Embedded Python: Boost Python

boost::python은 파이썬 확장(extended python)에 보다 초점을 맞춰져 있는 라이브러리입니다. 완전하게 임베딩을 완전히 지원하지는 않는다고 합니다[1]. 지원하지 않는 부분들은 직접 C API를 활용하여야 합니다. 그렇지만 그것만으로도 훨씬 더 생산성을 높일 수 있습니다. 사용할 만한 가치는 충분합니다.

Boost Python 개선되는 점

자동 레퍼런스 카운팅

PyObject의 레퍼런스 카운팅 개념은 사실 boost::shared_ptr의 개념과 매우 흡사합니다. 다만 C 언어의 한계로 인해 자동으로 객체의 생성, 소멸시 객체가 자동으로 호출되는 함수가 지원되지 않아 결국 그러한 카운팅을 프로그래머가 직접 해야 합니다. 매우 성가시고 불편합니다.

boost::python을 사용하면 `boost::python::object`라는 PyObject의 래퍼를 사용할 수 있습니다. 이를 통해 더이상 프로그래머가 까다로운 레퍼런스 카운팅을 신경쓰지 않아도 됩니다. 변수의 범위를 벗어나면 자동으로 카운터를 내려주고, 다른 레퍼런스에 대입되면 카운터를 올려줍니다. 이것은 일반 포인터 대 스마트 포인터의 사용에 비할 만합니다. `boost::python::object`만으로도 코드는 무척 간결해질 수 있습니다.

자동 레퍼런스 카운팅 테스트

boost 라이브러리야 믿고 쓰면 되는데, 그래도 좀 궁금하더군요. 진짜 자동으로 레퍼런스 카운팅 되는지. 진짜 동작하는지 정말 확인해 보고자 다음과 같은 실험을 해 보았습니다. 좀 바보 같아도 돌다리도 두들겨 보라고. 😎

1. Py_INCREF, Py_DECREF는 매크로입니다. 헤더에 선언되어 있으므로 소스를 재컴파일하지 않아도 헤더 파일에 쓰기 권한만 있으면 접근하여 수정할 수 있습니다. 그러므로 여기 접근하여 살짝 변경해 호출이 되는 것 확인할 수 있도로 합니다.
2. boost::python::object를 생성하여 레퍼런스 카운트를 직접 확인해 봅니다.
3. 종료될 때 명시적으로 Py_DECREF를 호출하지 않아도 호출되는지를 확인합니다.

우선 원래의 파이썬 소스의 헤더를 살짝 변경해 보았습니다.

```
/* object.h */
#include <stdio.h>
#define Py_INCREF(op) do { printf("Py_INCREF\n"); (      \    // do-while 삽입 후 'Py_INCREF' 문자열 출력
    _Py_INC_REFTOTAL  _Py_REF_DEBUG_COMMA                \
    ((PyObject*)(op))->ob_refcnt++); } while(0)

#define Py_DECREF(op)                                    \
    do {                                                 \
```

```
        if (_Py_DEC_REFTOTAL  _Py_REF_DEBUG_COMMA       \
        --((PyObject*)(op))->ob_refcnt != 0)            \
            _Py_CHECK_REFCNT(op)                        \
        else { printf("_Py_Dealloc\n");                 \    // else에 {} 삽
입 후 _Py_Dealloc'문자열 출력
        _Py_Dealloc((PyObject *)(op));  }               \
    } while (0)
```

이렇게 코드를 변경해 두면, 어느 시점에서 레퍼런스 카운터가 늘어나면 "Py_INCREF" 문자열이 출력될 것이고, 레퍼런스 카운터가 0이 되면 "_Py_Dealloc" 문자열이 출력될 것입니다. 그러면 소스 코드를 작성해 봅니다.

boost_pyhthon.cpp

```
#include <boost/python.hpp>
#include <Python.h>
#include <iostream>

int main(int argc, char** argv)
{
  Py_Initialize();
  //PyObject* naive = PyString_FromString("naive_string");
  boost::python::object
    bpobj(
      boost::python::handle<>(
        PyString_FromString("boost::python handling object")));

  boost::python::object
    bpobj_another = bpobj;

  //if (naive)
  //  Py_XDECREF(naive);
  Py_Finalize();
  return 0;
}
```

boost::python이 생성하는 객체를 하나 만들고, 다른 객체와 자원을 공유해 본 다음 프로그램을 종료합니다. 위 코드에서 우리가 직접 관리하는 PyObject 레퍼런스는 주석 처리를 하였습니다. 결과를 비교해 볼 수 있을 것입니다. 이 코드를 실행하면 다음과 같은 결과가 나옵니다.

```
Py_INCREF
Py_INCREF
_Py_Dealloc
```

첫번째 Py_INCREF는 문자열 하나가 생성되면서 객체의 카운트가 1이 될 때 출력된 것이라 추측할 수 있습니다. 두번째 Py_INCREF는 boost::python::object끼리 대입이 되면서 레퍼런스 카운트가 올라간 것이죠. 마지막에 예상대로 객체가 해제되면서 _Py_Dealloc이 불립니다. 확인이 되었습니다.

Try ~ Catch를 이용한 예외 처리

C API를 사용하는 경우, 모든 값에 대해 에러 체크를 명시적으로 수행해서 NULL이 리턴되는지 결과를 확인해야 합니다. 만약 어쩌다 리턴 값이 정상이 아닌 경우에 미리 선언된 자원은 해제하고 에러를 선언해야 하죠. 이 때 프로그래머는 종종 상당히 골치가 아픕니다. 이렇게 하나하나 코드의 에러를 확인하면서 짜면 필수불가결하게 if ~ else의 철장벽이 겹겹이 쌓이게 됩니다. 이렇게 높다랗게 쳐진 괄호를 뛰어넘어 메모리 해제를 하기 위해서 결국 어쩔 수 없이 goto 문에 의지하는 경우도 종종 발생한다.

```
if ( Py_SomeFunc != NULL ) {
  // 운이 좋다면
} else {
  // 여기서 자원 회수를 비롯한 에러 처리를 해야 한다.
}
```

boost::python에서는 try-catch 명령을 사용해 예외 처리가 됩니다. 파이썬 인터프리터에서 어떤 에러가 발생하면 `error_already_set` 예외가 발생합니다.

```
try {
  ...
} catch (boost::python::error_already_set const &) {
 // 에러 처리
}
```

예외 상황이 발생했을 때 모든 PyObject들이 `boost::python::object`로 관리되었다면 더욱 좋겠죠.

보다 편리한 객체 호출 및 리턴 값

파이썬 객체를 호출하기 위해 PyObject_CallObject, PyObject_CallMethod, 등의 함수보다는 ( ) 연산자를 사용해 호출합니다. 이 덕에 객체의 함수 호출이 보다 자연스러운 C++ 코드 구문과 유사해 자연스럽습니다. 그리고 인자 목록 또한 암시적으로 타입 캐스팅이 가능해 더욱 편리합니다.

C API를 바로 이용하는 경우 객체에 대해 Py<*PyObjType*>_As<*TypeToExtract*> 스타일의 함수를 사용합니다. boost::python에서는 `boost::python::extract`라는 템플릿 클래스를 제공하며, 이것은 파이썬 오브젝트 내부의 데이터에 대해 선언된 템플릿 타임으로의 암시적인 캐스팅을 해 줍니다.

여기서 템플릿 타입에는 C/C++의 기본 자료형도 가능하고, boost::python이 따로 선언한 리스트나 딕셔너리를 위한 오브젝트 클래스로도 변형이 가능합니다.

예제: urllib을 C++에서 사용하기

다음 코드는 C++에서 파이썬 'urllib'을 사용해 원격지의 자원을 로컬로 가져오는 예제입니다.

get_html.cpp

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

```
#include <iostream>
#include <fstream>

namespace bp = boost::python;
typedef bp::object bpobj;

inline bpobj CreateObject(PyObject* op)
{
  if (op == NULL) throw bp::error_already_set();
  return bpobj(bp::handle<>(op));
}

void PrintUsage(std::ostream& os)
{
  os << "< Save URL >\n";
  os << "get_html <url> <file_name>\n";
  os << "options:\n";
}

int main(int argc, char** argv)
{
  if (argc < 3) {
    PrintUsage(std::cerr);
    return 1;
  }

  if (Py_Initialize(), Py_IsInitialized()) {
    try {
      const char *url = argv[argc - 2];
      const char *file_name = argv[argc - 1];

      bpobj urllib  = CreateObject(PyImport_ImportModule("urllib"));
      bpobj connobj = urllib.attr("urlopen")(url);
      bpobj htmlobj = connobj.attr("read")();
      connobj.attr("close")();

      // save as file
      const char *html = bp::extract<const char*>(htmlobj);
     const int byte = bp::extract<const int>(htmlobj.attr("__len__")());

      std::ofstream os(file_name, std::ios::binary);

      if (os.is_open()) {
        os.write(html, byte);
        printf("%d bytes written to %s\n", byte, file_name);
        os.close();
      }
    }
    catch (bp::error_already_set const &) {
      PyErr_Print();
    }
```

```
    Py_Finalize();
  }
  return 0;
}
```

윈속이나 소켓이 같은 보다 저수준의 API은 파이썬 영역에서 처리되어 있습니다.

예제: finder C++

C API 기반의 finder C 코드를 boost::python을 이용해 고쳐 보았습니다.

boost_finder.cpp

```
#include <boost/python.hpp>
#include <iostream>

namespace bp = boost::python;
typedef bp::object bpobj;

inline bpobj createObject(PyObject* op)
{
  if (op == NULL) {
    throw bp::error_already_set();
  }

  return bpobj(bp::handle<>(op));
}

void boostFinder(const char* url)
{
  // import module
  bpobj module = createObject(PyImport_ImportModule("scripts.finder"));

  // get class object, instantiate the class
  bpobj klass = module.attr("url_finder")(url);

  // url_finder.links
  bp::list links = bp::extract<bp::list>(klass.attr("links"));

  if (PyList_Check(links.ptr()))
  {
    typedef Py_ssize_t psz_t;

    const psz_t linkSize = Py_SIZE(links.ptr());
    for (psz_t i = 0; i < linkSize; ++i)
    {
      // borrowed reference
```

```
      std::cout << PyString_AsString(PyList_GET_ITEM(links.ptr(), i))
                << '\n';
    }
  }
}

int main(int argc, char **argv)
{
  if (argc < 2) {
    std::cerr << "specify a url.\n";
    return 1;
  }

  if (Py_Initialize(), Py_IsInitialized())
  {
    try
    {
      PySys_SetArgv(argc, argv);
      boostFinder(argv[1]);
    }
    catch (bp::error_already_set const &)
    {
      PyErr_Print();
    }
    Py_Finalize();
  }
  return 0;
}
```

참고 사이트

- 파이썬 위키

[1] Boost.Python - 1.55.0 <Tutorial Introduction> "Embedding" section