py2jdbc Documentation

Release 0.0.6dev3

Scott Stephens

Aug 09, 2019

Contents

py2jdbc is a Python DBI 2.0 interface to JDBC databases.

- Documentation: http://py2jdbc.readthedocs.org/
- Source Code: https://github.com/swstephe/py2jdbc
- Download: https://pypi.python.org/pypi/py2jdbc
- Mailing List: http://groups.google.com/group/python-jdbc

Please feel free to ask questions via the mailing list (python-jdbc@googlegroups.com).

To report installation problems, bugs or any other issues please email python-jdbc@googlegroups.com or raise an issue on GitHub.

For an alphabetic list of all functions in the package, see the genindex.

CHAPTER 1

Introduction

1.1 Installation

This package is available from the Python Package Index. If you have pip you should be able to do:

```
$ pip install petl
```

You can also download maually, extract and run `python setup.py install`.

to verify installation, the test suite can be run with pytest, e.g.:

```
$ pip install pytest
$ pytest
```

py2jdbc has been tested with Python version 2.7 and 3.7 under Linux and Windows operating systems.

1.2 Dependencies and extensions

This package is written in pure Python. The only requirement is the six module, for Python 2 and 3 compatibility.

1.3 Design goals

This package is designed to conform to DBI 2.0, with an eye toward working well with database ORM's, like SQLAlchemy.

CHAPTER 2

Java Modified UTF-8 Encoding

2.1 Synopsis

This module creates a Python codecs interface for the Java Modified UTF-8 Encoding, for JNI interface calls. It is slightly different than the UTF-8 encoding.

The differences are:

- The null byte '\u0000' is encoded in 2-bytes rather than 1-byte, so that the encoded string never has an embedded zero-byte.
- Onle the 1-byte, 2-byte, and 3-byte formats are used.
- Supplementary characters are represented in the form of surrogate pairs, which take 6-bytes.

This gives us the following mapping:

Number of bytes	First code point	Last code point	Bits	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
2	U+0000	U+0000	–	11000000	10000000				
1	U+0001	U+007F	7	0xxxxxxx					
2	U+0080	U+07FF	11	110xxxxx	10xxxxxx				
3	U+0800	U+FFFF	16	1110xxxx	10xxxxxx	10xxxxxx			
6	U+10000	U+FFFFF	20	11101101	1010xxxx	10xxxxxx	11101101	1011xxxx	10xxxxxx

To implement as a Python codec, all that is needed is an encode and decode function. The codec is registered by passing a custom function to search for potentially multiple codecs and return the two functions in a CodecInfo object.

Sometimes this encoding is referred to as CESU-8 or Compatibility Encoding Scheme for UTF-16: 8-bit, but changes the way zero bytes ('\x00') are encoded. There doesn't seem to be an official designation for this encoding, and a request to officially added to Python was rejected, so I'll just use "mutf8" or "mutf-8" for my implementation.

2.2 Usage

To use this encoding, you could do this:

```
import codecs
import py2jdbc.mutf8
codecs.register(py2jdbc.mutf8.info)

codecs.encode(u'a string', 'mutf8')
codecs.encode(u'a string', 'mutf-8')
codecs.encode(u'a string', py2jdbc.mutf8.NAME)
```

The *JNI Interface* module registers and imports this module and maps it to `jni.encode()` and `jni.decode()` already, so you could also use it with:

```
from py2jdbc.jni import encode

encode(u'a string')
decode(b'a string')
```

Although JNI will do this automatically for any calls needing a character pointer argument or returning a character poiter result.

2.3 API Reference

CHAPTER 3

JVM Utilities

3.1 Synopsis

This module contains some utilities for finding the JVM dynamic link library, in a mostly portable way, and to figure out a default CLASSPATH setting.

3.2 API Reference

CHAPTER 4

JNI Interface

4.1 Synopsis

A pure Python JNI interface using ctypes. This is mostly a straight-forward mapping of jni.h from C++ to Python's FFI *ctypes*. There is some additional functionality to manage a singleton JVM and an JNIEnv object for each thread.

It should be functional enough that you could use it in any project needing a pure Python JNI interface, but may need some work to be more comprehensive.

More detailed documentation of the C side can be found in the JNI Specifications.

4.2 API Reference

CHAPTER 5

Java Signatures

5.1 Synopsis

To use the *JNI Interface*, functions need to be mapped to things like the return type of functions, or field data type, which come from the method or field signatures.

For example, a field with signature 'Z' should be called with *GetBooleanField*, while a static field with that signature should be called with *GetBooleanStaticField*.

A method with a signature '()B' should be called with 'CallByteMethod' and a static method with the same signature should be called with CallByteStaticMethod'.

To simplify this struture, this module will try to automatically map signatures to the functions that need to be called. It also tries to convert given Python values to a similar type.

For example:

For method signatures:

5.2 API Reference

CHAPTER 6

Wrappers

6.1 Synopsis

This module is a set of of classes which wrap jclass and jobject values, so they can be accessed approximately the same way as Python classes and objects.

1. Each *jni.JNIEnv* object must be tied to the local thread. So this module has an object *ThreadEnv*, which is a thread-specific "singleton". There is one instance per thread.
2. Each *ThreadEnv* object contains a list of classes called *classes*.
3. Each value is a Python wrapper for the class which wraps the jclass, mapping Java methods and fields to the class.
4. If a jobject is encountered, it can be wrapped with an *Instance* of the class, which is a nested class of the class.

For example:

6.2 API Reference

CHAPTER 7

Exceptions

CHAPTER 8

DBI 2.0

8.1 Synopsis

The module *__init__.py* rolls the DBI 2.0 interface from all the prior modules. From this level, there is no longer and JNI/JVM weirdness details and users should be able to use this code just like any other DBI module.

8.2 API Reference

CHAPTER 9

History

Originally, in my day job, I needed to access a Teradata database. After a lot of difficulty getting ODBC for Linux to work in Red Hat, I decided to look at integrating the Teradata JDBC drivers. Previous developers had actually ported everything to Jython, just to accommodate JDBC drivers.

My first approach was to convert my app to pure Python, then create a small Java WebSerice API to serve DBI requests. At first, I used Google's protobuf and ZeroMQ on both Java and Python sides to send messages back-and-forth.

Later, while teaching myself Hadoop, I found Thrift, which contained a server and a message protocol built-in, so I ported both sides to use Thrift on both sides.

Still later, I came across pyjnius, which is a Cython interface to JNI, with class autoloading introspection of classes and methods. However, it was painful to build pyjnius in our Windows environment, and it had some incompatibility problems with Python 3.

Finally I decided to write my own Pure Python interface to JNI, (using built-in ctypes), then wrap enough classes to get a Python interface to JDBC calls that approached DBI 2.0 compliance.

I hope to provide this as a connector, like pyodbc, to SQLAlchemy and other database frameworks.

As an expierment for my own education, I hope to create alternative branches where this package is ported back to Cython and even raw C++ on different branches, as well as trying out Python package distribution.

CHAPTER 10

Changes

10.1 Version 0.0.6

Put search to use PY2JDBC_JAVA_HOME, JAVA_HOME, JDK_HOME before trying to load library in path.

10.2 Version 0.0.5

Hide *signals* from Windows platform, fixes for Python 2.7, (mostly MUTF8).

10.3 Version 0.0.3

Unit tests against Derby database.

10.4 Version 0.0.1

This is the initial release of py2jdbc.

CHAPTER 11

Contributing

CHAPTER 12

Acknowledgements

This is community-maintained software. The following people have contributed to the development of this package:

- Scott Stephens (swstephe)

Development of py2jdbc used a professional version of PyCharm.

CHAPTER 13

Related Work

CHAPTER 14

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

P