
将扩展模块移植到 Python 3

发布 3.6.10rc1

Guido van Rossum
and the Python development team

十二月 18, 2019

Python Software Foundation
Email: docs@python.org

Contents

1	条件编译	1
2	对象 API 的更改	2
2.1	str/unicode 统一	2
2.2	long/int 统一	3
3	模块初始化和状态	3
4	CObject 替换为 Capsule	4
5	其他选项	7
	索引	8

作者 Benjamin Peterson

摘要

尽管更改 C-API 并不是 Python 3 的目标之一，但许多 Python 级别的更改使得 Python 2 的 API 无法完整实现。实际上，一些变化如 `int()` 和 `long()` 的统一在 C 级别更明显。本文档致力于记录不兼容性以及如何解决这些问题。

1 条件编译

仅编译 Python 3 的一些代码的最简单方法是检查 `PY_MAJOR_VERSION` 是否大于或等于 3。

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

不存在的 API 函数可以在条件块中别名为它们的等价物。

2 对象 API 的更改

Python 3 将一些具有类似功能的类型合并在一起，同时干净地分离了其他类型。

2.1 str/unicode 统一

Python 3 的 `str()` 类型相当于 Python 2 的 `unicode()`；C 函数被称为 `PyUnicode_*`。旧的 8 位字符串类型变为 `bytes()`，其中 C 函数称为 `PyBytes_*`。Python 2.6 及更高版本提供了一个兼容性头文件 `bytesobject.h`，将 `PyBytes` 名称映射到 `PyString`。为了保持与 Python 3 的最佳兼容性，`PyUnicode` 应该用于文本数据，并且 `PyBytes` 用于二进制数据。同样重要的是要记住 `pyBytes` 和 Python 3 中的 `PyUnicode` 不可互换，如 `PyString` 和 `PyUnicode` 在 Python 2 以下中示例显示了以下方面的最佳实践 `PyUnicode`、`PyString` 和 `PyBytes`。：

```
#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);
}
```

(下页继续)

```

    return result;
}

```

2.2 long/int 统一

Python 3 只有一个整数类型，`int()`。但它实际上对应于 Python 2 `long()` 类型——删除了 Python 2 中使用的 `int()` 类型。在 C-API 中，`PyInt_*` 函数被它们等价的 `PyLong_*` 替换。

3 模块初始化和状态

Python 3 有一个改进的扩展模块初始化系统。（参见 [PEP 3121](#)。）而不是将模块状态存储在全局变量中，它们应该存储在特定于解释器的结构中。创建在 Python 2 和 Python 3 中正确运行的模块非常棘手。以下简单示例演示了如何操作。：

```

#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 INITERERROR return NULL

PyMODINIT_FUNC
PyInit_myextension(void)

#else
#define INITERERROR 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)
        INITERERROR;
    struct module_state *st = GETSTATE(module);

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

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

```

4 CObject 替换为 Capsule

Capsule 对象是在 Python 3.1 和 2.7 中引入的，用于替换 CObject。CObject 是有用的，但是 CObject API 是有问题的：它不允许区分有效的 CObject，这导致不匹配的 CObject 使解释器崩溃，并且它的一些 API 依赖于 C 中的未定义行为。有关 Capsule 背后的基本原理的进一步阅读，请参阅 [bpo-5630](#)。）

如果你当前正在使用 CObject，并且想要迁移到 3.1 或更高版本，则需要切换到 Capsules。CObject 在 3.1 和 2.7 中已弃用，在 Python 3.2 中已完全删除。如果你只支持 2.7 或 3.1 及以上，你可以简单地切换到 Capsule。如果你需要支持 Python 3.0 或早于 2.7 的 Python 版本，则必须同时支持 CObject 和 Capsule。（请注意，不

再支持 Python 3.0，不建议将其用于生产用途。)

以下示例头文件 capsulethunk.h 可以为你解决问题。只需针对 Capsule API 编写代码，并在以下文件后包含此头文件 Python.h。你的代码将自动在带有 Capsule 的 Python 版本中使用 Capsules，并在 Capsule 不可用时切换到 CObjects。

capsulethunk.h 使用 CObject 模拟 Capsules。但是，CObject 没有提供存储胶囊的“名称”的地方。因此，模拟 Capsule 对象由 capsulethunk.h 创建，其行为与真实 Capsule 略有不同。特别地：

- 传递给 PyCapsule_New() 的 name 参数被忽略。
- 传入以下命令的 name 参数 PyCapsule_IsValid() 和 PyCapsule_GetPointer() 被忽略，并且不执行错误检查。
- PyCapsule_GetName() 总是返回 NULL。
- PyCapsule_SetName() 总是引发异常并返回失败。（由于无法在 CObject 中存储名称，因此 PyCapsule_SetName() 的明显失败被认为优于静默失败。如果这样不方便，请随意根据需要修改本地副本。）

你可以在 Python 源代码分发中的 Doc/includes/capsulethunk.h 找到 capsulethunk.h。为方便起见，我们还将其包含在此处：

```
#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) \
      ? (((PyObject *)capsule)->field) \
      : (default_value) \
    ) \

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

#define PyCapsule_Type PyObject_Type

#define PyCapsule_CheckExact(capsule) (PyObject_Check(capsule))
#define PyCapsule_IsValid(capsule, name) (PyObject_Check(capsule))

#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)
```

(下页继续)

```

#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 (PyCObject_Check(object)) {
    PyCObject *cobject = (PyCObject *)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 其他选项

如果你正在编写新的扩展模块，你可能会考虑 [Cython](#)。它将类似 Python 的语言转换为 C。它创建的扩展模块与 Python 3 和 Python 2 兼容。

索引

P

Python 提高建议
PEP 3121, [3](#)