Core

De data voor uitwisseling tussen webcomponenten, zoals servlets, pagina's gecreëerd met JavaServer Pages (JSP) of JavaServer Faces (JSF), wordt over het algemeen onderhouden in een database. Aan de hand van drie voorbeelden bespreken we hoe die uitwisseling, ook wel databinding genoemd, voor de verschillende technologieën bewerkstelligd kan worden.

Databinding - van JSP naar JSF

Rene van Wijk

We beperken ons tot presentatie georiënteerde webapplicaties, dat wil zeggen interactieve webpagina's die verschillende typen markup-talen en dynamische content als response voor een request bevatten. Webcomponenten zoals servlets en JSP's voorzien in de dynamische mogelijkheden. Interactie tussen een webclient en een webapplicatie kan in het kort beschreven worden als: webcomponenten op een server ontvangen een request van een client en genereren een response aan de hand van het request.

Zoals al eerder opgemerkt wordt de data voor webcomponenten over het algemeen onderhouden in een database. Hier doet zich een probleem voor: Hoe te communiceren met de database? Het tabellarisch presenteren van data in een relationeel systeem is fundamenteel anders dan de objectennetwerken die gebruikt worden in object georiënteerde Java-applicaties. Deze fundamentele misaanpassing wordt nog steeds onderschat. Aan de andere kant zijn er nu enkele frameworks en oplossingen om met dit probleem om te gaan. Door gebruik te maken van de juiste frameworks is databinding simpelweg een link tussen UI-componenten en de data.

In het vervolg van dit artikel worden drie voorbeelden beschreven met betrekking tot de databinding voor verschillende technologieën:

- Het eerste voorbeeld maakt gebruik van JSPs.
- Vervolgens wordt een voorbeeld besproken dat gebruik maakt van JSF.

JavaServer Pages (JSP)

Een JSP-pagina is een op tekst gebaseerd document dat twee typen tekst bevat: statische data, dat uitgedrukt kan worden in elk tekstgebaseerd formaat en JSP-elementen, die de dynamische content construeren. Om data uit de database te benaderen, wordt gebruik gemaakt van JDBC (elke CRUD operatie wordt met de hand gecodeerd). Opgehaalde data wordt toegevoegd aan list-of-beans-objecten om een makkelijke toegang tot de data te verkrijgen met behulp van Java-objecten. JSP biedt een mechanisme om terugkerende taken te omvatten in custom-tags. Met behulp van deze tags wordt het ontwerp van de webuserinterface vereenvoudigd.

Om functionaliteit te distribueren en de afhankelijkheden te minimaliseren tussen verschillende applicatie-objecten wordt een model-view-controller-architectuur gebruikt. Dat wil zeggen dat de applicatie in drie lagen opgedeeld wordt:

- Modellaag
- Viewlaag
- Controllerlaag

De modellaag bestaat uit een Java-klasse die de JDBC-statements bevat. De code voert een query uit en vult een list-of-beans-object met de resultaten. Om dit concept te hanteren, gebruiken we de JavaBeans conventies, dat wil zeggen,

- Private fields
- Toegang via public methoden
  - Leeseigenschappen: `PropertyClass getProperty() {...}`
  - Schrijfeigenschappen: `setProperty(PropertyClass pc) {...}`
- Constructor zonder parameters

De controllerlaag bestaat uit een Java-klasse die de paginalogica afhandelt. In het algemeen zal deze klasse ook aan de JavaBeans conventies voldoen. De reden voor deze ontwerpkeuze is het gebruik van expression language (EL) om de verschillende methoden van de Java-klasse te

benaderen. Het element `jsp:useBean` geeft aan dat de JSP-pagina een JavaBean gaat gebruiken. De JavaBean is toegankelijk in een gedefinieerde scope, bijvoorbeeld de session-scope. Refereren aan een eigenschap van de JavaBean kan met behulp van het element `jsp:getProperty`, dit roept een methode aan van de JavaBean. De methode genereert een HTML tabel waarvoor de data geleverd wordt door de modellaag.

Custom-tags

Custom-tags zijn door de programmeur gedefinieerde JSP-elementen. Om eigen tags (componenten) toe te voegen moeten we het volgende principe in acht nemen. Een JSP-pagina wordt van boven naar onder uitgevoerd. Tags die bij een bibliotheek horen geven een event. Dit event zorgt ervoor dat de methode doStartTag() van TagSupport wordt aangeroepen. Door te erven van TagSupport en de methode doStartTag() te overriden kunnen we eigen code en functionaliteit met een event associëren. Uiteindelijk wordt de sluit-tag voor het element bereikt en wordt de methode doEndTag() aangeroepen. Naast eigen gedefinieerde tags is het natuurlijk ook mogelijk om componentenbibliotheken te gebruiken, zoals jMaki. jMaki is een framework om web 2.0 applicaties te creëren door gebruik te maken van ingebouwde templates, een model voor Ajax-enabled widgets en een set services om widgets aan elkaar te knopen en te laten communiceren met externe services. In de tutorial bevindt zich ook een voorbeeld dat gebruikt maakt van jMaki.

JavaServer Faces

JSF, een componentenframework, bevat een API voor het representeren van UI-componenten en twee JSP custom tag bibliotheken.

Het programmeermodel en componentenbibliotheken, zoals Trinidad, maken het bouwen van webapplicaties vrij gemakkelijk. Het beschreven voorbeeld gebruikt Hibernate in de modellaag en Trinidad in de viewlaag. Een opmerking is hier op zijn plaats: Hibernate wordt in het voorbeeld in een JSF-omgeving gebruikt het is natuurlijk ook mogelijk om Hibernate te gebruiken in een JSP-omgeving.

Hibernate

Hibernate is een object/relationele mapping tool. Object/relationele-mapping (ORM) refereert aan de techniek om een datarepresentatie van een objectmodel te mappen aan een relationeel model.

Als leidraad nemen we het bovenstaande figuur. Hierin zien we aan de linkerkant een schematische opbouw van een applicatie en aan de rechterkant een schematische weergave van een object/relationele mapping. Om de object/relationele mapping te verkrijgen moeten we het persistentie framework (in dit geval Hibernate) informatie geven (in figuur 1 aangegeven middels de puzzelstukjes):

- hoe bijvoorbeeld de klasse Employee persistent gemaakt moet worden, en
- hoe instanties van de Employee klasse moeten worden geladen en opgeslagen.

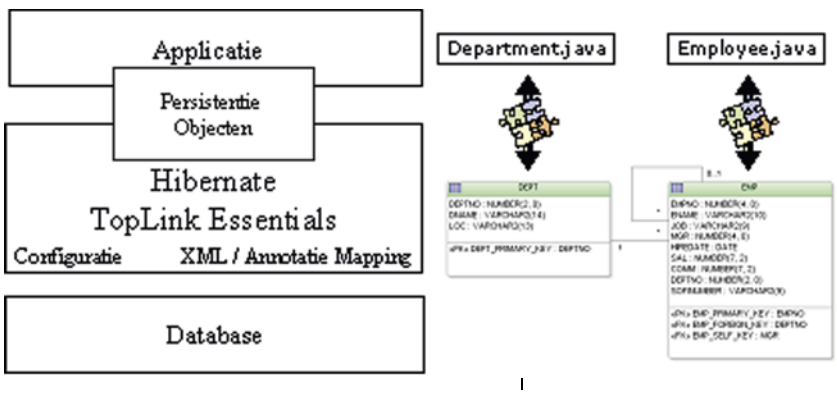
Voor de persistentieklassen worden JavaBeans gebruikt. Hibernate moet weten hoe de objecten van de persistentieklasse moeten worden geladen en opgeslagen. Hiervoor maakt Hibernate gebruikt van een zogenaamde mappingfile. Een mappingfile vertelt Hibernate welke tabel het moet benaderen en welke kolommen het moet gebruiken. Aangezien Hibernate de laag is in de applicatie die een connectie maakt met de database, moet connectie-informatie worden verstrekt. Deze informatie bevindt zich in een configuratiefile (`hibernate.cfg.xml`).

Het starten van Hibernate houdt in dat er een globaal `SessionFactory`-object wordt opgebouwd aan de hand van de verstrekte informatie in de configuratiefile. Op zijn beurt kan een `SessionFactory` nieuwe sessies openen (`Session`). Een Session weerspiegelt een 'unit-of-work', bijvoorbeeld een request van een client in een multi-user omgeving. Het streven is naar één Session-per-Request, dat wil zeggen,

- Een request wordt verstuurd van een client naar de server;
- Een nieuwe Hibernate `Session` wordt geopend;
- Database-operaties worden uitgevoerd (Een eenmalige databasetransactie voor een request van een client. Om dit te implementeren start er een transactie wanneer een server-request verwerkt moet worden. De transactie stopt voordat de response wordt verstuurd.);
- Als het werk gedaan is, wordt de `Session` gesloten.

Trinidad

Trinidad is een set van UI-componenten dat voorbouwt op de JSF API. Trinidad is door Oracle gedoneerd aan de Apache open-source-



Figuur 1.

community. Het voorbeeld beschreven in de tutorial moduleert een gestructureerde datalijst in de vorm van een master/detail hiërarchie (departments/employees). De data is verkregen met behulp van Hibernate die de data doorgeeft als een `List` object. Om de data aan een gebruiker te presenteren wordt gebruik gemaakt van de Table-component van Trinidad. Dit component is vergelijkbaar met de standaard `UIData` component in JSF, maar met extra eigenschappen, zoals:

- navigeren door het model;
- sorteren van het model;
- selecteren van enkele of meerdere items in het model.

Door gebruik te maken van expression language om de te presenteren informatie te benaderen, kunnen we het objectennetwerk (master/detail hiërarchie) aflopen dat door Hibernate wordt aangeboden.

Application Development Framework

ADF bouwt voort op Java EE standaarden en open-source technologieën. Het volgende voorbeeld gebruikt Business Components for Java (BC4J) in de modellaag en ADF Faces in de viewlaag.

Business Components for Java

Net als met Hibernate worden er een set Java-componenten aangemaakt die de databasetabellen representeren. Om dit te bewerkstelligen voorziet BC4J in de volgende componenten:

- Entity-objecten
  - Zijn componenten die gerelateerd zijn aan een tabel
  - Opleggen van de businesslogica
  - Opslaan van veranderingen aan de data
- View-objecten
  - Zijn query-componenten die de data voor de user-interface vormgeven
  - Kunnen aan entity-objecten refereren
- Applicatiemodule
  - Zijn business-service-clients die worden gebruikt om data te modificeren en door data te browsen
  - Gebruiken view-objecten in het datamodel

BC4J gebruikt XML-files die definiëren hoe de databasetabellen zijn gestructureerd (zoals kolomnamen, data typen, enzovoorts). Een datamodel gedefinieerd met behulp van BC4J wordt aan de viewlaag doorgegeven middels zogenaamde datacontrols. Datacontrols worden gecreëerd aan de hand van het datamodel en abstraheren de

business-services naar een algemene laag. Deze laag, genaamd datacontrol-palette, bevat:
- Datacollecties die instanties van view-objecten representeren;
- Beschikbare attributen van elke datacollectie (Java-objecten van de kolommen);
- Operation-folders met ingebouwde operaties, zoals `Create`, `Delete`, enzovoorts.

ADF Faces en databinding

ADF Faces is een uitgebreide bibliotheek met ongeveer honderd JSF-componenten (het was ADF Faces dat is gedoneerd als Trinidad aan de Apache open-source-community). Met ADF is het maken van pagina's een gedeeltelijke drag en drop ervaring. Bijvoorbeeld in JDeveloper, drag en drop
- Een `panelPage` vanuit de ADF Faces Core bibliotheek in het component-palette;
- De datacollectie `departments` vanuit het datacontrol-palette;
- De detail datacollectie `employees` vanuit het datacontrol-palette.

Nadat de datacollectie departments aan de pagina is toegevoegd, wordt aan de pagina specifieke XML file (paginadefinitiefile) het volgende toegevoegd:
- een departments-iterator die de data afhandelt
- een value-binding met een geschikte naam.

De paginadefinitie definieert de databindings die de UI-componenten op de pagina ondersteunen. Deze databindings worden als volgt gecreëerd. Een additionele handler (`ADFBindingFilter`) wordt getriggerd als een gebruiker een JSP-pagina opvraagt. Deze handler maakt een juiste bindingcontainer aan voor de huidige pagina en maakt de bindingcontainer toegankelijk middels de EL-expressie #{`bindings`}. In het onderstaande figuur wordt een schematische representatie gegeven.

De volgende concepten kunnen worden herkend:
- Bindingcontext, de data omgeving voor de applicatie;
- Datacontrol, abstracte implementatie van de business services applicatiemodule;
- Bindingcontainer, groep met gerelateerde iterator- en control-bindings gekoppeld aan de pagina.

De bindingcontext bevat de datacontrols en de bindingcontainers die de datacontrols gebruiken. Elke bindingcontainer heeft
- iterator-bindings die de datacollecties identificeren die gebruikt worden door de pagina, en
- control-bindings die de UI-controls op de pagina ondersteunen.

Tijdens het verwerken van een request wordt de JSF-lifecycle gestart. Met behulp van standaard mechanismen voert ADF voor of na bepaalde fasen extra stappen uit:
- Voorbereiden van de data die getoond moet worden;
- Aanroepen van operaties;
- Modificeren en valideren van data voor de view-objecten in het datamodel van de applicatiemodule.

Bij het opstarten van een ADF-applicatie zijn de volgende componenten betrokken:
- `FacesServlet`, onderhoudt de lifecycle voor alle JSF-applicaties;
- `ADFBindingFilter`, initialiseert de bindingcontext voor een HTTP-sessie van een gebruiker, licht datacontrol-instanties in dat er een request aankomt en meldt dat een response is afgehandeld;
- `DataBindings.cpx`, definieert de bindingcontext van de applicatie. Bevat metadata waarmee bindingobjecten worden aangemaakt, dat wil zeggen koppelt pagina's aan de bijbehorende paginadefinitiefiles en legt vast welke datacontrols gebruikt worden.

Bij het starten van de applicatie laadt de ADFBindingFilter de DataBindings.cpx-file. De lifecycle geeft de URL door aan de bindingcontext.
De bindingcontext koppelt de URL aan een paginadefinitie met de informatie uit de DataBindings.cpx-file. De bindingcontext instantieert de bindingcontainer en registreert de datacontrols.
Door de gehanteerde databinding wordt door ADF de mogelijkheid geboden om declaratief JSF-pagina's te bouwen.

Conclusies

Bij het doorlopen van de verschillende technologieën worden veel verschillende mogelijkheden geboden, hoe een datagebonden-applicatie gebouwd kan worden.

Data modelleren met behulp van JDBC is tijdrovend. Een framework als Hibernate neemt veel vervelend werk uit handen. ADF gaat nog een stap verder door naast het modelleren van data met behulp van BC4J, een koppeling te bieden die ons in staat stelt om op een declaratieve wijze pagina's te bouwen. Het is zelfs mogelijk om pagina's te laten genereren met behulp van een additionele plug-in JHeadstart.
In het algemeen zal de vraag van ontwikkelaars of het niet makkelijker kan ervoor zorgen dat de frameworks steeds meer werk uit handen nemen, waardoor Java EE gemakkelijk in het gebruik wordt. Met deze filosofie valt volgens mij best te leven. «

Referenties

Een tutorial (inclusief code) van de voorbeelden is te vinden op http://www.technicalconferencesolutions.com/pls/caat/caat_abstract_reports.display_presenter_abstract?conference_id=29&presenter_id=1953&abstract_id=73