProtected.

Conclusie

Dit artikel geeft slechts een klein overzicht van de mogelijkheden en wijzigingen die met JEE6 en Servlet 3.0 zullen komen. Toch is de impact hiervan voor JEE-ontwikkelaars aanzienlijk. JEE-applicaties kunnen steeds meer mogelijkheden van het JEE-platform aanspreken en de manier waarop dit kan wordt steeds eenvoudiger. Daarnaast worden de drempels qua hardware infrastructuur steeds lager. Dit zijn zeer positieve ontwikkelingen en ik kan niet wachten tot JEE6 uitkomt. «

De impact van JEE6 en Servlet 3.0 is aanzienlijk voor Java-developers.