Intro to Jython at Rackspace

Jim Baker

jim.baker@rackspace.com

Overview

- What is Jython
- Jython implementation and usage
- Rackspace opportunities
- Status of Jython 2.7 development

About me

- Core developer of Jython
- Co-author of *Definitive Guide to Jython* from Apress
- Software developer at Rackspace
- Formerly, founding team member, Ubuntu Juju
- Lecturer in CS at Univ of Colorado at Boulder, teach Principles of Programming Languages and especially functional programming

Jython

- Why do I care?
- What can it do for me?
- Compatible - CPython is our reference implementation and we use Python's regrtest
- Still maintain performance
- Future possibility: even possible to match or exceed PyPy...
- Most important: easy integration with Java

Jython background

- Implementation of Python for the Java platform
- Compiles to Java bytecode
- From the beginning great Java integration via some cleverness
- Small team of committers
- Under on-and-off development since 1997
- Extensively used

Who uses it?

- Wall Street banks like Nomura Securities
- Lockheed Martin, to build out software avionics like Lego blocks
- BMW, to run its factory production lines building BMWs
- Princeton Plasma Physics Lab, for research on nuclear fusion power
- Most commonly used for glue, to support scripting and more of Java components
- Do not underestimate the power of good glue!
- Like Python as a whole: more functionality moving into the glue

Compatibility

- Over 99% of the standard Python tests from CPython pass
- Some differences, mostly at the corners
- Code where we notice these corners: frameworks!
- But that's what we have been working hard to address
- (More to be covered in status of Jython)

No GIL

```
$ jython27
Jython 2.7b3+ (default:68aaff268c3c, Sep 10 2014, 20:
[Java HotSpot(TM) 64-Bit Server VM (Oracle Corporatio
Type "help", "copyright", "credits" or "license" for 
>>> from __future__ import GIL
  File "<stdin>", line 1
SyntaxError: Never going to happen!
```

Concurrency

- Jython uses standard Java threading model - "free threading"
- Also **free** for Jython in terms of underlying implementation - we get to use `java.util.concurrent` (!)
- Run N threads on compute loads, can be N times faster (depends on algorithm!)
- Thread safe builtins like `dict`, `list`, `set`
- Or `__dict__`
- Or classes you import

Memory model

- Only support mark-and-sweep GC - no ref counting
- Going out of scope does not mean immediate cleanup
- (Although young generation GC support sometimes can look quite similar)
- Best to use `with` or `try-finally` for cleanup, much like with PyPy
- Or in general, because going out of scope works best in scripts

Performance

- In some tests, can be faster than CPython
- But not necessarily. Example: need to figure out why Bottle's `re` expressions runs so slow
- Extant Python code will often depend on micro optimizations. . .
- Then again, readily can use Java from Jython. . .

Using Java from Jython

Importing and using Java packages is simple:

```
from java.util import HashMap

x = HashMap()
x["foo"] = 42
```

Semantic equivalence*

- Where possible, Java objects in Python space are treated as equivalent to Python
- The magic of **duck typing**
- Also works vice versa - Python objects can be used from Java if they implement Java interfaces/extend Java classes
- Even more the case now with Clamp, which supports direct import

`java.util.Map`, duck typed

- Works as if it's a regular `Mapping` object
- Standard Python introspection - eg `dir(HashMap)`
- (Jython console)

Even abstract base classes

Simple *pending* addition to _abcoll.py:

```
if _is_jython:
    import java
    Container.register(java.util.Collection)
    Iterable.register(java.lang.Iterable)
    Iterator.register(java.util.Iterator)
    MutableSequence.register(java.util.List)
    MutableMapping.register(java.util.Map)
    MutableSet.register(java.util.Set)
    Sized.register(java.util.Collection)
```

Implementation - C vs Java

- Where possible, we try to follow C implementation
- But we also can more readily use Python, because of direct Java import
- Example `itertools.permuations`

Code that you don't see

Exception management

```
if (PyErr_Occurred()) {
  if (PyErr_ExceptionMatches(PyExc_StopIteration))
  PyErr_Clear();
else
  return NULL;
}
```

Reference counting and allocations

```
static void
cycle_dealloc(cycleobject *lz)
{
    PyObject_GC_UnTrack(lz);
    Py_XDECREF(lz->saved);
    Py_XDECREF(lz->it);
    Py_TYPE(lz)->tp_free(lz);
}
```

jythonlib

- Can use threadsafe implementation of `java.util.ConcurrentMap`, like Guava's `MapMaker`
- Unlike `weakref.WeakValueDictionary`, which is not threadsafe, such as iteration
- `jythonlib.dict_builder` - currently exposed as a regular `dict`
- Probably should do some additional work on subclass relationships. . .

```
from jythonlib import MapMaker, dict_builder

_threads = dict_builder(
  MapMaker().weakValues().makeMap)()
```

Implementation conclusions

- JVM engineering support is fantastic
- Periodically JVM just gets much faster, especially in how we use its capabilities from Jython
- Example: housekeeping support, such as multithreading in GC, and improving object allocation
- Still need to take advantage of infrastructure like `invokedynamic` bytecode

Rackspace opportunities

- Keystone Jython - wrap existing Java-based identity infrastructure
- Storm for event processing
- Easy customization of Repose

What else and why?

- Rackspace invests in both Python and Java
- Easy to integrate with Jython, in process, with a single import as we have seen
- We know how to deploy and manage Java-based components, especially servlets
- Keystone Jython proves this can be worthwhile

OpenStack

- OpenStack is an open source platform for building out clouds
- Made up components - Nova, Swift, Keystone, . . .
- . . . written in Python
- High quality testing

Problem

- We need to support Keystone v3 for customers and internal apps
- Made a committment to do so by end of 2014
- Solution: Keystone Jython

Alternatives to Keystone Jython

- Port Keystone v3 to Java - we did this with v2
- Port Rackspace Identity to Python
- Wrap Rackspace Identity with JAX-RS to give us REST APIs

Thoughts

- These can be the right approaches!
- But they have potential issues - implementation costs (especially for ports/rewrites), coarse-grained vs fine grained and IPC overhead, server deployment
- Can potentially be complementary - SOA vs library
- Language implementations can give us extreme leverage - Jython is extensively used

Supporting Keystone v3 at Rackspace

- Keystone supports identity backend plugins
- Configurable for running under a wide range of setups
- $\Longrightarrow$ **no code changes in Keystone were required**
- Although we did find and fix some general bugs

Other OpenStack components?

Keystone is a good test case:

- Can be configured without C extension usage
- Does not use eventlets (which uses greenlets)
- Seamless identity support is generally expected by customers

Still applicable to other OpenStack components

- Keystone uses common Oslo libraries
- Package buildout with pbr, which builds upon pip, `site-packages` metadata
- Possible to emulate greenlet model with regular threads - ArtificialTurf package
- Emulation is really possible and even reasonable? **Yes.**

Unified logging with LogBridge

Easy support of SLF4J (Simple Logging Facade for Java):

```
import logging
import logbridge

logger = logging.getLogger('simple_example')
logger.setLevel(logging.DEBUG)
handler = logbridge.SLF4JHandler()
handler.setLevel(logging.DEBUG)
formatter = logging.Formatter(
   '%(asctime)s - %(name)s - %(message)s')
handler.setFormatter(formatter)
logger.addHandler(handler)

logger.error('error message')
...
```

Dependency injection

- Our Java identity infrastucture uses Spring DI
- Spring transforms the bytecode of annotated classes, so cannot directly use Python classes
- Requires wrapping your Python in Java classes
- Clamp project will be able to directly support Java annotations, so will not have to do this
- Plan to implement *real soon now*

Container

- OpenStack components generally run within a WSGI container
- WSGI can be readily mapped to the Java servlet API
- Just need to produce a war file that packages metadata, jars (including Python classes)
- Choose from Tomcat, Glassfish, Jetty, . . . to run your war file

Storm

- Partitioning problems, but with fault tolerance
- Simple integration with Jython (uber jars)
- Direct access to all Storm capabilities, as well as ZooKeeper (via Apache Curator)
- Or write a spout to consume Kafka, Rabbit, . . . using Java APIs from Python
- Just works without more support than Clamp, with reports from the field
- Example usage at Rackspace: decisioning for autoscaling

Repose

- Sits in front of REST APIs to provide rate limiting, other services
- Global counters, integrated with identity
- Can readily incorporate Python customizations via Jython and `javax.servlet.Filter` (which maps to WSGI middleware)

State of Jython

- Jython 2.7.0 is under very active development
- Should have a final release by Q4 - just in time for Keystone Jython!
- **Race through some of these changes** in this status update

Development focus

- Language changes - easy and completed early in the development cycle
- Runtime and libraries - mostly easy, some continuing work
- Ecosystem - current focus

Some recent changes to trunk

- Java 7 JVM is now the minimum version - get to use `AutoCloseable` and other goodies
- Can now mix Python and Java types in the bases of a class when using a metaclass
- Support for `buffer` and `memoryview` types
- Console and encoding support, such as `unicodedata`, IDNA, and CJK support
- Relative star imports
- Many, many small fixes (bz2 support, including tarfile, . . .)
- Finalizer support (`__del__`) for new-style classes

socket-reboot

- Part of beta 3
- Reimplements Python socket/select/ssl modules with Netty 4
- (Netty is a popular performant event loop networking framework for the JVM)
- Jython socket support is now very close to what is seen in Windows

Working in the Netty pipeline

```
class PythonInboundHandler(
  ChannelInboundHandlerAdapter):

    def __init__(self, sock):
        self.sock = sock

    def channelActive(self, ctx):
        self.sock._notify_selectors()
        ctx.fireChannelActive()

    def channelRead(self, ctx, msg):
        msg.retain()  # bump ref count
        self.sock.incoming.put(msg)
        self.sock._notify_selectors()
        ctx.fireChannelRead(msg)
    ...
```

Manage socket blocking and timeout

- All differences between nonblocking vs blocking with optional timeouts managed in one place
- All sockets can be selected on, regardless of blocking/nonblocking state

Enabling code

```
@raises_java_exception
def _handle_channel_future(self, future, reason):
  def workaround_jython_bug_for_bound_methods(_):
    self._notify_selectors()
  future.addListener(
    workaround_jython_bug_for_bound_methods)
  if self.timeout is None:
    return future.sync()
  elif self.timeout:
    self._handle_timeout(future.await, reason)
    if not future.isSuccess():
      raise future.cause()
    return future
  else:
    return future
```

requests package

- Popular client for working with HTTP/HTTPS with beautiful API
- Now works with Jython!
- Handy since it's used by pip
- socket-reboot enables requests

PyPA tooling support

- pip now works (again!) but requires a branch for the moment, including wheel support
- Dependency is completing a small PR against html5lib-python so that it doesn't use isolated UTF-16 surrogates in literals, since this is not actually legal unicode, nor does it work in Jython's UTF-16 based representation.
- Ironically this usage is to detect such illegal use in input streams
- Will also support virtualenv (via venv) and tox
- Plan to bundle pip support via ensurepip backport

Regular expressions

- Performance tuning of Jython's port of `sre` (underlying virtual machine for regular expressions)
- Currently requires expansion of UTF-16 encoded strings into codepoints array
- Memoization of this expansion means beautifulsoup now works with decent performance (no extra $O(n)$ factor)
- Implications for web frameworks like Django using `re`

Jython tools

- Develop tooling outside the usual release schedule and problems of being in core
- Clamp - improve integration of Jython from Java
- Jiffy - support CFFI for Jython
- Fireside - blazing fast WSGI bridge for servlet containers
- Logbridge - Use simple logging facade for Java as a Python logging handler
- What else should we do?

Clamp

- Precise integration with Java
- Java can **directly import** Python modules (at last!)
- Integrates with setuptools to produce jars
- Includes future integration as well with Maven via Aether

Python class, extending Java interfaces

Example:

```
from java.io import Serializable
from java.util.concurrent import Callable

class BarClamp(Callable, Serializable):

  def call(self):
    return 42
```

NB: automatically fills in a reasonable `serialVersionUUID`

Python class, clamped

To import a Python class that you want to import into Java, add a couple of lines:

```
from java.io import Serializable
from java.util.concurrent import Callable
from clamp import clamp_base

BarBase = clamp_base("bar")  # Java package prefix

class BarClamp(BarBase, Callable, Serializable):

  def call(self):
    return 42
```

Clamping your class

Key insight: ahead-of-time builds through setuptools to produce a jar for Java linkage:

```
import ez_setup
ez_setup.use_setuptools()

from setuptools import setup, find_packages

setup(
  name = "clamped",
  version = "0.1",
  packages = find_packages(),
  install_requires = ["clamp>=0.3"],
  clamp = ["clamped"],
)
```

Using from Java

Simply import clamped Python classes into Java code!

```
import bar.clamped.BarClamp;

public class UseClamped {
  public static void main(String[] args) {
    BarClamp barclamp = new BarClamp();
    try {
      System.out.println("BarClamp: " +
        barclamp.call());
    } catch (Exception ex) {
      System.err.println("Exception: " + ex);
    }
  }
}
```

Jiffy

- Provide a CFFI backend for Jython
- CFFI is a simple foreign function interface to C, gives great possible performance
- Jiffy is now pure vaporware
- Cursory examination of `cffi.backend_ctypes` suggests effort is straightforward/modest because of existing `jffi` package

JyNI

- Add JyNI jar to the Java classpath enables C extension API support
- Works for a number of packages, but need to add GC support (!) for anything real
- Sprinted on JyNI in August with its author (Stefan Richthofer) in Aachen Germany

Release schedule

- We have nearly completed bug triage
- Complete beta 4 (final beta)
- Release candidates as needed

Future

- Mostly around performance, Java integration, and of course the usual bug fixes
- Python bytecode compiler for Android, large complex methods
- More hooks for Java integration
- Integrating Zippy to provide PyPy-like performance (requires Graal JVM)
- Java 9 may also add more features to optimize dynamic languages such as *value* types

Jython 3.x?!

- Comes up periodically!
- Would be nice for unicode strings and bytestrings to have direct correspondence to Java
- Delete code!
- Plan to kickoff development at PyCon in Montreal Spring 2015
- Release schedule: we will get there at some point!

Discussion and Questions

- Contact me at jim.baker@rackspace.com
- jython-dev and jython-users mailing lists
- Source at hg.python.org/jython
- Book at jythonbook.com
- Main site jython.org
- Questions?