Elexis-Scripting

Gerry Weirich

21. Juni 2008

1 Einführung

Scripting ist eine sehr mächtige Möglichkeit, die Funktionalität von Elexis selbst zu erweitern, ohne in den Programmcode eingreifen zu müssen. Ein Script ist eine Art Miniprogramm, das innerhalb von Elexis ausgeführt wird, und das Zugriff auf alle von Elexis verwalteten Daten hat.

Scripts können an verschiedenen Stellen eingesetzt werden. Beispielsweise bei der Berechnung von formelbasierten Laborwerten oder Befunden, oder bei der Filterung der Patientenliste, oder auch als selbständige Funktionen.

2 Sprache und Syntax

Die zu verwendende Sprache ist ein vereinfachtes Java (aber auch 'richtiges' Java geht). Im Genensatz zu einem 'echten' Java-Programm brauchen Sie keinen Compiler. Stattdessen wird das Script Interpretiert, also in dem Moment analysiert und in Computerbefehle umgesetzt, in dem es ausgeführt wird. Ein genauere Erläuterung des Interpreters finden Sie bei dessen Hersteller Beanshell (http://www.beanshell.org). Hier nur eine ganz kurze Einführung:

2.1 Grundsätzlicher Aufbau

```
/* Dies ist ein Beispiel-Script. Text, welcher zwischen diesen Kommentarsymbolen
   steht, wird als Kommentar betrachtet, dient also nur der Erhellung oder
   Erheiterung des menschlichen Lesers. Der Interpreter hingegen ignoriert
   solchen Text schlicht. Sie sollten bei komplexeren Scripts nicht mit
   Kommentaren sparen, da sie Ihnen helfen, später zu verstehen, was Sie
   eigentlich machen wollten. Und anderen kann es helfen, Ihr Script zu
   verstehen.
*/

// Auch dies ist ein Kommentar, aber bei // geht der Kommentar immer nur
// bis zum Ende der aktuellen Zeile, bei /* immer bis zum nächsten */
```

```
irgendwas=10;
    /* Wir haben soeben eine Variable namens 'irgendwas' erstellt,
       und ihr den Wert 10 zugewiesen. Beachten Sie bitte auch, dass
       am Ende jedes Statements ein ; steht, damit der Computer weiss,
       dass das Statement hier fertig ist. */

heute="Mittwoch";
    /* Hier haben wir eine Variable namens 'heute' erstellt und ihr
       den Wert 'Mittwoch' zugewiesen. Beachten Sie, dass Zeichenketten
       immer in Anführungszeichen stehen müssen. */

grosseZahl=1000;    // Ich denke, das ist klar.

zusammen=irgendwas+grosseZahl;
    // 'zusammen' sollte jetzt 1010 sein, wenn Ihr Computer nicht kaputt ist

    // wir wollen das gleich mal testen:
if(zusammen==1010){
            // Geschweifte Klammern markieren zusammengehörende Blöcke
    resultat="Computer rechnet richtig";
    wertDesComputers=1000;
}else{
    resultat="Computer rechnet falsch";
    wertDesComputers=0;
}

    // Ihr Computer kann natürlich auch multiplizieren und dividieren
produkt=irgendwas*grosseZahl*3.4;
quotient=grosseZahl/(irgendwas-2);

    // Einfaches Rechnen mit Variablen
irgendwas=irgendwas+5;     // irgendwas ist jetzt 15
irgendwas+=3;               // irgendwas ist jetzt 18
irgendwas++;                // irgendwas ist jetzt 19

heute=heute+" 18. Juni";    // 'heute' ist jetzt "Mittwoch 18. Juni"
```

2.2 Objekte

Über objektorientierte Sprachen sind schon tausende von Büchern geschrieben worden. Ich versuche mal, das Wesentliche auf einer oder zwei Seiten zusammenzufassen:

Ein Objekt ist in diesem Zusammenhang ein Programm-Konstrukt, das bestimmte Eigenschaften und Fähigkeiten (sog. 'Methoden') hat, die in seiner 'Klasse' festgelegt werden. Beispiel:

```
/* Dieses Script erstellt und benutzt ein Auto */

 import de.volkswagen.autos.*;  // importiere die benötigten Klassen

 meinAuto=new Kaefer();         // erstelle ein Objekt der gewünschten Klasse
 meinAuto.setFarbe("Grün");     // setze Eigenschaften des Objekts
 if(meinAuto.istKaputt()){      // frage eine Eigenschaft des Objekts ab
     // wir bleiben zuhause
 }else{
     meinAuto.fahreNach("Bern"); // führe eine Methode des Objekts aus.
 }
```

Beachten Sie in obigem Beispiel, dass uns die genaue Implementation des 'Kaefers' nicht zu interessieren braucht. Wir müssen weder wissen, wie er hergestellt wird, noch woraus er besteht. Wir müssen nur wissen, welche seiner Eigenschaften und Methoden für uns interessant sind. Das macht Objekte so nützlich in der Programmierung. Man muss nicht alles selber machen, sondern kann die Vorarbeiten anderer (oder eigene frühere Vorarbeiten) nutzen.

Unterschied zu streng typisierten Programmiersprachen
'Echte' Java-, Pascal- C++ oder C# - Programmierer werden sich wahrscheinlich wundern, wieso da nicht steht:

```
Kaefer meinAuto = new Kaefer();
```

Dies ist, weil Beanshell untypisierte Variablen verwenden kann. Der Typ von 'meinAuto' wird erst zur Laufzeit festgelegt. Dies hat eine Reihe von Vorteilen (und eine Reihe von Nachteilen), auf die wir hier nicht im Einzelnen eingehen wollen. Wenn Sie diese Anarchie stört, dürfen Sie aber auch in Beanshell die streng typisierte Form verwenden. Standardmässig dürfen Sie meinAuto aber einen 'new Kaefer()' oder einen 'new Ferrari()', oder auch einfach die Zahl '10' zuweisen.

Eine Klasse ist also die Definition, ein Objekt ist eine konkrete Inkarnation einer Klasse. Wir nennen den Vorgang, ein Objekt einer bestimmten Klasse zu erstellen, auch 'instanzieren' oder instantiieren. Der dazu zu verwendende Programmbefehl heisst 'new' und das resultieende Objekt wird auch 'Instanz' der Klasse genannt. In obigem Beispiel ist meinAuto also eine Instanz der Klasse Kaefer.

3 Verbindung mit Elexis

Ein Script kann Verbindung mit Elexis aufnehmen, indem es in Elexis definierte Java-Klassen verwendet. Um dies auszunützen, braucht man allerdings gewisse Kennntnisse von Elexis-Interna.

Es ist daher wohl sinnvoll zunächst einfach die in Beispielen gezeigten Objekte zu verwenden. Auf diese Weise stellt sich das nötige Kennenlernen der sinnvoll verwendbaren Objekte mit der Zeit von allein ein. Oder man lässt sich benötigte Scripts von jemand anderem erstellen.

Hier ein einfaches Beispiel:

```
import ch.elexis.scripting.*;
Util.display("Hallo","Ich wünsche Ihnen einen guten Tag");
```

Dieses Script tut nichts wirklich Spektakuläres: In der ersten Zeile importiert es aus Elexis eine Sammlung von Hilfsklassen für Scripte und in der zweiten Zeile nutzt es eine dieser Hilfsklassen, nämlich die Klasse 'Util', indem es die Methode 'display' dieser Klasse aufruft. Als Ergebnis wird eine Mitteilungsbox mit dem genannten Text angezeigt. Der aufmerksame Leser hat vielleicht gemerkt, dass die Klasse Util hier gar nicht instanziert wurde - es steht nirgends 'new Util()'. Die Methode Util.display() ist eine 'statische Methode', welche der Einfachheit halber direkt über die Klasse referenziert werden kann. Wir hätten aber natürlich auch schreiben können:

```
import ch.elexis.scripting.*;
dingsda=new Util();
dingsda.display("Hallo","Ich wünsche Ihnen einen guten Tag");
```

3.1 Rechte

Da Scripte potentiell gefährliche Dinge tun können (sie könne ja auf alle Daten von Elexis zugreifen), ist ihre Erstellung und Verwendung an Rechte geknüpft. Um ein Script erstellen und/oder bearbeiten zu können, brauchen Sie das Recht Script-bearbeiten. Um ein Script auszuführen, benötigen Sie das Recht Script-ausführen. Diese Rechte erteilen Sie wie gewohnt unter Datei-Einstellungen-Gruppen und Rechte.

3.2 Script erstellen und Bearbeiten

Hierzu dient die View 'Script', die Sie wie gewohnt öffnen können. Klicken Sie auf den roten Stern, um ein neues Script zu erstellen. Sie werden zunächst nach einem Namen gefragt. Der Name darf nur aus den Buchstaben a-z und den Ziffern 0-9, sowie dem Unterstrich _ und dem Bindestrich - bestehen. Wenn Sie OK klicken, wird ein noch leeres Script mit diesem Namen erstellt. Um es zu editieren, klicken Sie es mit der rechten Maustaste an und wählen Sie 'Script bearbeiten'.

3.3 Script ausführen

Dies hängt vom Kontext und der Art des Scripts ab. Manche Scripte können Sie direkt in der Sctip-View ausführen: Klicken sie es mit der rechten Maustaste an und wählen Sie 'Script ausführen'. Scripte, die als Filter in der Patientenliste dienen sollen, ziehen Sie einfach per Drag&Drop in die Filterbox der PatientenListe-View.

Hier ein Beispiel für ein Script für den Patientenliste-Filter:

```
/* Dieses Script zählt aus den von vorangestellten Filtern
    übriggebliebenen Patienten alle Männer und Frauen und
    berechnet das Durchschnittsalter und den Median
*/

import ch.elexis.scripting.*; //  Hilfsklassen importieren

/* Vor dem ersten Durchlauf wird init aufgerufen */

    if(init){
      counter=new Counter();
      maenner=0;
      frauen=0;

/* nach dem letzten Durchlauf wird finished aufgerufen */
    }else if(finished){
        // Ergebnis auf 2 Stellen runden
      float sc=counter.getAverage(2);

      Util.display("Ausgewählte Patienten",
        Integer.toString(maenner)+" Männer, "+
        Integer.toString(frauen)+" Frauen, "+
        "Durchschnittsalter: "+Float.toString(sc)+
        ", Median: "+counter.getMedian()
     );

/* Dieser Block wird für jeden Patienten aufgerufen */
   }else{
      if([Patient.Geschlecht].equalsIgnoreCase("m")){
        maenner++;
      }else{
        frauen++;
      }
      int jahre=Integer.parseInt([Patient.Alter]);
      counter.add(jahre);

   }

return 0; // Wir wollen nur zählen, nicht filtern
```

Dieses Script nutzt die externe Klasse 'Counter' zum Berechnen des Duchschnittsalters und des Medians und die Klasse Util zur Anzeige des Resultats. Beide Hilfsklassen sind im package ch.elexis.scripting zu finden, das darum ganz am Anfang importiert wird. Der * am Ende bedeutet einfach: Alle Klassen dieses Package importieren.

Scripts im Patientliste-Filter werden immer ein erstes Mal mit dem gesetzten Parameter 'init' aufgerufen, bevor der Durchlauf startet. Sie können dann Initialisierungen vornehmen. Dann weden sie für jeden Patienten, der die weiter oben stehenden Filter passiert hat, genau einmal aufgerufen. Sie können den Patienten auswerten, indem sie entweder eine [Patient.xxx] Variable verwenden, oder das implizit übergebene Objekt 'patient' (die aktuelle Instanz der Klasse Patient) verwenden. Nach Abschluss des Filtervorgangs wird das Script nochmal mit dem Parameter 'finished' aufgerunfe, dann kann es Aufräumarbeiten erledigen, ein Resultat verarbeiten und anzeigen usw. Ein Script braucht die init und finished-Teile nicht zu verwenden, wenn es sie nicht benötigt.

Das Script sollte aber immer ein Resultat zurückliefern. 1 bedeutet: Diesen Patienten in die gefilterte Liste aufnehmen, -1 bedeutet: Diesen Patienten nicht in die Liste aufnehmen. 0 bedeutet: Dieses Script trifft keine Filterentscheidung. -2 bedeutet: Es ist ein schwerwiegender Fehler passiert, der Durchlauf muss abgebrochen werden. (Dies bewirkt, dass das Script aus dem Filter hinausgeworfen wird).