3.0 への拡張モジュール移植

リリース *2.7ja1*

Guido van Rossum
Fred L. Drake, Jr., editor

2011 年 12 月 25 日

Python Software Foundation
Email: docs@python.org

目次

author Benjamin Peterson

概要

C-API の変更は Python 3.0 の目標には入っていませんでしたが、Python レベルでの変更がたくさんあったので、2.x の API を無傷で済ませることはできませんでした。実際、 `int()` と `long()` の統合などは C レベルのほうが目立ちます。この文書では、なくなった互換性と、その対処方法について記述しようと思います。

1 条件コンパイル

一部のコードを 3.0 にだけコンパイルするための一番簡単な方法は、PY_MAJOR_VERSION が 3 以上かどうかチェックすることです。

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

存在しなくなった関数については、条件ブロックの中で同等品にエイリアスすれば良いでしょう。

2 オブジェクト API の変更

Python 3.0 では、似た機能を持つタイプのいくつかを、統合したり、きっちり分けたりしました。

2.1 str/unicode の統合

Python 3.0 の str() (C では PyString_* 関数) タイプは 2.x の unicode() (PyUnicode_*) と同じものです。昔の 8 ビット文字列タイプは bytes() です。Python 2.6 以降には互換性ヘッダ bytesobject.h が用意されており、 PyBytes 系の名前を PyString 系にマップしています。3.0 との互換性を最大限確保するには、 PyUnicode は文字データに、PyBytes はバイナリデータにだけ使うべきです。ほかにも、3.0 の PyBytes と PyUnicode は 2.x の PyString と PyUnicode とは違って交換不可能だということも重要です。以下の例では PyUnicode, PyString, PyBytes に関するベストプラクティスを見ることができます。

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

2.2 long/int の統合

Python 3.0 では整数タイプが一つしかありません。これは Python レベルでは `int()` と呼ばれますが、実際には 2.x の `long()` タイプと同じものです。C-API では `PyInt_*` 関数群の中身が、お隣さんの `PyLong_*` に交換されています。ここでとるべき最良の行動指針は、`intobject.h` で `PyLong_*` にエイリアスされている `PyInt_*` を使うというものです。ある場合には抽象 API 群 `PyNumber_*` も使えるかもしれません。

```
#include "Python.h"
#include "intobject.h"

static PyObject *
add_ints(PyObject *self, PyObject *args) {
    int one, two;
    PyObject *result;

    if (!PyArg_ParseTuple(args, "ii:add_ints", &one, &two))
        return NULL;

    return PyInt_FromLong(one + two);
}
```

3 モジュールの初期化と状態情報

Python 3.0 には、改良された拡張モジュール初期化システムがあります (**PEP 3121** 参照)。モジュールの状態はグローバル変数に持つのではなく、インタプリタ固有の構造体に持つべきだということになったのです。2.x と 3.0 のどちらでも動くモジュールを作るのにはコツが要ります。次の簡単な例で、その方法を実演してみます。

```
#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) {
    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

PyObject *
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 ほかの選択肢

新規に拡張モジュールを書こうと思っているのであれば、 Cython を検討してみても良いでしょう。これは Python 風の言語を C に翻訳してくれるもので、出力される拡張モジュールは Python 3.x と 2.x に両方対応しています。

索引