

Extending Python with C/C++

Simeon Miteff

simeon.miteff@gmail.com

Tuks Linux User Group
4 May 2009



Overview

- Differences between Python and C/C++
 - Why extending Python in C/C++ is useful
 - Spamsun wrapping example
 - The original C code
 - Wrapping with a C Python module
 - Wrapping a C++ class with boost::python
 - Wrapping a C shared library with ctypes
 - Wrapping with Pyrex/Cython code
 - Conclusion
-
-

C/C++ and Python differences

- Dynamic vs. static typing and binding
 - Storage allocation/deallocation
 - Native vs VM execution
 - Python
 - Scripting/Glue/Prototyping language
 - Interactive/Exploratory programming
 - C/C++
 - Heavy lifting: raw speed (Fortran?)
 - Systems programming
-
-

Why Extending is useful

- Existing C/C++ code
 - Leverage available libraries (bindings)
 - Code written before you became a Pythonista
 - Optimization
 - Factor out slow parts of your Python code
 - Number crunching (the same thing zillions of times)
 - Transaction processing (every microsecond counts)
 - “Embedding” Python considered harmful
 - Complicated and platform specific
 - Changes the programming environment
 - There are very few good reasons to embed
-
-

Spamsum example

- Robust message digest: comparing spam
- Andrew Tridgell's C program
 - spamsum function
 - spamsum_match function
- Wrapping approach
 - choose a Python extension API
 - provide equivalent functions or...
 - think in terms of objects

pyspamsum_c

- Uses the Python C API
 - static PyObject *func(arg) {} wrappers
 - static PyMethodDef array
 - Py_MODINIT_FUNC
 - 62 lines of code to wrap 2 functions
 - the Python.h API is not very friendly
-
-

pyspamsum_cpp

- OO way of thinking: Message objects
 - Python C API for OO is uninteresting
 - Enter boost::python
 - Uses C++ templates to generate wrapper
 - The C++ PySpamsum class (37 lines)
 - boost::python wrapper (14 lines)
 - define a class
 - add methods
 - compile and link with libboost_python
-
-

pyspamsum_ctypes

- ctypes module new in Python 2.5
 - Interface to C shared libs (.so/.dll)
 - All the magic happens in Python, no C code
 - Automatic type conversion
 - Python wrapper about 6 lines of code
 - import ctypes
 - load shared lib with ctypes.CDLL()
 - annotate exported symbols with arg/ret types
 - call functions like normal Python
-
-

pyspamsum_pyrex

- Cython and pyrex languages: C-esque neutered Python
 - Cython is better, apparently (Python 3.0 support, etc)
 - Compiles to a Python C API module
 - Pyx file is 9 lines
 - cdef extern from “header.h”
 - declare C types and functions
 - write mostly Python functions that call C code
 - run pyrex or cython to produce a C source file
 - build as a normal C extension module
-
-

Conclusion

- Discussion...
- Slides will eventually be on the website, really...
- We need more topics/speakers...
- See you next month!

