Jython: integrating Python and Java

COFFEE AND SNAKE

Implementing the Python interpreter in Java instead of C brings advantages for both languages. Andreas Jung shows us just some of the possibilities that Jython offers

As programming languages go, it may appear that Python and Java have nothing in common. Take a closer look and you'll find that's not so. Both languages are object-oriented and organise their extensions in modules or libraries, which they import at runtime. For this reason alone it makes sense to combine the advantages of both languages. A Python interpreter makes it very easy to provide Java applications with extensive scripting facilities and on the other side of the coin, the odd Java class can be very useful in Python programs.

Consequently, a Java implementation of the Python scripting language has been developed over the last few years. It's equivalent to the C-Python version maintained by Guido van Rossum; the aim of the developers being to provide the same functionality as C-Python in the Java version. Initially the project was named Jpython; nowadays it is now simply as Jython.

Table 1: Jython options

Option	Description
-i	Starts the Jython interpreter in interactive mode as soon as a script that has also been specified is finished
-jar jar-file	Starts the program "_run_.py" in the jar archive
-c cmd	Starts the program passed as argument "*cmd*"
file	Starts the file "*file*" as a program
-	Reads the program from standard input
args	Jython will pass all subsequent arguments to the program that is to be executed

Table 2: Method parameters

Java data type	permitted Python data type
char	string (length = 1)
boolean	int (true != 0)
byte, short, int, long	int
float, double	float
java.lang.String, byte[], char[]	string

Graphical installer

Jython is currently in version 2.1a3 and this can be downloaded from *http://www.jython.com*. The version numbers always follow the official Python releases, so that the features and functionality of the Java variant should always mirror those of the C version.

Due to its very nature, Jython requires the installation of a Java environment, such as the JDK from Blackdown or IBM under Linux. Under Windows the JDK from Sun or Microsoft's VM (Virtal Machine), also known as Jview, are the most suitable alternatives.

To install Jython you'll need the file jython-21a3.class; the link for downloading this can be found on the Jython Web page. You can start the installation process with the following command:

```
java -cp . jython-21a3
```

A wizard will then lead you through the installation process. Once it is complete Jython can be started in interactive mode using the command *jython*. The familiar Python shell appears:

```
>> a=2; b=4
>> print a,b
2 4
>> print a+b
6
>> import sys
>> print sys.path
['', '/home/ajung/.', '/opt/jython-2.1/Lib']
```

The Jython program you have called is only a shell wrapper that starts the JDK with the appropriate classes. Table 1 shows the most important options and arguments when calling Jython.

Importing Java classes

Up to this point everything looks fairly familiar, so what's the point? Well, to take one example, Jython can import Java classes directly. Classes in Java are arranged hierarchically into packages. An *import*

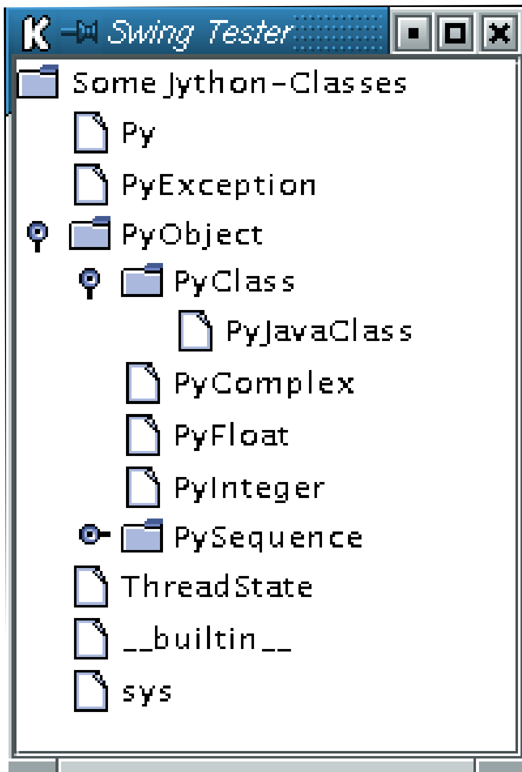

Figure 1: Some Jython classes displayed with the Tree widget from the Java package "Swing". The program creating this tree is a Python script

command enables the current class to also use classes from other packages without having to specify the package name every time:

```
import java.util.*
import java.awt.Frame
```

A similar mechanism exists in Python, which also uses a hierarchy of packages. Jython, too, is able to access Java classes using an *import* command:

```
from java.util import *
import java.awt.Frame
```

Method calls

Things get a bit more complicated when calling the methods of Java classes. Although this process is identical in both Jython and Java, the two languages' different data types make a conversion from Python to Java necessary. The same is true for the reverse process of returning results to Python. In Jython this type conversion, also called "coercing", happens automatically. Table 2 gives an overview of which Python data types are permitted for which Java data types; Table 3 relates to the representation of Java in Python when returning results.

A simple example

Jython programs can use any Java classes. Listing 1 shows how simple it is to build and display a tree with Jython and Swing. The tree's structure is defined with data in nested dictionaries. The function *makeTree*, with the help of functions *addNode* and *addLeaves*, generates the appropriate structure of *swing.tree.DefaultMutableTreeNode* instances, required by Swing to display the tree (see Figure 1).

Table 3: Return values

Java data type of return value	converted Python data type
char	string (length = 1)
boolean	int (true = 1, false = 0)
byte, short, int, long	int
float, double	float
java.lang.String	string
java.lang.Class	instance of Java class
Foo[]	list with Foo objects or objects derived from Foo
org.python.corePyObject	No conversion
Foo	Java instance representing Foo class

Listing 1: treedemo.py

```
01 from pawt import swing
02
03 data = {
04     'PyObject': {
05         'PyInteger':None,
06         'PyFloat':None,
07         'PyComplex':None,
08         'PySequence': {
09             'PyArray':None,
10             'PyList':None,
11             'PyTuple':None,
12             'PyString':None,
13         },
14         'PyClass': {
15             'PyJavaClass':None,
16         },
17     },
18     'sys':None,
19     'Py':None,
20     'PyException':None,
21     '_builtin_':None,
22     'ThreadState':None,
23 }
24
25 Node = swing.tree.DefaultMutableTreeNode
26
27 def addNode(tree, key, value):
28     node = Node(key)
29     tree.add(node)
30     if value:
31         addLeaves(node, value.items())
32
33 def addLeaves(node, items):
34     items.sort()
35     for key, value in items:
36         addNode(node, key, value)
37
38 def makeTree(name, data):
39     tree = Node('Some Jython-Classes')
40     addLeaves(tree, data.items())
41     return tree
42
43 if _ _name_ _ == '_ _main_ _':
44     tree = makeTree('Some Jython-Classes', data)
45     swing.test(swing.JScrollPane(swing.JTree(tree)))
```

Listing 2: PyInJava.java

```
01 import org.python.util.PythonInterpreter;
02 import org.python.core.*;
03
04 public class PyInJava{
05     public static void main(String []args)
06     throws PyException
07     {
08         PythonInterpreter P =
09             new PythonInterpreter();
10
11         P.exec("import sys");
12
13         P.set("a", new PyInteger(10));
14         P.set("b", new PyFloat(32.0));
15         P.exec("s=a+b");
16         P.exec("print s");
17         PyObject x = P.get("s");
18
19         System.out.println("Total: "+x);
20     }
21 }
```

The author

Python expert Andreas Jung currently lives near Washington D.C. and works for Zope Corporation (formerly Digital Creations) as a software engineer in the Zope core team.

Python embedded in Java

The combination of Java and Python in the opposite direction is also very promising. It is often desirable to build a Python interpreter into a Java application (embedding) in order to provide that application with scripting functions. The example in Listing 2 demonstrates this integration using the class "PythonInterpreter".

Small but important differences

With all these advantages it's almost inevitable that there's a downside. The differences between Jython and C-Python are caused on one hand by ambiguities within Python's language definition in the Language Reference, and on the other by Java limitations. The most important differences between Jython and C-Python are:

- Extension modules written in C do not work with Jython.
- Key words can also be used as identifiers in Jython.
- Every Jython object is an instance of a class (in C-Python this will only be implemented with version 2.2).
- The functionality of file objects is limited.
- Some of the extension modules included with C-Python are missing or have only been partially implemented.

Java and Python also differ regarding the overloading of methods. Java permits methods with the same name and different signatures – i.e. differences in the number of arguments and their types. Python doesn't allow overloading because its lack of typing and variable function parameters would interfere with the clarity of the semantics. Jython resolves this dilemma by sorting the methods according to their signatures and then selecting the most appropriate.

Java arrays

Arrays are another problem area. Many Java methods use array parameters, but Python doesn't contain any relevant data types. Using the jarray module, Python sequence types can be converted into a PyArray, which can then be passed directly to a Java method. For this, the "array()" method requires a Python sequence as its first parameter and the basic type of the Java array ("h"=short, "d"=double, "c"=char, etc.) as the second.

```
from jarray import array

print array( (1,2,3,4),'h')
print array( (1,2,3,4),'d')
print array( 'linux','c')
```

This little program produces the following output:

```
array([1, 2, 3, 4], short)
array([1.0, 2.0, 3.0, 4.0], double)
array(['l', 'i', 'n', 'u', 'x'], char)
```

Speed is not everything

The speed of Jython primarily depends on the Java VM used. A VM with an integrated just-in-time compiler is significantly faster than one without. Under Windows, using Microsoft's Jview, Jython achieves about 80 per cent of the C-version's performance (measured with Pystone benchmark). Under Linux, however, the speed can be as low as 20 to 30 per cent, for instance with the Blackdown VM.

Jython is very well suited for integrating Python into an existing Java infrastructure. It is also a simple way of familiarising yourself with Java and its class libraries, as the interactive mode lets you play with the Java components on the fly, without having to use an editor or Java compiler. Jython can also communicate with external systems for which there is no C-Python extension, for instance through RMI or with JDBC database drivers.

Info

Jython Web site	*http://www.jython.org/*
Blackdown JDK Linux	*http://www.blackdown.org*
IBM Linux JDK 1.3.0	*http://www-106.ibm.com/developerworks/java/jdk/linux130/?dwzone=java*
Python Language Reference	*http://www.python.org/doc/ref*