www.sxc.hu

Vai aí um cafezinho para acompanhar o jornal?

O melhor de dois mundos: Java e Python

Não dá para usar a ferramenta redonda no buraco quadrado. Às vezes, o ambiente de programação a que estamos acostumados (mesmo o excelente Python) não é adequado para aquele problema que estamos tentando resolver. Apresentamos então o Jython, a forma mais simples que se conhece de desenvolver programas em Java... usando a linguagem Python.

POR JOSÉ MARÍA RUÍZ E JOSÉ PEDRO ORANTES

O *Jython* [1] é, nada mais nada menos, que uma implementação do *Python* [2] na plataforma *Java* [3]. O resultado é que, programando com o Jython, temos como resultado final um programa em Java no qual podemos usar todo o poder da platafoma Java e seus pacotes. Entretanto, para o desenvolvedor ainda é um programa em Python, cujo tempo de desenvolvimento é de duas a quatro vezes mais curto que em Java.

O Jython é, sem tirar nem pôr, um Python como qualquer outro, com as mesmas características, recursos e sintaxe. Mas tem uma vantagem sobre o Python "original": pode trabalhar com qualquer uma das inúmeras bibliotecas do Java. Isso inclui usar o belíssimo *swing* para desenhar interfaces gráficas, usar os renomados *JavaBeans* e até criar *applets java* – tudo muito mais fácil de fazer, pois usamos a facílima linguagem Python.

Isso faz do Jython um ambiente de programação poderoso, com tempo de desenvolvimento menor e código muito mais legível que Java. Ademais, podemos compilar o programa para que não tenhamos a necessidade de instalar o Jython nas máquinas dos clientes.

Outro motivo para usar Jython: máquinas Windows não costumam ter o interpretador de Python instalado. Você precisa pedir ao usuário que instale (por exemplo) o *ActivePython* [4], da empresa *ActiveState*, antes de poder usar o seu programa. No Jython, bastará uma máquina virtual Java instalada – coisa que praticamente todo computador já tem.

Agora a desvantagem. Por conta da dupla camada de abstração, programas desenvolvidos em Jython tendem a ser muito mais lentos que o mesmo programa desenvolvido em Java puro. Mas a potência dos computadores de hoje em dia diminuem essa desvantagem.

Muitos de vocês podem se perguntar: mas eu não preciso saber programar em Java para usar o Jython? A resposta, em um primeiro momento, é não. A única linguagem que você precisa realmente saber de cor e salteado é o Python. Mas sempre é bom ter uma idéia de como o Java funciona e ter à mão a documentação da *API* do Java, disponível no site da *Sun* em [4]. Ter ao alcance a referência da API é bastante produtivo, já que precisamos saber o que queremos que o Java faça pelo nosso programa em Python e como temos que pedir isso a ele. Mesmo assim, nosso programa terá a sintaxe e a forma de programar do Python. Portanto, não é obrigatório ter a documentação da API do Java para programar – mas altamente recomendável.

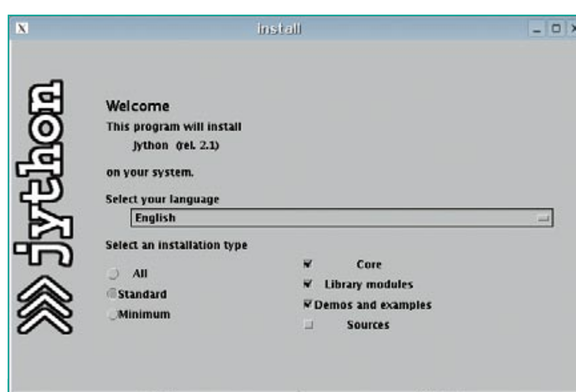

Figura 1: O programa de instalação do *Jython*.

Vamos considerar, então, que o estimado leitor aprendeu direitinho o que mostramos nos artigos anteriores. Eles serão suficientes para seguir o que trataremos aqui. Este mês, aprenderemos a usar elementos básicos do Java dentro do Jython, trabalharemos com o `swing` e usaremos alguns objetos do Java – como, por exemplo, os vetores. Para fins de exemplo, nos valeremos de um aplicativo externo ligeiramente modificado – o código original pode ser encontrado no artigo *Dive into XML* do site xml.com [5].

O programa, desenvolvido em Python, serve para interpretar as transmissões de manchetes, pelo protocolo `RSS`, de blogs, sites de notícias e afins.

Café com pão, digo, Python

Para trabalhar com o Jython, precisamos ter o Java instalado no computador. Para nós, tanto faz usarmos o *Java Runtime Edition* (`j2re`) ou o *Java Developers Kit* (`j2sdk`), desde que a versão seja **1.4.2** ou posterior. Ambos os ambientes podem ser baixados do site oficial do Java [3]. Com o Java instalado, precisamos do interpretador Jython, disponível em [1]. A última versão estável do Jython, ao menos até enviarmos esta matéria para o editor, era a **jython-2.1**. Precisamos também do interpretador Python, que provavelmente sua máquina Linux já possui. Verifique, apenas, se a versão do Python é igual ou superior à **2.3**.

O único passo não trivial (embora fácil) é a instalação do interpretador Jython. O comando `java jython_21` apresentará o instalador (**figura 1**), que nos pedirá confirmação para uma série de opções. Por fim, informe o diretório em que quer o Jython. Pronto! Tudo instalado. Agora, vamos criar links simbólicos entre os binários do Jython e o diretório usual em que os programas do usuário ficam. Pelas normas, esse diretório é `/usr/local/bin` para disponibilizar o Jython para todos os usuários e `/home/`*`usuário`*`/bin` para um único usuário. O comando, para quem não lembra, se parece com:

```
ln -s jython-2.1/jython /usr/local/bin/jython
ln -s jython-2.1/jythonc /usr/local/bin/jythonc
```

Você poderia usar `/usr/bin`, se quisesse, em vez de `/usr/local/bin`, mas sempre é melhor seguir os padrões, não é mesmo? Quisera todas as distribuições fizessem o mesmo...

Café no bule

Tudo pronto? É hora de ver como funciona esse tal de Jython. Para começar, que tal brincar um pouco com o interpretador, como fizemos com o Python? Assim, podemos ter certeza de que o comportamento de ambos é idêntico:

```
$ jython
Jython 2.1 on java1.4.2_05➥
 (JIT: null)
Type "copyright", "credits" or➥
 "license" for more information.
 >>> print 'Fala, mano!'
 Fala, mano!
 >>>
```

Agora que tal cutucar com um pouquinho de Java? Vamos produzir um "Olá, Mundo!", só que agora com o swing. Nosso exemplo exibirá uma janela chamada Olá Mundo, com um quadro de texto. Por enquanto, você precisará encerrar o interpretador com ***[Ctrl] + [C]*** para fechar a janela, já que nosso programa ainda não tem um botão de *sair*...

```
$ jython
Jython 2.1 on java1.4.2_05➥
 (JIT: null)
Type "copyright", "credits"➥
 or "license" for more information.
>>> import javax.swing as swing
>>> win = swing.JFrame("Olá mundo!)
>>> texto =➥
 swing.JLabel("Ei. Olha o mundo!")
>>> win.contentPane.add(texto)
>>> win.pack()
>>> win.show()
```

Graças ao `swing`, o programa é bastante agradável aos olhos e, ao mesmo tempo, bem simples. O Java inclui ainda um sistema para alterar a aparência da interface gráfica, se a que aparece por padrão não nos agradar.

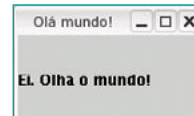

Figura 2: Um "Olá mundo!" gráfico.

Programando

Assim como em Python, o mais cômodo é fazer com que o Jython execute arquivos contendo o código, em vez de ficar digitando comandinhos diretamente no interpretador. O Jython não possui nenhum tipo de extensão predefinida para os arquivos. O sistema determina o tipo de arquivo pelo conteúdo, não pela extensão. Colocamos o sufixo `.py` simplesmente para que nós possamos distinguir o arquivo.

Para usar as bibliotecas do Java, primeiro temos que importá-las, no mesmo estilo que usamos com as bibliotecas do próprio Python. Como vimos no exemplo anterior, no qual importamos as bibliotecas do `swing`, só precisamos importar o que vamos usar. Imagine que precisaremos da classe `vector`, que está dentro do pacote `util`. Só o que temos que fazer é:

```
import java.util as util
```

ou, mais diretamente,

```
import java.util.vector as vector
```

Viu? Não precisamos de todo o pacote `util`, podemos importar apenas a classe `util.vector`. Para utilizá-la, bastaria chamar seu *construtor* (Arrá! Viu como um guia de referência da API do Java faz falta?) e criar com ele uma instância da classe `vetor`. Por exemplo:

```
pepe = util.vector()
```

Isso cria um objeto chamado `pepe`, que é uma instância da classe `vetor`. Uma vez criado, podemos aceder a qualquer dos métodos desse vetor chamado `pepe`. O comando

```
pepe.add("Juazeiro")
```

adiciona um objeto de tipo `String` (cadeia de caracteres) com o valor `Juazeiro` ao vetor `pepe`.

Cada instância de qualquer classe tem o mesmo comportamento tanto em Java quanto em Jython. É importante que nos lembremos disso, já que agora vamos misturar tudo. Vamos criar, então, três botões, cada um criado de uma forma diferente. Em cada um deles, colocaremos um *listener* (auscultador) para detectar os eventos que agem em cada botão. Cada um deles, quando pressionado, irá disparar uma rotina que escreverá seu nome em um quadro de texto. Confira a **listagem 1** e a **figura3**.

Bastante simplório, não é mesmo? Se você observar bem, para criar os botões usamos código próprio do Python, usando uma lista com os nomes dos botões e os criando com um laço `for`, dentro do qual chamo o construtor `JButton`. Depois de passar alguns argumentos ao construtor, os botões são adicionados ao painel com o método `.add()`, que implementa o `JPanel`.

Listagem 1: Três botões

```
import javax.swing as swing
import java.awt as awt

quadroTexto = swing.JTextField(10)

def __init__():
 win =           swing.JFrame("Exemplo com           botões")
 acoes = ['um', 'dois', 'tres']

 pnlBotoes = swing.JPanel(awt.FlowLayout())

 pnlBotoes.add(quadroTexto)
        for cadaBotao in acoes:
            pnlBotoes.add(swing.JButton(cadaBotao,actionPerformed=acao))
 win.contentPane.add(pnlBotoes)
 win.size=(300,300)
 win.pack()
     win.show()

def acao(event):
 acaoBotao = event.getActionCommand()
 if acaoBotao == "um":
     quadroTexto.setText("UM")
 elif acaoBotao == "dois":
     quadroTexto.setText("DOIS")
 else:
     quadroTexto.setText("TRÊS")
root = __init__()
```

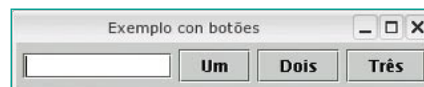

Figura 3: Clicar nos botões preenche o campo.

JyRSS – nosso leitor de notícias

Chegou a hora de montarmos nosso aplicativo principal. Alguns de vocês talvez não saibam o que é `RSS`, portanto aí vai uma explicação sucinta. `RSS` não é nada mais nada menos que um arquivo XML com uma série de marcas ("*tags*"). Essas marcas seguem um padrão definido pela xml.com para transmitir as manchetes das notícias e um pequeno texto que as descreve. Um aplicativo que use esse arquivo XML poderia baixar as notícias para que as possamos ler, sem que precisemos abrir um navegador. Nosso programinha vai baixar as notícias e prepará-las para que as possamos ler.

Para começar, precisamos do código do programa que mencionamos no início do artigo. Ele será responsável por baixar e interpretar o arquivo RSS com as notícias. Como resultado, teremos um outro arquivo, contendo as notícias já interpretadas para que nosso programa em Jython possa apresentar. O sistema todo tem algumas deficiências, como não reconhecer todas as codificações de caracteres. Por isso, recomendamos que se façam testes primeiro com sites de notícias em inglês. Uma boa pedida: Slashdot (http://www.slashdot.org/index.rss, **figura 1**), já que em alguns sites em português (como o http://www.vivaolinux.com.br/) contém acentos e caracteres especiais que podem fazer com que o programa não funcione direito.

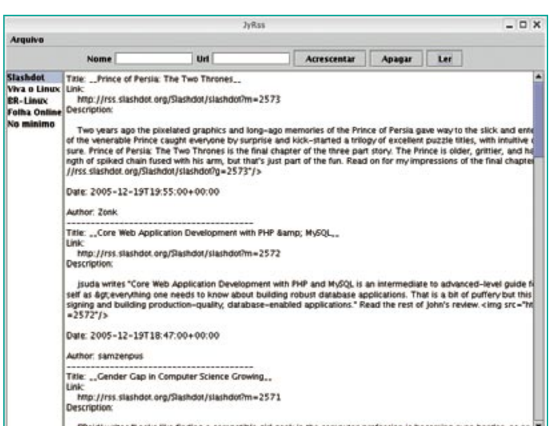

Figura 4: O *JyRSS* em ação, lendo um feed do *Slashdot*

O JyRSS é composto de várias partes. Há um `JFrame` em que estarão embutidos o `Painel de botões` e os `quadros de texto`. Um dos quadros de texto é um `JList`, no qual mostraremos os nomes dos sites que contêm as notícias. O outro é um `JEditorPane`; nele as notícias serão exibidas. Ambos fazem uso do `JScrollPane` para que se possa rolar a tela nos casos em que o texto é maior do que a janela.

Também usaremos `JMenu` e `JMenuBar` para criar um pequeno menu com a opção `Salvar` (que salva **dois** arquivos, um com os nomes dos sites e outro com as URLs de cada site). Esse menu nem seria necessário; só o incluímos para que o leitor perceba como é fácil criar todos esses penduricalhos de que as interfaces gráficas são feitas.

O *painel de botões* terá três `JButtons`, um para *adicionar* uma nova URL à lista, outro para *apagar* uma URL da lista e

Listagem 2: JyRSS.py

```
#!/usr/bin/jython

import javax.swing as swing
import java.lang as lang
import java.awt as awt
import java.util as util
import os



class Lector:

    def exit(self, event):
        lang.System.exit(0)

    def __init__(self):

        self.vectorrss = util.Vector()
        self.vectorurl = util.Vector()
        self.listaRSS()
        self.listaNoticias()
        self.pnlBotoes()
        self.menu()
        if os.path.exists(‘listarss.txt’):
            self.leArquivoRss()
        self.win = swing.JFrame(“JyRss”, size=(300,300),windowClosing=self.exit)
        self.win.setJMenuBar(self.menu)
        self.win.contentPane.add(self.pnlBotao,awt.BorderLayout.NORTH)
        self.win.contentPane.add(self.jscplista, awt.BorderLayout.WEST)
        self.win.contentPane.add(self.jscpNoticias, awt.BorderLayout.CENTER)
        self.win.setSize(600, 400)
        self.win.show()

    def pnlBotoes(self):
        self.pnlBotao = swing.JPanel(awt.FlowLayout())
        acoes = [“Acrescentar”,”Apagar”,”Ler”]
        self.txtUrl = swing.JTextField(10)
        lblNome = swing.JLabel(“Nome”)
        self.txtNome = swing.JTextField(10)
        lblUrl = swing.JLabel(“Url”)
        self.pnlBotao.add(lblNome)
        self.pnlBotao.add(self.txtNome)
        self.pnlBotao.add(lblUrl)
        self.pnlBotao.add(self.txtUrl)

        for cadaBotao in acoes:
            self.pnlBotao.add(swing.JButton(cadaBotao, actionPerformed=self.acaoMenu))



    def menu(self):
        opcoes = [“Salvar”]
        self.menu = swing.JMenuBar()
        arquivo = swing.JMenu(“Arquivo”)
        for eachOpcao in opcoes:
            arquivo.add(swing.JMenuItem(eachOpcao, actionPerformed=self.acaoMenu))
        self.menu.add(arquivo)

    def listaRSS(self):
        self.lstLista = swing.JList()
        self.jscplista = swing.JScrollPane(self.lstLista)
        self.jscplista.setSize(100,100)

    def listaNoticias(self):
        self.lstNoticias = swing.JEditorPane()
        self.jscpNoticias = swing.JScrollPane(self.lstNoticias)

    def leArquivoRss(self):

        f = open(‘listarss.txt’,’r’)
        fu = open(‘listaurl.txt’, ‘r’)
        linha = f.readline()
        lurl = fu.readline()
        while linha:
            self.vectorrss.add(linha)
            self.vectorurl.add(lurl)
            linha = f.readline()
            lurl = fu.readline()
        f.close()
        fu.close()
        self.lstLista.setListData(self.vectorrss)

    def leArquivoNoticias(self):
        fg = open(‘news.txt’,’r’)
        texto = fg.read()
        fg.close()
        self.lstNoticias.setText(texto)

    def salvarArquivo(self):
        fg = open(‘listarss.txt’,’w’)
        furl = open(‘listaurl.txt’,’w’)
        j = self.vectorrss.size()
        i = 0
        while i<=j-1:
            texto = self.vectorrss.get(i)
            fg.write(texto +’\n’)
            texto = self.vectorurl.get(i)
            furl.write(texto +’\n’)
            i = i+1
        fg.close()
        furl.close()

    def acaoMenu(self, event):
        self.acao = event.getActionCommand()
        if self.acao == ‘Acrescentar’:
            if self.txtNome.getText() == “”:
                self.vectorrss.add(“SEM NOME\n”)
            else:
                self.vectorrss.add(self.txtNome.getText())
            if self.txtUrl.getText() == “”:
                self.vectorurl.add(“SEM URL\n”)
            else:
                self.vectorurl.add(self.txtUrl.getText())

            self.lstLista.setListData(self.vectorrss)
            self.txtNome.setText(“”)
            self.txtUrl.setText(“”)

        elif self.acao == ‘Ler’:
            item = self.lstLista.getSelectedIndex()
            url = self.vectorurl.get(item)
            os.system(‘python lrss.py ‘+ url)
            self.leArquivoNoticias()

        elif self.acao == ‘Apagar’:
            itemapagar = self.lstLista.getSelectedIndex()
            self.vectorrss.remove(itemapagar)
            self.vectorurl.remove(itemapagar)
            self.lstLista.setListData(self.vectorrss)

        elif self.acao == ‘Salvar’:
            self.salvarArquivo()

root = Lector()
```

outro que chamará o interpretador de RSS (lrss.py) para nos mostrar as notícias de cada um dos sites, que aparecerão em `JEditorPane`.

Também fazemos uso da classe `vector`. Usaremos dois vetores, um para guardar os nomes e outro para os endereços. O vetor de nomes será passado à instância `JList` para que adicione todos os nomes dos sites à tela. Usaremos o pacote `java.lang` para implementar a função de sair da janela quando clicarmos no botão `X`, no canto superior direito de nosso aplicativo. Ao contrário do que se poderia esperar, isso não é criado por padrão.

Do Python usaremos a biblioteca `os`, pois nos valeremos do método `os.path.exists()` para comprovar se o arquivo de nomes existe mesmo. Usaremos também `os.system()`, que executará o script Python responsável pela leitura das noticias. Este guardará as notícias em um arquivo que será posteriormente lido por nosso programa. Não estranhe se o programa demorar para mostrar alguma coisa na tela. Isso tem motivo – aliás, dois motivos: é preciso baixar o arquivo a partir do site de notícias (e isso leva tempo) e, depois, temos que interpretar esse arquivo. Além disso, as bibliotecas usadas (`minidom`) são bastante lentas. Além desses itens, usaremos os comandos normais de abertura, leitura, escrita e fechamento de arquivos (`open()`, `write()`, `read()`, etc...).

Os nomes das bibliotecas estão sendo citados para que o estimado leitor vá se acostumando com eles e os identifique à medida que lê o código. Dessa forma, será mais fácil achar a referência a eles na API do Java. Voltamos a repetir: tenha sempre à mão um guia de referência rápida de Java.

Como qualquer um pode ver, programar em Java usando o Jython não é nada complicado. Pelo contrário, é sobremaneira cômodo e muito mais simples.

Listagem 3: lrss.py

```
from xml.dom import minidom
import urllib

DEFAULT_NAMESPACES = \
  (None, # RSS 0.91, 0.92, 0.93, 0.94, 2.0
  ‘http://purl.org/rss/1.0/’, # RSS 1.0
  ‘http://my.netscape.com/rdf/simple/0.9/’ # RSS 0.90
  )
DUBLIN_CORE = (‘http://purl.org/dc/elements/1.1/’,)

def load(rssURL):
  return minidom.parse(urllib.urlopen(rssURL))

def getElementsByTagName(node, tagName, possibleNamespaces=DEFAULT_NAMESPACES):
  for namespace in possibleNamespaces:
    children = node.getElementsByTagNameNS(namespace, tagName)
    if len(children): return children
  return []

def first(node, tagName, possibleNamespaces=DEFAULT_NAMESPACES):
  children = getElementsByTagName(node, tagName, possibleNamespaces)
  return len(children) and children[0] or None

def textOf(node):
  return node and “”.join([child.data for child in node.childNodes]) or “”

if __name__ == ‘__main__’:
  import sys
  rssDocument = load(sys.argv[1])
  fn = open(‘news.txt’,’w’)
  Noticia=””
  for item in getElementsByTagName(rssDocument, ‘item’):
    Noticia =  ‘Title: __’ + textOf(first(item, ‘title’))+ “__\n”
    Noticia = Noticia + ‘Link: \n ‘+ textOf(first(item, ‘link’))+ “\n”
    Noticia = Noticia + ‘Description: \n\n    ‘ + textOf(first(item, ‘description’))+ “\n”
    Noticia = Noticia + ‘\nDate: ‘ + textOf(first(item, ‘date’, DUBLIN_CORE))+ “\n”
    Noticia = Noticia + ‘\nAuthor: ‘+ textOf(first(item, ‘creator’, DUBLIN_CORE))+ “\n”
    Noticia = Noticia + “------------\n”
    fn.write(Noticia)
  fn.close()
```

Na **listagem 3** temos o código em Python do pequeno aplicativo externo que nos ajudou nessa jornada.

Melhorando o JyRSS

Como qualquer um pode ver, a esse programa faltam muitos recursos. Além disso, deve haver diversos *bugs* à espreita. Exortamos o estimado leitor a que tente depurar todos esses bugs.

Também sugerimos a execução de algumas obras de melhoria. Por exemplo, faça com que o quadro `JEditorPane` mude a aparência do texto (negrito, itálico, etc...); o nome selecionado na lista `JList` poderia disparar a leitura automática da notícia (dica: você precisará manipular um *listener* associado à `JList`); incluir uma opção de salvar apenas os sites preferidos... E por aí vai. Divirta-se! E até a próxima... ■

SOBRE O AUTOR

José María Ruíz está terminando seu projeto de conclusão de curso na Faculdade de Engenharia Técnica em Informática e Sistemas, na Espanha. Já há sete anos, usa e desenvolve software livre, desde os velhos tempos da telinha preta do DOS até o moderno FreeBSD.

José Pedro Orantes está cursando o 3º ano de Engenharia Técnica em Informática e Sistemas e, simultaneamente, o 3º ano de Engenharia Técnica em gestão de informática. Usa Linux há seis anos no computador de trabalho, tanto para trabalhos de escritório como para desenvolvimento.

INFORMAÇÕES

[1] *Jython*: http://www.jython.org

[2] *Python*: http://www.python.org

[3] *Java*: http://java.sun.com

[4] *ActivePython*: http://www.activeperl.com/python.plex

[5] Mergulhe no *XML*: http://www.xml.com/lpt/a/2002/12/18/dive-into-xml.html