Python & C++. The beauty & the beast, dancing together.

C++ python extensions and embedding

Diego Rodriguez-Losada
@diegorlosada

CppCon 2016, Seattle, Wa, USA

20-Sept-2016

Intro

- Extending python with C/C++ extensions
  - Performance
  - Wrapping existing libraries
  - Integrations
- Embeding python in C/C++ apps
  - Python scripting in your app

Python/C&C++ extensions

Python

- ctypes
- cffi (pypy)

C & C++

- Python C API (CPython)
- Pybind11
- Boost.Python

SWIG

- IDL

Python/C API

```
>>import mymath
>>mymath.add(2, 3)
>>5.0
```

Python/C API

```c
// mymath.c
#include <Python.h>  // FIRST, before any other header!!
```

```c
static PyObject *
module_function(PyObject *self, PyObject *args){
    float a, b, c;
    if (!PyArg_ParseTuple(args, "ff", &a, &b))
        return NULL;
    c = a + b;
    return Py_BuildValue("f", c);
}
```

```c
static PyMethodDef MyMethods[] = {
    {"add",  module_function, METH_VARARGS, "Adds two numbers"},
    {NULL, NULL, 0, NULL}
};
```

```c
PyMODINIT_FUNC initmymath(void){ // This NAME COMPULSORY
    (void) Py_InitModule3("mymath", MyMethods,
                          "My doc of mymath");
}
```

Python 3.5

```
static struct PyModuleDef mymathmodule = {
    PyModuleDef_HEAD_INIT,
    “mymath", "My documentation of mymath“,
    -1,
    MyMethods
};

PyMODINIT_FUNC
PyInit_mymath(void){
    return PyModule_Create(&mymathmodule);
}
```

OO with Python/C API

```c
static PyTypeObject noddy_NoddyType = {
PyObject_HEAD_INIT(NULL)
0,                          /*ob_size*/
"noddy.Noddy",              /*tp_name*/
sizeof(noddy_NoddyObject),  /*tp_basicsize*/
0,                          /*tp_itemsize*/
0,                          /*tp_dealloc*/
0,                          /*tp_print*/
… /*MANY MORE*/
0,                          /*tp_str*/
0,                          /*tp_getattro*/
0,                          /*tp_setattro*/
0,                          /*tp_as_buffer*/
Py_TPFLAGS_DEFAULT,         /*tp_flags*/
"Noddy objects",            /* tp_doc */
};
```

OO with Python/C API

```c
static PyObject *
Noddy_new(PyTypeObject *type, PyObject *args, PyObject
*kwds) {
    Noddy *self;
    self = (Noddy *)type->tp_alloc(type, 0);
    if (self != NULL) {
        self->first = PyString_FromString("");
        if (self->first == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->last = PyString_FromString("");
        if (self->last == NULL) {
            Py_DECREF(self);
            return NULL;
        }
        self->number = 0;
    }
    return (PyObject *)self;
}
```

Extensions: Boost.Python & Pybind11

```cpp
//pybind11_math.cpp
#include <pybind11/pybind11.h>

int add(int i, int j) {
    return i + j;
}

namespace py = pybind11;

PYBIND11_PLUGIN(pybind11_math) {
    py::module m("pybind11_math");
    m.def("add", &add);
    return m.ptr();
}
```

```cpp
//boost_math.cpp
#include <boost/python.hpp>

int add(int i, int j) {
    return i + j;
}

namespace py = boost::python;


BOOST_PYTHON_MODULE(boost_math) {
    py::def("add", add);
}
```

E.g.: BOOST_PYTHON_MODULE

```
# if PY_VERSION_HEX >= 0x03000000

#   define _BOOST_PYTHON_MODULE_INIT(name) \
  PyObject* BOOST_PP_CAT(PyInit_, name)()  \
  { \
    static PyModuleDef_Base initial_m_base = { \
        PyObject_HEAD_INIT(NULL) \
        0, /* m_init */ \
        0, /* m_index */ \
        0 /* m_copy */ };  \
    static PyMethodDef initial_methods[] = { { 0, 0, 0, 0 } }; \
 \
    static struct PyModuleDef moduledef = { \
        initial_m_base, \
        BOOST_PP_STRINGIZE(name), \
        0, /* m_doc */ \
        -1, /* m_size */ \
        initial_methods, \
```

Basic OO

```
#include <pybind11/pybind11.h>

struct Food {
    float quantity;
};
struct Water {
    float amount;
};

namespace py = pybind11;

PYBIND11_PLUGIN(pybind11_math) {
    py::module m("pybind11_math");

    py::class_<Food>(m, "Food")
        .def(py::init<>())
        .def_readwrite("quantity", &Food::quantity);

    py::class_<Water>(m, "Water")
        .def(py::init<>())
        .def_readwrite("amount", &Water::amount);
}
```

```
>>import mymodule
>>food = mymodule.Food()
>>food.quantity = 3.5
>>print food.quantity
```

OO

```
struct Looney {
    Looney(const std::string &name_ = "Silvester") : name(name_),
                                                     happiness(0.0f) { }
    void setName(const std::string &name_ = "Tweety") {name = name_;}
    const std::string &getName() const { return name; }

    void give(const Food& food) { happiness += food.quantity; }
    void give(const Water& water) { happiness += water.amount; }

    std::string name;
    float happiness;
};
```

```
>>import mymodule
>>food = mymodule.Food()
>>food.quantity = 3.5
>>duffy = mymodule.Looney(“duffy”)
>>duffy.give(food)
```

OO-Boost.Python

```
namespace py = boost::python;
BOOST_PYTHON_MEMBER_FUNCTION_OVERLOADS(setname_overloads,
                                        Looney::setName, 0, 1)
BOOST_PYTHON_MODULE(boost_math) {

    py::class_<Looney>("Looney",
                        py::init<py::optional<std::string>>())
        .def("setName", &Looney::setName,
            setname_overloads())
        .def("getName", &Looney::getName,
            py::return_value_policy<py::copy_const_reference>())
        .def("give", (void (Looney::*)(const Food &)) &Looney::give)
        .def("give", (void (Looney::*)(const Water &)) &Looney::give);
}
```

OO-Pybind11

```
namespace py = pybind11;

PYBIND11_PLUGIN(pybind11_math) {
    py::module m("pybind11_math");

    py::class_<Looney>(m, "Looney")
        .def(py::init<const std::string &>(),
            py::arg("name") = std::string("Silvester"))
        .def("setName", &Looney::setName,
            py::arg("name") = std::string("Tweety"))
        .def("getName", &Looney::getName)
        .def("give", (void (Looney::*)(const Food&)) &Looney::give)
        .def("give", (void (Looney::*)(const Water&)) &Looney::give);

    return m.ptr();
}
```

STL

```
struct Looney{
    float happiness;
    std::vector<std::string> friends;
};

float average(const std::vector<Looney>& v) {
    return std::accumulate(std::begin(v), std::end(v), 0.0f,
        [](float a, Looney b) { return a + b.happiness;}) /
       v.size();
}

std::set<std::string> collect(const std::vector<Looney>& v) {
    std::set<std::string> result;
    for (Looney p : v)
        result.insert(std::begin(p.friends), std::end(p.friends));
    return result;
}
```

STL-Pybind11

```
PYBIND11_PLUGIN(pybind11_math) {
    py::module m("pybind11_math");

    m.def("average", &average);
    m.def("collect", &collect);

    py::class <Looney>(m, "Looney")
        .def_readwrite("friends", &Looney::friends)

    return m.ptr();
}
```

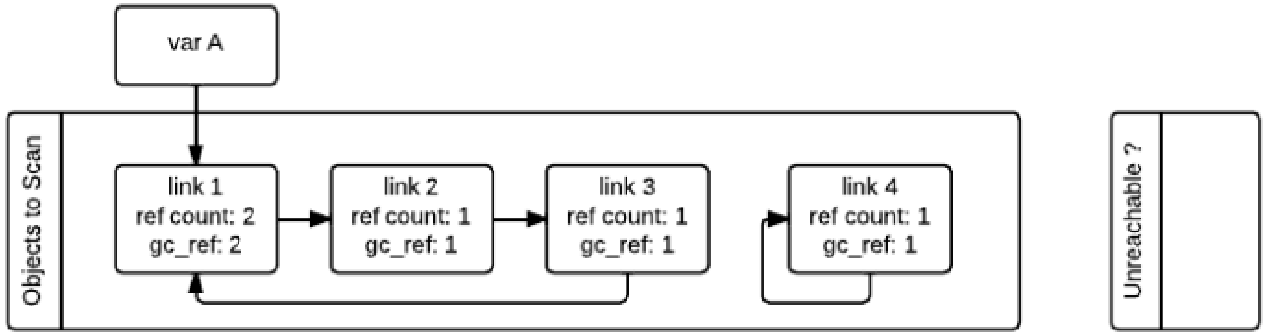
STL-Boost (to_python converter)

```
struct set_to_set{
    static PyObject* convert(const std::set<std::string>& v) {
        PyObject* result = PySet_New(NULL);
        for (const std::string& s : v) {
            PySet_Add(result, Py_BuildValue("s", s.c_str()));
        }
       return result;
    }
};

BOOST_PYTHON_MODULE(boost_math) {

    py::to_python_converter<std::set<std::string>,
                                        set_to_set>();
```

STL – Boost (from_python)

```
iterable_converter()
  .from_python<std::vector<std::string> >()
  .from_python<std::vector<Looney> >()
;
```

Python Auto GC



[1] http://9gag.com/gag/anB2KzE/this-is-how-your-multi-core-cpu-works
[2] https://pythoninternal.wordpress.com/2014/08/04/the-garbage-collector/

E.g: callback

```
def my_log(msg) :
    print "\n MY MSG! ", msg


my_extension.set_log_function(my_log)
```

Callback: set (python/C API)

```
static PyObject *my_log_function = NULL;

static PyObject *
set_log_function(PyObject *dummy, PyObject *args) {
    PyObject *temp;

    if (PyArg_ParseTuple(args, "O:set_log_function", &temp)) {
        if (!PyCallable_Check(temp)) {
            PyErr_SetString(PyExc_TypeError, "param not callable");
            return NULL;
        }
        Py_XINCREF(temp);
        Py_XDECREF(my_log_function);
        my_log_function = temp;
        Py_RETURN_NONE
    }
    return NULL;
}
```

Callback: call (python/C API)

```
static PyObject * my_log_function = NULL;

void call_log(const std::string& msg) {
    PyObject* value = Py_BuildValue("(s)", msg.c_str());

    PyObject* result = PyObject_CallObject(my_log_function, value);

    Py_XDECREF(value);
    Py_XDECREF(result);
}

//extension code
call_log("Hello world log msg")
```

Callback: set (Boost.Python)

```
void set_log_function(PyObject *f){
    Py_XDECREF(log_function);
    Py_INCREF(f);
    log_function = f;
}
```

Pybind11 callbacks

```
std::function<void(std::string)> log_function;

void set_log_function(const std::function<void(std::string)>& f){
    log_function = f;
}

float average(const std::vector<Looney>& v) {
    log_function("Computing the average");
    return std::accumulate(std::begin(v), std::end(v), 0.0f,
        [](float a, Looney b) { return a + b.happiness;}) /
       v.size();
}
```

Pybind11 std::function callback

```
template <typename Return, typename... Args>
struct type_caster<std::function<Return(Args...)>> {
    typedef std::function<Return(Args...)> type;
    typedef typename std::conditional<std::is_same<Return,
        void>::value, void_type, Return>::type retval_type;
public:
    bool load(handle src_, bool) {
        src_ = detail::get_function(src_);
        if (!src_ || !PyCallable_Check(src_.ptr()))
            return false;
        object src(src_, true);
        value = [src](Args... args) -> Return {
            gil_scoped_acquire acq;
            object retval(src(std::move(args)...));
            return (retval.template cast<Return>());
        };
        return true;
    }
```

GIL

- As [David Beazley](#) writes in The Unwritten Rules of Python:
  - *1. You do not talk about the GIL.*
    2. You do NOT talk about the GIL.
    3. Don't even mention the GIL. No seriously.

[1] http://www.slideshare.net/dabeaz/understanding-the-python-gil

Pybind11 std::function callback

```
template <typename Return, typename... Args>
struct type_caster<std::function<Return(Args...)>> {
    typedef std::function<Return(Args...)> type;
    typedef typename std::conditional<std::is_same<Return,
        void>::value, void_type, Return>::type retval_type;
public:
    bool load(handle src_, bool) {
        src_ = detail::get_function(src_);
        if (!src_ || !PyCallable_Check(src_.ptr()))
            return false;
        object src(src_, true);
        value = [src](Args... args) -> Return {
            gil_scoped_acquire acq;
            object retval(src(std::move(args)...));
            return (retval.template cast<Return>());
        };
        return true;
    }
```

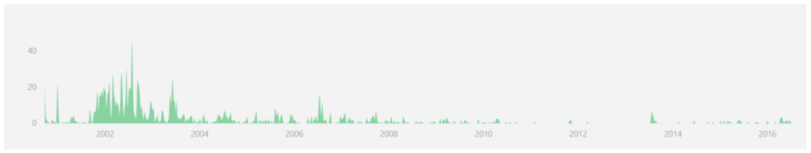
GIL

```
float average(const std::vector<Looney>& v) {
    py::gil_scoped_release release;
    return std::accumulate(std::begin(v), std::end(v), 0.0f,
        [](float a, Looney b) { return a + b.happiness;}) /
              v.size();
}
```

Summary comparison

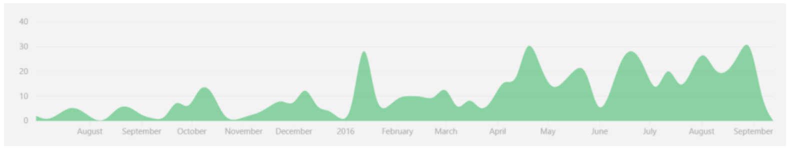
Boost.python

- More manual features: converters
- Github:
  - 51 stars
  - 61 forks
  - 40 contributors
- boostorg.github.io/python/doc/html/index.html
- Compiled library
- Depends on boost



pybind11

- More automagic features: STL, callbacks...
- Github:
  - 1500 stars
  - 150 forks
  - 34 contributors
- pybind11.readthedocs.io/
- Header only



Embedding Python

```c
#include <Python.h>

int main(int argc, char *argv[])
{
    Py_SetProgramName(argv[0]);
    //set PYTHONHOME=C:/python27
    Py_SetPythonHome("C:/Python27");
    Py_Initialize();

    PyRun_SimpleString("print 'Hello World'");

    Py_Finalize();
    return 0;
}
```

Repo!

VS 14, CMake 3.5, Python2.7, git (cmder)

```
$ git clone
    https://github.com/drodri/cppcon2016
```

```
$ pip install conan
```

CONAN
C/C++ package manager

```
$ python test.py
```

Thank you!

Diego Rodriguez-Losada
@diegorlosada