15.12 将函数指针转换为可调用对象¶

问题¶

你已经获得了一个被编译函数的内存地址，想将它转换成一个Python可调用对象， 这样的话你就可以将它作为一个扩展函数使用了。

解决方案¶

`ctypes` 模块可被用来创建包装任意内存地址的Python可调用对象。 下面的例子演示了怎样获取C函数的原始、底层地址，以及如何将其转换为一个可调用对象：

```
>>> import ctypes
>>> lib = ctypes.cdll.LoadLibrary(None)
>>> # Get the address of sin() from the C math library
>>> addr = ctypes.cast(lib.sin, ctypes.c_void_p).value
>>> addr
140735505915760

>>> # Turn the address into a callable function
>>> functype = ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double)
>>> func = functype(addr)
>>> func
<CFunctionType object at 0x1006816d0>

>>> # Call the resulting function
>>> func(2)
0.9092974268256817
>>> func(0)
0.0
>>>
```

讨论¶

要构建一个可调用对象，你首先需要创建一个 `CFUNCTYPE` 实例。 `CFUNCTYPE()` 的第一个参数是返回类型。 接下来的参数是参数类型。一旦你定义了函数类型，你就能将它包装在一个整型内存地址上来创建一个可调用对象了。 生成的对象被当做普通的可通过 `ctypes` 访问的函数来使用。

本节看上去可能有点神秘，偏底层一点。 但是，但是它被广泛使用于各种高级代码生成技术比如即时编译，在LLVM函数库中可以看到。

例如，下面是一个使用 `llvmpy` 扩展的简单例子，用来构建一个小的聚集函数，获取它的函数指针， 并将其转换为一个Python可调用对象。

```
>>> from llvm.core import Module, Function, Type, Builder
>>> mod = Module.new('example')
>>> f = Function.new(mod,Type.function(Type.double(), \
                     [Type.double(), Type.double()], False), 'foo')
>>> block = f.append_basic_block('entry')
>>> builder = Builder.new(block)
>>> x2 = builder.fmul(f.args[0],f.args[0])
>>> y2 = builder.fmul(f.args[1],f.args[1])
>>> r = builder.fadd(x2,y2)
>>> builder.ret(r)
<llvm.core.Instruction object at 0x10078e990>
>>> from llvm.ee import ExecutionEngine
>>> engine = ExecutionEngine.new(mod)
>>> ptr = engine.get_pointer_to_function(f)
>>> ptr
4325863440
>>> foo = ctypes.CFUNCTYPE(ctypes.c_double, ctypes.c_double, ctypes.c_double)(ptr)

>>> # Call the resulting function
```

```
>>> foo(2,3)
13.0
>>> foo(4,5)
41.0
>>> foo(1,2)
5.0
>>>
```

并不是说在这个层面犯了任何错误就会导致Python解释器挂掉。 要记得的是你是在直接跟机器级别的内存地址和本地机器码打交道，而不是Python函数。