15.5 从扩展模块中定义和导出C的API¶

问题¶

你有一个C扩展模块，在内部定义了很多有用的函数，你想将它们导出为一个公共的C API供其他地方使用。 你想在其他扩展模块中使用这些函数，但是不知道怎样将它们链接起来， 并且通过C编译器/链接器来做看上去特别复杂（或者不可能做到）。

解决方案¶

本节主要问题是如何处理15.4小节中提到的Point对象。仔细回一下，在C代码中包含了如下这些工具函数：

```
/* Destructor function for points */
static void del_Point(PyObject *obj) {

  free(PyCapsule_GetPointer(obj,"Point"));
}

/* Utility functions */
static Point *PyPoint_AsPoint(PyObject *obj) {
  return (Point *) PyCapsule_GetPointer(obj, "Point");
}

static PyObject *PyPoint_FromPoint(Point *p, int must_free) {
  return PyCapsule_New(p, "Point", must_free ? del_Point : NULL);
}
```

现在的问题是怎样将 `PyPoint_AsPoint()` 和 `Point_FromPoint()` 函数作为API导出， 这样其他扩展模块能使用并链接它们，比如如果你有其他扩展也想使用包装的Point对象。

要解决这个问题，首先要为 `sample` 扩展写个新的头文件名叫 `pysample.h` ，如下：

```
/* pysample.h */
#include "Python.h"
#include "sample.h"
#ifdef __cplusplus
extern "C" {
#endif

/* Public API Table */
typedef struct {
  Point *(*aspoint)(PyObject *);
  PyObject *(*frompoint)(Point *, int);
} _PointAPIMethods;

#ifndef PYSAMPLE_MODULE
/* Method table in external module */
static _PointAPIMethods *_point_api = 0;

/* Import the API table from sample */
static int import_sample(void) {
  _point_api = (_PointAPIMethods *) PyCapsule_Import("sample._point_api",0);
  return (_point_api != NULL) ? 1 : 0;
}
```

```
/* Macros to implement the programming interface */
#define PyPoint_AsPoint(obj) (_point_api->aspoint)(obj)
#define PyPoint_FromPoint(obj) (_point_api->frompoint)(obj)
#endif

#ifdef __cplusplus
}
#endif
```

这里最重要的部分是函数指针表 `_PointAPIMethods` . 它会在导出模块时被初始化，然后导入模块时被查找到。 修改原始的扩展模块来填充表格并将它像下面这样导出：

```
/* pysample.c */

#include "Python.h"
#define PYSAMPLE_MODULE
#include "pysample.h"

...
/* Destructor function for points */
static void del_Point(PyObject *obj) {
  printf("Deleting point\n");
  free(PyCapsule_GetPointer(obj,"Point"));
}

/* Utility functions */
static Point *PyPoint_AsPoint(PyObject *obj) {
  return (Point *) PyCapsule_GetPointer(obj, "Point");
}

static PyObject *PyPoint_FromPoint(Point *p, int free) {
  return PyCapsule_New(p, "Point", free ? del_Point : NULL);
}

static _PointAPIMethods _point_api = {
  PyPoint_AsPoint,
  PyPoint_FromPoint
};
...

/* Module initialization function */
PyMODINIT_FUNC
PyInit_sample(void) {
  PyObject *m;
  PyObject *py_point_api;

  m = PyModule_Create(&samplemodule);
  if (m == NULL)
    return NULL;

  /* Add the Point C API functions */
  py_point_api = PyCapsule_New((void *) &_point_api, "sample._point_api", NULL);
  if (py_point_api) {
    PyModule_AddObject(m, "_point_api", py_point_api);
  }
  return m;
}
```

最后，下面是一个新的扩展模块例子，用来加载并使用这些API函数：

```
/* ptexample.c */

/* Include the header associated with the other module */
#include "pysample.h"

/* An extension function that uses the exported API */
static PyObject *print_point(PyObject *self, PyObject *args) {
  PyObject *obj;
  Point *p;
  if (!PyArg_ParseTuple(args,"O", &obj)) {
    return NULL;
  }

  /* Note: This is defined in a different module */
  p = PyPoint_AsPoint(obj);
  if (!p) {
    return NULL;
  }
  printf("%f %f\n", p->x, p->y);
  return Py_BuildValue("");
}

static PyMethodDef PtExampleMethods[] = {
  {"print_point", print_point, METH_VARARGS, "output a point"},
  { NULL, NULL, 0, NULL}
};

static struct PyModuleDef ptexamplemodule = {
  PyModuleDef_HEAD_INIT,
  "ptexample",          /* name of module */
  "A module that imports an API",  /* Doc string (may be NULL) */
  -1,                /* Size of per-interpreter state or -1 */
  PtExampleMethods       /* Method table */
};

/* Module initialization function */
PyMODINIT_FUNC
PyInit_ptexample(void) {
  PyObject *m;

  m = PyModule_Create(&ptexamplemodule);
  if (m == NULL)
    return NULL;

  /* Import sample, loading its API functions */
  if (!import_sample()) {
    return NULL;
  }

  return m;
}
```

编译这个新模块时，你甚至不需要去考虑怎样将函数库或代码跟其他模块链接起来。 例如，你可以像下面这样创建一个简单的 `setup.py` 文件：

```
# setup.py
from distutils.core import setup, Extension
```

```
setup(name='ptexample',
      ext_modules=[
        Extension('ptexample',
                  ['ptexample.c'],
                  include_dirs = [],  # May need pysample.h directory
                  )
      ]
)
```

如果一切正常，你会发现你的新扩展函数能和定义在其他模块中的C API函数一起运行的很好。

```
>>> import sample
>>> p1 = sample.Point(2,3)
>>> p1
<capsule object "Point *" at 0x1004ea330>
>>> import ptexample
>>> ptexample.print_point(p1)
2.000000 3.000000
>>>
```

讨论¶

本节基于一个前提就是，胶囊对象能获取任何你想要的对象的指针。 这样的话，定义模块会填充一个函数指针的结构体，创建一个指向它的胶囊，并在一个模块级属性中保存这个胶囊， 例如 `sample._point_api` .

其他模块能够在导入时获取到这个属性并提取底层的指针。 事实上，Python提供了 `PyCapsule_Import()` 工具函数，为了完成所有的步骤。 你只需提供属性的名字即可（比如sample._point_api），然后他就会一次性找到胶囊对象并提取出指针来。

在将被导出函数变为其他模块中普通函数时，有一些C编程陷阱需要指出来。 在 `pysample.h` 文件中，一个 `_point_api` 指针被用来指向在导出模块中被初始化的方法表。 一个相关的函数 `import_sample()` 被用来指向胶囊导入并初始化这个指针。 这个函数必须在任何函数被使用之前被调用。通常来讲，它会在模块初始化时被调用到。 最后，C的预处理宏被定义，被用来通过方法表去分发这些API函数。 用户只需要使用这些原始函数名称即可，不需要通过宏去了解其他信息。

最后，还有一个重要的原因让你去使用这个技术来链接模块——它非常简单并且可以使得各个模块很清晰的解耦。 如果你不想使用本机的技术，那你就必须使用共享库的高级特性和动态加载器来链接模块。 例如，将一个普通的API函数放入一个共享库并确保所有扩展模块链接到那个共享库。 这种方法确实可行，但是它相对繁琐，特别是在大型系统中。 本节演示了如何通过Python的普通导入机制和仅仅几个胶囊调用来将多个模块链接起来的魔法。 对于模块的编译，你只需要定义头文件，而不需要考虑函数库的内部细节。

更多关于利用C API来构造扩展模块的信息可以参考 Python的文档