Web Services Made Easy

Christophe de Vienne
@cmdevienne

Introduction

- Web Service Made Easy (WSME) permet d'implémenter simplement des services web au sein d'une application web Python.
- Réécriture de TGWebServices mettant l'accent sur :
  - Extensibilité et modularité
  - Indépendance vis-à-vis du framework
  - Meilleure gestion des types
- Démarré en 2011
- Version 0.4b1 disponible

Caractéristiques

- Simple à utiliser
- Multi-protocoles
- Gestion native des types de base (texte, nombres, dates, listes...)
- Gestion des types complexes
- Simple à documenter
- Indépendant du framework web (s'intègre dans différents framework)
- Extensible (ajout de types, protocoles)
- Compatible python 2 (>= 2.5) et python 3 (testé 3.2)

Simple à utiliser

On définit une API en créant une classe dérivée de WSRoot dont on expose des fonctions :

```
from wsme import WSRoot, expose, validate


class MyService(WSRoot):
    @expose(unicode)
    @validate(unicode)
    def hello(self, who=u'World'):
        return u"Hello {0} !".format(who)
```

On expose cette racine dans notre application. Exemple en WSGI :

```
from myservice import MyService

application = MyService(protocols=['restjson']).wsgiapp()
```

Multi-protocoles

La fonction "hello" peut-être appelée, suivant les protocoles activés :

URL	Retourne
`http://<server>/hello.json?who=you`	`"Hello you !"`
`http://<server>/hello.xml`	`<result>Hello World !</result>`
`http://<server>/api.wsdl`	Fichier WSDL pour un client SOAP

Protocoles implémentés à l'heure actuelle :

- REST+Json
- REST+XML
- SOAP
- ExtDirect (de Sencha ExtJS)

Gestion des types

- Types natifs (texte, nombres, dates...)
- Types simples (binaires, énumération, fichier)
- Types 'utilisateur' : définir son propre encodage pour des données de base ou des types variés.
- Types complexe : définir des structures

Simple à documenter

Module Sphinx : `wsme.sphinxext`, ajoute un domaine `wsme` qui définit de nouvelle directives.

Directives simples

```
.. wsme:type:: UnType

    Documentation du type ``UnType``

    .. wsme:attribute:: aname

        Documentation de l'attribut ``aname``

.. wsme:service:: name/space/ServiceName

    Documentation du service ServiceName

    .. wsme:function:: afunction

        Documentation de la fonction ``afunction``
```

Directives 'auto'

Utilisent les docstrings pour documenter automatiquement, en ajoutant des exemples de données encodées pour les différents protocoles activés.

```
.. wsme:autotype:: myapp.MyType

    .. Génère la documentation du type ``myapp.MyType``

.. wsme:autoservice:: myapp.MyService

    .. Génère la documentation du controlleur ``myapp.MyService``
```

Exemple

```python
class SampleService(wsme.WSRoot):
    @wsme.expose(SampleType)
    @wsme.validate(SampleType, int, str)
    def change_aint(data, aint, dummy='foo'):
        """
        :param aint: The new value

        :return: The data object with its aint
                 field value changed.
        """
        data.aint = aint
        return data
```

service /

function **change_aint**(*data*, *aint*, *dummy='useless'*) → SampleType

Parameters samples:

REST+Json

```
{
    "aint": 5,
    "data": {
        "aint": 10
    },
    "dummy": "samplestring"
}
```

REST+Xml SOAP ExtDirect

Return samples: REST+Json REST+Xml SOAP ExtDirect

Parameters:

- **data** (SampleType) –
- **aint** (int) – The new value
- **dummy** (str) –

Returns: The data object with its aint field value changed.

Return type: SampleType

Voir également : http://packages.python.org/WSME/document.html#full-example

Indépendant du framework

- Pas de framework imposé. N'importe quel conteneur WSGI fait l'affaire.
- Des adaptateurs permettent une intégration plus efficace dans certains cas (Turbogears 1).

Extensible

- Ajout de nouveaux protocoles
- Gestion de types personnalisés
- Encodage spécifique de certains types pour certains protocoles

Gestion des types

Quelques détails sur la gestion des types.

Types "natifs"

`bytes`
: Chaine ascii : `str` en Python 2, `bytes` en Python 3

`text`
: Chaine unicode : `unicode` en Python 2, `str` en Python 3

`int`
: Entier

`float`
: Flottant

`bool`
: Booléen

`Decimal`
: Décimal à taille fixe (`decimal.Decimal`)

`date`, `time`, `datetime`
: Date (`datetime.date`), Heure (`datetime.time`), Date/Heure (`datetime.datetime`)

Types simples

`binary`

Transfert des objets binaires, généralement encodés en base64 (peut varier pour certains protocoles).

`Enum`

Permet de restreindre les valeurs possibles

`File`

Transfert de fichiers. Supporte les fichiers envoyés par formulaire.

Tableaux & Dictionnaires

Tableaux

Les tableaux sont typés, on les déclare ainsi :

```
class SomeWebService(object):
    @expose([wsme.text])
    def getlist(self):
        return ['a', 'b', 'c']
```

Dictionnaires

Comme pour les tableaux on spécifie le type de données, le type de la clé devant nécessairement être un type natif :

```
class SomeWebService(object):
    @expose({wsme.text: int})
    def getdict(self):
        return {
            'Pierre-Joseph': 1809,
            'Mikhaïl': 1814
        }
```

Types complexes

Permet de définir des structures :

```
Gender = Enum(str, 'male', 'female')
Title = Enum(str, 'M', 'Mrs')
ImageKind = Enum(str, 'jpeg', 'gif')

class Image(wsme.types.Base):
    name = unicode
    kind = ImageKind
    data = binary

class Person(wsme.types.Base):
    lastname = wsattr(str, mandatory=True)
    firstname = wsattr(str, mandatory=True)

    gender = Gender
    title = Title

    birthdate = datetime.date

    hobbies = [str]

    avatar = Image
```

Exemple Json

```
{
    "firstname": "Proudhon",
    "lastname": "Pierre-Joseph",

    "gender": "male",
    "title": "M.",

    "birthdate": "1809-01-15",

    "hobbies": ["Philosophie"],

    "avatar": {
        "name": "20px-Pierre-Joseph_Proudhon.jpg",
        "kind": "jpg",
        "data": "/9j/4AAQSkZJRgABAQEBLAEsAAD//gBQRmlsZSBzb3VyY2U6IGh0dHA6
    }
}
```

Exemple XML

```
<person>
    <firstname>Proudhon</firstname>
    <lastname>Pierre-Joseph</lastname>

    <gender>male</gender>
    <title>M.</title>

    <birthdate>1809-01-15</birthdate>

    <hobbies>
        <item>Philosophie</item>
    </hobbies>

    <avatar>
        <name>20px-Pierre-Joseph_Proudhon.jpg</name>
        <kind>jpg</kind>
        <data>/9j/4AAQSkZJRgABAQEBLAEsAAD//gBQRmlsZSBzb3VyY2U6IGh0dHA6Ly9
    </avatar>
</person>
```

Extension SOAP

- Implémente un sous-ensemble du protocole SOAP 1.1
- Publie une description WSDL de l'api à l'url `/api.wsdl`.
- Activation :

```
root.addprotocol('soap',
    tns='http://example.com/demo',
    typenamespace='http://example.com/demo/types',
    baseURL='http://127.0.0.1:8989/',
)
```

- Exemple de client suds :

```
from suds.client import Client


url = 'http://127.0.0.1:8989/api.wsdl'

client = Client(url, cache=None)
assert client.service.Hello('You') == 'Hello You !'
```

Extension SQLAlchemy

- Définition automatique des types webservices à partir des classes mappées.
- Implémentation automatique d'API CRUD.

Définition automatique des types

A partir d'une classe mappée (dans un module `model`) :

```
class Person(Base):
    id = Column(Integer, primary_key=True)

    firstname = Column(Unicode)
    lastname = Column(Unicode)

    age = Column(Integer)
```

On définit un type webservice :

```
class Person(wsmeext.sqlalchemy.types.Base):
    __saclass__ = model.Person
```

Equivalent à :

```
class Person(wsme.types.Base):
    id = int
    firstname = wsme.text
    lastname = wsme.text
    age = int
```

Et un peu plus...

```
class Person(wsme.types.Base):
    # ...

    def to_instance(self, instance):
        pass

    def from_instance(self, instance, attrs=None, eagerload=None):
        """Copie les valeurs depuis une instance de classe mappée.

        :param instance: L'instance de classe mappée
        :param attrs: Liste des attributs simples à copier
                      (par défaut: tous)
        :param eagerload: Liste des attributs relation à copier
                          intégralement (par défaut: aucun)"""
        pass
```

API CRUD

Définition automatique d'API Create/Read/Update/Delete pour une classe mappée :

```
from wsmeext.sqlalchemy.controllers import CRUDController

class PersonController(CRUDController):
    __saclass__ = model.Person
    __dbsession__ = DBSession


class Root(WSRoot):
    person = PersonController()
```

-> définit les fonctions suivantes :

- `/person/create` (en REST: `PUT` sur `/person`)
- `/person/read` (en REST: `GET` sur `/person`)
- `/person/update` (en REST: `POST` sur `/person`)
- `/person/delete` (en REST: `DELETE` sur `/person`)

Extension ExtDirect

- Implémente le protocole ExtDirect intégralement :
  - Gestion des appels en batch
  - Gestion des appels 'formulaire'
- Compatible ExtJS 4 (v3 pas testée récemment)
- Implémentation rapide de DataStore, notamment à partir d'un modèle SQLAlchemy

```
class Person(Base):
    __saclass__ = model.Person


class PersonController(sadatastore.SADataStoreController):
    __dbsession__ = model.DBSession
    __datatype__ = Person
```

- Génère les sources javascript définissant le DirectStore et les classes Model (bientôt avec les relations entre classes).
- Utilisé avec bottle, un moyen de coder très simplement une application ExtJS / SQLAlchemy : essayez la démo !

A Venir

- XML-rpc, Json-rpc, autres... Contributions bienvenue !
- Sécurité : gestion des droits, en particulier sur WSME-SQLAlchemy. Devra s'intégrer proprement au framework hôtes (idées bienvenues !).
- Parallélisation des appels en batch.
- Tests unitaires : objectif 100% de code coverage.
- WSME-Soap : Se passer de Genshi.

Les outils utilisés

six

Utilitaires de compatibilité Python 2 and 3. http://pypi.python.org/pypi/six/.

WebOb

Wrappers WSGI. http://webob.org/.

Tox

Automatisation des tests. http://tox.testrun.org/latest/.

Sphinx

Outil de documentation. http://sphinx.pocoo.org/.

Jenkins

Serveur d'intégration continue. https://jenkins.shiningpanda.com/wsme/.

Mercurial (& bitbucket)

http://mercurial.selenic.com/. https://bitbucket.org/cdevienne/wsme.

Vim, syntastic & flake8

http://www.vim.org/, https://github.com/scrooloose/syntastic, http://pypi.python.org/pypi/flake8/

rst2pdf

Utilisé pour faire cette présentation. http://rst2pdf.ralsina.com.ar/.

Sources sur https://bitbucket.org/cdevienne/wsme-pyconfr-2012.

En savoir plus

Documentation

http://packages.python.org/WSME

Les sources

https://bitbucket.org/cdevienne/wsme/ https://bitbucket.org/cdevienne/wsme-sqlalchemy/
https://bitbucket.org/cdevienne/wsme-soap/ https://bitbucket.org/cdevienne/wsme-extdirect/

Contact

- http://groups.google.com/group/python-wsme
- touiteur : @cmdevienne
- mél : cdevienne@gmail.com

Sommaire