기술문서	'14. 10. 07. 작성

파이썬 C Extension 기술

작성자 : 인천대학교 OneScore 동아리 최창원

qwefgh90@naver.com

<제목 차례>

<표 차례>

<그림 차례>

I. Introduction of Python C-Extension

가. 파이썬 인터프리터의 종류

- C파이썬 : C로 작성된 인터프리터이다. 파이썬 인터프리터의 표준 구현체로 볼 수 있다.
- 스택리스 파이썬 : C 스택을 사용하지 않는 인터프리터이다.
- Jython : 자바 가상 머신 용 인터프리터이다.
- IronPython : .NET 플랫폼 용 인터프리터이다.
- PyPy : 파이썬으로 작성된 파이썬 인터프리터이다. RPython과 meta tracing JIT을 사용하여 새롭게 파이썬 인터프리터를 만들었다.

나. C-Extension이 필요한 이유

Python은 동적 타이핑이 지원되고 객체지향적인 특징을 지닌 언어로 1991 귀도 반 로섬이 발표하였다. Python으로 작성한 코드는 바이너리로 컴파일되지 않으며 인터프리터에서 동작하게 된다. Python은 손쉽고 직관적인 문법 때문에 다른 언어들과 많이 비교되고는 한다. 성능적인 측면에서 보았을 때 Python으로 작성된 코드가 실행될 때 인터프리터에서 해석되고 동적 타이핑이 되므로 일반적으로 컴파일 언어에 비해 실행속도가 느리다. 이런 성능적인 이슈를 해결하기 위해 C-Extension이 사용되곤 한다. C-Extension은 일반적인 파이썬 인터프리터 구현체인 CPython에서 동작하며 인터프리터로 해석되는 방식이 아닌 Machine Code가 삽입되어 동작하므로 성능도 좋은 편이다. 또한 기존에 C 코드로 작성된 라이브러리를 가져와서 사용할 수 있기 때문에 라이브러리 확장에 매우 좋은 도구이다. 따라서 C-Extension은 Python 프로젝트시 성능, 확장적인 측면에서 사용이 고려될 만하다.

다. 샘플 프로젝트를 통한 C-Extension 개발

C-Extension을 개발하기 위해선 확장 전용 파이썬 API를 사용하여야 한다. 파이썬 API는 C:\Python27\include\python.h 에 정의되어 있다. Windows 환경에서 Python API를 사용하고 컴파일하기 위해선 C:\Python27\include의 폴더를 include하고 C:\Python27\libs를 라이브러리 폴더로 지정하여야 한다. 다음은 전체 샘플 코드이다. 샘플 코드는 main.cpp 와 include.h로 구성되어 있으며 VS2010에서 작업 하였다.

구현 절차는 다음과 같다.

1) C 확장 모듈의 이름을 결정한다. (changext 로 가정한다.)
2) 결정한 모듈 이름으로 win32 - DLL 개발 프로젝트를 생성한다.
3) C:\Python27\include 폴더를 include 하고 C:\Python27\libs를 라이브러리 경로로 지정한다.

4) “PyMODINIT init모듈이름 ()” 함수를 정의한다.

5) Python API를 사용하여 함수를 작성한 후 PyMethodDef 배열을 통해 함수목록을 생성한다.

6) Py_InitModule 함수를 통해 테이블을 등록하고 모듈을 초기화 한다.

7) VS2010에서 컴파일한 후 결과물인 changext.dll 파일을 changext.pyd로 바꾼다.

```
#include “include.h“

static PyObject* changError;

PyObject* changext_add(PyObject* self, PyObject* args){
int operand1;
int operand2;

if(!PyArg_ParseTuple(args,“ii“,&operand1,&operand2)){
return NULL;
}

PyObject* returnVal = Py_BuildValue(“i“,operand1+operand2);
if(returnVal == NULL){
return NULL;
}else{
return returnVal;
}
}

static PyMethodDef ChangExt1_MTable[] = {
{ “add“, changext_add, METH_VARARGS, “a function of adding param1 to param2“ },
{ NULL, NULL, 0, NULL }
};

PyMODINIT_FUNC initchangext(void){
PyObject* module = Py_InitModule(“changext“, ChangExt1_MTable);
if (module == NULL)
return

changError = PyErr_NewException(“ChangExt1.error“, NULL, NULL);
Py_INCREF(changError);
PyModule_AddObject(module, “error“, changError);
}
```

<main.cpp>

표 3 main.cpp 소스코드

1). main.cpp 분석

i) PyObject* changext_add(PyObject* self, PyObject* args)

직접 구현한 함수로 파이썬으로부터 호출될 함수이다. 파이썬으로부터 호출될 함수는 “self“인 모듈 객체와 “args“인 인자값을 넘겨받는다. 이때 PyObject* 형을 사용하는데 이는 인터프리터에서 관리되는 파이썬 객체이다. 파이썬 primitive 타입도 PyObject 타입을 가진 객체이다.

파이썬으로부터 전달 받은 2개의 인자를 받기 위해 “PyArg_ParseTuple” 라는 파이썬 API를 사용한다. 추후 나오겠지만 메서드 테이블 등록 시 “METH_VARARGS” 를 테이블에 등록할 경우 해당 메서드를 호출하는 파이썬으로 부터 전달되는 인자는 튜플로 존재하게 된다. 따라서 예제에선 해당 API를 사용하여 개발하였다.

마지막 함수의 반환값인 파이썬 객체를 생성하기 위해 "Py_BuildValue" 라는 파이썬 API를 사용한다. 해당 API는 1번째 인자로 간단한 포맷스트링을 주어서 쉽게 파이썬 객체를 생성하도록 도와준다.

ii) PyMethodDef ChangExt1_MTable[] = {...}

해당 배열은 파이썬에서 호출할 수 있는 함수의 테이블로 만든 것 이다. 추후에 함수 테이블을 InitModule에 인자로 전달함으로써 파이썬에서 해당 함수를 호출할 수 있다.

주의할 점은 테이블 변수는 static으로 선언되어야 한다는 점이다. static으로 선언함으로써 다른 확장 모듈과 충돌하지 않도록 한다. "initchangext" 인 초기화 함수를 제외한 모든 함수와 변수는 static 으로 선언되어야 하는 것도 같은 이유이다. 또한 함수의 이름은 모듈이름 + '_' +함수이름 으로 구성된다.

각 함수 테이블의 항목들은 4가지로 구성된다. 함수이름, 함수포인터, 인자형태, 함수설명이다. 이때 인자형태 부분에는 "METH_VARARGS" 나 "METH_VARARGS ¦ METH_KEYWORDS" 를 사용할 수 있다. 첫 번째 형태는 인자가 튜플로 넘어오길 기대하는 형태이다. 두 번째 형태는 인자가 키워드로 넘어오길 기대하는 형태이다. 이 경우 "PyArg_ParseTupleAndKeywords" 함수를 통해 인자 객체를 파싱할 수 있다.

iii) PyMODINIT_FUNC initchangext(void)

해당 함수는 C-Extension 구현 시 반드시 존재해야하는 함수이다. 함수 이름은 "init모듈이름" 그리고 "PyMODINIT_FUNC" 라는 매크로를 사용해야한다.

해당 함수는 처음 파이썬 프로그램이 모듈은 사용할 때 호출된다. 내부에선 반드시 Py_InitModule() 함수를 호출하여 모듈을 초기화 하여야 한다. 해당 함수는 sys.modules에 모듈이름을 키로 객체를 생성하고 두 번째 인자로 함수테이블을 전달받아 함수들을 삽입한다.

```
>>> import sys; import changext;sys.modules['changext']
<module 'changext' from 'changext.pyd'>
>>>
```

그림 3 sys.modules 확인

마지막으로 "PyErr_NewException" 는 사용자 정의 에러를 만드는 함수이다. 반환값 으로 나온 파이썬 객체를 "PyModule_AddObject" 의 인자로 넣어주면 해당 모듈에 "error" 라는 이름의 객체를 추가하게 된다.

```
#ifdef _DEBUG
#define _DEBUG_WAS_DEFINED 1
#undef _DEBUG
#endif

#include <Python.h>

#ifdef _DEBUG_WAS_DEFINED 1
#define _DEBUG 1
#endif

#include <memory.h>
```

<include.h>

표 4 include.h 소스코드

"Python.h" 를 포함하면 모든 Python API를 사용할 수 있다.

라. 파이썬 확장 아키텍처

그림 4 파이썬 확장 아키텍처

II. Algorithm Implementation with C-Extension

가. 피보나치 수열 구현

1). 구현코드 (파이썬 Vs 파이썬 확장)

```python
def pyFibonachi(index) :
    prepreValue = 1
    preValue = 1
    offset = 2
    if(index < 1) :
        print 'invalid parameter'
        return 0;
    if(index == 1) :
        return prepreValue
    elif(index == 2):
        return preValue
    else :
        while(offset < index):
            temp = prepreValue + preValue
            prepreValue = preValue
            preValue = temp
            offset += 1
        return preValue
```

```java
long fibonachi(int index){
    long result=0;
    if(index<1){
        result=0;
    }else if(index==1){
        result=1;
    }else if(index==2){
        result=1;
    }else{
        long ppreValue=1;
        long preValue=1;
        result = 1;
        for(int i=0; i<index-2; i++){
            //for operation
            ppreValue = preValue;
            preValue = result;
            result = ppreValue + preValue;
        }
    }
    return result;
}
```

그림 5 피보나치 수열 파이썬 확장

2). 성능 비교

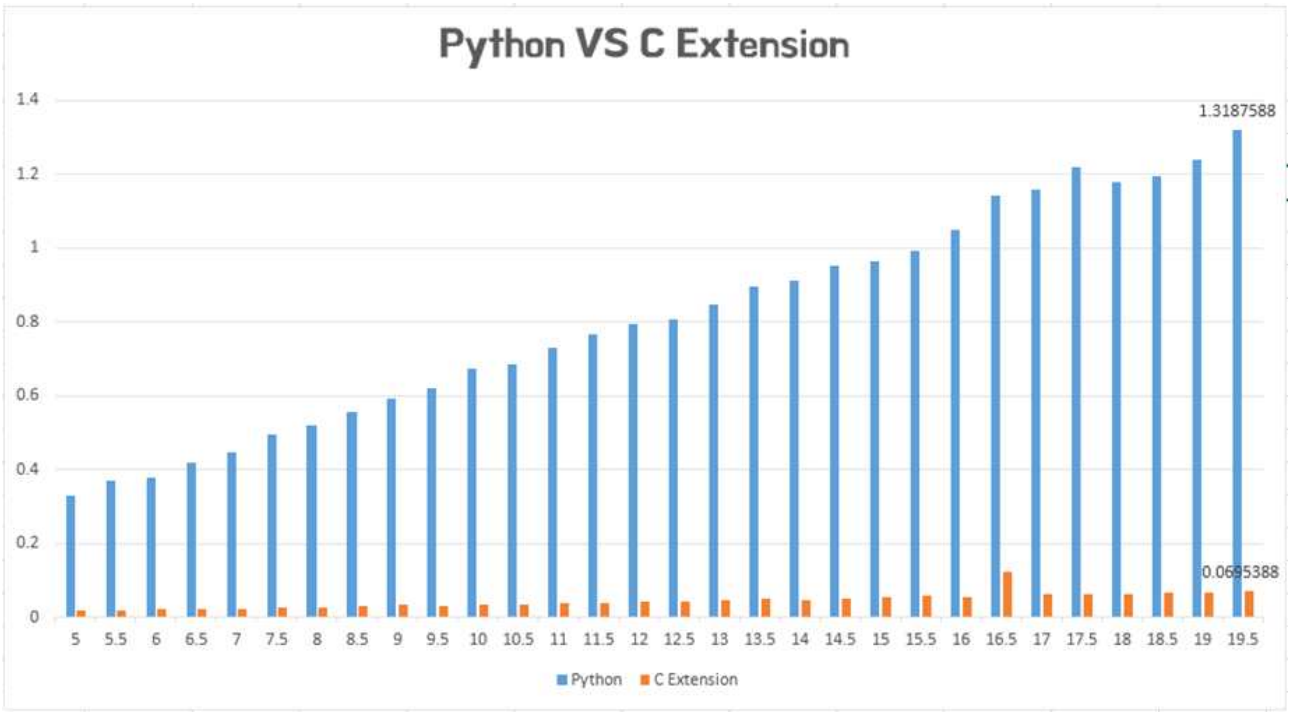

그림 6 피보나치 수열 성능 비교

나. 레드 블랙 트리 구현

1). 구현코드 (트리 헤더)

기존의 헤더를 분석한 후 아래와 같은 단계로 확장을 구현 하였다.

1) 라이브러리의 기능을 바탕으로 인터페이스를 구성한다. (예: 계산기 덧셈, 뺄샘)

2) C 변수로 관리할 자료구조를 생각해본다.

3) C 확장과 파이썬이 어떤 데이터를 주고 받을지 생각한다.

4) 위 내용을 바탕으로 인터페이스를 구현한다.

-팁-

Python 변수에서 C자료 구조를 사용할 땐 Capsule이라는 클래스에 포인터를 삽입해서 사용한다.

Python 변수는 참조되지 않으면 쉽게 사라지므로 Capsule이 사라질 때 소멸자로 자원 해제를 한다.

```
pNode createNode(KeyType key,DataPtr data);
void appendNode(Tree& tree, pNode childNode);
void rebuildAfterAppend(Tree& tree, pNode childNode);
void removeNode(Tree& tree, KeyType data, void (*destructor) (DataPtr dataPtr));
void rotateLeft(Tree& tree, pNode parent);
void rotateRight(Tree& tree, pNode parent);
void destroyTree(Tree& tree, void (*destructor) (DataPtr dataPtr));
DataPtr searchNode(Tree& tree, KeyType key);
```

2). 구현코드 (파이썬 확장 함수 헤더)

```
//Nil 변수 초기화
void rbInit();
//레드블랙트리 생성 파이썬 함수
PyObject* changext_rbCreateTree(PyObject* self, PyObject* args);
//레드블랙트리 노드 생성 파이썬 함수
PyObject* changext_rbCreateNode(PyObject* self, PyObject* args);
//레드블랙트리 노드 추가
PyObject* changext_rbAppendNode(PyObject* self, PyObject* args);
//레드블랙트리 노드 삭제
PyObject* changext_rbRemoveNode(PyObject* self, PyObject* args);
//레드블랙트리 노드 검색
PyObject* changext_rbSearchNode(PyObject* self, PyObject* args);
//레드블랙트리 트리 삭제
```

```
PyObject* changext_rbDestroyTree(PyObject* self, PyObject* args);
//레드블랙트리 출력
PyObject* changext_rbPrintTree(PyObject* self, PyObject* args);
```

참고문헌

✓ 파이썬 확장 및 포팅 Github 주소
https://github.com/qwefgh90/AlgorithmSolution/blob/master/PythonExtension2010/PythonExtension2010

✓ 파이썬 번역주소 http://qwefgh90.github.io/build/html/trans/translating_extending.html

✓ 파이썬 독스 https://docs.python.org/2/extending/extending.html