
확장 모듈을 파이썬 3에 이식하기

출시 버전 3.7.17

Guido van Rossum
and the Python development team

6월 28, 2023

Contents

1	조건부 컴파일	2
2	객체 API의 변경 사항	2
2.1	str/unicode 통합	2
2.2	long/int 통합	3
3	모듈 초기화와 상태	3
4	캡슐로 대체된 CObject	5
5	다른 옵션	8
	색인	9

저자 Benjamin Peterson

요약

C-API를 변경하는 것이 파이썬 3의 목표 중 하나는 아니었지만, 파이썬 수준의 많은 변경으로 인해 파이썬 2의 API를 그대로 남겨두는 것은 불가능했습니다. 사실, `int()`와 `long()` 통합과 같은 일부 변경 사항은 C 수준에서 더 분명합니다. 이 문서는 비 호환성을 기록으로 남기고 그 문제를 해결하는 방법을 설명합니다.

1 조건부 컴파일

어떤 코드를 파이썬 3에서만 컴파일하는 가장 쉬운 방법은 PY_MAJOR_VERSION이 3 이상인지 확인하는 것입니다.

```
#if PY_MAJOR_VERSION >= 3
#define IS_PY3K
#endif
```

존재하지 않는 API 함수는 조건 블록 내에서 동등한 것으로 별칭을 만들 수 있습니다.

2 객체 API의 변경 사항

파이썬 3은 비슷한 기능을 가진 일부 형을 병합하면서 다른 것들은 깨끗하게 분리합니다.

2.1 str/unicode 통합

파이썬 3의 str() 형은 파이썬 2의 unicode()와 동등합니다; C 함수는 모두 PyUnicode_*입니다. 이전의 8비트 문자열형은 bytes()가 되었고, C 함수는 PyBytes_*가 되었습니다. 파이썬 2.6 이상은 호환성 헤더 bytesobject.h를 제공하며, PyBytes 이름을 PyString으로 매핑합니다. 파이썬 3과의 최상의 호환성을 위해, 텍스트 데이터에는 PyUnicode를, 바이너리 데이터에는 PyBytes를 사용해야 합니다. 파이썬 3의 PyBytes와 PyUnicode가 파이썬 2의 PyString과 PyUnicode처럼 교환할 수 없다는 것을 기억하는 것도 중요합니다. 다음 예는 PyUnicode, PyString 및 PyBytes에 대한 모범 사례를 보여줍니다.

```
#include "stdlib.h"
#include "Python.h"
#include "bytesobject.h"

/* text example */
static PyObject *
say_hello(PyObject *self, PyObject *args) {
    PyObject *name, *result;

    if (!PyArg_ParseTuple(args, "U:say_hello", &name))
        return NULL;

    result = PyUnicode_FromFormat("Hello, %S!", name);
    return result;
}

/* just a forward */
static char * do_encode(PyObject *);

/* bytes example */
static PyObject *
encode_object(PyObject *self, PyObject *args) {
    char *encoded;
    PyObject *result, *myobj;

    if (!PyArg_ParseTuple(args, "O:encode_object", &myobj))
        return NULL;

    encoded = do_encode(myobj);
```

(다음 페이지에 계속)

```

if (encoded == NULL)
    return NULL;
result = PyBytes_FromString(encoded);
free(encoded);
return result;
}

```

2.2 long/int 통합

파이썬 3은 오직 하나의 정수형 `int()` 만을 가집니다. 그러나 이것은 실제로 파이썬 2의 `long()` 형에 해당합니다 - 파이썬 2에서 사용된 `int()` 형은 제거되었습니다. C-API에서, `PyInt_*` 함수는 동등한 `PyLong_*` 함수로 대체됩니다.

3 모듈 초기화와 상태

파이썬 3에는 개선된 확장 모듈 초기화 시스템이 있습니다. (PEP 3121를 참조하십시오.) 모듈 상태를 전역에 저장하는 대신, 인터프리터별 구조체에 저장해야 합니다. 파이썬 2와 파이썬 3 모두에서 올바르게 동작하는 모듈을 만드는 것은 까다롭습니다. 다음의 간단한 예제는 방법을 보여줍니다.

```

#include "Python.h"

struct module_state {
    PyObject *error;
};

#if PY_MAJOR_VERSION >= 3
#define GETSTATE(m) ((struct module_state*)PyModule_GetState(m))
#else
#define GETSTATE(m) (&_state)
static struct module_state _state;
#endif

static PyObject *
error_out(PyObject *m) {
    struct module_state *st = GETSTATE(m);
    PyErr_SetString(st->error, "something bad happened");
    return NULL;
}

static PyMethodDef myextension_methods[] = {
    {"error_out", (PyCFunction)error_out, METH_NOARGS, NULL},
    {NULL, NULL}
};

#if PY_MAJOR_VERSION >= 3

static int myextension_traverse(PyObject *m, visitproc visit, void *arg) {
    Py_VISIT(GETSTATE(m)->error);
    return 0;
}

static int myextension_clear(PyObject *m) {

```

(다음 페이지에 계속)

```

Py_CLEAR(GETSTATE(m)->error);
return 0;
}

static struct PyModuleDef moduledef = {
    PyModuleDef_HEAD_INIT,
    "myextension",
    NULL,
    sizeof(struct module_state),
    myextension_methods,
    NULL,
    myextension_traverse,
    myextension_clear,
    NULL
};

#define INITERROR return NULL

PyMODINIT_FUNC
PyInit_myextension(void)

#else
#define INITERROR return

void
initmyextension(void)
#endif
{
    #if PY_MAJOR_VERSION >= 3
        PyObject *module = PyModule_Create(&moduledef);
    #else
        PyObject *module = Py_InitModule("myextension", myextension_methods);
    #endif

    if (module == NULL)
        INITERROR;
    struct module_state *st = GETSTATE(module);

    st->error = PyErr_NewException("myextension.Error", NULL, NULL);
    if (st->error == NULL) {
        Py_DECREF(module);
        INITERROR;
    }

    #if PY_MAJOR_VERSION >= 3
        return module;
    #endif
}

```

4 캡슐로 대체된 CObject

Capsule 객체는 CObject를 대체하기 위해 파이썬 3.1과 2.7에 도입되었습니다. CObject는 유용했지만, CObject API는 문제가 있었습니다: 유효한 CObject를 구분할 수 없어서 불일치한 CObject가 인터프리터에 충돌을 일으킬 수 있도록 했고, 일부 API는 C의 정의되지 않은 동작에 의존했습니다. (캡슐을 뒷받침하는 이유에 대한 자세한 내용은 bpo-5630를 참조하십시오.)

현재 CObject를 사용하고 있고, 3.1 이상으로 이전하려고 한다면, 캡슐로 전환해야 합니다. CObject는 3.1과 2.7에서 폐지되었고, 파이썬 3.2에서 완전히 제거되었습니다. 2.7이나 3.1 이상만 지원한다면, 단순히 Capsule로 전환할 수 있습니다. 파이썬 3.0이나 2.7 이전의 파이썬 버전을 지원해야 한다면, CObject와 캡슐을 모두 지원해야 합니다. (파이썬 3.0은 더는 지원되지 않으며 프로덕션 용도로 권장되지 않음에 유의하십시오.)

다음 예제 헤더 파일 capsulethunk.h는 문제를 해결할 수 있습니다. Capsule API로 코드를 작성하고, Python.h 다음에 이 헤더 파일을 포함하십시오. 여러분의 코드는 자동으로 캡슐이 있는 파이썬 버전에서 캡슐을 사용하고, 캡슐을 사용할 수 없을 때 CObject로 전환합니다.

capsulethunk.h는 CObject를 사용하여 캡슐을 시뮬레이션합니다. 그러나, CObject는 캡슐의 “이름 (name)”을 저장할 장소를 제공하지 않습니다. 결과적으로 capsulethunk.h가 만든 시뮬레이트된 Capsule 객체는 실제 캡슐과 약간 다르게 동작합니다. 구체적으로:

- PyCapsule_New() 에 전달된 name 매개 변수는 무시됩니다.
- PyCapsule_IsValid() 와 PyCapsule_GetPointer() 에 전달된 name 매개 변수는 무시되며, name 에 대한 예러 검사는 수행되지 않습니다.
- PyCapsule_GetName() 은 항상 NULL을 반환합니다.
- PyCapsule_SetName() 은 항상 예외를 발생시키고 실패를 반환합니다. (CObject에 이름을 저장할 수 있는 방법이 없으므로, 여기서 조용한 실패보다 요란한 PyCapsule_SetName() 의 실패를 선호합니다. 이것이 불편하다면, 적절하다고 생각하는 대로 로컬 복사본을 자유롭게 수정하십시오.)

capsulethunk.h는 파이썬 소스 배포판의 Doc/includes/capsulethunk.h에서 찾을 수 있습니다. 여러분의 편의를 위해 여기에도 포함합니다:

```
#ifndef __CAPSULETHUNK_H
#define __CAPSULETHUNK_H

#if ( (PY_VERSION_HEX < 0x02070000) \
    || ((PY_VERSION_HEX >= 0x03000000) \
    && (PY_VERSION_HEX < 0x03010000)) )

#define __PyCapsule_GetField(capsule, field, default_value) \
    ( PyCapsule_CheckExact(capsule) \
      ? ((PyCObject *)capsule)->field) \
      : (default_value) \
    ) \

#define __PyCapsule_SetField(capsule, field, value) \
    ( PyCapsule_CheckExact(capsule) \
      ? (((PyCObject *)capsule)->field = value), 1 \
      : 0 \
    ) \

#define PyCapsule_Type PyCObject_Type

#define PyCapsule_CheckExact(capsule) (PyCObject_Check(capsule))
#define PyCapsule_IsValid(capsule, name) (PyCObject_Check(capsule))
```

(다음 페이지에 계속)

```

#define PyCapsule_New(pointer, name, destructor) \
    (PyObject_FromVoidPtr(pointer, destructor))

#define PyCapsule_GetPointer(capsule, name) \
    (PyObject_AsVoidPtr(capsule))

/* Don't call PyObject_SetPointer here, it fails if there's a destructor */
#define PyCapsule_SetPointer(capsule, pointer) \
    __PyCapsule_SetField(capsule, cobject, pointer)

#define PyCapsule_GetDestructor(capsule) \
    __PyCapsule_GetField(capsule, destructor)

#define PyCapsule_SetDestructor(capsule, dtor) \
    __PyCapsule_SetField(capsule, destructor, dtor)

/*
 * Sorry, there's simply no place
 * to store a Capsule "name" in a CObject.
 */
#define PyCapsule_GetName(capsule) NULL

static int
PyCapsule_SetName(PyObject *capsule, const char *unused)
{
    unused = unused;
    PyErr_SetString(PyExc_NotImplementedError,
        "can't use PyCapsule_SetName with CObjects");
    return 1;
}

#define PyCapsule_GetContext(capsule) \
    __PyCapsule_GetField(capsule, descr)

#define PyCapsule_SetContext(capsule, context) \
    __PyCapsule_SetField(capsule, descr, context)

static void *
PyCapsule_Import(const char *name, int no_block)
{
    PyObject *object = NULL;
    void *return_value = NULL;
    char *trace;
    size_t name_length = (strlen(name) + 1) * sizeof(char);
    char *name_dup = (char *)PyMem_MALLOC(name_length);

    if (!name_dup) {
        return NULL;
    }
}

```

```

memcpy(name_dup, name, name_length);

trace = name_dup;
while (trace) {
    char *dot = strchr(trace, '.');
    if (dot) {
        *dot++ = '\0';
    }

    if (object == NULL) {
        if (no_block) {
            object = PyImport_ImportModuleNoBlock(trace);
        } else {
            object = PyImport_ImportModule(trace);
            if (!object) {
                PyErr_Format(PyExc_ImportError,
                    "PyCapsule_Import could not "
                    "import module \"%s\"", trace);
            }
        }
    } else {
        PyObject *object2 = PyObject_GetAttrString(object, trace);
        Py_DECREF(object);
        object = object2;
    }
    if (!object) {
        goto EXIT;
    }

    trace = dot;
}

if (PyObject_Check(object)) {
    PyObject *cobject = (PyObject *)object;
    return_value = cobject->cobject;
} else {
    PyErr_Format(PyExc_AttributeError,
        "PyCapsule_Import \"%s\" is not valid",
        name);
}

EXIT:
Py_XDECREF(object);
if (name_dup) {
    PyMem_FREE(name_dup);
}
return return_value;
}

#endif /* #if PY_VERSION_HEX < 0x02070000 */

#endif /* __CAPSULETHUNK_H */

```

5 다른 옵션

새 확장 모듈을 작성하고 있다면, `Cython`을 고려하십시오. 파이썬과 비슷한 언어를 C로 변환합니다. 만들어진 확장 모듈은 파이썬 3과 파이썬 2 모두와 호환됩니다.

색인

Y

파이썬 향상 제안
PEP 3121, 3