2-2013 | Sommer | www.ijug.eu

Java aktuell

Praxis. Wissen. Networking. Das Magazin für Entwickler

Java aktuell

Java verbreitet sich überall

Ausblicke
JDeveloper 12*c*, Seite 8
Android goes Gradle, Seite 29

Hochverfügbarkeit
JBoss AS7, Seite 21

Web-Entwicklung
Play!, Seite 32
Linked Data, Seite 38

Java und Oracle
Continous Integration, Seite 59

D: 4,90 EUR A: 5,60 EUR CH: 9,80 CHF Benelux: 5,80 EUR ISSN 2191-6977

iJUG Verbund

Sonderdruck

4 191978 304903 02

3 Editorial
Wolfgang Taschner

5 Das Java-Tagebuch
Andreas Badelt

8 Oracle JDeveloper 12*c* – ein Ausblick
Frank Nimphius

13 Asynchrone Datenabfragen mit DataFX
Hendrik Ebbers

17 Stolperfallen bei der Software-Architektur
Frank Pientka

21 Hochverfügbarkeit mit dem JBoss AS 7
Heinz Wilming und Immanuel Sims

29 Android goes Gradle
Heiko Maaß

32 Web-Apps mit „Play!" entwickeln – nichts leichter als das!
Andreas Koop

38 Linked-Data-Praxis: Daten bereitstellen und verwerten
Angelo Veltens

42 Vagrant: Continuous Delivery ganz einfach
Sebastian Laag

45 Cobol und Java: Zwei Sprachen kommen sich näher
Rolf Becking

48 Continuous Bugfixing in großen Projekten
Jürgen Nicolai

53 „Wir sollten das Oracle-Bashing unterlassen …"
Interview mit Falk Hartmann, Java UserGroup Saxony

54 Pragmatisches Testen mit System
Martin Böhm

59 Drillinge – bei der Geburt getrennt. Wie PL/SQL, Apex und Continuous Integration wieder zusammenfinden
Markus Heinisch

62 Datenschutz-konformes Social Sharing mit Liferay
Michael Jerger

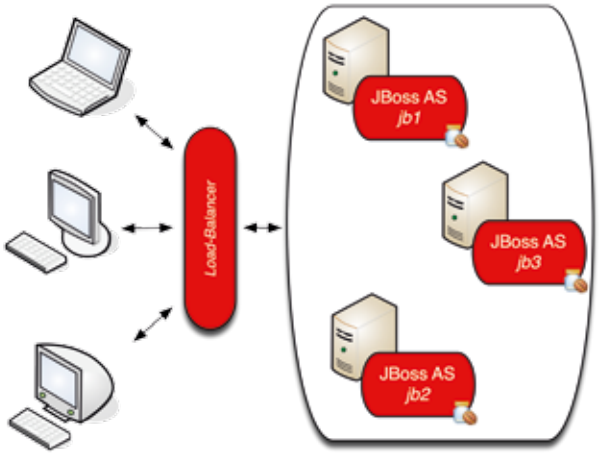
65 Unbekannte Kostbarkeiten des SDK Heute: Der ZIP-File-System-Provider
Bernd Müller

66 DevFest Vienna 2012
Dominik Dorn

25 Unsere Inserenten

28 Die iJUG-Mitglieder auf einen Blick

37 Impressum



Lastverteilung zum Erreichen der Hochverfügbarkeit, Seite 20

Build-Kreislauf beim Continuous Integration, Seite 59

Asynchrone Datenabfragen mit DataFX

Hendrik Ebbers, Java User Group Dortmund

Mit JavaFX 2.x hat Oracle ein mächtiges neues API für Desktop-Anwendungen veröffentlicht. Das ursprünglich als Skriptsprache veröffentlichte JavaFX wurde komplett in Java umgeschrieben und wird seit Java 7 Update 6 standardmäßig mit Java ausgeliefert. Das Framework liefert eine Fülle an UI-Komponenten und Features, die alle modernen Ansprüche an User Interfaces wie Animationen, Multitouch, CSS-Styling, Charts etc. bedienen.

Mit JavaFX ist es möglich, moderne Rich Clients für eine wachsende Zahl an Endgeräten wie Desktop-PCs, Tablets oder Embedded-Devices zu entwickeln. So wurde beispielsweise vor Kurzem eine erste JDK-/JavaFX-Preview für das Raspberry Pi veröffentlicht (siehe http://www.guigarage.com/2012/12/my-first-steps-with-javafx-on-raspberry-pi/). DataFX, das API, das dieser Artikel vorstellt, erweitert JavaFX um einige nützliche Funktionen in den Bereichen der Daten-Visualisierung und -Beschaffung.

DataFX setzt sich aktuell aus zwei verschiedenen Bestandteilen zusammen. Der erste bietet verschiedenste Implementierungen für List-, Table- und Tree-Zellen. Ein Beispiel dafür ist die „MoneyTableCell", mit der sich Zahlen mit einer Währungs-Formatierung direkt in einer Tabelle darstellen lassen. Dank CSS-Unterstützung kann man sogar verschiedene Styles für positive und negative Beträge definieren. In diesem Artikel wird das Hauptaugenmerk auf den zweiten Teil von DataFX gelegt. Dieser unterstützt Entwickler mit einem „DataSource & DataReader"-API, durch das beliebiger Content asynchron zur Nutzung in JavaFX Controls abgerufen werden kann.

DataFX liegt seit Ende 2011 in der Version 1.0 vor und kann auf der Projektseite (siehe http://www.javafxdata.org) heruntergeladen oder per Maven/Gradle zu einem Projekt hinzugefügt werden. Listing 1 zeigt die Maven Dependency.

```
<dependency>
                    <groupId>org.javafxdata</groupId>
                    <artifactId>datafx-core</artifactId>
                    <version>1.0</version>
</dependency>
```

Listing 1

Multi-Threading in Desktop-Anwendungen

Ähnlich dem Event Dispatch Thread in Swing gibt es in JavaFX auch einen Thread, der für die Aktualisierung der GUI zuständig ist. Darin sind das komplette Layouten und Neuzeichnen der Benutzeroberfläche sowie sämtliche Benutzer-Interaktionen behandelt. Damit dürfen allerdings – wie bereits in Swing – alle Variablen, die an die Oberflächenkontrollen gebunden sind, nur in diesem Thread bearbeitet werden. Der Inhalt beziehungsweise das Datenmodell einer Tabelle darf also nicht außerhalb des JavaFX-Application-Thread geändert werden. Viele Business-Anwendungen beziehen ihre Daten jedoch von Backend-Systemen, Datenbanken oder Web-Services. Die Abfrage dieser Systeme kann einige Zeit in Anspruch nehmen und würde im schlimmsten Fall den JavaFX-Thread komplett blockieren, sodass ein Neuzeichnen der Oberfläche oder das Reagieren auf Benutzereingaben nur noch mit einer extremen Zeitverzögerung durchgeführt wird. Listing 2 zeigt ein Negativbeispiel, in dem durch einen Buttonklick die Oberfläche für längere Zeit komplett einfriert.

```
Button myButton = new Button("click me");
    myButton.setOnAction(new EventHandler<ActionEvent>() {

                    @Override
                    public void handle(ActionEvent arg0) {
                            String result = longDurationRequest();
                            myTextfield.setText(result);
                    }
            });
```

Listing 2

DataFX bietet nun die Möglichkeit, genau solche zeitaufwändigen Aufgaben in einen Hintergrund-Thread zu verlagern und nach einem erfolgreichen Laden der Daten diese im JavaFX-Thread bereitzustellen. Das Ganze ist im Grunde mit dem durch Java 6 eingeführten „SwingWorker" vergleichbar. Durch Nutzung des Property-API von JavaFX und dessen Binding-Features können Ergebnisse zusätzlich viel einfacher an der Oberfläche dargestellt werden.

iTunes-Suche mit DataFX

Anhand einer Web-Service-Abfrage ist die Nutzung von DataFX hier beispielhaft dargestellt. Als Basis dient das „iTunes Search"-

API von Apple, das, wie die meisten Web-Services heutzutage, REST nutzt und Daten im JSON-Format liefert. Den Web-Service kann man über die URL „https://itunes.apple.com/search?term=Nirvana“ selbst ausprobieren. In diesem Fall werden 50 in iTunes erhältliche Medien (Videos, Songs etc.) mit dem Schlagwort „Nirvana“ gesucht und zurückgeliefert. Listing 3 zeigt die JSON-Antwort des Service als Beispiel.

Zum Auslesen von Daten bietet DataFX das Interface „org.javafxdata.datasources.reader.DataSourceReader“ und verschiedene Implementierungen an. Als konkrete Implementierung wird für das Beispiel die Ableitung „org.javafxdata.datasources.reader.RestRequest“ genutzt, die das Ergebnis eines HTTP-Request als Rohdaten zurückliefert. Dieser kann einfach über einen Builder erstellt und parametrisiert werden (siehe Listing 4).

Durch den Builder kann der Endpoint komplett definiert werden. Betrachtet man die Dokumentation des iTunes-API unter „http://www.apple.com/itunes/affiliates/resources/documentation/itunes-store-web-service-search-api.html” genauer, findet man noch eine Fülle an Parametern zur besseren Definition der Suche. Im ersten Aufruf weiter oben wird zum Beispiel der „term“-Parameter in der URL angegeben. Dieser könnte ebenfalls direkt durch den Builder definiert werden. Zusätzlich wird noch angegeben, dass der Request JSON-Daten liefert (siehe Listing 5).

Um die Daten nun in Java-Objekte umzuwandeln, ist eine Klasse nötig, die das Datenmodell beschreibt. Da DataFX „Jackson” als JSON-Parser nutzt, kann diese Klasse mit Jackson-Annotations versehen werden. Die hier genutzte Klasse soll den Namen, den Preis und ein Vorschaubild zu iTunes Content kapseln (siehe Listing 6).

Um die rohen JSON-Daten zu parsen und für JavaFX bereitzustellen, ist eine „DataSource“ notwendig. DataSources dienen in DataFX dazu, Daten asynchron in einem Hintergrund-Thread zu laden. Für das konkrete Beispiel wird die DataFX Klasse „ObjectDataSource“ genutzt, die XML- oder JSON-Daten in Java-Objekte umwandelt.

Auch für diese Klasse gibt es wieder einen Builder, mit dem sich einfach eine konfigurierte Instanz erstellen lässt. Dem Builder wird der „DataSourceReader“ übergeben. Zusätzlich gibt es noch eine Besonderheit in den eingehenden JSON-Daten. Solange, wie der JSON-Input aus einem einzelnen Element oder aus einem Array von gleichen Elementen besteht, ist es klar, dass diese Elemente geparst werden sollen.

```
{
 “resultCount“:50,
 “results“: [
{“trackName“:“Smells Like Teen Spirit“,
“artworkUrl100“:“http://a190.phobos.apple.com/us/r1000/032/Features/4b/c4/cb/
dj.mjhyndcl.100x100-75.jpg“,
...},
...]
}
```

Listing 3

```
RestRequest request = new RestRequestBuilder(“http://itunes.apple.com“).
path(“search“).build();
```

Listing 4

```
RestRequest request = new RestRequestBuilder(“http://itunes.apple.com“).
path(“search“).format(Format.JSON).queryParam(“term“,“Nirvana“).
queryParam(“media“,“music“).build();
```

Listing 5

```
@JsonIgnoreProperties(ignoreUnknown = true)
public class ITunesMedia implements Serializable {
        private static final long serialVersionUID = 1L;
        private String artworkUrl100;
        private String trackName;
        private double trackPrice;
        public ITunesMedia() {}
        public String getArtworkUrl100() {
                return artworkUrl100;
        }
        public void setArtworkUrl100(String artworkUrl100) {
                this.artworkUrl100 = artworkUrl100;
        }
        public String getTrackName() {
                return trackName;
        }
        public void setTrackName(String trackName) {
                this.trackName = trackName;
        }
        public double getTrackPrice() {
                return trackPrice;
        }
        public void setTrackPrice(double trackPrice) {
                this.trackPrice = trackPrice;
        }
}
```

Listing 6

In dem hier genutzten Beispiel möchten wir allerdings nur einen Teil der Daten, und zwar das „results"-Array, parsen. Für diesen Zweck kann man dem Builder der „ObjectDataSource" das gewünschte Tag übergeben. Zuletzt muss nur noch eine „ObservableList" als Ergebnisliste übergeben werden. Dies ist eine mit JavaFX neu eingeführte Liste, deren Zustandsänderungen durch Listener überwacht werden können. Übergibt man diese Liste einer „ListView" oder „TableView" als Datenmodell, passt sich die Oberfläche automatisch an, sobald Daten in der Liste geändert werden. Sollte der Builder eine nicht leere Liste erhalten, werden alle neu erstellten Objekte ans Ende der Liste gehängt (siehe Listing 7).

Der Aufruf der „retrieve()"-Methode startet die Datenbeschaffung im Hintergrund. Die erhaltenen Daten werden dann im JavaFX-Application-Thread automatisch der Ergebnisliste hinzugefügt. Um die Daten nun in einer grafischen Oberfläche anzuzeigen, nutzen wir eine Tabelle, der wir die „ObservableList" als Datenmodell übergeben. Zusätzlich müssen noch die Tabellenspalten definiert werden. Hier wollen wir den Namen, das Vorschaubild und den Preis jeweils in einer eigenen Spalte anzeigen.

Wie man in Abbildung 1 sehen kann, werden momentan noch die Standardzellen-Implementierungen der JavaFX-Tabelle genutzt. Diese stellen die Werte einfach über ein „toString()" dar. Beim Namen ist dies absolut in Ordnung. Allerdings soll auch das Bild angezeigt werden und die Preisangabe in einer vernünftigen Formatierung erscheinen. Für Letzteres bietet DataFX die bereits in der Einleitung angesprochene „MoneyTableCell". Zellen zur Darstellung von Bildern werden mit dem nächsten Release von DataFX folgen. Listing 8 zeigt eine einfache Implementierung.

Über Cell-Factories sind die beiden Zellen-Implementierungen nun zur Darstellung der Daten in der Tabelle nutzbar. Die Erstellung der einzelnen Instanzen und das Setzen der Daten in die Zelle erfolgt automatisch durch JavaFX. Erweitert man die Anwendung noch um ein einfaches Suchfeld, kann man mit wenigen Zeilen Code eine einfache iTunes-Suche in JavaFX realisieren. Unter http://www.doag.org/go/java_aktuell/ebbers steht der komplette Sourcecode zum Download bereit.

Zusätzlich zu dem bisher besprochenen Code wird im Beispiel noch die JavaFX-GUI erstellt. Hierzu werden mehrere Builder-Klassen genutzt, durch die man schnell ein einfaches User-Interface aufbauen kann. Die Datenabfrage ist in einem „EventHandler" realisiert, der jedes Mal ausgeführt wird, sobald der Button eine Aktion feuert.

Abbildung 2 zeigt anhand eines Web-Service, wie einfach es ist, ein JavaFX-Programm zu erstellen, das mittels DataFX problemlos Daten aus einer externen Quelle visualisieren kann. Das fertige Programm ist als Maven-Projekt unter „https://github.com/guigarage/DataFX-iTunes-Demo" zu finden.

Weitere DataFX-Features

Neben den im Beispiel vorgestellten Funktionen bietet das DataSource-API von DataFX noch einige weitere Features, die hier kurz angeschnitten werden sollen.

Da die „retrieve()"-Methode der DataSource ein Objekt vom Typ „javafx.concurrent.Worker" zurückgibt, kann man zusätzlich die im Hintergrund laufende Datenbeschaffung überwachen oder abbrechen. Der Worker arbeitet intern mit JavaFX-Properties, an die jederzeit „Change-

Abbildung 1: Suchergebnisse in JavaFX

Abbildung 2: Die fertige iTunes-Suche

```
ObservableList<ITunesMedia> items = FXCollections
                                    .<ITunesMedia> observableArrayList();
ObjectDataSource<ITunesMedia> dataSource = ObjectDataSourceBuilder.<ITunesMedia>create().
dataSourceReader(request).resultList(items).itemTag("results").itemClass(ITunesMedia.class).build();
dataSource.retrieve();
```

Listing 7

```
public class ImageTableCell extends TableCell<ITunesMedia, String> {

        public ImageTableCell() {
                final ImageView view = new ImageView();
                setGraphic(view);

                itemProperty().addListener(new ChangeListener<String>() {

                        @Override
                        public void changed(ObservableValue<? extends String> observableValue,
                                        String oldValue, String newValue) {
                                try {
                                        if (newValue != null) {
                                                view.setImage(new Image(newValue, true));
                                        } else {
                                                view.setImage(null);
                                        }
                                } catch (Exception e) {
                                        view.setImage(null);
                                        e.printStackTrace();
                                }
                        }
                });
        }
}
```

Listing 8

```
dataSource.retrieve().stateProperty().addListener(new ChangeListener<State>() {
                        @Override
                        public void changed(ObservableValue<? extends State> observableValue,
                                        State oldValue, State newValue) {
                                System.out.println("Current State:" + newValue);
                        }
                });
```

Listing 9

Listener" registriert werden können (siehe Listing 9).

Mit den bereits vorgestellten Klassen „ObjectDataSource" und „RestRequest" ist es auch sehr einfach möglich, Daten an einen REST-Endpunkt zu senden. Hierzu kann für den Request zum Beispiel „POST" als HTTP-Methode ausgewählt und der DataSource explizit mitgeteilt werden, dass es kein Result gibt. Der HTTP-POST wird automatisch im Hintergrund durchgeführt und blockiert nicht den JavaFX-Application-Thread. Zusätzlich lassen sich neben den Query-Parametern auch Form- oder Request-Parameter übergeben. So kann man beispielsweise das Senden einer HTML-Form emulieren. Auch „Oauth" wird durch Setzen eines Key/Secret in den Grundzügen unterstützt. Die jeweiligen Builder-Klassen besitzen Methoden zur Konfiguration aller dieser Parameter. Neben der hier vorgestellten „ObjectDataSource" zur Erstellung von Java-Objekten aus XML- oder JSON-Daten gibt es in der Version 1.0 von DataFX noch eine „JdbcDataSource" sowie eine „CSVDataSource", die Java-Objekte aus Datenbank-Abfragen beziehungsweise CSV-Dateien erstellen können.

Fazit und Ausblick

In der aktuellen Version bietet DataFX viele Hilfestellungen, um Daten aus verschiedenen Quellen für eine JavaFX-Anwendung bereitzustellen. Dem Entwickler wird sämtliche Multi-Threading-Programmierung abgenommen, sodass er sich voll auf User-Interface und Datenmodell konzentrieren kann.

Für die nächste Version von DataFX wird das DataSource-API aktuell deutlich weiterentwickelt. Alle „Reader" und „DataSources" werden auf gemeinsame Interfaces und Basis-Klassen migriert, wodurch eine generische Programmierung einfacher fällt. Zusätzlich sollen sich die „DataSources" besser konfigurieren lassen. So kann man beispielsweise die Executor-Instanzen angeben, in denen die Hintergrund-Tasks ablaufen sollen. All diese Erweiterungen sollen die Nutzung und Integration von DataFX in großen Business-Anwendungen deutlich einfacher und attraktiver gestalten. Dazu kommen neue, konkrete Implementierungen von „Readern" und „DataSources", um Bilder und andere Daten im Hintergrund zu laden. Neue Zellen, wie die bereits angesprochene Zelle für Grafiken, und Utility-Klassen werden die nächste Version abrunden.

Hendrik Ebbers
hendrik.ebbers@web.de

Hendrik Ebbers ist Leiter der Java User Group Dortmund und beschäftigt sich größtenteils mit Rich-Client-Programmierung und Middelware. Sein aktuelles Hauptaugenmerk liegt auf JavaFX. Neben DataFX arbeitet er noch an weiteren Open-Source-Projekten wie JFXtras (siehe http://jfxtras.org) mit. Seine aktuellen Arbeiten und Forschungsergebnisse veröffentlicht er regelmäßig auf seinem eigenen Blog www.guigarage.com

www.ijug.eu

JETZT ABO BESTELLEN

Sichern Sie sich 4 Ausgaben für 18 EUR

Für Oracle-Anwender und Interessierte gibt es das Java aktuell Abonnement auch mit zusätzlich sechs Ausgaben im Jahr der Fachzeitschrift DOAG *News* und vier Ausgaben im Jahr Business News zusammen für 70 EUR. Weitere Informationen unter www.doag.org/shop/

FAXEN SIE DAS AUSGEFÜLLTE FORMULAR AN

0700 11 36 24 39

ODER BESTELLEN SIE ONLINE

go.ijug.eu/go/abo

Interessenverbund der Java User Groups e.V.
Tempelhofer Weg 64
12347 Berlin

Java aktuell

+++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN +++ AUSSCHNEIDEN +++ ABSCHICKEN +++ AUSFÜLLEN

☐ **Ja**, ich bestelle das Abo Java aktuell – das IJUG-Magazin: 4 Ausgaben zu 18 EUR/Jahr

☐ **Ja**, ich bestelle den kostenfreien Newsletter: Java aktuell – der iJUG-Newsletter

ANSCHRIFT

Name, Vorname

Firma

Abteilung

Straße, Hausnummer

PLZ, Ort

GGF. ABWEICHENDE RECHNUNGSANSCHRIFT

Straße, Hausnummer

PLZ, Ort

E-Mail

Telefonnummer

X

Die allgemeinen Geschäftsbedingungen* erkenne ich an, Datum, Unterschrift

*Allgemeine Geschäftsbedingungen:

Zum Preis von 18 Euro (inkl. MwSt.) pro Kalenderjahr erhalten Sie vier Ausgaben der Zeitschrift "Java aktuell - das iJUG-Magazin" direkt nach Erscheinen per Post zugeschickt. Die Abonnementgebühr wird jeweils im Januar für ein Jahr fällig. Sie erhalten eine entsprechende Rechung. Abonnementverträge, die während eines Jahres beginnen, werden mit 4,90 Euro (inkl. MwSt.) je volles Quartal berechnet. Das Abonnement verlängert sich automatisch um ein weiteres Jahr, wenn es nicht bis zum 31. Oktober eines Jahres schriftlich gekündigt wird. Die Wiederrufsfrist beträgt 14 Tage ab Vertragserklärung in Textform ohne Angabe von Gründen.