

Construa um player simples com Java no Python

Reprodutor de MP3 com Jython

Vamos avançar mais um pouco no Jython: neste mês, usaremos as bibliotecas Java para a reprodução de sons e criaremos um mini tocador de MP3.

POR JOSÉ PEDRO ORANTES

E JOSÉ MARIA RUÍZ



O MP3 é um dos formatos de som mais difundidos no mundo da música digital. Todos os dias, milhares de pessoas criam arquivos em MP3, e muitos aparelhos de som, inclusive de carros, já tocam esse formato. Atualmente, a maioria dos programas para tocar MP3 suporta esse formato. Mas como criar nosso próprio reprodutor de MP3?

Programar um reprodutor de som em Java, com a exceção de um compatível com os formatos comuns que o *Java* suporta, é bastante complicado. É necessário muito tempo para decodificar o formato MP3. Por isso, para criar nosso reprodutor, vamos utilizar bibliotecas preparadas para essa decodificação. Nesse caso, vamos utilizar as bibliotecas com licença LGPL (*Lesser General Public License*).

Geralmente, é preciso definir várias etapas para reproduzir som em Java: obter o stream de áudio, decodificá-lo, tratá-lo e escrevê-lo no dispositivo do nosso PC.

Requisitos

Como requisitos fundamentais, iremos precisar do nosso querido intérprete *Jython* (abordado inicialmente no tutorial de Python da edição 16). Para obtê-lo, vá à página da Internet e baixe sua última versão (2.1, no fechamento desta edição). Também precisaremos da nova versão do Java (*JRE 1.5* ou *JDK 1.5*), disponível em [3]. Essa versão incorpora algumas novidades e melhorias necessárias para nosso pequeno reprodutor de MP3. Por último, temos que obter as bibliotecas que vamos utilizar: trata-se do pacote *MPSPI*, encontrado no site Javazoom [2], ou em www.javazoom.net/mp3spi/mp3spi.html.

Uma vez instalados o intérprete de *Jython* e o *JDK 1.5* (ou *1.5*), devemos prosseguir instalando as bibliotecas do *Javazoom* (`unzip -x pasta.zip` ou `tar xvzf pasta.tar.gz`). O pri-

meiro passo é copiar as bibliotecas necessárias para o diretório `lib/ext/` do nosso *JRE* (máquina virtual Java). No nosso caso, depois de instalar o *JDK 1.5*, esse diretório se encontra em `/usr/Java/jdk1.5.0_02/jre/lib/ext/`; devemos copiar aqui os arquivos `mp3spi1.9.2.jar`, `j11.0.jar` e `tritonus_share.jar` (esses dois últimos estão no diretório `lib` do arquivo descompactado).

Feito isso, ao iniciar o intérprete *Jython*, aparecerá uma mensagem para cada arquivo copiado, indicando que o respectivo pacote está sendo processado para uso. Para confirmar que tudo está correto, digite `import javazoom` no interpretador. Nenhuma mensagem de erro deve aparecer (caso isso ocorra, verifique se todos os arquivos estão copiados no local certo).

Com essas bibliotecas, temos agora um decodificador do stream de nossos

Listagem 1: jython-mp3.py

```

001 import org.tritonus.share.sampled.TAudioFormat as taf
002 import org.tritonus.share.sampled.file.TAudioFileFormat
    as ftaf
003 import javax.sound.sampled as sampled
004 import java.io as io
005 import jarray
006 import java.util as util
007 import javax.swing as swing
008 import java.lang as lang
009 import java.awt as awt
010 import java.util as util
011 import javazoom.jl.player.advanced as pad
012 import java.io as io
013
014 class jyMusica(lang.Thread):
015     def __init__(self):
016         self.panelReprodutor()
017         self.menu()
018         self.win = swing.JFrame("JyMusica", size=(150,
019 150), windowClosing=self.exit)
020         self.win.setJMenuBar(self.menu)
021         self.win.contentPane.add(self.pnlBotao, awt.
022 BorderLayout.NORTH)
023         self.win.contentPane.add(self.pnlBotao2, awt.
024 BorderLayout.CENTER)
025         self.win.show()
026         self.win.setResizable(0)
027         self.win.pack()
028
029     def exit(self, event):
030         lang.System.exit(0)
031
032     def run(self):
033         arquivo = io.File(self.nome)
034         inp = sampled.AudioSystem.getAudioInputStream(
035 arquivo)
036         baseFileFormat = sampled.AudioSystem.getAudioFi
037 leFormat(arquivo)
038         baseFormat = baseFileFormat.getFormat()
039         self.getTag(baseFileFormat)
040         self.player.play()
041
042     def getTag(self, baseFileFormat):
043         properties = ftaf.properties(baseFileFormat)
044         key = "author"
045         val = properties.get(key)
046         key = "title"
047         tag = properties.get(key)
048         key = "album"
049         alb = properties.get(key)
050         self.txtArea.setText("")
051         self.txtArea.append("\nArtista: ")
052         self.txtArea.append(val + "\n")
053         self.txtArea.append("Faixa: ")
054         self.txtArea.append(tag + "\n")
055
056     self.txtArea.append("Ⓚbum: ")
057         self.txtArea.append(alb + "\n")
058
059     def panelReprodutor(self):
060         self.pnlBotao = swing.JPanel(awt.FlowLayout())
061         self.pnlBotao2 = swing.JPanel(awt.FlowLayout())
062         acoes = ["Tocar", "Parar"]
063         self.txtArea = swing.JTextArea(6,18)
064         self.areaSrl = swing.JScrollPane(self.txtArea)
065         self.pnlBotao.add(self.areaSrl)
066         for cadaBotao in acoes:
067             self.pnlBotao2.add(swing.JButton(cadaBotao,
068 actionPerformed=self.acaoMenu))
069
070     def menu(self):
071         opcoes = ["Tocar", "Parar"]
072         self.menu = swing.JMenuBar()
073         arquivo = swing.JMenu("Arquivo")
074         for eachOpcion in opcoes:
075             arquivo.add(swing.JMenuItem(eachOpcion,
076 actionPerformed=self.acaoMenu))
077         arquivo.addSeparator()
078         self.menu.add(arquivo)
079
080     def acaoMenu(self, event):
081         self.acao = event.getActionCommand()
082         if self.acao == 'Tocar':
083             selector = swing.JFileChooser()
084             rtn = selector.showOpenDialog(self.win)
085             arquivo = selector.getSelectedFile()
086             self.player = pad.AdvancedPlayer(io.FileInp
087 utStream(arquivo))
088             self.nome=arquivo.toString()
089             self.play()
090
091         elif self.acao=='Parar':
092             self.stop()
093
094     def play(self):
095         self.thread = lang.Thread(self)
096         self.thread.start()
097
098     def stop(self):
099         self.player.stop()
100         self.line.drain()
101         self.line.stop()
102         self.line.close()
103         self.din.close()
104         self.thread.interrupt()
105         self.thread.finalize()
106
107 if __name__=='__main__':
108     jyMusica()

```

arquivos MP3 além de algumas outras ferramentas (bibliotecas para conversão de formatos, leitor de informações de canções, artistas, álbum etc).

Ainda falta um último passo: devemos eliminar uma das limitações do Jython. O Java possui diferentes tipos de variáveis, métodos, construtores etc, que podem ser privados (*private*) ou públicos (*public*). É importante saber que no Jython não se pode declarar variáveis, métodos etc, como públicas nem como privadas. Ao tentarmos acessar, por exemplo, uma interface abstrata privada, aparecerá uma mensagem de erro dizendo que não se pode acessar a interface com o denominador public.

Solucionar essa limitação é muito simples. Basta editar a pasta de registro do Jython no diretório onde ele está instalado (no nosso caso, `$HOME/Desenvolvimento/jython-2.1/registry`) e buscar a linha `python.security.respectJavaAccessibility = true` e mudar "true" para "false". Agora o nosso sistema já está pronto.

Por fim, é muito importante ter à mão a documentação da API do Java, também disponível em sua página da Internet, e a das bibliotecas que iremos utilizar. Do Javazoom, será necessária a documentação do *JLayer* (a do MPSPi vem com o arquivo bai-

xado). A API das bibliotecas Tritonus podem ser baixadas em [4].

Som em Java?

Produzir sons em Java requer que os programadores conheçam o funcionamento de suas classes principais. As que iremos utilizar são `AudioSystem`, `AudioFormat` e `SourceDataLine`. A `AudioSystem` está conectada aos dispositivos do sistema, de modo que podemos acessá-los. Dentro dela, dispomos de métodos para tratar o stream de áudio `AudioFormat`, com o qual manipulamos o formato de áudio de que necessitamos e o `SourceDataLine`, que nos possibilita entrar no dispositivo de som, ou seja, introduzir som no dispositivo e, dessa forma, escutá-lo.

O Java proporciona uma classe mais direta, que é o `CLIP`. Costuma-se utilizá-lo mais para sons pequenos em aplicações ou em applets. Uma das limitações do `CLIP` é que é necessário carregar o som inteiro na memória antes de reproduzi-lo, o que aumenta os recursos utilizados, além de estar preparado para reproduzir somente os formatos típicos em Java (*Wav*, *A-Law*, *U-Law* etc). Vamos utilizar um `SourceDataLine` e trabalharemos com um pequeno buffer com o stream de som decodificado, a ser introduzido em nosso dispositivo de som.

Os passos para tratar o som são os seguintes. Primeiro, temos que obter o stream e o formato de som (utilizando `AudioSystem` e seus métodos `getAudioInputStream` e `getAudioFileFormat`). Além do formato de áudio (`AudioFormat`), iremos ler os metadados do MP3 (título, álbum, autor etc). Com isso feito (o decodificador já está fazendo seu trabalho), precisamos criar um *buffer*, que não é nada mais que uma matriz (*array*) de bytes. Para criar o chamado array, o Jython nos fornece um módulo chamado `jarray`, que tem dois construtores: `zeros` e `array`. O primeiro cria uma matriz

iniciada com 0 do tamanho e tipo que precisamos; o segundo cria uma matriz já inicializada (por nós mesmos) de um tipo de dados. Vejamos um exemplo do uso do `Jarray`:

```
>>> import jarray
>>> a = jarray.zeros(4, 'i')
>>> print a array([0, 0, 0, 0], int)
>>> b = jarray.array([1,2,3,4], 'i')
>>> print b array([1, 2, 3, 4], int)
```

Como se pode ver, o funcionamento é bem simples. `jarray.zeros` recebe os argumentos que são o número de elementos e o tipo de dados – nesse caso, 'i' são inteiros (*integer*). Já `jarray.array` recebe a seqüência que forma o array e o tipo de dados também. Para nosso buffer é necessário um array de 4096 Bytes, que criaremos usando `jarray.zeros(4096, 'b')`.

Se quiséssemos que nosso array fosse formado por cadeias de caracteres (*strings*), teríamos um `array.zeros(45, Java.lang.String)`. Se precisarmos (de fato, precisamos) passar a longitude do nosso array como argumento a um método, usamos `len(array)`, já que os arrays do Jython não têm um método como o `.length` dos arrays Java.

Por último, criaremos o `SourceDataLine` e, a partir do seu método `write`, gravaremos o som no dispositivo de áudio (então já começaremos a ouvir música).

Assim, restará apenas preencher o buffer (a partir do `AudioInputStream`) e voltar a escrevê-lo no dispositivo de som, até chegar ao final da canção ou até que se interrompa a reprodução.

Finalmente, já que usaremos uma pequena interface (feita com *swing*), vamos precisar programar o Java com *threads* – através do que nosso reproduzidor herdou da classe `Thread` – e, no momento de reproduzir, será produzido um *thread*, que se encarregará de tocar a música,

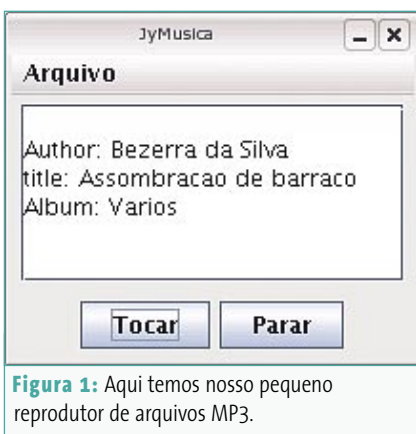


Figura 1: Aqui temos nosso pequeno reproduzidor de arquivos MP3.

enquanto continuamos tendo controle em nosso pequeno reproduutor.

Um thread é uma seqüência de instruções executadas em paralelo com a aplicação que o chama. Podemos dizer que enquanto usamos nossa aplicação, produzimos outra pequena aplicação que reproduzirá o som sem interferir na primeira (execução em paralelo).

Métodos e bibliotecas

Vamos implementar um total de onze métodos. Começaremos descrevendo o “motor” do reproduzidor e terminaremos com os de criação da interface. Os métodos são quatro:

⇒ `Run()` - É neste método que começa o processo de tratamento de nosso arquivo MP3. Daqui obtemos o `InputStream` e o `AudioFormat`. Depois, chamaremos o método seguinte. `Reproduzir()` vai se chamar

`run()`, já que é uma implementação do método `run()` que herdamos da classe `Thread`. Dessa forma, ao chamar o método `start()` do novo thread, o método `run()` será executado diretamente.

⇒ `Reproduzir()`: aqui é criado o buffer de Bytes e o `SourceDataLine`. Assim, recebemos o `AudioInputStream` (o stream de áudio) e criamos o `SourceDataLine` (recebido do método `getLine`, que definiremos mais tarde). Entramos então em um loop no qual contamos os bytes que vamos ler e os que serão escritos, e com o qual leremos (preenchendo o buffer) através do `AudioInputStream`, e então o escrevemos no dispositivo de som com o `SourceDataLine`. Finalmente, quando a canção terminar, fechamos o `SourceDataLine` e o `AudioInputStream`.

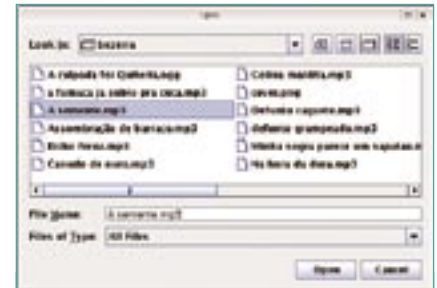


Figura 2: Esse é o selecionador de arquivos (`JFileChooser`) que será produzido ao se executar o método `showOpenDialog`.

⇒ `GetLine()`: Com este método criamos o `SourceDataLine` (através do `AudioSystem`), a partir do stream já decodificado e que nos será devolvido. Usaremos um método que implementa o JDK1.5, que é o `get-SourceDataLine`. Dessa forma, evitamos ter que fazer o downcasting (fazer o molde de uma classe baixa a uma classe mais específica). O molde

Simples 
Consultoria

Tecnologia + Humana



Usabilidade
Arquitetura da Informação
Acessibilidade
Gerenciamento de Conteúdo
Treinamentos Especializados Zope e Plone

www.simplesconsultoria.com.br

não converte o objeto, é apenas um objeto prolongado mais específico) do método `getLine` do `audioSystem` (que produz um `DataLine.info`).

→ Finalmente, temos o `getTag()`: esse método será encarregado de nos fornecer a informação da canção (autor, título etc). Ele irá receber o formato do arquivo sem decodificá-lo e criar uma ocorrência do tipo `Map`, com a qual acessaremos os dados que mais nos interessam – neste caso, álbum, título e autor.

Para os métodos da interface (figura 1), teremos o criador do botão e do quadro de texto (`panelReprodutor()`), com o qual criaremos os botões (`Jbuttons`) e a área de texto (`JTextArea()`), que nos mostrará a informação do MP3; o criador do menu (`menu()`), que criará um menu com duas opções (`Tocar` e `Parar`, que realizarão as mesmas funções dos botões); o receptor das funções do mouse (`acaoMenu()`), que executará os métodos `play` e `stop`; o método `Play()`, que iniciará o thread que irá reproduzir a canção, além de produzir o selecionador de pastas; e o método `Stop()`, que irá interromper a reprodução em curso.

A interface é muito simples. Uma área de texto (onde serão inseridos os dados do MP3), o botão `Tocar`, que abrirá um selecionador de arquivos (figura 2), com o qual selecionaremos o arquivo a ser reproduzido, e o botão `Parar`, com o qual se interrompe a reprodução da música. Cada vez que clicarmos no `Play`, teremos que selecionar novamente a canção que será tocada. É possível que você queira melhorá-lo, adicionando uma lista de reprodução, um botão para avançar ou retroceder etc. No número 16 da *Linux Magazine*, vimos como usar outros componentes da *List* e armazenar dados em pastas. Com essas informações é possível criar uma lista de reprodução para o nosso “player”.

Para reduzir o tamanho da aplicação, tivemos que omitir alguns pontos, como guardar as exceções. Isso é notado ao se abrir o selecionador de arquivos e, clicando no botão `Cancelar`, abre-se um arquivo que não é um MP3 etc. Esse é um programa exemplo que precisa ainda de depuração.

E, por último, o método `__init__()`, típico das aplicações Java e Jython, que iniciará os valores da aplicação, criando o painel que contém a interface gráfica, e inicializará a variável booleana com a qual será controlada a parada do reprodutor (`Parar`) e o método `exit()`, que permitirá sair da aplicação ao clicarmos no botão para fechar a janela (o típico X, acima, à direita).

Outro aspecto que devemos comentar é o tratamento que o Jython dá às variáveis booleanas, já que não existem os valores `true` e `false`. Para solucionar este problema, ao invés de `true`, utilizamos o valor 1, e 0 no lugar de `false`, que são os valores equivalentes no Java.

As bibliotecas que vamos utilizar são as seguintes: `org.tritonus.share.sampled.TAudioFormat` e `org.tritonus.share.sampled.file.TAudioFileFormat`, que será usado para obter a informação do nosso arquivo MP3. `Java.sound.sample`, onde estão as classes (`SourceDataLine`, `AudioSystem` etc) de que vamos precisar; `ponto.java.io`, para acessar as pastas. A biblioteca `jarray` nos permite criar arrays compatíveis com Java. Por último, `java.util`, `javax.swing`, `java.lang`, `java.awt`, `java.util`, que servirão para o resto de nossas tarefas.

No caso de quisermos reproduzir músicas da Internet, abrimos o endereço criando uma ocorrência URL (devemos acrescentar a biblioteca `java.net`, à qual passaremos o protocolo que vamos usar, o host que deverá ser conectado, o ponto de co-

nexão e o nome da pasta). Preparado o endereço, o objeto URL com o método `openStream` começa a extrair o stream desse arquivo, e temos apenas que passá-lo para nosso reprodutor (com `get.AudioInputStream()`) e logo (dependendo da velocidade e tamanho da conexão) começaremos a escutar a música.

O programa completo pode ser visto na **listagem 1**.

Outro reprodutor

É claro que há uma forma mais fácil de criar um reprodutor. Mas o objetivo deste artigo foi ver como podemos criar, a partir dos elementos do Java, um reprodutor de acordo com nossas necessidades (como o tamanho do buffer). Nas bibliotecas instaladas com o `javazoom`, mais concretamente no `JPlayer`, existe uma classe que cria um reprodutor de MP3 completo, ao qual devemos apenas passar o `InputStream` (com `FileInputStream`) quando o criamos. Ao executar o método `.play()` do respectivo reprodutor, ele se encarregará do resto.

No próximo mês voltaremos com mais scripts em Python e Jython. ■

INFORMAÇÕES

- [1] Jython: www.jython.org
- [2] Javazoom: www.javazoom.net
- [3] Java: java.sun.com
- [4] Tritonux: www.tritonus.org

SOBRE O AUTOR

José Pedro Orantes Casado cursa o 3º ano de Engenharia Técnica em Informática de Sistemas e, há muitos anos, utiliza Linux como ferramenta de trabalho. José Maria Ruíz está terminando seu projeto de conclusão de curso na Faculdade de Engenharia Técnica em Informática de Sistemas, também na Espanha. Usa e desenvolve Software Livre há mais de 7 anos, dois deles no FreeBSD.

Conheça o maior portal para desenvolvedores da América Latina.

www.devmedia.com.br

The screenshot shows the DevMedia website interface within a Microsoft Internet Explorer browser window. The browser's address bar displays the URL <http://www.devmedia.com.br/portal/teste.asp>. The website header features the DevMedia logo and a navigation menu with links for Home, Clube Delphi, SQL Magazine, MSDN Magazine, Web Mobile, JAVR Magazine, Fórum, Cursos, and Edições Anteriores. A yellow banner at the top of the page reads "O resultado pode surpreender você." Below the navigation menu, the page is organized into several columns:

- Dev_Shopping:** Promotes a "Evento à Distância - Introdução ao .NET" with a price of R\$ 99,00 and an "exclusivo - Delphi e Relatórios - Com Delphi 7 e Delphi 2005 Win32!" with a price of R\$ 299,00.
- Portal do Assinante:** Lists recent articles such as "Curso de dbExpress e DataSnap - Parte II" by Günther Paulk, "Função FireDbase" by Everson Volato, and "Clube Delphi DevExpress NavBar" by Luciano Pimenta.
- Últimas Atualizações:** Features updates like "WebMobile Aplicativos - Concurso WebMobile" and "SQL Magazine Firebird no Delphi 2005 - Parte II".
- Books:** Promotes a "Normalização" book by Borland, Edição Nº21, with a description of its content on database normalization.

The browser window also shows the Google search bar and various browser controls like Back, Forward, and Stop buttons.

