Imagens de satélite com Python

Python planetário

Quem nunca quis se sentir como os técnicos da Nasa em seu centro de controle? Hoje vamos criar o nosso próprio para monitorar o planeta e seus arredores.

por José María Ruiz e Pedro Orantes

Chris Lienert – www.sxc.hu

Toda vez que vemos o lançamento de um foguete ficamos assombrados diante da explosão da decolagem, os cientistas olhando atentamente seus painéis de controle e as cifras monstruosas que revelam quanto foi investido no projeto.

Figura 1 A imagem original que será modificada com a biblioteca *PIL*.

Ao nosso alcance

É então que surge a pergunta: "E como isso me afeta?" Certa vez tive uma conversa no escritório da IEEE (*Institute of Electrical and Electronics Engineers*) de Málaga na qual me contaram que a maioria dos satélites transmitem para todo o mundo as imagens e dados que recolhem. Isso significa que, com o equipamento necessário, é possível receber na sua própria casa imagens fascinantes do universo, de Marte ou da Terra.

A temperatura do oceano, imagens meteorológicas do campo magnético do sol e das missões a Marte são enviadas constantemente à Terra a partir dessas máquinas espaciais. E o efeito é sempre o mesmo: o apresentador na televisão deixa os telespectadores deslumbrados com imagens incríveis, enquanto escutam acordes de sintetizador.

Mas essas imagens não são de domínio público? Onde posso consegui-las? Neste artigo, vamos utilizar o *Python* para criar um script *CGI* que nos permite obter e manter atualizadas as imagens que queremos, em uma espécie de colagem ou painel. Construiremos nosso próprio centro de controle espacial.

Obter as imagens

A primeira coisa a fazer será encontrar as imagens e reuni-las. Vamos usar como exemplo quatro imagens de ca-

Curiosidade

Pouco tempo depois de este artigo ter sido finalizado, apareceu uma notícia no *Slashdot* (ver [4]) sobre uma labareda solar de tamanho suficiente para alterar as comunicações. Quando ocorre esse tipo de evento, em muitos centros de controle os engenheiros cruzam os dedos para que seus satélites não caiam ou se percam diante da onda de vento solar gerada. A labareda pode ser vista na **figura 4**.

ráter científico. Elas são atualizadas em intervalos diferentes, de maneira que poderemos ver como os eventos registrados evoluem. As URLs podem ser encontradas em [1].

Devemos baixar as imagens e armazená-las em nosso programa. Faremos uso da biblioteca *httplib* que é parte da distribuição padrão do Python. Essa biblioteca nos permitirá falar diretamente com um servidor web remoto sem termos que nos preocupar com os detalhes de nível mais baixo. A conversa será realizada com uso do protocolo *HTTP*. Esse protocolo é bastante simples e só precisaremos de uma parte mínima dele.

Quando Tim Berners Lee criou o desenho original da Web, quis que o protocolo para solicitação de documentos fosse o mais simples possível. O trabalho do HTTP se resume à recepção e envio de informações ao servidor, apenas isso. É composto de vários comandos, sendo os mais conhecidos o *GET*, que podemos traduzir como "pegar", e o *POST*, "mandar" ou "enviar". É assim que baixamos documentos e enviamos informações.

Uma parte importante do HTTP é a *URL*, que serve para dar nomes a esses documentos. Todos estamos acostumados a lidar com URLs, geralmente do tipo http://www.linuxmagazine.com.br/issue/. A URL é composta de: [protocolo]://[máquina]/[caminho]/[objeto]. Vamos ver agora por que é tão importante que saibamos disso.

A biblioteca *httplib* do Python estabelece o primeiro passo para uma conexão com o servidor remoto através do método `HTTPConnection`.

```
>>> c = httplib.HTTPConnection("www.
linuxmagazine.com.br")
>>>
```

Armazenamos na variável `c` o objeto que representa a conexão realizada, e podemos enviar solicitações.

```
>>> c.request("GET","/issue/20")
>>>
```

Utilizamos o comando `GET`, e assim solicitamos um objeto. O segundo parâmetro do método é o "caminho" até o objeto. A URL que estamos solicitando é http://www.linuxmagazine.com.br/. É importante que o caminho comece com uma barra "/", como se fosse um caminho de diretório numa máquina. Como podemos saber se tudo correu bem?

```
>>> r = c.getresponse()
>>> print r.status, r.reason
200 OK
>>>
```

Com o `getresponse`, obtemos um objeto que representa os dados devolvidos pela conexão. Esse objeto tem, entre outros, os atributos `status` e `reason`, que nos indicam o estado, um número com um significado especial, e a explicação do mesmo. Nesse caso correu tudo bem e por isso recebemos um "OK". Caso contrário, se a rota que pedimos não existisse, obteríamos:

```
>>> r = c.getresponse()
>>> print r.status, r.reason
400 Bad Request
>>>
```

Agora que já temos a página, podemos lê-la usando o método `read()` do objeto com a resposta.

```
>>> print r.read()
<html>
<head>

<base
➥ href="http://www.linuxmagazine.com.br/
➥issue/20/" />

<title>Linux Magazine -   20
...
```

Quando terminarmos, devemos fechar a conexão chamando o método `close()` do objeto que representa a conexão. Nesse caso seria:

```
>>> c.close()
>>>
```

Figura 2 A imagem do pequeno demônio do *BSD* rotacionada em 45 graus com *PIL*.

Para obter mais imagens, vamos fazer exatamente a mesma coisa: abrir uma conexão, pedir a imagem, armazená-la em um dicionário e fechar a conexão.

Python Imaging Library

Nossa idéia original era fazer um mural ou colagem com as imagens obtidas. O Python não oferece uma biblioteca de tratamento de imagens na sua distribuição padrão. Isso não quer dizer que não exista tal biblioteca. Não só ela existe, como além disso é muito útil e poderosa. Estamos nos referindo à *Python Imaging Library* (ver URL [2] em **Mais informações**).

A biblioteca *PIL* (Python Imaging Library) nos permitirá tratar imagens em uma grande quantidade de formatos. Podemos convertê-las para outro formato, rotacioná-las, medi-las, fazer fusões etc. Quem já tiver experiência com progra-

Exemplo 1: Exemplo de uso do PIL

```
01 >>> painel = Image.new('RGB',(600,480))
02 >>> im = Image.open("daemon.jpg")
03 >>> im.thumbnail((300,200),Image.ANTIALIAS)
04 >>> painel.paste(im,(0,0))
05 >>> painel.paste(im,(300,0))
06 >>> painel.show()
07 >>>
```

Exemplo 2: colagem.conf

```
01 [tamanho]
02 horizontal = 800
03 vertical = 600
04
05 [imagens]
06 url1 = http://www-mgcm.arc.nasa.gov/MarsToday/marstoday.gif
07 url2 = http://www.sec.noaa.gov/sxi/current_sxi_4MKcorona.png
08 url3 = http://www.ssec.wisc.edu/data/sst/latest_sst.gif
09 url4 = http://www.wetterzentrale.de/pics/D2u.jpg
```

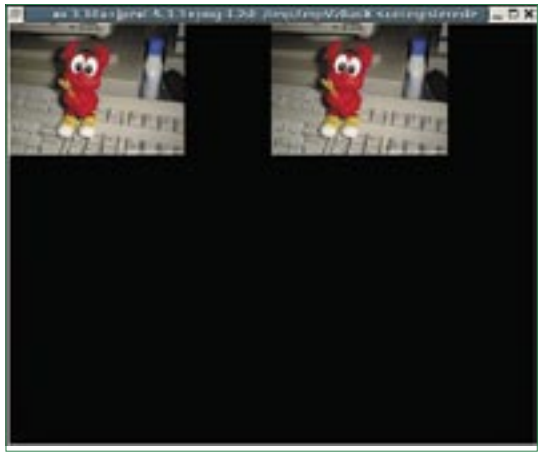

Figura 3 Criamos uma imagem vazia com *PIL* e depois inserimos outras imagens dentro dela como em um mosaico.

mas de manipulação gráfica, como por exemplo o *GIMP* [3], vai compreender a importância de uma biblioteca com essas funcionalidades.

Como o PIL não é um "item de série" do Python, devemos instalá-lo em nossa distribuição. Há pacotes *RPM* e *DEB* da biblioteca.

Como se trabalha com PIL? Através da manipulação de objetos da classe `Image`. Essa classe é capaz de armazenar imagens de quase todos os formatos, permitindo-nos manipulá-las. Vejamos um exemplo. Na **figura 1** podemos ver a imagem original do arquivo `daemon.jpg` da minha equipe. Vamos rotacioná-la em 45 graus:

```
>>> import Image
>>> im = Image.open("daemon.jpg")
>>> img.rotate(45).show()
>>>
```

Na **figura 2** podemos ver o resultado. Usamos o método `rotate()`, ao qual passamos um ângulo de 45 graus, e no resultado invocamos o método `show()`, que mostrará o resultado com o programa *xv* (para fechar o xv temos apenas que digitar `q`).

Mas no nosso exemplo não queremos rotacionar imagens, e sim medi-las. As imagens da web são grandes e nós queremos criar um painel de tamanho fixo. Então devemos adaptar as imagens baixadas para que caibam no mural.

Para isso, vamos inserir as imagens em uma maior, mas há muitas maneiras de se fazer isso. A solução que adaptaremos para nosso caso é dividir a imagem-painel num número de quadros igual às imagens que vamos inserir. Mas como saberemos a quantidade de quadros? Vamos escolher a menor potência de 2 que seja maior que nosso número de imagens. Não é muito complicado. Por exemplo, se temos 7 imagens, 8 quadros (2 elevado a 3) serão suficientes. Basicamente multiplicaremos 2 por ele mesmo até que seja maior que o número de imagens que queremos mostrar. Graficamente o que faremos será ir dividindo a imagem em largura e altura em quadrados. Em cada iteração o número de quadrados será multiplicado por 2. Com esse método, perderemos espaço na imagem, mas será tão pouco que isso não complicará muito o código.

Criemos o thumbnail

Primeiro criaremos uma imagem vazia (veja o **exemplo 1**). A **figura 3** mostra o resultado. Desta vez não carregamos nenhuma imagem, mas usamos o método `new()`, que necessita que definamos o tipo de pixel ("RGB" vem de *Red,Green,Blue*, vermelho,verde, azul, e é um dos formatos padrão) e as dimensões da imagem em pixels. No nosso caso escolhemos 600 pixels de largura por 480 de altura (preste atenção nos parênteses, porque a resolução é expressa como uma seqüência do tipo `(x, y)`). Essa nova imagem não contém nada além de um decepcionante fundo preto. Vamos colocar um pouco de cor!

Pegamos a imagem do diabinho e chamamos o método `thumbnail()`, que mede a imagem tanto vertical quanto horizontalmente. Temos que passar ao método o tamanho desejado como uma seqüência. A nova imagem terá um tamanho de 300x200 pixels. Ela pode aceitar um parâmetro adicional, que em nosso caso é `Image.ANTIALIAS`, que serve para melhorar a resolução da nova imagem.

Em seguida, usamos o método `paste()` de `Image`, que nos permite "colar" uma imagem dentro da outra nas coordenadas indicadas como segundo parâmetro. Colamos a imagem "daemon" duas vezes, a primeira na posição (0,0) do mural e a segunda na posição (300,0). Podemos ver o resultado usando o método `show()`.

Arquivo de configuração

As URLs e a resolução devem ser informadas ao programa, mas como? Há várias opções. Elas poderiam ser passadas para o programa em sua execução. Algumas URLs têm o problema de serem muito longas, e assim a linha de comando para executar o programa pode ser um empecilho.

Ao invés disso, usaremos um arquivo de configuração. Cada vez que o programa for executado, lerá o arquivo e saberá os parâmetros desejados.

E que formato terão os arquivos? A tendência atual é criar arquivos de configuração *XML*. Mas o XML pode ser bastante complicado se levarmos em consideração que nosso arquivo de configuração pode não ter mais de 10 linhas. No UNIX, a tendência é usar o formato "chave = valor" e é exatamente isso que faremos. O arquivo será como mostra o **exemplo 2**.

Leremos cada linha do arquivo, dividindo-a pelo `=` e usando a primeira parte como chave de um dicionário e a segunda parte como seu valor. Se a chave já existir, usaremos uma lista como seu valor, com os diferentes valores da linha como entrada. Mas por que faríamos o trabalho sujo se alguém já tem uma solução?

O Python traz em sua distribuição padrão uma biblioteca de grande utilidade para nós. Alguém achou oportuno elaborar um analisador de arquivos de configuração, e chamou-o de *ConfigParser*. Com essa biblioteca podemos extrair a informação do arquivo de configuração.

O arquivo de configuração é composto por "Seções" e "Opções". Cada seção contém várias opções, sendo que

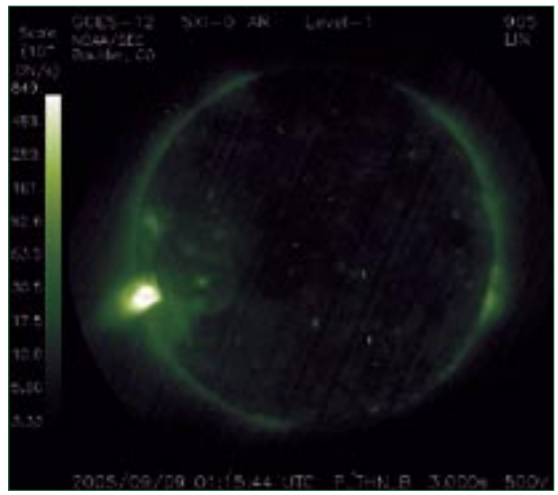

Figura 4 Labaredas solares que ameaçam deixar fora de combate os satélites de comunicação.

os nomes das seções e opções devem ser únicos. Por isso as URLs começam com "url1", "url2" e "url3". Mas isso não será um problema. Vejamos como funciona o *ConfigParser* (confira o **exemplo 3**). Como podemos observar no exemplo, o uso do ConfigParser é muito simples. Primeiro é criado um analisador, guardando-o na variável `config`. Depois carregamos com o método `readfp()` o arquivo de configuração – esse método também analisa o arquivo. A partir desse momento, podemos fazer perguntas ao objeto armazenado em `config`. Com `sections()` obtemos uma lista de seções e, com `options()`, a de opções. Com essa informação, já podemos recolher os dados necessários usando o método `get()`, ao qual passamos uma seção e uma opção.

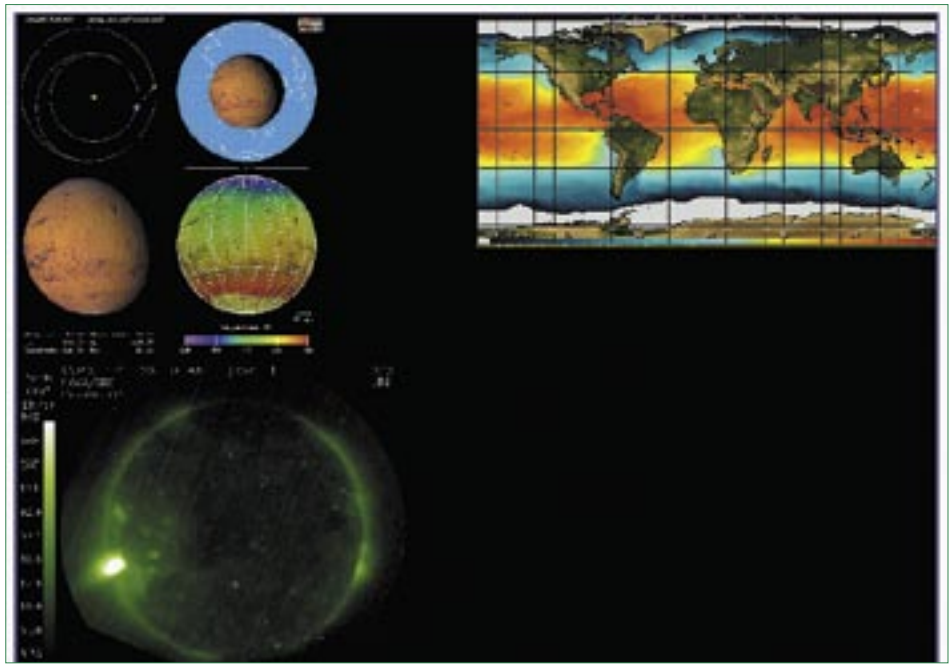

Figura 5 Nosso painel de controle espacial pronto e uma página web gerada dinamicamente.

Encaixe

Agora já temos:

- Um sistema de configuração usando *ConfigParser*.
- Um sistema para descarregar as imagens, usando *httplib*.
- Um sistema para manipular as imagens usando *PIL*.

Resta então juntar tudo para que possamos gerar a página que aparece na **figura 5**. O resultado final pode ser baixado de [5].

Criaremos uma classe `Colagem` com os métodos:

- `_carregaConf()`
- `_baixa()`
- `_totalXY()`
- `geraColagem()`
- `_geraImagem()`
- `_geraHTML()`

Quando um método começa com `_`, torna-se privado. Qualquer tentativa de uso desse método gerará uma exceção. Por isso esses métodos não podem ser chamados de fora do objeto `Colagem`. Dessa maneira, `Colagem` só possui um método acessível de fora: `geraColagem()`. A geração do HTML foi separada daquela da colagem para possibilitar as futuras extensões do objeto. Por exemplo, poderíamos não querer gerar um arquivo HTML, mas incorporar a imagem em um programa. Nesse caso, herdaríamos de `Colagem` e criaríamos um novo método `geraColagem()`, que só gerasse e retornasse a imagem.

O método `_geraHTML()` gera o código HTML da página web. Um ponto importante é que gera um mapa sobre a colagem, de maneira que seja possível clicar sobre as diferentes imagens que aparecem nele. Ao fazer isso, a imagem será carregada no tamanho natural. O mapa é gerado consultando-se o dicionário de imagens. Cada entrada do dicionário contém um objeto da classe `Imagem`.

`Imagem` guarda a informação de cada imagem baixada enquanto o programa a armazena. São armazenados os dados próprios de cada imagem, como, por exemplo, as coordenadas que ocuparão finalmente a colagem.

Como sempre, espera-se que o leitor dedique algum tempo para testar o programa para adaptá-lo às suas necessidades ou idéias.

Conclusão

A complexidade de um programa em Python não está na quantidade de linhas de código, e sim no nível em que se trabalha. No programa deste artigo, fizemos uso intensivo das bibliotecas que realizaram ações bastante complicadas por nós. O Python possui um amplo leque de bibliotecas a serem exploradas, muitas delas com anos de desenvolvimento, esperando por programadores com idéias originais para pô-las em prática. ■

Mais Informações

[1] Imagens utilizadas: http://www-mgcm.arc.nasa.gov/MarsToday/marstoday.gif; http://www.sec.noaa.gov/sxi/current_sxi_4MKcorona.png; http://www.ssec.wisc.edu/data/sst/latest_sst.gif; http://www.wetterzentrale.de/pics/D2u.jpg

[2] Python Imaging Library: http://www.pythonware.com/products/pil/

[3] GIMP: http://www.gimp.org

[4] Notícia sobre a labareda solar no Slashdot: http://science.slashdot.org/science/05/09/08/1933205.shtml?tid=215&tid=14

Exemplo 3: Uso do ConfigParser

```
>>> config = ConfigParser.ConfigParser()
>>> config.readfp(open('colagem.conf'))
>>> config.sections()
['tamanho', 'imagens']
>>>
>>> config.options('imagem')
['url1', 'url3', 'url2']
>>>
>>> config.get('imagem','url1')
'"http://www-mgcm.arc.nasa.gov/MarsToday/marstoday.gif"'
```