JyNI is a compatibility layer with the goal to enable Jython to use native CPython extensions like NumPy or SciPy. This way we aim to enable scientific Python code to run on Jython. Since Java is rather present in industry, while Python is more present in science, JyNI is an important step to lower the cost of using scientific code in industrial environments.

To create JyNI we took the source code of CPython 2.7 and stripped away all functionality that can be provided by Jython and is not needed for mirroring objects (see below). We kept the interface unchanged and reimplemented it to delegate calls to Jython via JNI and vice versa.

The most difficult thing is to present JNI-jobjects from Jython to extensions such that they look like PyObject* from Python (C-API). For this task, we use the three different approaches explained below, depending on the way a native type is implemented.

1. To reserve the additional memory, allocation is adjusted wherever it occurs. The other lookup-direction is done via a hash map.

2. This is the basic approach. It is not feasible if the C-API provides access to the object’s structure via macros.

   Examples: PyDict, PySlice, PyModule

3. This approach is the only way to deal with types that are unknown to Jython, i.e. types natively defined by an extension.

   Example: PyCFunction

PyObject* is the basic handling of JNI objects. We use this only if the given PyObject* is already defined by an extension.

JyNI is currently available for Linux only. Once it is sufficiently complete and stable, we will work out a cross-platform version compilable on Windows, MacOS and others. The following built-in types are already supported:

- PyString
- PyBool
- PyComplex
- PyFrozenSet
- PyTuple
- PyInt
- PySlice
- PyClass
- PyDict
- PyLong
- PyModule
- PyInstance
- PyList
- PyFloat
- PySet
- PyMethod
- Singletons PyNone, PyNotImplemented, PyEllipsis
- Natively defined types up to some constraints

The function families PyArg_ParseTuple and Py_BuildValue are also supported.

The main goal of JyNI is compatibility with NumPy and SciPy, since these extensions are of most scientific importance.

The next steps toward this goal are support for exceptions, the remaining built-in types and garbage collection. Cross-platform support will be addressed as a subsequent goal.

Examples:
- PyTuple, PyList, PyString
- PyInt, PyLong, PyFloat
- PyComplex

JyNI wraps Java

Python wraps Java

Java wraps Python

objects are mirrored

To lookup the corresponding jobject of a PyObject*, we prepend an additional header before each PyObject in memory. (If a PyGC_Head is present, we prepend our header even before that.)

... jobject, flags, meta info | PyGC_Head, PyObject ...

PyObject* is the basic handling of JNI objects. We use this only if the given PyObject* is already defined by an extension.

JyNI wraps Java

Python wraps Java

Java wraps Python

objects are mirrored

To create JyNI we took the source code of CPython 2.7 and stripped away all functionality that can be provided by Jython and is not needed for mirroring objects (see below). We kept the interface unchanged and reimplemented it to delegate calls to Jython via JNI and vice versa.

The most difficult thing is to present JNI-jobjects from Jython to extensions such that they look like PyObject* from Python (C-API). For this task, we use the three different approaches explained below, depending on the way a native type is implemented.

1. To reserve the additional memory, allocation is adjusted wherever it occurs. The other lookup-direction is done via a hash map.

2. This is the basic approach. It is not feasible if the C-API provides access to the object’s structure via macros.

   Examples: PyDict, PySlice, PyModule

3. This approach is the only way to deal with types that are unknown to Jython, i.e. types natively defined by an extension.

   Example: PyCFunction

PyObject* is the basic handling of JNI objects. We use this only if the given PyObject* is already defined by an extension.

JyNI is currently available for Linux only. Once it is sufficiently complete and stable, we will work out a cross-platform version compilable on Windows, MacOS and others. The following built-in types are already supported:

- PyString
- PyBool
- PyComplex
- PyFrozenSet
- PyTuple
- PyInt
- PySlice
- PyClass
- PyDict
- PyLong
- PyModule
- PyInstance
- PyList
- PyFloat
- PySet
- PyMethod
- Singletons PyNone, PyNotImplemented, PyEllipsis
- Natively defined types up to some constraints

The function families PyArg_ParseTuple and Py_BuildValue are also supported.

The main goal of JyNI is compatibility with NumPy and SciPy, since these extensions are of most scientific importance.

The next steps toward this goal are support for exceptions, the remaining built-in types and garbage collection. Cross-platform support will be addressed as a subsequent goal.

Examples:
- PyTuple, PyList, PyString
- PyInt, PyLong, PyFloat
- PyComplex

JyNI is available under the GNU GPL v. 3 with classpath exception. See www.jyni.org for more details.

Copyright © 2013 Stefan Richthofer