魅力的なJython

PythonのJava実装で開発はこんなに楽になる

Uche Ogbuji

2003年 5月 06日

Jythonは、プログラミング言語Pythonの100% Pure Javaな実装であり、Pythonの長所をJava仮想マシンおよびライブラリーと組み合わせ、Javaプラットフォームを補完します。今回の記事では、ソフトウェア・コンサルタントであり、developerWorks でもお馴染みのUche Ogbuji氏が、PythonとJava言語におけるクラスの作成方法やインタープリターの使用方法を比較しながら、Java開発者に向けてJython 2.1の概要を説明します。この記事では、Javaのライブラリー・アクセスや、Jythonインタープリター・シェルおよびコード・ファイルのサンプルを示しながら、両者の違いを説明します。

従来JPython として知られていたJythonは、すべてのコードがJavaで記述されたアプリケーションです。開発者はJythonを使用することにより、プログラミング言語Pythonの構文とほとんどの機能を使用することができます。Jythonは、次のような理由から、Javaプログラマーにとって興味深いものとなっています。

- JythonバージョンのPythonインタープリター・シェルでは、通常のJavaのコンパイル / 実行という手順を踏まずに、アイデアやAPIの実験および調査を簡単に実行できます。
- Pythonはその設計が動的かつ汎用的であるため、複雑な (Javaのリフレクションやイントロスペクションで必要とされるような) ライブラリーを使ってこれらの機能を追加する必要がありません。この特性により、ある種の開発はより簡単になります。またこの特性は、自動化されたテストのフレームワークで特に役立ちます。
- Pythonの構文とその使い勝手は、多くの開発者に人気があります。このような開発者たちは、PythonをJavaアプリケーションの開発と保守のためのきわめて生産性の高い言語だと考えています。

今回の記事では、最新のリリースであるJython 2.1について、Javaライブラリーへのアクセス、Jythonのインタープリター・シェルの使用法、およびわかりやすいJythonのコード・ファイルを例に挙げながら説明します。この記事を理解するにあたってPythonの知識は必ずしも必要ではありませんが、この記事で紹介する基本的なサンプルを超えたレベルでさらにJythonを使用する予定の皆さんは、ぜひPythonを学習してください。

ここで使用する環境は、Red Hat 8.0 (2.4.18カーネル) とJ2SE 1.4.0です。Jythonの使用にあたってのプラットフォームとJava環境の選択については、Jythonのサイト (参考文献を参照) で各プラットフォーム固有の備考をご覧ください。

注意: JythonおよびJavaは、Javaランタイム上で動作します。

Jython入門

Jythonは、インストーラーを含む単一のJavaクラス・ファイルとして配布されます。`jython-21.class` をダウンロードして`CLASSPATH` の任意の場所に配置して、`java jython-21` を実行してください。インストールするコンポーネントを選択し (サンプルではすべてデフォルトを選択しています)、ライセンス (オープン・ソースのBeOpen/CNRIライセンス) に合意してインストール先ディレクトリーを選択すれば、後はインストーラーが処理してくれます。

インストールに問題が発生した場合は、Jython Webサイトのインストール情報のページを参照してください。UNIXプラットフォームの場合は、選択したJythonのインストール・パスを`PATH` 環境変数に追加してください。これで、「jython」と入力すればインタラクティブなシェルを実行できる状態になりました。

リスト1. Jythonシェルの実行

```
$ jython
 Jython 2.1 on java1.4.0_01 (JIT: null)
 Type "copyright", "credits" or "license" for more information.
 >>>
```

`>` プロンプトでコマンドを入力すると、すぐに結果が返ってきます。Javaプログラミングでは、各プログラムで少なくとも1つのクラスを定義する必要があります。リスト2は、画面にメッセージを書き込む完全なJavaプログラムです。

リスト2. 完全なJavaプログラム

```
class App
 {
   public static void main(String args[])
   {
     System.out.println("I don't like spam!");
   }
 }
```

JPythonを使用すると、プログラムの行数が次のように少なくなります。

リスト3. Jythonによって軽減するJavaコードのオーバーヘッド

```
>>> print "I don't like spam!"
 I don't like spam!
 >>>
```

`print` キーワードは重要なツールの1つです。このキーワードは、ただちに引数を画面に表示してシェル・プロンプトに戻るインタラクティブなシェルでは特に重要です。こうすれば、入力するコードやデバッグの分量が少なくなるだけでなく、コンパイル、実行という手順を踏まずにすぐ結果を得ることができます。また、次の例のように、複数の値を同時に表示したり、単純な式を簡単に使用することもできます。

リスト4. Jythonの重要なツール、print

```
>>> print "one plus one is", 1+1
 one plus one is 2
 >>>
```

Jythonの式はJavaの式に似ています。`1+1` の結果は整数値ですが、`print` によって強制的に文字列に変換され、区切り文字のカンマの指定によって前の文字列に連結されています。

さらにJythonを使用すれば、多くの手順を踏まずに標準Javaライブラリーにアクセスできます。次の例では、`java.util.Random` にアクセスしています。

リスト5. Jythonを介した標準Javaライブラリーへのアクセス

```
>>> from java.util import Random
 >>> rng = Random()
 >>> i = rng.nextBoolean()
 >>> print i
 1
 >>>
```

Jythonの`import` キーワードは、あるモジュールの内容を別のモジュールで使用できるようにするという点でJava言語のimportに似ていますが、構文と動作には少し違いがあります。

リスト5の例では、関連する`from` キーワードを使用して、`java.util` からインポートするシンボルを絞り込んでいます。その次の行は、`Random` クラスのインスタンスの生成を示しています。ご覧のとおり、ここでは`new` キーワードは不要です。

また、生成した新しいインスタンスを格納する変数にも型宣言は必要ありません。この事実は、Jythonにおける簡略化の重要性と、その動的な性質から得られる1つのメリットをよく表しています。つまり、型付けを考慮する必要がほとんどないのです。

リスト5のその次の行は、メソッドの起動を示しています。メソッドの起動は、結果の型宣言がないことを除けばJava言語のものとまったく同じです。Javaコードの`nextBoolean()` はブーリアンです。しかし、Jython 2.1にはブーリアン型がありません (ただし、この状況はすぐに変わる可能性があります。Python 2.3ではブーリアン型が追加されているからです)。そのため、Jythonでは代わりに値0 または1 の整数を使用しています。同様に、ブーリアン値を期待するJavaメソッドを起動するには、このような制約に適合する整数値を渡す必要があります。

また`import` キーワードは、リスト6に示すように、インポートするすべてのシンボルを完全修飾名で指定する方法でも使用できます。

リスト6. importでインポートするすべてのシンボルを完全修飾名で指定した場合

```
>>> import java.util.Random
 >>> rng = java.util.Random()
 >>> print rng.nextFloat()
 0.9567907452583313
 >>>
```

Jythonの浮動小数点値は、Java言語とまったく同じです。

ソース・ファイルでのコード作成

インタープリターは手早くチェックや検査を行う場合に便利ですが、すべての作業をインタープリターで行う必要はありません。Jythonでは、ソース・ファイルにコードを書き込んでそのコードを実行することもできます (ただし、Jythonではコンパイルはオプションです)。例として、以下にスタンドアロンのJythonプログラムを示します。

リスト7. コイン・トスをシミュレートするJythonのサンプル・プログラム (listing7.pyという名前のファイルに保存してください)

```
from java.util import Random
 rng = Random()
 #This is a comment in Jython
 print "Flipping a coin..."
 if rng.nextBoolean():
     print "Came up heads"
 else:
     print "Came up tails"
```

プログラムの実行方法の前に、まずコードについて説明します。この例では、Jython (およびその前身であるPython) に関してある程度のユーザーが最初に気づく特徴である、`if` 文を紹介しています。Jythonでは、`if` 文の条件が真のときに実行されるブロックを示す区切り文字がありません (Javaプログラミングとは異なり、Jythonでは条件文を括弧で囲む必要はありません)。該当するブロックのコードは、周りのコードより大幅にインデントすることだけで示されます。

Jythonにおけるコードのブロックは、中括弧などを使用せず、常にインデントによって示されます。コード・ブロックの先頭に来る`if` などの文はコロンで終わります。Jythonのこの特徴は、言い換えれば、コードのインデント方法によってコードの意味が実際に変わってくるため、コードの記述時にはインデントに注意を払う必要があります。たとえば、リスト8aの結果は、3 という数字だけが表示されます。これは、その上の2つの文が決して真になることのない`if` ブロックに含まれるためです。

リスト8a. インデント: "3" だけが表示される例

```
if 0:
     print "1"
     print "2"
 print "3"
```

1つの行のインデントを次のように変更しただけで、2 と3 という数字が表示されるようになります。

リスト8b. インデント: "2" と "3" が表示される例

```
if 0:
     print "1"
 print "2"
 print "3"
```

またインデントは、一貫性を持ち、コードをブロックに編成する文と関連があり、通常はコードの流れを制御するものでなければなりません。たとえば、次の例をご覧ください。

リスト8c. インデント: 構文エラー

```
print "1"
     print "2"
 print "3"
```

この例は単なる構文エラーになります。ブロックを他のコードから分離する制御文がどこにもないためです。

インデントを使用してコード・ブロックを示すことは、PythonおよびJythonで議論の的となる特徴の1つですが、多くの場合、私にはこの件が大げさに取り上げられすぎているように思えます。結局、望ましいコーディング標準に従ってインデントを行っていれば、この件は何の問題にもならないはずです。コーディングで望ましいインデント方法が守られていれば、その確認を同じプログラマーではなく、コンピューターが実施しても同じことです。

さらに、この言語を使用し始めて何時間も経ってからこの制限に気づくような開発者を見たことがありません。正しいインデントは習慣になります。確かに、インデントと構文のこの関連は、これまでに経験したことのないようなエラーを生む可能性がありますが、明示的な区切り文字が存在しないことにより、区切り文字を使用する他の言語でよく発生する一部のエラーもなくなります。

リスト7 のファイル (listing7.py) は、次のように、ファイル名を引数にして`jython` コマンドを起動することによりコンパイルなしでも実行できます。

リスト9. 「コイン・トス」プログラムをコンパイルせずに実行する

```
$ jython listing7.py
 Flipping a coin...
 Came up tails
 $
```

上記の例の`$` はUNIXのシェル・プロンプトで、Windowsシステムの`C:\>` と同じです。また`jpythonc` コマンドを使用してモジュールをJavaのバイトコード (`.class`) ファイルにコンパイルし、それを`java` または`jre` コマンドを使用して直接実行することも可能です。この方法でコンパイルされたJythonモジュールにはいくつか制限がありますが、その件についてはこの記事で取り上げる範囲ではありませんので、ここでは詳しくは説明しません。

グローバル関数の作成

Jythonでは、簡単にグローバル関数を作成できます。またグローバル変数を定義することも可能です (これは通常、クラス・ラッパーを作成せずに定数を設定するためです)。たとえば、次の例をご覧ください。

リスト10. 一連の数値を文字列形式で返すグローバル関数 (listing10.pyという名前のファイルに保存してください)

```
START = 1
 SPACER = " "
 def CounterString(length):
     buffer = ''
     for i in range(START, length):
         buffer = buffer + str(i) + SPACER
     return buffer
 print CounterString(10)
```

まず最初に、このプログラムで定数として使用される2つのグローバル変数、`START` と`SPACER` を、それぞれ整数および文字列として定義します。

次に`def` キーワードを使用して、関数`CounterString` を定義します。この関数は、`length` という名前の整数を単一の引数として取ります。Jythonで引数が整数であるかどうか明示的にチェック

されないという事実は、Jythonの動的な特性による1つの長所です。しかし、この事実は、ある種の型エラーがJavaプログラミングの場合より遅い時期まで発見されないという短所にもなります。

この関数のシグネチャー行がコロンで終わり、その後から行がインデントされて新しいブロックが始まっていることに注目してください。この新しいブロックの最初の行は、文字列バッファーを空の文字列で初期化しています。このバッファーは、関数に期待されている結果を出すために操作されます。

次の行では、ループを作成しています。Jythonの`for` 文は、根本的にJava言語の文と異なります。Javaプログラミングでは、初期条件と終了条件、およびループの各手順を設定します。Jythonのループは、最初から最後まで特定の順序をたどって実行されます。通常この順序は、Jythonの非常に重要なデータ型であるリストの形をとっています。

3つの文字列を含むリストは次のようになります。

```
["a", b", "c"]
```

このプログラムで実行しているように、ループを1 からN までの間繰り返したい場合は、任意の範囲の数字のリストを返す`range()` 関数を使用します。インタラクティブなJythonのプロンプトで何回か実験してみると、このツールをよく理解できると思います。

リスト11. range() 関数の例

```
>>> range(5)
 [0, 1, 2, 3, 4]
 >>> range(1, 5)
 [1, 2, 3, 4]
 >>> range(1, 10, 2)
 [1, 3, 5, 7, 9]
 >>> range(10, 1, -3)
 [10, 7, 4]
```

リスト10 を見直してみると、`for` ループで繰り返される処理は、この関数本体の他の部分よりも一段余分にインデントされたコード・ブロックになっています。そのブロックは1行から成り、ブロック内では現在のバッファーと新しい数字が連結されています。この新しい数字は、連結の前にまず`str()` 関数 (Javaプログラミングで使用される`cast` ではなく) を使用して強制的に文字列に変換され、その後ろにスペースが付加されています。このループの終了後、結果のバッファーが返されます。関数本体のすぐ後ろにあるのは、この関数をテストするための1行のコードです。これもJythonの特徴ですが、アプリケーション・クラスに`main` メソッドなどを装備しなくても、この関数を実行することができます。リスト10の結果は次のようになります。

リスト12. リスト10の結果

```
$ jython listing10.py
 1 2 3 4 5 6 7 8 9
```

関数の作成と同様に簡単なクラスの作成

Jythonでは、グローバル関数の作成と同様の簡単な手順でクラスを作成できます。リスト13はその例です。

リスト13. ユーザー定義クラスの簡単な例 (listing13.pyという名前のファイルに保存してください)

```
class Dog:
     def __init__(self, bark_text):
         self.bark_text = bark_text
         return

     def bark(self):
         print self.bark_text
         return
     def annoy_neighbors(self, degree):
         for i in range(degree):
             print self.bark_text
         return
 print "Fido is born"
 fido = Dog("Bow wow")
 print "Let's hear from Fido"
 fido.bark()
 print "Time to annoy the neighbors"
 fido.annoy_neighbors(5)
```

上記のコードでは、最初の行でクラスに名前を付け、1つの大きなコード・ブロックにその内容をすべて定義しています。

最初に定義されているイニシャライザー は特別なメソッドです (Javaのコンストラクターのようなものです)。このメソッドは常に`__init__` という名前で、クラスの新しいインスタンスが生成されるときには必ず呼び出されます。Jythonでは、現在呼び出されている (イニシャライザーの場合は作成されている) インスタンスを引数として明示的に宣言します。従来どおり、この引数は`self` と呼ばれます。

`Dog` イニシャライザーでは、引数として渡された文字列`bark_text` が、`self` を使用してインスタンス変数として保存されています。メソッド`bark()` は、呼び出し時に明示的なパラメーターを取りませんが、`self` は指定する必要があります。

メソッド`annoy_neighbors` は明示的な引数を1つ取ります。この引数は犬が近所の住人を困らせるために吠える回数を示すもので、`self` の後ろに付加される形で指定されます。コードが流れるようにこの深いネストに、つまりインデントにつながっていることに注目してください。`annoy_neighbors` の場合は、クラス定義の中にこのメソッドの定義があり、その中にループのブロックがあります。この例でも、`print "Fido is born"` で始まるコードで、クラスが実際に使用されています。リスト13の結果は次のようになります。

リスト14. リスト13の結果

```
$ jython listing13.py
 Fido is born
 Let's hear from Fido
 Bow wow
 Time to annoy the neighbors
 Bow wow
 Bow wow
 Bow wow
 Bow wow
 Bow wow
```

プログラミング言語間の橋渡し

今回の記事では、皆さんのJavaプログラムの知識庫にJythonを追加するメリットについて、次の内容を簡単に説明しました。

- Jython言語により、さまざまなタスクを実行するために必要なコードの分量が減少します。
- Jythonインタープリターではコンパイルを行わずにコードを実行できるため、迅速なコード開発が促進されます。
- Jythonでは、Java言語でサポートされていないグローバル関数とグローバル変数を作成できます。
- Jythonでは、静的な型付けを行う仮想マシンで推論とキャストを使用して適切な動作を確保することにより、動的な型付けを導入しています。
- Jythonでは総称データ型の使用を導入しています (もっとも、総称型は今後リリースされるJavaのTigerなどのバージョンでも既に導入済みです)。
- Jythonを使用することにより、開発者は自動化したテストのフレームワークを簡単に作成できます。

またこの記事では、さまざまな例をとおして、開発者が知っておくべき構文や型付けにおける違いにもふれました。このような違いには、Jythonではインデントに構文的な意味があることや、現在サポートされていないブーリアン型の代わりに整数が使用されていることなどがあります。

Jythonを使用するからといって、Java言語から離れてしまう必要は決してありません。Jythonは、Java言語を補完する非常に便利な言語として、検査、プロトタイピング、テストなどを手早く実行したい場合や、Jythonが適している一部のコーディング・タスクを処理する場合などに役立てることができます。

関連トピック

- Jythonホームページでの実装をダウンロードし、使い方を詳しく学んでください。既にJythonをお持ちの方も、このサイトでインストールやプラットフォーム固有の問題などについて有益な情報を得ることができます。
- Jythonは、Python言語の実装の1つです。Jythonの使用を計画されている方は、Python.org で提供されている資料やその他の情報をご覧になることをお勧めします。
- 「Javaコードの診断: replによる対話式評価」(developerWorks、2002年3月) では、Eric Allen氏がJythonを使用して簡潔なrepl (read-eval-print-loop) を作成する例を紹介しています。
- Jythonに関するオンライン・ヘルプや、開発者間のJythonに関するディスカッションの場として最適なJythonユーザーのメーリング・リストにご参加ください。
- O'ReillyとNoel Rappin氏による『Tips for Scripting Java with Jython, Part 1』では、Javaプログラマーにとって特に時間の節約になり、かつ興味深いJythonの11の特殊な機能が説明されています。
- 『Jython Essentials』(Samuele Pedroni and Noel Rappin、O'Reilly、2002年3月) は、Jythonの実質的な入門書であり、JythonとJavaの相互作用を示す多くの例や、Jythonプログラマーにとって役立つモジュールやライブラリーの参照資料を提供しています (Chapter 1はオンラインでも提供されています)。
- Jythonを使用したWebおよび企業アプリケーションの作成方法を学習されたい方は、『Python Programming with the Java Class Libraries』(Richard Hightower、Addison-Wesley/Pearson、2002年6月) をお読みください。
- ActiveState Programmer Network では、Jythonで実装された簡単なJSPカスタム・タグと簡単なJythonサーブレットという2つのJython関連情報が提供されています。
- Pythonを使用したWebサービスの開発について幅広く扱った一連の記事を参照してください。これらの記事は、Jythonでの開発に必要な作業を理解する上で役立ちます。
- developerWorks のJava technology zone では、Javaテクノロジー関連の多くの記事を参照することができます。