Fachhochschule
Münster University of
Applied Sciences

Java Scripting

Java Erweiterungen mittels *Groovy-* oder *JavaScript* Sprachen

Prof. Dr. Nikolaus Wulff

Vorüberlegungen

Müssen Anwendungen dynamisch um Funktionalität durch den Benutzer erweitert werden empfiehlt sich der Einsatz einer Scriptsprache, wie z.B. VB in MS-Office.

- Seit dem JDK1.6 gibt es das Paket ***javax.script***.
- Es enthält die wichtigen Abstraktionen.
  - ***ScriptEngine*** und ***ScriptEngineFactory***.
  - ***ScriptContext*** und ***Binding***.
  - Sowie ***CompiledScript*** und ***Invocable***.
- Werden gegen diese API Interpreter entwickelt, lassen sich eigene Scripte einfach in Java einbinden.

Java Script API

Package javax.script

The scripting API consists of interfaces and classes that define Java ™ Scripting Engines and provides a framework for their use in Java applications.

See: Description

Interface Summary

Interface	Description
Bindings	A mapping of key/value pairs, all of whose keys are `Strings`.
Compilable	The optional interface implemented by ScriptEngines whose methods compile scripts to a form that can be executed repeatedly without recompilation.
Invocable	The optional interface implemented by ScriptEngines whose methods allow the invocation of procedures in scripts that have previously been executed.
ScriptContext	The interface whose implementing classes are used to connect Script Engines with objects, such as scoped Bindings, in hosting applications.
ScriptEngine	`ScriptEngine` is the fundamental interface whose methods must be fully functional in every implementation of this specification.
ScriptEngineFactory	`ScriptEngineFactory` is used to describe and instantiate `ScriptEngines`.

Class Summary

Class	Description
AbstractScriptEngine	Provides a standard implementation for several of the variants of the `eval` method.
CompiledScript	Extended by classes that store results of compilations.
ScriptEngineManager	The `ScriptEngineManager` implements a discovery and instantiation mechanism for `ScriptEngine` classes and also maintains a collection of key/value pairs storing state shared by all engines created by the Manager.
SimpleBindings	A simple implementation of Bindings backed by a `HashMap` or some other specified `Map`.
SimpleScriptContext	Simple implementation of ScriptContext.

JavaScript

- JavaScript ist direkt aus Java heraus aufrufbar:

```
public static void main(String[] args)
        throws ScriptException {
    ScriptEngineManager manager = new ScriptEngineManager();
    ScriptEngine js =  manager.getEngineByName("JavaScript");
    assert js != null : "no java script found";

    String script = "x=2;  y = 1/x; z = y*y; z";
    System.out.printf("%s = %s\n",script,js.eval(script));
}
```

```
Ausgabe:

x=2;  y = 1/x; z = y*y; z = 0.25
```

- Einfacher geht es nicht. Aber auch der Einsatz von anderen Sprachen ist möglich und lohnenswert.

Groovy Scripts

- Auch Groovy bietet eine ScriptEngine an, welche die *javax.script* API implementiert.

```
public static void main(String[] args) throws ScriptException {
    ScriptEngineManager manager = new ScriptEngineManager();
    List<ScriptEngineFactory> factories = manager.getEngineFactories();
    for(ScriptEngineFactory factory: factories) {
        List<String> ext  = factory.getExtensions();
        System.out.printf("%s: %s  %s \n",
                factory.getLanguageName(),
                factory.getEngineName(), ext);
    }
    ScriptEngine groovy =  manager.getEngineByName("groovy");
    assert groovy != null : "no groovy script found";

    String script = "x=2;  y = 1/x; z = y*y; z";
    System.out.printf("%s = %s\n",script,groovy.eval(script));
}
```

```
Ausgabe

Groovy:   Groovy Scripting Engine  [groovy]
ECMAScript:       Mozilla Rhino  [js]
x=2;  y = 1/x; z = y*y; z = 0.25
```

Java aus JavaScript

- Aber auch Java ist von JavaScript heraus aufrufbar:

```
static String   definition = "var d = new java.util.Date(); "
                           + "var s = d.toString(); "
                           + "print(s); ";

public static void scriptTest() throws Exception {
    engine.eval(definition);
}
```

```
Ausgabe: Tue Jan 13 14:11:23 CET 2015
```

- Java Pakete und Klassen werden per *importPackage* und *importClass* in JavaScript eingebunden.
- Achtung zwischen dem JDK 1.7 und 1.8 gibt es einige Syntaxunterschiede, da die JavaScript Engine in V1.8 (Rhino ↔ Nashorn) neu implementiert wurde.

Script Funktionen

- Innerhalb von Scripts lassen sich Funktionen definieren und aufrufen:

GroovyScript:

```
String script = "def f(n) { n <= 1 ? 1 : n * f(n - 1) }";
groovy.eval(script);
Invocable exe =  (Invocable) groovy;
Object[] parms = {new Integer(4)};
System.out.printf("%s = %s\n",script, exe.invokeFunction("f",parms));
```

JavaScript:

```
String script = "function f(n) { return n <= 1 ? 1 : n * f(n - 1); }";
js.eval(script);
Invocable exe = (Invocable) js;
Object[] parms = {"4"};
System.out.printf("%s = %s\n",script, exe.invokeFunction("f",parms));
```

- Die genaue Syntax hängt von der Scriptsprache ab...

Eigne ScriptEngines

- Eigene Script-Sprachen sind einfach einzubinden, wenn sie ebenfalls gegen die Script API entwickelt werden:

```
class JavaASTParser implements ScriptEngine
```

- Dies kann auch die Native C Wrapper Klasse sein...
- Scripting ist mit dieser API ein Selbstläufer.
- Der Aufwand lohnt sich allerdings nur, wenn Funktionalität benötigt wird, die durch JavaScript oder Groovy nicht oder inperformant geliefert wird.
  - Z.B. Scripting für Matrizen und Vektoroperationen oder eine eigene fachliche Domäne.

Was bleibt übrig zu tun?

- Groovy und JavaScript „verstehen“ mathematische Ausdrücke:
  - Punkt- vor Strichrechnung
  - Klammerrechnung Assoziativ und Distributivgesetze
  - etc. werden automatisch richtig behandelt.
- Mathematische Funktionen, wie sin, tan, etc sind nur als Math.sin bekannt und der Ausdruck `x*sin(x)` muss entsprechend erweitert werden.
- Auch selbst definierte Funktionen und Klassen müssen entsprechend angepasst werden.

EBNF Codegenerierung

- Mit Hilfe von EBNF Grammatiken lassen sich sehr effektiv eigene Skriptsprachen entwickeln.
- Java Werkzeuge hierzu sind **AntLR** und **JavaCC**, die angelehnt sind an **flex** und **yacc** der Unix Welt.
- Voraussetzung für die Codegenerierung ist eine durchdachte EBNF Grammatik.
- Diese Generatoren generieren passend zur Grammatik einen *Lexer*, einen *Parser* und einen *TreeWalker*.
- Der Walker travestiert den generierten AST Baum und gestattet es per Visitor-Muster die Knoten zu besuchen und rekursiv auszuwerten.

AntLR Beispiel

- Anbei ein Auszug aus der AntLR Generierung des TreeWalkers zum Ausdruck

```
expr = term ( ('+'|'-') term)*:
```

```
/**
 * expr = term ( (+|-) term )*
 * enhanced with return value.
 */
expr returns [double ret]
@init { $ret = 0.0;}: (
    x=term {$ret=x.ret;}
     (
      ('+'^ y=term {$ret+=y.ret;})
    | ('-'^ y=term {$ret-=y.ret;})
     )*
   );
```

JavaCC Beispiel

- Ein analoger EBNF Ausdruck implementiert für den JavaCC JJTree Preprocessor.

```
void addition() #void : {} {
    multiplication()
  ( LOOKAHEAD(3)
    (
     ("+" multiplication() #Plus(2))
    |("-" multiplication() #Minus(2))
    )
  )*
}
```

- JJTree generiert daraus eine Grammatik für JavaCC, der lauffähigen Java Source-Code erzeugt.

TreeVisitor

- Das geparste Script wird als ein Baum abgebildet und die Auswertung erfolgt durch Besuch der Knoten:

```
public double visit(final SimpleNode node, final double... data)
                                              throws ScriptException {
  double x, y, ret = data[0];
  int id = node.id;
  String key, name;
  SimpleNode left, right;

  switch (id) {

    case JJTPLUS: {
        x = valueOf(node, 0, data);
        y = valueOf(node, 1, data);
        ret = x + y;
        //logger.info(format("Plus %f + %f = %f \n",x,y, ret));
    }
    break;
```

Eigene Script Sprache

Die Entwicklung eigener Script Sprachen erfordert etwas Aufwand, kann aber sehr lohnend sein.

Beispiel: Ein Script für beliebige Funktionsdefinitionen passend zur *de.lab4inf.math.Function* Schnittstelle:

```
public static void simpleScript() throws ScriptException {
    ScriptEngineManager manager = new ScriptEngineManager();
    ScriptEngine engine = manager.getEngineByExtension("l4m");
    double x,y;
    String def = " poly(x) = 0.1*x**4 - 2*x**2 + 1;";
    Script script = (Script) engine.eval(def);
    Function fct  = script.getFunction("poly");
    for(x=0;x<=2.0;x+=0.25) {
        y = fct.f(x);
        printf("f(%.2f) = %+.3f \n",x,y);
    }
}
```

Selbst definierter Potenzoperator: $x^n \equiv$ x**n

JavaCC Power

- Definition neuer Operatoren und Funktionen:

```
void power() #void: {} {
  unary()
  (
    (
      ("**" unary() #Pow(2) )
    | ("^"  unary() #Pow(2) )
    )
  )*
}

void unary() #Unary(1) : {} {
  (jjtThis.value="-")?
 (
    LOOKAHEAD(2) function()
   | ( "(" expression() ")" #Bracket(1) )
   | identifier()
   | number()
 )
}
```

JUnit ScriptTest

- Eine Funktion *f(x,y,...)* lässt sich leicht innerhalb eine eigenen Script Sprache implementieren und testen.

```
public class ScriptFunctionTest extends TestCase {
    String name,args,def;
    Function fct;
    public ScriptFunctionTest(String name) {
        super(name);
    }
    protected void setUp() throws Exception {
        super.setUp();
        name = "harmony"; args = "x,y"; def = "x*y/(x+y)";
        fct = new ScriptFunction(name, args, def);
    }
    public void testF() {
        for(double x=0;x<4;x+=0.25) {
            for(double y=0;y<x; y+=0.125) {
                assertEquals(x*y/(x+y), fct.f(x,y), 1.E-8);
            }
        }
    }
}
```

Zusammenfassung

- Mit Scriting ist es sehr leicht möglich eine Anwendung um dynamische Inhalte zu erweitern.
- Das neue *javax.scripting* Paket bietet hierfür eine gute und erweiterbare Schnittstelle.
- Falls JavaScript nicht ausreicht ist Groovy eine gute Option, wenn das nicht ausreicht stehen mit Werkzeugen wie AntLR oder JavaCC alle Mittel offen.
- Wichtig ist es den Anwendungsfall vorher genau zu analysieren und für die Codegenerierung eine ausdrucksstarke EBNF zu spezifizieren.