Enterprise

Als er één framework is dat het afgelopen jaar sterk aan populariteit heeft gewonnen is dat JBoss Seam. Seam is een applicatieframework met als doel het ontwikkelen van webapplicaties eenvoudiger te maken. Het framework doet dit door bestaande technieken zoals JSF, EJB3 en jBPM met elkaar te integreren, en daar een aantal nieuwe concepten aan toe te voegen.

Webapplicaties maken met JBoss Seam

Applicatieframework voegt nieuwe concepten toe

De basis van Seam is een combinatie van JSF en EJB3. Het gebruik van EJB3 in de weblaag is ook ongetwijfeld een van de meest opvallende zaken aan het framework, en ook direct het punt waarop de meeste kritiek wordt gegeven. Een normale JSF applicatie bestaat uit JSP (of Facelets) pagina's en Managed Beans. Laatst genoemde zijn POJO's waarin alle logica achter een pagina wordt opgenomen. Seam vervangt de Managed Beans door EJB3 Session Beans. Voordat we hier verder naar kijken is het zinvol om eerst te kijken naar wat Seam vanuit een functioneel oogpunt biedt.

De functionaliteit van Seam

Seam helpt bij het ontwikkelen van webapplicaties door onder meer de volgende zaken:

- Een uitgebreider state management model
- Betere integratie met de middenlaag
- Eenvoudige remoting
- Dependency Injection / Outjection
- Integratie met jBPM

Het verbeterde state management model is waarschijnlijk het belangrijkste onderdeel van Seam. Het idee hierachter is dat HTTP sessions te beperkt zijn om geavanceerde applicaties mee te ontwikkelen. Het grootste probleem is dat er geen eenvoudige manier is om data van verschillende onderdelen uit de sessie van een gebruiker van elkaar te onderscheiden. Een gebruiker meerdere taken parallel laten uitvoeren levert dan ook al snel het probleem op dat sessiedata van verschillende taken door elkaar gaat lopen. Seam lost dit probleem op door de introductie van 'conversations'. Hier zullen we verderop in het artikel verder naar kijken.

Basisvoorbeeld

Nu we een overzicht hebben van de mogelijkheden van Seam is het tijd voor een eerste code voorbeeld. Codevoorbeeld 1 is een zeer eenvoudige Facelets pagina waar een gebruiker zijn naam kan invoeren. Na het klikken op de Say Hello knop, krijgt de gebruiker een persoonlijke begroeting. Let er op dat de JSP/Facelets pagina altijd precies hetzelfde is als er geen gebruik wordt gemaakt van Seam.

```
<body>
 <ui:define name="body">
  <h:form id="frm_Hello">
    Please fill in your name:
      <h:inputText  id="txt_Name"  value="#{hello.
name}"/>
     <h:commandButton  id="btn_sayHello"  value="Say
Hello!"
      action="#{hello.sayHello}"/>
   <br/>
   <h:outputText id="greeting" value="Hello #{hello.
name}"
  rendered="#{not empty hello.name}" />
  </h:form>
 </ui:define>
</body>
```

Codefragment 1. Hello.xhtml

In codevoorbeeld 2 is de Backing Bean van de pagina weergegeven. Dit is een Stateful Session Bean met een methode sayHello, en een get/set methode voor het opslaan van de ingevoerde naam.

Paul Bakker
is ontwikkelaar bij Info Support B.V. Hij is actief in de Advanced Web Applications werkgroep en doet hiervoor onder andere onderzoek naar verschillende webframeworks. E-mail: paulb@infosupport.com, Blog: http://blogs.infosupport.com/blogs/paul_bakker/

```
@Local
public interface Hello {
   public String sayHello();
   public String getName();
   public void setName(String
                              name);
   public void remove();
}


@Stateful
@Scope(ScopeType.EVENT)
@Name("hello")
public class HelloBean implements Hello {
   private String name;

   public String getName() {
  return name;
   }

   public String sayHello() {
  return "";
   }

   public void setName(String name){
      this.name = name;
   }

  @Remove
   public void remove() {
   }
}
```

Codefragment 2. Hello.java en HelloBean.java

De @Scope en @Name annotaties zijn onderdeel van Seam en zijn geen onderdeel van EJB3. De @Scope annotatie geeft aan wat de scope is van de bean. Dit kan onder andere Event, Session en Conversation zijn. In dit geval is er gekozen voor de Event scope, wat ongeveer neerkomt op request scope van een standaardwebapplicatie. De naam opgegeven met @Name is de naam waarmee JSP/Facelets pagina's en andere beans de bean kunnen aanspreken. In codevoorbeeld 1 gebeurt dit bijvoorbeeld bij de binding #{hello.name}. Een methode geannoteerd met @Remove is verplicht voor elke Stateful Bean en wordt aangeroepen als de context van het component eindigt. Een aardige bijkomstigheid van Seam is dat beans niet gedefinieerd hoeven te worden in de faces-config.xml, wat bij een normale JSF applicatie wel moet.

Gelaagdheid

De meeste webapplicaties zijn opgedeeld in lagen. Vaak wordt er gesproken over een weblaag, een businesslaag en een data-/integratielaag. Een van de dingen die hiermee bijvoorbeeld bereikt wordt is dat de middenlaag bruikbaar kan worden gemaakt voor meerdere clientapplicaties. Het argument tegen het gebruik van EJB in de weblaag is dan ook dat als EJB's vanuit de middenlaag naar de weblaag verhuizen, de herbruikbaarheid hiermee ook verdwijnt. Dit argument is echter onjuist. Het feit dat er EJB in de weblaag wordt gebruikt, betekent niet dat er niet ook nog een aparte (EJB) middenlaag kan zijn. Aan de structuur van de applicatie hoeft niets te veranderen. Het enige wat er verandert, is dat de POJO Backing Beans worden vervangen door Session Beans. De architectuur van een systeem bepaal je dus nog steeds zelf, Seam legt je hier niet in vast. Uiteraard wordt EJB3 niet zonder reden gebruikt. Door het gebruik hiervan wordt het stateful model van Seam mogelijk gemaakt. Bovendien biedt dit een veel betere integratie tussen de weblaag en de middellaag van een applicatie.

Conversations

Conversations zijn een van de vernieuwende concepten die Seam met zich meebrengt. Vanuit het oogpunt van een gebruiker is een Conversation een 'unit-of-work', oftewel een taak die de gebruiker uitvoert met een duidelijk start- en eindpunt. Technisch gezien kan een conversation het best worden gezien als een subsessie. Net als bij een normale HTTP-sessie kunnen data worden toegekend die geldig zijn gedurende de levensloop van de conversation. Conversations verschillen op twee punten van sessies. Ten eerste wordt een Conversation altijd expliciet vanuit code gestart en gestopt. Dit gebeurt met behulp van annotations. Het tweede verschil is dat conversations een uniek ID hebben, en er zodoende meerdere conversations tegelijkertijd voor een enkele gebruiker actief kunnen zijn. Door het gebruik van conversations is het bijzonder eenvoudig om een gebruiker meerdere taken parallel aan elkaar te laten uitvoeren. Ook kan er bijvoorbeeld worden gewisseld tussen openstaande taken, dit concept wordt 'workspace management' genoemd.

Op grotere schaal is het concept van conversations ook geïmplementeerd. Seam kan worden gekoppeld aan een jBPM configuratie. jBPM is een framework waarmee businessprocessen kunnen worden gedefinieerd. Data die geldig zijn voor een bepaald proces kan op een soortgelijke manier als

Seam voegt nog iets toe aan het bekende dependency injection concept, namelijk het 'outjecten' van dependencies

Een veel voorkomend probleem bij het werken met een ORM framework in een webapplicatie zijn fouten bij het gebruik van lazy loading

met sessies worden gekoppeld aan het proces. Data kan zelfs worden gedeeld door meerdere gebruikers die een rol hebben binnen het proces. Deze koppeling werkt op basis van relatief eenvoudige XML-configuratie, en annotaties in de code.

Remoting

Een veelbesproken technologie op dit moment is ongetwijfeld AJAX. Ook Seam bevat mogelijkheden om dit op een elegante manier aan te pakken. Wat Seam biedt, is een hele eenvoudige manier om vanuit JavaScript direct methoden aan te roepen op een Session Bean. Hier hoef je zelf vrijwel geen code voor te schrijven. Seam is echter geen UI framework, en biedt dus ook geen grafische componenten. Na het ontvangen van data vanaf de server, zul je dus nog wel zelf iets met deze data in Javascript moeten doen. Uiteraard kun je hier wel weer gebruik maken van andere library's zoals Dojo en Prototype. Codevoorbeeld 3 laat zien hoe je van Seam Remoting gebruik kunt maken. Aan de server-kant hoeft er alleen een methode als @WebRemote geannoteerd te worden. De implementatie van deze methode kan bijvoorbeeld een lijst met Customer objecten teruggeven, gefilterd met de parameter 'name'.

```
@Local
public interface RemotingDemo {
   @WebRemote
   public List<Customer>
      getCustomers(String name);
}
```

Codefragment 3. Session Bean interface

Codevoorbeeld 4 laat vervolgens zien hoe je deze @WebRemote methode kunt gebruiken. Hiervoor moet Seam in elk geval weten van welke interface je gebruik wilt maken. Seam genereert hier tijdens runtime een proxy voor. Vervolgens kun je de met @WebRemote geannoteerde methoden via zo'n proxy benaderen.

```
<script type="text/javascript" src="seam/remoting/
resource/remote.js"/>

<script type="text/javascript" src="seam/remoting/
interface.js?remotingDemo"/>
```

```
<script>
function filterCustomers(textField) {
Seam.Component.getInstance("remotingDemo").
getCustomers(textField.value, filterCallback);
}

function filterCallback(result) {
//Hier de ontvangen data verwerken
}
</script>
```

Codefragment 4. Facelets pagina met Remoting

Deze manier van werken met Javascript proxies is op zichzelf niet heel erg bijzonder, er zijn meer frameworks die dit op vergelijkbare manier kunnen. Het voordeel van Seam is echter dat de remoting geïntegreerd is in het framework, en er hierdoor aan de serverkant vrijwel niets gedaan hoeft te worden.

Dependency Injection

Dependency Injection houdt in dat dependency's niet zelf door classes worden opgehaald of worden gecreëerd, maar worden geïnjecteerd door de container. Dit concept komt terug in EJB3 en vormt de basis van bijvoorbeeld het Spring framework. In Seam gebruik je dependency injection door aan een klasse variabele de @In annotatie toe te voegen. Deze annotatie zorgt ervoor dat de variabele een waarde krijgt geïnjecteerd vanuit de container. Op deze manier kun je bijvoorbeeld een service injecteren. Een bijzonder aspect van het Dependency Injection mechanisme in Seam is dat injectie plaatsvindt bij een methode-aanroep, en niet tijdens het creëren van een object (zoals bijvoorbeeld in Spring). Dit biedt betere mogelijkheden voor stateful componenten.

Dependency Outjection

Seam voegt nog iets toe aan het bekende dependency injection concept, namelijk het 'outjecten' van dependencies. Outjecten houdt in dat de waarde van een variabele terug wordt gepubliceerd naar de container nadat deze gewijzigd is, zodat deze bij andere Session Beans geïnjecteerd kan worden. Niet geheel verrassend gebeurt dit met de annotatie @Out.

JPA

Een veel voorkomend probleem bij het werken met een ORM framework in een webapplicatie

zijn fouten bij het gebruik van lazy loading. Bij lazy loading worden relaties van een klasse uit de database gehaald op het moment dat er om deze relatie gevraagd wordt, zodat er geen data worden opgehaald die nooit gebruikt zal worden. In een webapplicatie zal pas bekend worden welke waarden er met lazy loading moeten worden opgehaald als de view opgebouwd wordt. De view is in dit geval bijvoorbeeld een JSP. Het probleem is dat op dit moment de transactie waarin de data is opgehaald al gesloten is. Hierdoor zullen er excepties optreden. Seam lost dit probleem op een hele transparante manier op. Seam gebruikt onder water een Extended Persistent Context met converstation scope en een extra transactie per request. Zonder uitgebreide kennis van EJB3 lijkt dit misschien ingewikkeld, maar gelukkig wordt dit volledig door het framework afgehandeld. Als ontwikkelaar van een Seam-applicatie ben je hiermee verlost van lazy loading problemen, zonder dat dit noemenswaardige impact op de performance heeft.

JSR WebBeans

De ideeën achter Seam zijn ondergebracht in JSR 299, genaamd WebBeans. Op deze manier wil men deze ideeën standaardiseren. Er zijn plannen om WebBeans onderdeel te maken van Java EE 6, dat gepland staat voor eind 2008. In welke vorm dit precies zal zijn is op dit moment nog niet helemaal duidelijk, een interessante ontwikkeling is het in elk geval wel.

Type applicaties

Een mooie eigenschap van Seam is dat het framework erg goed schaalt. Zelfs bij het ontwikkelen van een eenvoudige data invoerapplicatie helpt Seam zonder dat je veel extra tijd kwijt bent aan allerlei configuratie. Aan de andere kant is Seam net zo goed inzetbaar in grote Enterpriseapplicaties, en biedt zelfs integratie met bijvoorbeeld jBPM. Dit komt ook omdat Seam je niet dwingt tot een bepaalde architectuur. Ook is er een plugin beschikbaar voor Eclipse en NetBeans waarmee er vanuit een database schema een Seam-applicatie kan worden gegenereerd. Deze gegenereerde code kan als startpunt dienen voor een eigen Seam-project.

Conclusie

Seam maakt het een stuk eenvoudiger om webapplicaties te ontwikkelen. Het werken met annotaties neemt een hoop configuratie weg, en zaken zoals Remoting en de integratie met JPA schelen een hoop problemen die je niet zelf meer hoeft op te lossen. De echte kracht van Seam zit echter vooral in het stateful model, zoals Conversations. Dit maakt het mogelijk om met veel minder moeite applicaties te ontwikkelen die veel gebruiksvriendelijker zijn dan met een meer traditionele aanpak mogelijk is. Seam is te gebruiken in elke applicatieserver die EJB3 ondersteunt. Vanaf de website is er echter alleen een download beschikbaar die direct op een JBoss applicatieserver werkt. Voor het gebruik in een andere applicatieserver (zoals Glassfish) zijn er extra, niet meegeleverde, library's noodzakelijk. Ook is Seam te gebruiken vanuit Tomcat in combinatie met de JBoss Microcontainer.

Referenties

Seam home: http://labs.jboss.com/jbossseam
JSR 299: http://jcp.org/en/jsr/detail?id=299
Documentatie: http://labs.jboss.com/jbossseam/docs
Blog: http://blogs.infosupport.com/blogs/paul_bakker/