Jython - Improving Java Integration

Jim Baker, Rackspace

jim.baker@rackspace.com

Problem

- Support large-scale distributed computation systems like Hadoop, Storm, GraphLab

Problem

- Support large-scale distributed computation systems like Hadoop, Storm, GraphLab
- With language of choice

Problem

- Support large-scale distributed computation systems like Hadoop, Storm, GraphLab
- With language of choice
- Example: Jython has seen some success on Hadoop with Pig for defining UDFs

Storm

- “Real-time” complex event processing system

Storm

- “Real-time” complex event processing system
- Runs topology of storms, bolts to process events (“tuples”)

Storm

- "Real-time" complex event processing system
- Runs topology of storms, bolts to process events ("tuples")
- Can support at-least-once, exactly-once semantics

In Jython

Pieces of imports:

```
from backtype.storm import Config, Constants
from backtype.storm.topology import TopologyBuilder
from backtype.storm.topology.base import BaseRichBolt
from backtype.storm.tuple import Fields, Values

from clamp import PackageProxy
```

PolicyBoltnitoringSpout‘


```
class PolicyBolt(BaseRichBolt):

    __proxymaker__ = PackageProxy("otter")

  # noarg constructor; depends on Storm initializing
  # through prepare
  def prepare(self, conf, context, collector):
    self._collector = collector
    self.policies = \
      defaultdict(partial(Policy, age=15.))

  def declareOutputFields(self, declarer):
    declarer.declare(Fields(["asg", "decision"]))

  def getComponentConfiguration(self):
    # Every 5 seconds, make a decision
```

Supporting code


```
from org.python.compiler import CustomMaker

class SerializableProxies(CustomMaker):
  # NOTE: SerializableProxies is itself a java proxy,
  # but it's not a custom one!

  def doConstants(self):
    self.classfile.addField("serialVersionUID",
      CodegenUtils.ci(java.lang.Long.TYPE),
      Modifier.PUBLIC | Modifier.STATIC | Modifier.FIN
    code = self.classfile.addMethod("<clinit>",
      ProxyCodeHelpers.makeSig("V"), Modifier.STATIC)
    code.visitLdcInsn(java.lang.Long(1))
    code.putstatic(self.classfile.name,
      "serialVersionUID",
      CodegenUtils.ci(java.lang.Long.TYPE))
```

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop
- Java, from a classfile perspective

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop
- Java, from a classfile perspective
- For Storm: serializablity (mostly don't care about Kryo optimization - just moving code here), resolution on the `CLASSPATH`

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop
- Java, from a classfile perspective
- For Storm: serializablity (mostly don't care about Kryo optimization - just moving code here), resolution on the `CLASSPATH`
- In general: support a specific output shape

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop
- Java, from a classfile perspective
- For Storm: serializablity (mostly don't care about Kryo optimization - just moving code here), resolution on the `CLASSPATH`
- In general: support a specific output shape
- => clamp project to bind again `__proxymaker__` protocol

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop
- Java, from a classfile perspective
- For Storm: serializablity (mostly don't care about Kryo optimization - just moving code here), resolution on the `CLASSPATH`
- In general: support a specific output shape
- => clamp project to bind again `__proxymaker__` protocol
- One extra piece is linking to the Jython runtime

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop
- Java, from a classfile perspective
- For Storm: serializablity (mostly don't care about Kryo optimization - just moving code here), resolution on the `CLASSPATH`
- In general: support a specific output shape
- => clamp project to bind again `__proxymaker__` protocol
- One extra piece is linking to the Jython runtime
- Don't need to worry about corner cases seen in separate compilation - duck typing still applies AND we are just implementing interfaces

Clamp

- Standard issue - our Jython code needs to work with your system on the JVM
- Java is the standard level of interop
- Java, from a classfile perspective
- For Storm: serializablity (mostly don't care about Kryo optimization - just moving code here), resolution on the `CLASSPATH`
- In general: support a specific output shape
- => clamp project to bind again `__proxymaker__` protocol
- One extra piece is linking to the Jython runtime
- Don't need to worry about corner cases seen in separate compilation - duck typing still applies AND we are just implementing interfaces
- Amenable to invokedynamic - we will get there

Exposing

- Core type of Jython runtime is `PyObject`

Exposing

- Core type of Jython runtime is `PyObject`
- No interface injection => wrapper classes

Exposing

- Core type of Jython runtime is `PyObject`
- No interface injection => wrapper classes
- **Precisely** match Python semantics to Java semantics

Python types

Exposes Java classes with annotations:

```
public PyObject get(PyObject key, PyObject defaultO
    return dict_get(key, defaultObj);
}

@ExposedMethod(defaults = "Py.None", doc = BuiltinD
final PyObject dict_get(PyObject key, PyObject defa
  PyObject o = getMap().get(key);
  return o == null ? defaultObj : o;
}
```

Annotation processor

Reusing our annotation processor, in Jython:

```
def process_class_file(f):
  etp = ExposedTypeProcessor(f)
  for exposer in etp.getMethodExposers():
    generate(exposer)
  for exposer in etp.getDescriptorExposers():
    generate(exposer)
  if etp.getNewExposer():
    generate(etp.getNewExposer())
  generate(etp.getTypeExposer())
  write(etp.getExposedClassName(), etp.getBytecode())
```

Use ASM


```
def generate(exposer):
  writer = ClassWriter(ClassWriter.COMPUTE_FRAMES)
  exposer.generate(writer)
  write(exposer.getClassName(), writer.toByteArray())
```

etc.

Next steps

- Finish support for `invokedynamic`, across the board

Next steps

- Finish support for `invokedynamic`, across the board
- Maybe we could get expose semantics written in Python, same performance as Java

Next steps

- Finish support for `invokedynamic`, across the board
- Maybe we could get expose semantics written in Python, same performance as Java
- Consider using dynalink

Next steps

- Finish support for `invokedynamic`, across the board
- Maybe we could get expose semantics written in Python, same performance as Java
- Consider using dynalink
- We will get there, but highly pragmatic/conservative/also time constrained

Pipeline: treat presentations like code

This presentation uses a nice set of tools:

- github

https://github.com/jimbaker/jvm-language-summit

Pipeline: treat presentations like code

This presentation uses a nice set of tools:

- github
- markdown

https://github.com/jimbaker/jvm-language-summit

Pipeline: treat presentations like code

This presentation uses a nice set of tools:

- github
- markdown
- pandoc

https://github.com/jimbaker/jvm-language-summit

Pipeline: treat presentations like code

This presentation uses a nice set of tools:

- github
- markdown
- pandoc
- beamer

https://github.com/jimbaker/jvm-language-summit

Pipeline: treat presentations like code

This presentation uses a nice set of tools:

- github
- markdown
- pandoc
- beamer
- LaTeX, but only for isolated bits - favor markdown

https://github.com/jimbaker/jvm-language-summit

Pipeline: treat presentations like code

This presentation uses a nice set of tools:

- github
- markdown
- pandoc
- beamer
- LaTeX, but only for isolated bits - favor markdown
- Maybe a good template for your own presentations, feel free to use!

https://github.com/jimbaker/jvm-language-summit