Rich Internet Applicaties (RIA) worden steeds populairder. Er bestaan verschillende technologieën en frameworks voor het ontwikkelen van Rich Internet Applicaties. Eén van deze frameworks is Vaadin. Dit is een framework waarbij het mogelijk is om met alleen Java-code een webapplicatie te ontwikkelen. Dit artikel beschrijft het ontwikkelen van applicaties met het Vaadin-framework.

Webapplicaties ontwikkelen met Vaadin

Compleet framework dat volledig in Java werkt

Jamie Craane
is software engineer/architect en Java/Flex competence leader bij QNH Application Development & Solutions. Tevens is hij werkzaam in het bestuur van de Nederlandse Flex User Group (FLUGR).

Vaadin is ontwikkeld door het bedrijf IT-Mill. De ontwikkeling van het framework startte in 2000 onder de naam IT-Mill Toolkit. Eind 2007 is release 5 van IT-Mill Toolkit als open source project ondergebracht onder de Apache License 2, waarbij in maart 2009 de stabiele versie 5.3.0. is gereleased. Pas in de lente van 2009 heeft men de naam IT-Mill Toolkit omgedoopt naar Vaadin met als reden naamsverwarring te voorkomen tussen de organisatie IT-Mill en het open source project Vaadin.

Puur Java-code

Vaadin-applicaties worden volledig geschreven in Java. Wie met Google Web Toolkit (GWT) bekend is, herkent dit. Het voordeel van deze werkwijze is dat de ontwikkelaar geen kennis nodig heeft van HTML en JavaScript. Voor de cross-browser rendering maakt Vaadin gebruik van GWT.

Het werken in Java biedt de volgende voordelen:

- compile-time type checking;
- het kunnen unit testen van de user interface code;
- zowel server als user interface code kunnen in één geïntegreerde omgeving ontwikkeld worden.

Wie ervaring heeft met desktop user interface toolkits zoals Swing is vrij snel bekend met het ontwikkelen van Vaadin-applicaties. Codefragment 1 laat een voorbeeld zien van een eenvoudige Vaadin-applicatie.

```
public class VaadinartikelApplication extends
Application {
    private TextField textField;

    @Override
    public void init() {
        Window mainWindow =
        new Window("Vaadin Application");
        mainWindow.setContent(new HorizontalLayout());
        mainWindow.addComponent(new Label
        ("What is your name: "));

        textField = new TextField();
        mainWindow.addComponent(textField);

        mainWindow.addComponent(new Button("Click me",
        new Button.ClickListener() {
            @Override
            public void buttonClick(ClickEvent event) {
                Window popup = new Window("Hello");
                popup.addComponent(new Label((String)
                textField.getValue()));
                getMainWindow().addWindow(popup);
            }
        }));

        setMainWindow(mainWindow);
    }
}
```

Codefragment 1: eenvoudige Vaadin applicatie.

Codefragment 1 toont een eenvoudige Vaadin-applicatie met een Label, TextField en Button component. De HorizontalLayout zorgt ervoor dat de drie componenten naast elkaar getoond worden. De Button bevat een event handler die wordt uitgevoerd wanneer de gebruiker op de button klikt. Hierbij wordt een popup getoond met daarin de waarde die is ingevoerd in het TextField.

Vaadin ondersteunt (nog) geen op XML-gebaseerde markup taal. Men is wel aan het bekijken of dit in een toekomstige release wordt toegevoegd. Voor Eclipse is er een WYSIWYG editor beschikbaar om de user interface vorm te geven. Een WYSIWYG

editor voor andere ontwikkelomgevingen dan Eclipse staat op de planning.

Volledig serverside

Een groot verschil tussen Vaadin en GWT is dat in Vaadin alle events op de server worden afgehandeld. Dit maakt Vaadin tot een zogenaamd server-side RIA framework. De state van de user interface componenten leeft dan ook op de server (in de sessie). Dit heeft een aantal voordelen:

- event handlers kunnen eenvoudig debugged worden;
- een Vaadin-applicatie is niet client-side te beïnvloeden waardoor je als ontwikkelaar geen zorgen meer hoeft te maken over security hacks zoals het enablen van een disabled button. De server-side button is namelijk nog steeds disabled, ook wanneer diezelfde button client-side enabled wordt;
- AJAX communicatie tussen de client- en servercomponent is volledig afgeschermd door het framework.

Doordat events op de server worden afgehandeld, wordt de state van elk component bij elke gebruikersinteractie verstuurd naar de server. Het formaat van dit bericht is vastgelegd in de User Interface Definition Language (UIDL) wat feitelijk JSON (Javascript Object Notation) is. Vaadin optimaliseert de hoeveelheid data die naar de server wordt verstuurd door alleen de state van de relevante (lees gewijzigde) componenten te versturen waardoor het netwerkverkeer wordt beperkt. Figuur 1 toont de globale architectuur van het Vaadin framework.

De voornaamste verschillen in ontwikkelen tussen Vaadin en GWT zijn:

- Vaadin is een server-side RIA framework. GWT is een client-side RIA framework;
- Vaadin schermt de AJAX communicatie volledig af van de gebruiker van het framework en is impliciet aanwezig. Bij GWT implementeert de developer de server- en client-side services zelf, die nodig zijn voor de client-server communicatie;
- Vaadin heeft wat meer network roundtrips nodig dan GWT aangezien het een server-side RIA framework is. In de praktijk betekent dit 10-20% meer server roundtrips;
- Vaadin heeft geen development mode zoals GWT dit heeft. Bij Vaadin wordt de code dan ook niet elke keer naar Javascript gecompileerd wanneer er wijzigingen in de Java-code worden gemaakt. Uitzondering hierbij is wanneer er een nieuwe GWT widget is toegevoegd. Hierbij is een eenmalige compilatie noodzakelijk;
- Uit ervaring blijkt dat de ontwikkeltijd van een Vaadin-applicatie over het algemeen een stuk korter is dan van een GWT applicatie.

Impliciete AJAX communicatie

Doordat de applicatie volledig in Java wordt geschreven, heeft de ontwikkelaar niet te maken met low-level AJAX communicatie. Een typische gebruikersactie ziet er als volgt uit:
1) de gebruiker klikt op een button;
2) de Vaadin client-side engine verstuurt een UIDL bericht naar de server;
3) het click event van de button wordt op de server afgevangen;
4) de event handler update één of meerdere componenten, bijvoorbeeld het toevoegen van rijen aan een tabel;
5) de gewijzigde status wordt als UIDL response teruggegeven;
6) de Vaadin client-side engine update de gewijzigde componenten.

Zoals in bovenstaande flow is te zien, worden alleen de gewijzigde componenten geüpdate waardoor page-refreshes niet aan de orde zijn.

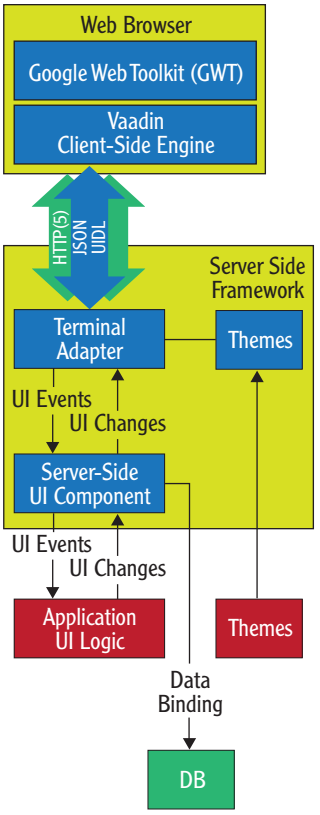
Componentenbibliotheek

Eén van de eisen van een goed applicatieframework is de beschikbaarheid en volledigheid van een goede componentenbibliotheek. Vaadin beschikt over een groot aantal bruikbare componenten. Een aantal van deze componenten zijn:

- Button, Label, TextField;
- DateSelector, MenuBar, ComboBox, etc;
- Table. Een Table is geschikt voor het tonen van tabulaire data. De Vaadin Table component ondersteunt bijvoorbeeld het selecteren van één of meerdere items, resizen/reorderen en collapsen van kolommen en het lazy laden van data;
- Tree. Een Tree is geschikt voor het tonen van hiërarchische data.

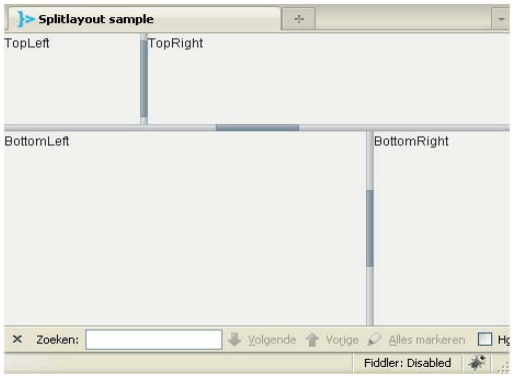
Naast de user interface componenten beschikt Vaadin over een groot aantal layoutmanagers waarvan de belangrijkste zijn:

- Horizontal- en VerticalLayout. De Horizontal- en VerticalLayout ordenen de componenten naast elkaar/onder elkaar;
- GridLayout. De GridLayout plaatst componenten in een door de ontwikkelaar te definiëren grid waarbij het mogelijk is om componenten over meerdere cellen te verdelen;
- SplitPanel. De SplitPanel layout bestaat uit twee gebieden die door de gebruiker zijn te resizen. SplitPanels zijn verticaal of horizontaal georiënteerd en de split positie kan worden vergrendeld. Zie figuur 2 voor een voorbeeld van een layout met een SplitPanel. Codefragment 2 laat hiervan de code zien;
- CustomLayout. De CustomLayout maakt het mogelijk om een layout te ontwikkelen in HTML die bijvoorbeeld met CSS wordt vormgegeven. In



Figuur 1: Vaadin architectuur.

Vaadin handelt alle events af op de server in tegenstelling tot GWT.



Figuur 2: voorbeeld van geneste SplitPanel layout's.

de HTML worden placeholders (in de vorm van DIV's) gedefinieerd die worden vervangen door Vaadin-componenten;

- CssLayout. De CssLayout plaatst alle componenten binnen de layout in een DIV. Deze DIV's kunnen dan weer met CSS worden vormgegeven. De CssLayout is de snelste layout om te renderen aangezien de CssLayout het minste HTML-elementen nodig heeft.

De meeste layoutmanagers ondersteunen onder andere de volgende eigenschappen die bepalen hoe componenten zich binnen de layout gedragen:

- spacing: bepaalt de ruimte tussen de componenten binnen een layout;
- margin: bepaalt de ruimte om de layout heen. Standaard heeft een layout geen marge;
- alignment: bepaalt hoe componenten binnen een layout cell zijn uitgelijnd. Bijvoorbeeld top-left of middle-center.

Soms is het zinvol om layoutmanagers te vervangen door CssLayout.

Hoe complexer de applicatie, hoe complexer het is om de layout van de applicatie te implementeren. Let bij het ontwikkelen van de layout op het overmatig gebruik van geneste layouts. Hoe complexer de layout, hoe meer tijd het kost om deze layout te renderen. In die gevallen kan het zinvol zijn om bepaalde layoutmanagers te vervangen door bijvoorbeeld de CssLayout. Zeker op wat oudere browsers (lees IE6, IE7) kan dit een performancewinst tijdens het renderen opleveren.

Zoals eerder vermeld maakt Vaadin voor de rendering van componenten gebruik van GWT. Dit maakt het eenvoudig om bestaande GWT componenten te integreren binnen Vaadin. Daarnaast is sinds eind maart 2010 de zogenaamde Vaadin directory live. Dit is een Marketplace voor open source en commerciële componenten waarin op dit moment circa 145 componenten beschikbaar zijn.

De feature sampler geeft een goed overzicht van alle beschikbare componenten en layoutmanagers.

```
public class VaadinLayout extends Application {
    @Override
    public void init() {
        Window mainWindow =
       new Window("Splitlayout sample");
        SplitPanel mainSplitPanel =
       new SplitPanel(SplitPanel.ORIENTATION_VERTICAL);
        SplitPanel upper =
       new SplitPanel(SplitPanel.ORIENTATION_HORIZONTAL);
        upper.setFirstComponent(new Label("TopLeft"));
        upper.setSecondComponent
         (new Label("TopRight"));
        SplitPanel lower =
       new SplitPanel(SplitPanel.ORIENTATION_HORIZONTAL);
        lower.setFirstComponent
       (new Label("BottomLeft"));
        lower.setSecondComponent(new
Label("BottomRight"));
        mainSplitPanel.setFirstComponent(upper);
        mainSplitPanel.setSecondComponent(lower);

        mainWindow.setContent(mainSplitPanel);
        setMainWindow(mainWindow);
    }
}
```

Codefragment 2: voorbeeld van geneste SplitPanel layout's.

Custom componenten

Naast de uitgebreide set van standaardcomponenten is het ook mogelijk om zelf componenten te ontwikkelen. Door af te leiden van CustomComponent worden gemakkelijk, herbruikbare componenten gemaakt. In Codefragment 3 is een custom component geïmplementeerd die een zoekveld en een button combineert. Deze component kan op eenvoudige wijze worden gebruikt door een instantie hiervan mee te geven aan de addComponent() methode.

```
public class VaadinCustomComponent extends Application {
    @Override
    public void init() {
        Window mainWindow =
       new Window("Sample Application");
        mainWindow.addComponent(new Search());
        setMainWindow(mainWindow);
    }

    public static class Search extends
CustomComponent implements Button.ClickListener {
        private TextField q;

        public Search() {
        AbstractLayout mainLayout
        = new HorizontalLayout();
            q = new TextField();
            mainLayout.addComponent(q);
            mainLayout.addComponent
           (new Button("Search", this));
            setCompositionRoot(mainLayout);
        }

        public void buttonClick(ClickEvent event) {
            System.out.println("Search for: "
             + q.getValue());
        }
    }
}
```

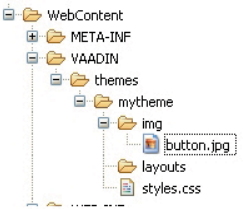
Codefragment 3: implementatie en gebruik van custom componenten.

Verder is het ook mogelijk om geheel nieuwe componenten te ontwikkelen door bestaande GWT componenten te integreren of client-side functionaliteit van bestaande componenten uit te breiden. Dit valt echter buiten de scope van dit artikel maar is uitstekend beschreven in de documentatie.

Styling

In Vaadin is het uiterlijk van de user interface vast-

gelegd in themes. Een theme bevat stylesheets (CSS), HTML layouts en statische resources zoals plaatjes. Themes staan onder de WebContent/VAADIN/themes directory. De volgende conventies zijn hierbij van toepassing: de naam van het theme is de naam van de subdirectory en elk theme moet minimaal de file styles.css bevatten. Figuur 3 laat deze directorystructuur zien:



Figuur 3: plaatsing van custom themes onder WebContent.

In plaats van alles afzonderlijk te stylen, is het aan te raden een bestaand theme als uitgangspunt te nemen. Door hiervan de CSS te wijzigen wordt eenvoudig het gewenste effect bereikt. Een goed uitgangspunt hierbij is het Chameleon theme. Dit theme bevat een online theme editor waarmee een aantal basiseigenschappen zijn te stylen. Na het aanpassen van deze eigenschappen kan de betreffende CSS worden gedownload. Verdere aanpassingen kunnen daarna in de CSS worden gemaakt.

Resources

Naast het aanbieden van reguliere views, is het voor veel webapplicaties noodzakelijk om dynamische content in alternatieve vorm aan te bieden, bijvoorbeeld PDF. Vaadin biedt hiervoor de Resource interface met diverse implementaties. Eén van deze implementaties is de StreamResource class. Deze class kan worden gebruikt om dynamische content aan te bieden, zoals een PDF of een gegenereerde grafiek. Codefragment 4 creëert een StreamResource uit een PDF die wordt weergegeven in de applicatie. De (niet getoonde) DynamicPdf is een custom class die de Vaadin StreamSource interface implementeert. Deze interface definieert de methode InputStream getStream(). In dit voorbeeld retourneert de getStream methode uit de DynamicPdf class een ByteArrayInputStream van de PDF. De DynamicPdf maakt gebruik van iText om de daadwerkelijke PDF te genereren.

```
// this verwijst naar de Application instance
StreamResource resource = new StreamResource
(new Pdf(), "test.pdf?", this);
resource.setMIMEType("application/pdf");
mainWindow.addComponent(new Embedded("", resource));
```

Codefragment 4: tonen van een dynamische gegenereerde PDF.

Bookmarking

Een veelvoorkomend probleem bij AJAX applicaties die als een single page application in de browser draaien, is het ontbreken van browser history functionaliteit. Dit maakt het bookmarken van specifieke pagina's en het gebruik van de browser back button onmogelijk. Vaadin beschikt over de UriFragmentUtility component die het mogelijk maakt om het fragment van een URL te gebruiken. Het fragment van een URL is het gedeelte achter het '#' teken zoals in http://example.com/path#myfragment.
Een fragment kan worden gezet met de setFragment() methode van de UriFragmentUtility component, bijvoorbeeld bij het klikken op een tab. Wanneer een fragment wijzigt (door het klikken op een bookmark dat een fragment bevat) wordt een FragmentChangedEvent gedispatched. In de handler van dit event kan het fragment worden uitgelezen met de getFragment() methode. Hiermee kan de applicatie het juiste scherm aan de gebruiker laten zien. De back button werkt hierbij uiteraard ook aangezien de verschillende URL's in de browser history terechtkomen.

Documentatie

Hoe goed een framework ook is, de documentatie ervan is minstens zo belangrijk. Vaadin komt met een 300 pagina's dik boek dat online beschikbaar is en als PDF. Dit boek biedt een zeer compleet overzicht van de werking van het Vaadin-framework inclusief alle beschikbare componenten. Daarnaast zijn er diverse tutorials en sample applicaties beschikbaar waardoor men snel up-to-speed is.

Toolsupport

Vaadin komt gebundeld met een Eclipse-plugin die de ontwikkeling van Vaadin-applicaties ondersteunt. Met de plugin is het eenvoudig om een nieuw raamwerk Vaadin-applicatie te creëren en te deployen op een webserver zoals Tomcat. De plugin biedt ook een WYSIWYG editor.
De debug window wordt gebruikt om layout problemen op te sporen. De debug window wordt gestart door '?debug' achter de URL van de applicatie te plaatsen.
Figuur 4 toont een voorbeeld van de debug window waarin het conflicterende component rood is gemarkeerd.

Figuur 4: Vaadin debug window.

Een Vaadin-applicatie wordt gepackaged als een webarchive (war) file waarbij de Vaadin jar file is opgenomen in de WEB-INF/lib directory. Verder biedt Vaadin out-of-the-box ondersteuning voor Google App Engine.

Conclusie

Vaadin is een zeer complete en volwassen technologie voor het ontwikkelen van Rich Internet Applications. Doordat de ontwikkeling volledig in Java plaatsvindt, zijn in korte tijd indrukwekkende applicaties te realiseren. Doordat het framework al geruime tijd in ontwikkeling en productie is, ben ik tijdens het werken met Vaadin weinig issues tegengekomen.
Het voelt allemaal erg robuust en solide aan. De goede documentatie, behulpzame forumleden en consistente release cycle maken Vaadin tot waardig alternatief van bestaande Rich Internet Application frameworks. «

Referenties

1. http://vaadin.com
2. http://demo.vaadin.com/sampler/#
3. http://vaadin.com/directory
4. http://vaadin.com/wiki/-/wiki/Main/all_pages
5. http://demo.vaadin.com/chameleontheme