15.4 在C扩展模块中操作隐形指针¶

问题¶

你有一个扩展模块需要处理C结构体中的指针， 但是你又不想暴露结构体中任何内部细节给Python。

解决方案¶

隐形结构体可以很容易的通过将它们包装在胶囊对象中来处理。 考虑我们例子代码中的下列C代码片段：

```
typedef struct Point {
    double x,y;
} Point;

extern double distance(Point *p1, Point *p2);
```

下面是一个使用胶囊包装Point结构体和 `distance()` 函数的扩展代码实例：

```
/* 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);
}

/* Create a new Point object */
static PyObject *py_Point(PyObject *self, PyObject *args) {

  Point *p;
  double x,y;
  if (!PyArg_ParseTuple(args,"dd",&x,&y)) {
    return NULL;
  }
  p = (Point *) malloc(sizeof(Point));
  p->x = x;
  p->y = y;
  return PyPoint_FromPoint(p, 1);
}

static PyObject *py_distance(PyObject *self, PyObject *args) {
  Point *p1, *p2;
  PyObject *py_p1, *py_p2;
  double result;

  if (!PyArg_ParseTuple(args,"OO",&py_p1, &py_p2)) {
    return NULL;
  }
  if (!(p1 = PyPoint_AsPoint(py_p1))) {
    return NULL;
  }
  if (!(p2 = PyPoint_AsPoint(py_p2))) {
    return NULL;
  }
  result = distance(p1,p2);
  return Py_BuildValue("d", result);
}
```

在Python中可以像下面这样来使用这些函数：

```
>>> import sample
>>> p1 = sample.Point(2,3)
>>> p2 = sample.Point(4,5)
>>> p1
<capsule object "Point" at 0x1004ea330>
>>> p2
<capsule object "Point" at 0x1005d1db0>
>>> sample.distance(p1,p2)
2.8284271247461903
>>>
```

讨论¶

胶囊和C指针类似。在内部，它们获取一个通用指针和一个名称，可以使用 `PyCapsule_New()` 函数很容易的被创建。另外，一个可选的析构函数能被绑定到胶囊上，用来在胶囊对象被垃圾回收时释放底层的内存。

要提取胶囊中的指针，可使用 `PyCapsule_GetPointer()` 函数并指定名称。 如果提供的名称和胶囊不匹配或其他错误出现，那么就会抛出异常并返回NULL。

本节中，一对工具函数—— `PyPoint_FromPoint()` 和 `PyPoint_AsPoint()` 被用来创建和从胶囊对象中提取Point实例。在任何扩展函数中，我们会使用这些函数而不是直接使用胶囊对象。 这种设计使得我们可以很容易的应对将来对Point底下的包装的更改。 例如，如果你决定使用另外一个胶囊了，那么只需要更改这两个函数即可。

对于胶囊对象一个难点在于垃圾回收和内存管理。 `PyPoint_FromPoint()` 函数接受一个 `must_free` 参数， 用来指定当胶囊被销毁时底层Point * 结构体是否应该被回收。 在某些C代码中，归属问题通常很难被处理（比如一个Point结构体被嵌入到一个被单独管理的大结构体中）。 程序员可以使用 `extra` 参数来控制，而不是单方面的决定垃圾回收。 要注意的是和现有胶囊有关的析构器能使用 `PyCapsule_SetDestructor()` 函数来更改。

对于涉及到结构体的C代码而言，使用胶囊是一个比较合理的解决方案。 例如，有时候你并不关心暴露结构体的内部信息或者将其转换成一个完整的扩展类型。 通过使用胶囊，你可以在它上面放一个轻量级的包装器，然后将它传给其他的扩展函数。