

Teil I - Nodes

Jaroslav Tulach, Sun Microsystems
Ergänzungen: David Strupl, Sun Microsystems
Geertjan Wielenga, Sun Microsystems

Deutsche Überarbeitung - Aljoscha Rittner
Sepix GmbH
NetBeans Dream Team

Agenda Teil I

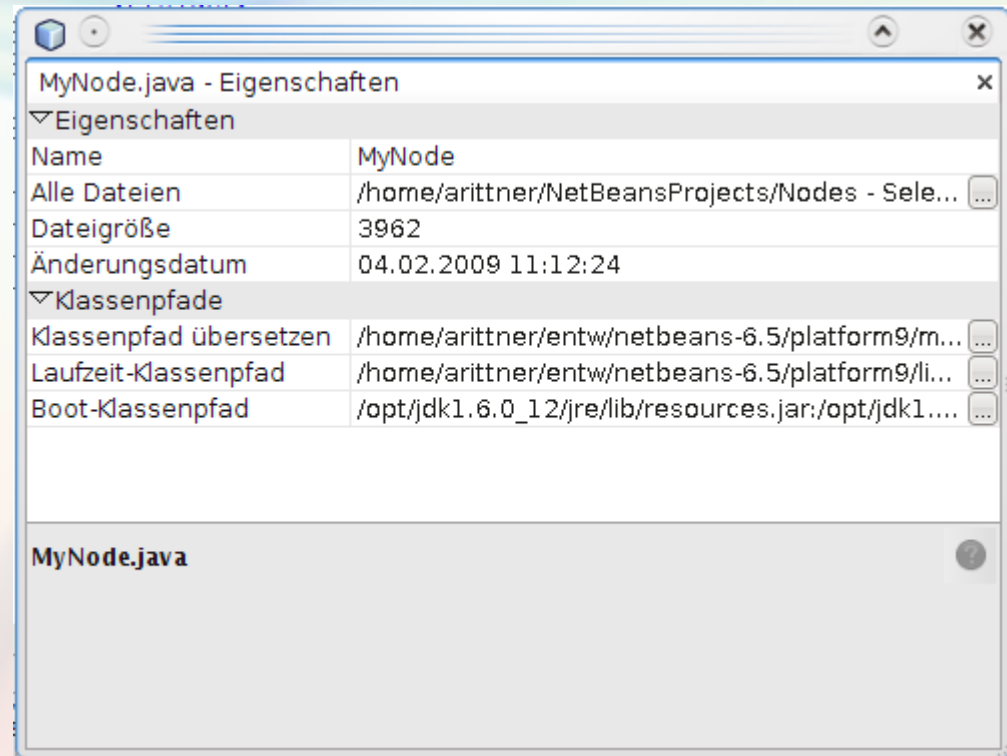
- Nodes
 - > Warum heißt NetBeans: NetBeans?
 - > Nodes API
 - > Fähigkeiten
 - > Hierarchien (Children-Container)
 - > Actions und Kontext
 - > Fragen und Antworten

Agenda Teil II

- Explorer Views
 - > Explorer View
 - > Von NetBeans gelieferte Views
 - > Der Explorer Manager
 - > Views einsetzen
 - > Eigene Views erstellen
 - > Fragen und Antworten

JavaBeans

- JavaBeans für das Netzwerk
- Beans “überall” 1.0
 - > bean context
 - > property sheet



Nodes

- Typisierte JavaBeans
 - > Kein Reflection
 - > Standard Listeners
 - > Erweiterungsfähig
- Unterstützen Hierarchien
- Brücke zu Beans über BeanNode

Präsentations Ebene

- Nodes sind *presentation layer*
- Nodes sind Hierarchisch
- Nodes enthalten beliebige Objekte und liefern benutzerfreundliche Fähigkeiten

Node Fähigkeiten

- extends `java.beans.FeatureDescriptor`
- Clipboard Operationen, D'n'D
- Actions, Customizer
- Help, Icon, HTML Beschriftung
- Persistence
- Properties
 - > für den Endanwender über die Property Sheets zu bearbeiten

DEMO

Wie wird ein Node erzeugt?

Nodes API

```
import org.openide.nodes.AbstractNode;  
import org.openide.nodes.Children;  
class MyNode extends AbstractNode {  
    public MyNode() {  
        super(new MyChildren());  
    }  
}  
class MyChildren extends Children.Keys<String> {  
    protected void addNotify() {  
        setKeys(Collections.nCopies(1, "Child"));  
    }  
    protected Node[] createNodes(String key) {  
        MyNode n = new MyNode();  
        n.setName(key);  
        return new Node[] { n };  
    }  
}
```

Children

- Nodes können Sub-Nodes haben
- Sub-Nodes werden in extra Container-Objekte verwaltet:
org.openide.nodes.Children
- Children ist Container und Node-Factory zugleich
- Verschiedene Children-Implementierungen decken alle möglichen Verhaltensweisen ab
 - > Children.Keys (extends Children.Array)
 - > Asynchrone Erzeugung per ChildFactory
 - > Children.SortedMap (extends Children.Map)
 - > Children.SortedArray (extends Children.Array)
- Nodes ohne Sub-Nodes müssen Children.LEAF als „Container“ zurückgeben

ChildFactory

- Das Ermitteln und Erzeugen von Daten für Sub-Nodes kann teilweise lange dauern
 - > Beispiele: Dateien auf einem entfernten Server, Versioning-History einer Datei, Datenzeilen einer aufwändigen Datenbankabfrage
- Children-Objekte müssen also manchmal asynchron erzeugt werden
- ChildFactory bietet eine Factory, die im Hintergrund (Nebenläufig) und Threadsafe die Daten (Keys) für die Nodes erzeugen kann
- Statische Methode `Children.create` (ChildFactory, asynchronous) erzeugt einen Children-Container mit einer ChildFactory)
- Achtung: Das Erstellen der Nodes selbst muss wieder schnell gehen, da das wieder im Event-Dispatcher Thread erfolgt (GUI Blocking!)
- Immer wann man `Children.Keys` verwenden kann, sollte man ChildFactory nutzen.

> So ist es einfach, sofort auf asynchron anzustellen, wenn es

DEMO

Einsatz der ChildFactory

Regeln

- Nodes arbeiten als Modell
 - > Nodes sind keine Daten
- Erzeuge Nodes so spät wie möglich
 - > ChildFactory + Children.create(ChildFactory, async)
 - > Das Erstellen wird in einen Hintergrund-Thread verschoben
 - > Children.addNotify → Children.setKeys → Children.createNodes (keys)
- Gehe sicher, dass Nodes aufgeräumt werden können (gc)
 - > Listener nicht vergessen
 - > Nutze removeNotify
- Wandle ein Node niemals in einen speziellen Typ
 - > Nutze Lookups, um Daten aus einem Node zu ermitteln

Nodes sind keine Daten

- Falsch

```
public class MyNode extends AbstractNode {  
    Person person;  
  
    public MyNode (Person person) {  
        super (Children.LEAF);  
        this.person = person;  
    }  
  
    public String getName() {  
        return person.getName();  
    }  
  
    public Date getBirth() {  
        return person.getBirth();  
    }  
}
```

Nodes sind keine Daten

- Richtig

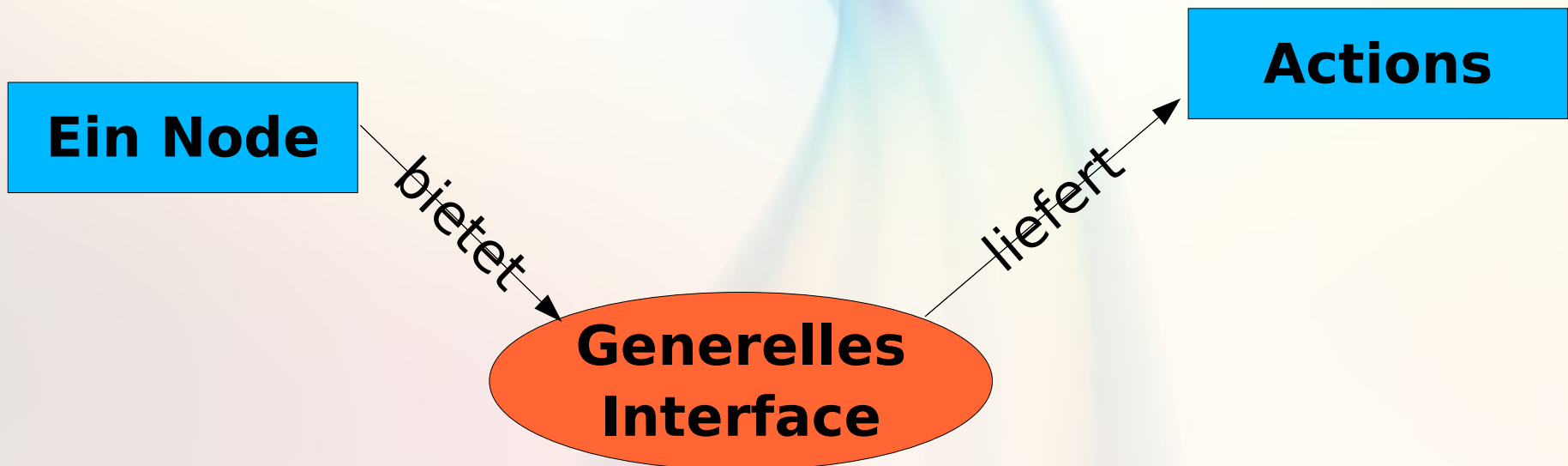
```
public class MyNode extends AbstractNode {  
    public MyNode (Person person) {  
        this (Children.LEAF, new InstanceContent());  
    }  
  
    protected MyNode (Person person, InstanceContent content) {  
        super (Children.LEAF, new AbstractLookup (content));  
        content.add (person);  
    }  
}
```

- Nun kann man das Objekt Person direkt aus dem Lookup ermitteln:
 - > Kein Casten notwendig, keine Verwendung der MyNode-Klasse, kein Wissen um die besondere Implementierung von MyNode. Keine Duplizierung von Methoden, die in den Datenobjekten schon programmiert sind. Es interessieren uns nur die Daten im Lookup:

```
Node node = ... // z.B. aus einer Benutzerauswahl.  
Person person = node.getLookup().lookup (Person.class);
```

Nodes Actions

- Nodes haben Actions
 - > Das fehlt JavaBeans
- Swing Actions
 - > `Action[] Node.getAction(boolean)`



Nodes Context

- Lookup `Node.getLookup()`
 - > Wird im Constructor (`AbstractNode`) übergeben
 - > Ersatz für das alte `getCookie(Class)`
 - Kein Marker-Interface
- `OpenCookie`, `EditorCookie`, usw...
 - > Setze die Implementation der Cookies in das Nodes Lookup
 - > Programmiere Actions, die auf das Objekt (über das Lookup) reagieren.
- Multiselection
 - > `ProxyLookup`

Nodes Actions

- Eine spezielle Implementation ist NodeAction
- Eine NodeAction darf nur einmal (statisch) erzeugt werden. Sie erhält den Kontext während des Aufrufes
- Zwar erhält eine NodeAction alle (selektierten) Nodes, die im notwendigen Kontext stehen, aber die Action muss trotzdem die verknüpften Objekte aus dem Lookup auslesen
 - > instanceof ist böse!
- NodeAction ist immer an den actionGlobalContext gebunden und reagiert nur auf Nodes, die per ExplorerManager selektiert werden

DEMO

Die NodeAction

Kontextsensitive Actions

- Actions, die im Kontext von Node-Daten aktiv sein sollen, reagieren nicht auf die Nodes, sondern auf die **Daten im Lookup**
- Context Actions sollten **keine Actions sein, die auf Nodes selbst reagieren.**
- Im Gegensatz zu einer NodeAction, muss der Kontext selbst ermittelt werden. Das kann wieder über den **actionsGlobalContext** geschehen oder auch durch eine Eigenprogrammierung, die andere Kriterien bietet.

Context Actions

- <http://wiki.netbeans.org/wiki/view/DevFaqActionContextSensitive>

```
public class FooAction extends AbstractAction
    implements LookupListener, ContextAwareAction {
    private Lookup context;
    Lookup.Result lkpInfo;
    public FooAction() {
        this(Utilities.actionsGlobalContext());
    }
    private FooAction(Lookup context) {
        this.context = context;
    }
    void init() {
        Lookup.Template tpl =
            new Lookup.Template(Whatever.class);
        lkpInfo = context.lookup (tpl);
        lkpInfo.addLookupListener (this);
        resultChanged (null);
    }
    public boolean isEnabled() {
        init();
        return super.isEnabled();
    }
}
```

```
public Action createContextAwareInstance
    (Lookup context)
{
    return new FooAction(context);
}
public void resultChanged(LookupEvent ev) {
    setEnabled (lkpInfo.allItems().size() != 0);
}
}
```

Dieses Beispiel macht fast nichts anderes als eine NodeAction. Der Kontext wird an den actionsGlobalContext „gebunden“.

Die Node Typen der Nodes-API

- **AbstractNode**
 - > Entgegen den Namen nicht wirklich abstract (obwohl man gezwungen ist, mindestens ein Constructor zu deklarieren).
 - > Die Basisklasse aller Nodes
- **BeanNode**
 - > Ein praktische Node-Klasse, die per Reflection POJO-Eigenschaften durch reicht
- **FilterNode**
 - > Repräsentiert ein anderes Node als Proxy. Damit kann man von Nodes Eigenschaften übernehmen, aber durch eigene Actions und Attribute ergänzen. Dies wird häufig mit DataNodes gemacht
- **DataNode**
 - > Repräsentiert eine Datei mit spezifischem Mime-Type.

DEMO

Die Node-Typen der API

Fragen & Antworten

<http://bits.netbeans.org/dev/javadoc/org-openide-nodes/>

Teil II - Explorer Views

Jaroslav Tulach, Sun Microsystems
Ergänzungen: David Strupl, Sun Microsystems
Geertjan Wielenga, Sun Microsystems

Deutsche Überarbeitung und Ergänzungen - Aljoscha Rittner
Sepix GmbH
NetBeans Dream Team

Agenda Teil II

- Explorer Views
 - > Der Swing-Weg
 - > Von NetBeans gelieferte Views
 - > Der Explorer Manager
 - > Property Sheet
 - > Views einsetzen
 - > Eigene Views erstellen
 - > Fragen und Antworten

Der Swing-Weg

- Unterschiedliche Modelle hinter den Swingkomponenten
 - > TreeModel
 - > ListModel
 - > ComboBoxModel
- Unterschiedliche Renderer zur visuellen Darstellung
 - > ListCellRenderer
 - > TableCellRenderer

Der Swing-Weg (JList)

```
public class JListDemo extends JFrame {
    public static void main(String args[]) {
        final JList myList = new JList();
        myList.setModel
        (new AbstractListModel() {
            String[] strings = {
                "Tom", "Dick", "Harry "
            };
            @Override
            public int getSize() {
                return strings.length;
            }
            @Override
            public Object getElementAt(int i)
            {
                return strings[i];
            }
        });
    }
}
```

```
EventQueue.invokeLater(new Runnable() {
    public void run() {
        JListDemo demo = new JListDemo();
        demo.setDefaultCloseOperation(
            WindowConstants.EXIT_ON_CLOSE);
        demo.setTitle("JList Demo");
        demo.setSize(new Dimension(200, 200));
        demo.add(myList);
        demo.setVisible(true);
    }
});
```

Der Swing-Weg (JTree)

```
public class JTreeDemo extends JFrame {
```

```
    public static void main(String args[]) {
```

```
        final JTree jTree1 = new JTree();  
        DefaultMutableTreeNode treeNode1  
            = new DefaultMutableTreeNode("Names");  
        DefaultMutableTreeNode treeNode2  
            = new DefaultMutableTreeNode("Tom");  
        treeNode1.add(treeNode2);  
        treeNode2  
            = new DefaultMutableTreeNode("Dick");  
        treeNode1.add(treeNode2);  
        treeNode2  
            = new DefaultMutableTreeNode("Harry");  
        treeNode1.add(treeNode2);  
        jTree1.setModel(  
            new DefaultTreeModel(treeNode1));
```

```
        java.awt.EventQueue.invokeLater(new Runnable() {  
            public void run() {  
                JTreeDemo list = new JTreeDemo();  
                list.setDefaultCloseOperation(  
                    WindowConstants.EXIT_ON_CLOSE);  
                list.setTitle("JTree Demo");  
                list.setSize(new Dimension(200, 200));  
                list.add(jTree1);  
                list.setVisible(true);  
            }  
        });  
    }
```

Explorer Views

- Viele unterschiedliche Swing-Komponenten zur Darstellung
 - > BeanTreeView
 - > ContextTreeView + ListView
 - > IconView, MenuView
 - > ChoiceView
 - > OutlineView (ehemals TableTreeView, ListTableView, TableView)
 - > PropertySheet
- Die Nodes bieten eine universelle Möglichkeit Daten hierarchisch zu strukturieren.

BeanTreeView

- Zeigt Nodes als Baumstruktur
- Es werden auch Nodes ohne Children angezeigt (Leaf-Nodes)
- Projekt-, Datei- und Favoriten-Ansicht in der NetBeans IDE verwenden das BeanTreeView

ContextTreeView

- Zeigt Nodes als Baumstruktur
- Es werden keine Leaf-Nodes angezeigt
- Wird häufig im Zusammenhang mit ListView verwendet
 - > Dabei werden im ListView die Leaf-Nodes aus dem Explored Context des ContextTreeView angezeigt
- ContextTreeView mit ListView wird gerne in Dateidialogen verwendet

ListView

- Zeigt die Nodes einer Ebene als Liste an
- Es wird keine Hierarchie dargestellt
- Wird auch als Detailedarstellung des ContextTreeView verwendet
- Wird häufig in Dateidialogen oder Assistenten verwendet

IconView

- Zeigt die Nodes einer Ebene als Symbole an
- Es wird keine Hierarchie verwendet
- Wird auch als Symbol-Detaildarstellung des ContextTreeView verwendet
- Wird seltener verwendet, ggf. in Dateidialogen oder Assistenten

MenuView

- Zeigt eine Button an, der bei Klick die Nodes-Hierarchie als Menü ausbaut und darstellt
- Wird recht selten verwendet

ChoiceView

- Bietet ein Node in einer Auswahlbox (ComboBox) an
- Sub-Nodes werden im Popup der ComboBox angezeigt
 - > Es wird keine Hierarchie verwendet.
- Wird in der IDE selten verwendet.

OutlineView

- Zeigt eine Nodehierarchie in struktureller und tabellarischer Form an
- Erste Spalte zeigt die Hierarchie
- Weitere Spalten zeigen Eigenschaften der Nodes an
- In der IDE ersetzt das OutlineView immer mehr das TreeTableView (welches sehr buggy ist)

DEMO

Vorstellung der Standard-Views

Besonderheiten der Views

- Fast alle Views sind JScrollPane mit fertig eingerichteter View-Komponente
 - > Ausgenommen ChoiceView
- Den Views kann man kein Modell explizit zuweisen.
 - > Die Views suchen sich selbst den ExplorerManager in der Swing-Komponenten-Hierarchie
- Die Views verwenden Node-Wrapper.
 - > Aus Optimierungsgründen!

D.h. die Views haben wieder ein eigenes Objekt-Modell und verwenden nicht direkt die Nodes des ExplorerManager. Das ist (nur) wichtig, wenn man eigene Renderer schreibt.

DEMO

Explorer Views in die GUI Builder Palette einfügen

Explorer Manager

- ExplorerManager (der Modell-Controller)
 - > Root Context
 - > Bietet damit ein View (einen Teilast) auf eine Node Hierarchie
 - > Ausgewählte Nodes (vetoable)
 - > Erlaubt grundsätzlich Multiselection von Nodes (View übergreifend, ist Multiselection über den actionGlobalContext möglich)
 - > Explored Context
 - > Beschreibt den Elternknoten der sichtbaren Nodes. Wird üblicherweise von ListView, IconView und ContextTreeView verwendet. Wird sonst nicht häufig verwendet.

Root Container

```
class MyPanel extends JPanel implements  
    ExplorerManager.Provider {  
private myManager;  
public MyPanel() {  
    myManager = new ExplorerManager ();  
    add (new BeanTreeView ());  
    add (new PropertySheetView ());  
    myManager.setRootContext (myNode);  
}  
public ExplorerManager getExplorerManager () {  
    return myManager;  
}
```

DEMO

Explorer Views anwenden

Property Sheet

```
protected Sheet createSheet() {
    Sheet sheet = super.createSheet();
    Sheet.Set props = sheet.get(Sheet.PROPERTIES);
    if (props == null) {
        props = Sheet.createPropertiesSet(); sheet.put(props);
    }
    props.put(new PropertySupport.ReadWrite("name", String.class, "Nice Name", "Short desc") {
        String val = "";
        public Object getValue() throws IllegalAccessException, InvocationTargetException {
            return val;
        }
        public void setValue(Object val)
            throws IllegalAccessException, IllegalArgumentException, InvocationTargetException
        {
            String old = this.val; this.val = val.toString();
            firePropertyChange("name", old, val);
        }
    });
    return sheet;
}
```

DEMO

Property Sheet für Nodes anlegen

Schreibe Dein eigenes Node-View

- Es ist nur eine Swing-Komponente
- Überschreibe `addNotify` und `removeNotify` (plain Swing!)
 - > Suche Eltern für `ExplorerManager.Provider`
 - > Füge Listener hinzu
 - > Reagiere auf den `ExplorerManager`
- Kontrolliere den `ExplorerManager`
 - > Verwende die Setter-Methoden
 - > Verwende `Vetoable` Listeners

Erweitere bestehende ListViews

- Nur für ListViews
 - > (+ IconView)
- Überschreibe createList
 - > Verwende die Swingkomponente aus der super-Klasse (so bleibt Quicksearch erhalten)
 - > Verändere das Verhalten, durch Aufruf öffentlicher Methoden
 - > Setze so Deinen eigenen Renderer
- Konvertiere im Renderer das Objekt in Dein Node und zurück
 - > Siehe Visualizer-Klasse!

Explorer Views ausreizen

- Checkboxes in Node Visualizers?
 - > Node Lookup muss `org.openide.explorer.view.CheckableNode` zurückgeben
- Eigene Renderer schreiben?
 - > `org.openide.explorer.view.Visualizer` für das Konvertieren der Visualizer-Nodes verwenden

DEMO

Ein ListView erweitern

Nodes und Auswahlverhalten

- Jede Fensterkomponente (TopComponent) hat ein Lookup
- Nodes haben Lookups
- Mit einem Proxy ist es einfach das TopComponent-Lookup mit den Lookup der Nodes zu verknüpfen
 - > `TopComponent.associateLookup`
- Über den ExplorerManager wird die Auswahl (Selection-Management) der Nodes gesteuert.
 - > `TopComponent implements ExplorerManager.Provider`

Zusammenfassung

- Nodes sind typisierte JavaBeans
- Hierarchie wird unterstützt
- Erweiterungsfähig
- Eine reiche Auswahl von Views
- Ein Modell für alle Views
- API ist (auch) unabhängig von NetBeans, sowohl die Nodes-API als auch die Lookup-API

DEMO

- Tutorial Builder – Gallery

Fragen & Antworten

- <http://bits.netbeans.org/dev/javadoc/org-openide-nodes/>
- <http://bits.netbeans.org/dev/javadoc/org-openide-explorer/>