

Contents

- [1 Introdução](#)
- [2 Módulo urllib](#)
- [3 Serialização de objetos](#)
- [4 Twitter API](#)
- [5 Exemplo de aplicação](#)
 - ◆ [5.1 Screenshots](#)
- [6 Source Code](#)
- [7 Referências](#)

Introdução

Vários Web Services recentes, como Twitter, Flickr and Yahoo! Maps, vêm sendo apresentados com REST como opção de acesso. A Transferência de Estado Representacional (Representational State Transfer - REST), como nomeador pelo seu autor, [Roy Fielding](#), é definido como um estilo de arquitetura distribuída para sistema de hipermídia. Mais do que as pessoas comumente rotulam como "web service", *REST enfatiza as interações entre componentes, a generalização das interfaces, independência de implantação de componentes e componentes intermediários para reduzir a latência de interação, reforça a segurança e encapsula sistemas legados* (Roy Fielding).

Algumas vantagens são geralmente creditadas aos serviços REST, como:

- Suporte a cache, diminuindo o tempo de resposta e diminuindo a carga do servidor.
- Não são necessárias sessões, permitindo uma melhor distribuição de carga entre vários servidores].
- Arquitetura cliente-servidor, com poucos requerimentos para o lado cliente. Em geral, somente um navegador é necessário.
- No longo prazo, melhor compatibilidade.

Devido à "natureza web" intrínseca do REST, um módulo Python bem adequado para acessar serviços REST é o urllib. Algumas características do urllib são descritas na próxima seção.

Módulo urllib

urllib é um módulo Python versátil para coleta de dados através da Internet. Tem várias características interessantes, tais como:

- permite abrir URLs com uma interface semelhante a encontrada em operações de arquivo.
- Funções para processamento de URL, tais como suporte a caracteres HTML especiais (*HTML escaping*) e processamento de parâmetros.
- Suporte a proxy e autenticação HTTP.

urllib permite a criação de programas poderoso. Por exemplo, suponha que se deseje buscar o conteúdo da página "<http://wiki.forum.nokia.com/>" e salvá-lo no arquivo local "**wikiforumnokia.html**". Isto pode ser feito com a urllib com algumas linhas de código:

urllib_demo1.py

```
import urllib

# returns a file like interface
furl = urllib.urlopen("http://wiki.forum.nokia.com/")
# reading the "file"
contents = furl.read()
# saving the page contents
flocal = open("wikiforumnokia.html", "wt")
flocal.write(contents)
```

Ou, se preferir, o método **urlretrieve()** pode realizar esta tarefa com apenas duas linhas de código:

urllib_demo2.py

```
import urllib
urllib.urlretrieve("http://wiki.forum.nokia.com/", "wikiforumnokia.html")
```

Na verdade, **urlopen()** executa uma requisição HTTP do tipo GET e recupera o conteúdo da página desejada, removendo o cabeçalho HTTP. Se parâmetros adicionais são necessárias para a sua requisição, eles podem ser acrescentados na URL, como em uma requisição GET típica.

urllib_demo3.py

```
import urllib
params = urllib.urlencode({'name': 'My name is bond', 'phone': '007007'})
url = "www.exemplo.com/yourname?" + params
print url
```

A saída é uma URL com todos os parâmetros codificados, como você pode ver na barra de endereços quando pesquisa no Yahoo! ou no Google.

```
>>> 'www.exemplo.com/yourname?phone=007007&name=My+name+is+bond'
```

É possível simular uma requisição HTTP do tipo POST também. Requisições POST não usam a URL para codificar os parâmetros. Em vez disso, os parâmetros são incluídos no corpo do html do pedido. Os formulários html são um bom exemplo de requisições POST, onde todos os parâmetros do formulário estão escondidas no interior do corpo do html.

Pode-se acessar a URL **www.exemplo.com** com a mesma argumentação anterior, mas agora usando POST, conforme demonstrado no seguinte trecho de código.

urllib_demo4.py

```
import urllib

params = urllib.urlencode({'name': 'My name is bond', 'phone': '007007'})
result = urllib.urlopen("www.exemplo.com/yourname", params).read()
```

Neste caso, um parâmetro adicional deve ser fornecido ao **urlopen**, indicando a requisição POST.

Requisições POST podem ser traduzido como operações REST do tipo criar/apagar e requisições GET como operações REST do tipo ler. Eles são métodos básicos para acessar e modificar remotamente os seus recursos REST, como será demonstrado neste artigo.

Um recurso muito interessante do urllib é a autenticação HTTP básica. Com a autenticação HTTP básica é possível especificar as suas credenciais (usuário e senha) para acessar determinadas URLs. O urllib manipula as solicitações de autenticação, dando ao usuário a chance de digitar o seu login e senha. Esse recurso é especialmente útil para a API do Twitter e será demonstrado no exemplo seguinte (a URL do Twitter utilizada será explicado mais tarde, não se preocupe).

urllib_demo5.py

```
import urllib

url = 'http://twitter.com/statuses/friends_timeline.json'
f = urllib.urlopen(url)
d = f.read()
```

Os dispositivos S60 irão pedir ao usuário para entrar o login e a senha, como nas imagens abaixo:



urllib peding o login



urllib pedindo a senha

No entanto, estes diálogos podem ser evitados utilizando-se [FancyURLopener](#), uma classe especial do urllib para tratar certos códigos de resposta do HTTP, como o 401 (autenticação requerida). Neste caso, é necessário sobrecarregar o método `prompt_user_passwd`, utilizando-o para fornecer o login e a senha quando solicitados pelo servidor HTTP. Isto pode ser visto no exemplo a seguir.

urllib_demo6.py

```
import urllib

class _FancyURLopener(urllib.FancyURLopener):
    """ This class handles basic auth, providing user and password
        when required by http response code 401
    """
    def __init__(self, usr, pwd):
        """ Set user/password for http and call base class constructor
        """
        urllib.FancyURLopener.__init__(self)
        self.usr = usr
        self.pwd = pwd

    def prompt_user_passwd(self, host, realm):
        """ Basic auth callback
        """
        return (self.usr, self.pwd)

# Create a customized url opener for handling authentication request
url opener = _FancyURLopener("twitter_username", "twitter_password")
# read friends timeline, using credential provided
f = url opener.open('http://twitter.com/statuses/friends_timeline.json')
# twitter response
d = f.read()
```

Requisições POST/GET são dois elementos importantes na comunicação com a API do Twitter. No entanto, para a comunicação completa, um componente essencial ainda está faltando: serialização de objetos.

Serialização de objetos

A API do Twitter pode enviar resposta usando [XML](#), [JSON](#), [RSS](#) ou [Atom](#). O formato da resposta depende da URL solicitada. Por exemplo, ao solicitar o *timeline* de um usuário e dos seus amigos (*friends timeline*), a seguinte URL é usada:

```
http://twitter.com/statuses/friends_timeline.format
```

Neste caso, quem chama a API deve escolher o formato da resposta, utilizando json, XML, RSS ou Atom no lugar de **format**. Para respostas JSON, a requisição deverá ser:

```
http://twitter.com/statuses/friends_timeline.json
```

A [Notação de Objetos para Javascript](#) (JavaScript Object Notation - JSON) é um formato leve para intercâmbio de dados, facilmente legível por seres humanos e também facilmente analisado e geradas por máquinas. Várias linguagens, incluindo Python, têm uma (ou várias) implementações de JSON. Especificamente para Python para S60, pelo menos três implementações estão disponíveis:

- [json.py](#), por Patrick D. Logan e Ville H. Tuulos.
- [simplejson 2.0.9 in single file](#), portado por [Aapo Rista](#) ([inspirado no trabalho de Marcelo Barros](#)).
- [simplejason](#), portado por [Marcelo Barros](#), com apenas algumas alterações quando comparado com a [implementação original](#) (este artigo irá utilizar esta implementação).

Usando JSON, quase todos os tipos básicos podem ser codificados como strings e enviados através de uma conexão de rede. Dois métodos do simplejson (**dumps** e **loads**) são responsáveis pela codificação dos dados como uma representação JSON e posterior decodificação. Desta forma, as informações enviadas pelo Twitter pode ser recebidas e devidamente processadas.

Como exemplo, considere o seguinte dicionário e sua posterior serialização/desserialização:

json_demo.py

```
import simplejson

d = { "name":"Marcelo", "age": 36, "weight":1.77, "devices":["N95", "E71", "N800"] }
ser = simplejson.dumps(d)

print len(ser)
print type(ser)
print ser

rd = simplejson.loads(ser)

print len(rd)
print type(rd)
print rd
```

A saída da serialização é uma string, pronta para ser transmitida por uma conexão TCP/IP:

```
>>> 81
>>> <type 'str'>
>>> '{"age": 36, "devices": ["N95", "E71", "N800"], "name": "Marcelo", "weight": 1.77}'
```

Esta string pode ser convertida para um objeto Python novamente utilizando o método **loads**, como pode ser visto a seguir:

```
>>> 4
>>> <type 'dict'>
>>> {'age': 36, u'weight': 1.77, u'name': u'Marcelo', u'devices': [u'N95', u'E71', u'N800']}
```

A serialização de objetos é o último elemento para a compreensão da API do Twitter. A próxima seção irá apresentar uma classe base para utilização da API do Twitter.

Twitter API

A [API REST do Twitter](#) é bem documentada e é uma leitura obrigatória para desenvolvedores que pretendem construir ferramentas que acessem o Twitter. Embora este tutorial cubra apenas três métodos, eles são suficientes para uma clara compreensão da API do Twitter API. Os métodos cobertos são os seguintes:

- get_friendstimeline: Retorna os 20 status mais recentes do usuário autenticado e dos seus amigos.
- update: Adiciona um update para o usuário autenticado.
- destroy: Apaga um update.

O código está a seguir. Partes dele já foram explicadas em seções anteriores, tais como a autenticação HTTP e usos do JSON e urllib.

s60twitter.py

```
import urllib
import simplejson as json

class _FancyURLOpener(urllib.FancyURLOpener):
    """ This class handles basic auth, providing user and password
        when required by http response code 401
    """
    def __init__(self, usr, pwd):
        """ Set user/password for http and call base class constructor
        """
        urllib.FancyURLOpener.__init__(self)
        self.usr = usr
        self.pwd = pwd

    def prompt_user_passwd(self, host, realm):
        """ Basic auth callback
        """
        return (self.usr, self.pwd)

class TwitterApi(object):
    """ Twitter API basic class
    """
    def __init__(self, tw_usr, tw_pwd):
        """ Set user/password for twitter
        """
        self._tw_usr, self._tw_pwd = tw_usr, tw_pwd

    def _get_urlopener(self):
        """ Return an urlopener with authentication support
        """
        return _FancyURLOpener(self._tw_usr, self._tw_pwd)

    def get_friends_timeline(self, page=1, count=20):
        """ Return friends timeline for current user
        """
        params = urllib.urlencode({"page":page, "count":count})
        url = "http://twitter.com/statuses/friends_timeline.json?" + params
        f = self._get_urlopener().open(url)
        d = f.read()
        return json.loads(d)

    def update(self, stat_msg):
        """ Update twitter with new status message
        """
        params = urllib.urlencode({"status":stat_msg})
        url = "http://twitter.com/statuses/update.json"
```

```

        f = self._get_urlopener().open(url, params)
        d = f.read()
        return json.loads(d)

    def destroy(self, udpt_id):
        """ Destroy the status specified by udpt_id
        """
        url = "http://twitter.com/statuses/destroy/%s.json" % udpt_id
        f = self._get_urlopener().open(url, "")
        d = f.read()
        return json.loads(d)

```

Cada método é representado como uma URL com o método de codificação no final (json) e alguns possíveis parâmetros. Para **get_friends_timeline**, os parâmetros são adicionados à URL, criando uma requisição GET com autenticação para o servidor do Twitter. O número de página ("page") pode ser especificado a partir de 1, ou seja, a página mais recente e com, no máximo, "count" atualizações do Twitter. A resposta é decodificado utilizando simplejson e um array de updates é retornado para quem solicitou a URL. Cada entrada neste array é um dicionário com chaves representando vários campos do update. Em especial:

- **text**: texto do update
- **created_at**: data do update
- **user**: informação sobre o autor
- **id**: ID do update

É possível enviar um update para o Twitter utilizando o método **update**. Neste caso, uma requisição POST autenticada é utilizada e o texto do update deve ser codificado no corpo do html.

Por último, uma atualização pode ser apagada com método **destroy**. Esta é uma URL especial onde o ID do update é inserido no final. Novamente, uma requisição POST autenticada, mesmo que sem parâmetros.

Mais métodos podem ser implementados utilizando estes três apresentados. Basta estudar a [API REST do Twitter](#).

Exemplo de aplicação

Uma pequena aplicação que utiliza a TwitterApi está abaixo. Ele usa o [framework básico para criação de interfaces gráficas](#). Basta colocar o seu login (self.twitter_user) e senha (self.twitter_password) do Twitter, copiar os arquivos para cartão de memória (**E:\python**) e executá-lo. Use as teclas de navegação direita/esquerda para mover entre as páginas e as opções de menu para atualizar o timeline, enviar ou apagar um update.

twitter_demo.py

```

# -*- coding: utf-8 -*-
import sys
sys.path.append(r"e:\python")
import key_codes
from appuifw import *
from window import Application, Dialog
from s60twitter import TwitterApi

__all__ = [ "twitter_demo" ]
__author__ = "Marcelo Barros de Almeida (marcelobarrosalmeida@gmail.com) "

```

Trazendo_a_arquitetura_REST_para_dispositivos_S60_com_Python:_um_tutorial_guiado_utilizando_Twitter_API

```
__version__ = "0.0.1"
__copyright__ = "Copyright (c) 2009- Marcelo Barros de Almeida"
__license__ = "GPLv3"

class Notepad(Dialog):
    """ Minimum text editor for writing new updates
    """
    def __init__(self, cbk, txt=u""):
        menu = [(u"Send", self.close_app),
                (u"Discard", self.cancel_app)]
        Dialog.__init__(self, cbk, u"New update", Text(txt), menu)

class TwitterDemo(Application):
    """ Twitter API demo programa
    """
    def __init__(self):
        menu = [(u"Refresh", self.refresh_pages),
                (u"Send update", self.send_update),
                (u>Delete",self.delete),
                (u"Close", self.close_app)]
        self.body = Listbox([(u"", u"")])
        Application.__init__(self, u"What I am doing ...", self.body, menu)
        self.dlg = None
        # current displayed page
        self.page = 1
        self.headlines = [(u"",u"")]
        # timeline, indexed by page number
        self.timeline = {}
        self.last_idx = 0
        self.update_msg = ""
        # Twitter credential here !
        self.twitter_user = "twitter_username"
        self.twitter_password = "twitter_password"
        self.twitter_api = TwitterApi(self.twitter_user,self.twitter_password)
        # Some key binding for better navigation
        self.bind(key_codes.EKeyRightArrow, self.inc_page)
        self.bind(key_codes.EKeyLeftArrow, self.dec_page)
        self.bind(key_codes.EKeyUpArrow, self.key_up)
        self.bind(key_codes.EKeyDownArrow, self.key_down)

    def inc_page(self):
        """ Download an older page
        """
        if not self.ui_is_locked():
            self.page += 1
            self.refresh_timeline()

    def dec_page(self):
        """ Download a newer page
        """
        if not self.ui_is_locked():
            self.page -= 1
            if self.page < 1:
                self.page = 1
            else:
                self.refresh_timeline()

    def key_up(self):
        """ Update title with current position
        """
        if not self.ui_is_locked():
            p = app.body.current() - 1
```

```

        m = len( self.headlines )
        if p < 0:
            p = m - 1
        self.set_title(u"%d/%d] Page %d" % (p+1,m,self.page))

def key_down(self):
    """ Update title with current position
    """
    if not self.ui_is_locked():
        p = app.body.current() + 1
        m = len( self.headlines )
        if p >= m:
            p = 0
        self.set_title(u"%d/%d] Page %d" % (p+1,m,self.page))

def refresh_pages(self):
    """ Clear all displayed updates and refresh with page again
    """
    self.timeline = {}
    self.page = 1
    self.refresh_timeline()

def refresh_timeline(self):
    """ Update timeline info, download current page
    """
    if not self.timeline.has_key(self.page):
        self.lock_ui(u"Downloading page %d..." % self.page)
        try:
            self.timeline[self.page] = self.twitter_api.get_friends_timeline(self.page)
        except:
            note(u"Impossible to download page %d..." % self.page,"error")
            self.unlock_ui()
            self.refresh()
            return
        self.unlock_ui()

    self.headlines = []
    # First line: author infor and reply info
    # Second line: update text
    for msg in self.timeline[self.page]:
        r1 = msg[u'user'][u'screen_name']
        if msg[u'in_reply_to_screen_name']:
            r1 += u" to %s" % msg[u'in_reply_to_screen_name']
        r2 = msg[u'text']
        self.headlines.append((r1,r2))
    self.last_idx = 0
    self.refresh()

def send_update_cbk(self):
    """ Send a new update (dialog callback)
    """
    if not self.dlg.cancel:
        msg = self.dlg.body.get()
        self.set_title(u"Sending update...")
        try:
            # Twitter uses UTF-8
            self.twitter_api.update(msg.encode('utf-8'))
        except:
            note(u"Impossible to send messages. Try again", "error")
            self.unlock_ui()
            return False
    self.unlock_ui()

```

```
        self.refresh()
        return True

    def send_update(self):
        """ Send a new update
        """
        self.dlg = Notepad(self.send_update_cbk, self.update_msg)
        self.dlg.run()

    def delete(self):
        """ Delete an update
        """
        idx = self.body.current()
        try:
            updt_id = self.timeline[self.page][idx][u'id']
        except:
            return
        self.lock_ui(u"Deleting ...")
        try:
            self.twitter_api.destroy(updt_id)
        except:
            note(u"Impossible to delete update. Try again", "error")
        else:
            del self.timeline[self.page][idx]
            del self.headlines[idx]
        self.unlock_ui()
        self.refresh()

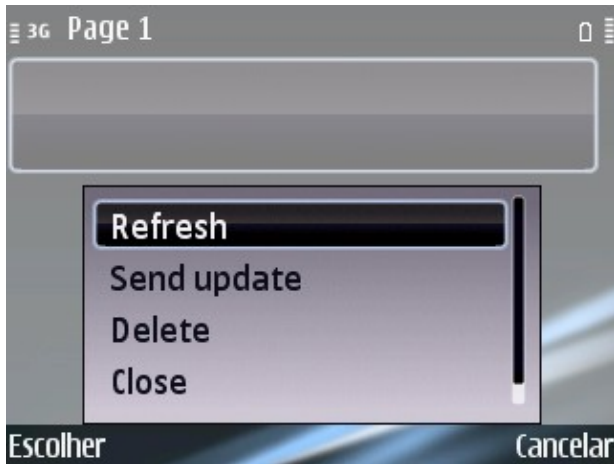
    def refresh(self):
        """ Refresh Listbox with current statuses
        """
        Application.refresh(self)
        idx = self.body.current()
        if not self.headlines:
            self.headlines = [(u"", u"")]
        self.last_idx = min( self.last_idx, len(self.headlines)-1 )
        app.body.set_list(self.headlines, self.last_idx)
        self.set_title(u"Page %d" % (self.page))

if __name__ == "__main__":

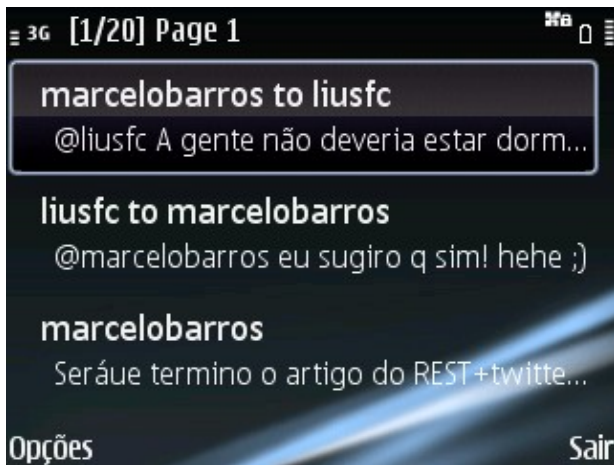
    imd = TwitterDemo()
    imd.run()
```

Screenshots

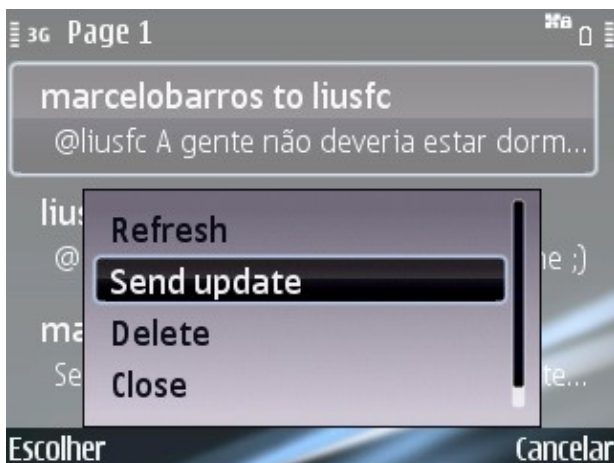
Alguns screenshots deste demo:



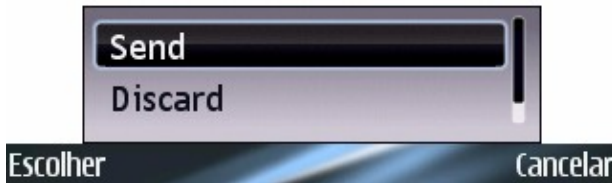
Menu inicial, com a opção refresh em destaque. Esta opção irá chamar `get_friends_timeline()`.



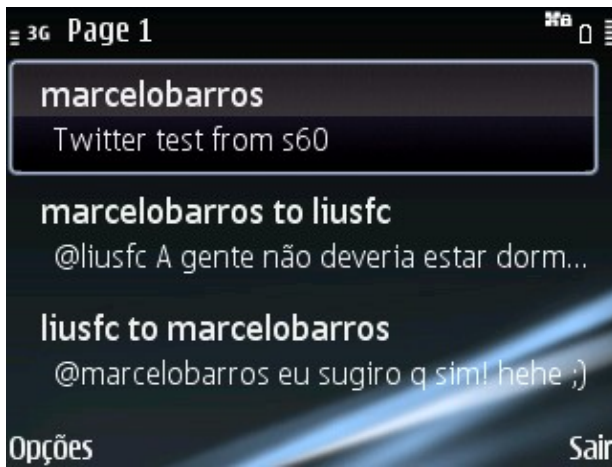
Timeline do Twitter.



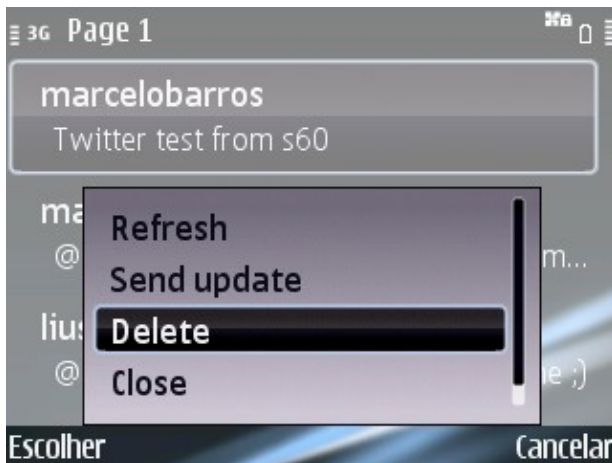
Menu Update.



Criando e enviando um novo update.



Novo update no timeline (use a opção refresh antes).



Apagando um update.

Source Code

Baixe o código fonte de todos os exemplos apresentados neste artigo: [MBA_twitter_demo_src.zip](#)

Referências

Original em Inglês: [Bringing REST architecture to S60 devices with Python: a guided tutorial using Twitter API](#)