# [Cython三分钟入门](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

分类： [Python](http://blog.csdn.net/lanphaday/article/category/328646) [Translation](http://blog.csdn.net/lanphaday/article/category/328653)2009-09-17 10:17 30119人阅读 [评论](http://blog.csdn.net/gzlaiyonghao/article/details/4561611#comments)(19) [收藏](javascript:void(0);) [举报](http://blog.csdn.net/gzlaiyonghao/article/details/4561611#report)

[float](http://www.csdn.net/tag/float)[python](http://www.csdn.net/tag/python)[math.h](http://www.csdn.net/tag/math.h)[import](http://www.csdn.net/tag/import)[math](http://www.csdn.net/tag/math)[c](http://www.csdn.net/tag/c)

作者：perrygeo

译者：赖勇浩（[http://laiyonghao.com](http://laiyonghao.com/)）

原文：<http://www.perrygeo.net/wordpress/?p=116>

我最喜欢的是Python，它的代码优雅而实用，可惜纯粹从速度上来看它比大多数语言都要慢。大多数人也认为的速度和易于使用是两极对立的——编写C代码的确非常痛苦。而 Cython 试图消除这种两重性，并让你同时拥有Python 的语法和 C 数据类型和函数——它们两个都是世界上最好的。请记住，我绝不是我在这方面的专家，这是我的第一次Cython真实体验的笔记：

编辑：根据一些我收到的反馈，大家似乎有点混淆——Cython是用来生成 C 扩展到而不是独立的程序的。所有的加速都是针对一个已经存在的 Python 应用的一个函数进行的。没有使用C 或 Lisp 重写整个应用程序，也没有手写C扩展 。只是用一个简单的方法来整合C的速度和C数据类型到 Python 函数中去。

现在可以说，我们能使下文的great\_circle 函数更快。所谓great\_circle 是计算沿地球表面两点之间的距离的问题：

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. **import** math
3. **def** great\_circle(lon1,lat1,lon2,lat2):
4. radius = 3956 #miles
5. x = math.pi/180.0
7. a = (90.0-lat1)\*(x)
8. b = (90.0-lat2)\*(x)
9. theta = (lon2-lon1)\*(x)
10. c = math.acos((math.cos(a)\*math.cos(b)) +
11. (math.sin(a)\*math.sin(b)\*math.cos(theta)))
12. **return** radius\*c

让我们调用它 50 万次并测定它的时间 ：

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. **import** timeit
3. lon1, lat1, lon2, lat2 = -72.345, 34.323, -61.823, 54.826
4. num = 500000
6. t = timeit.Timer("p1.great\_circle(%f,%f,%f,%f)" % (lon1,lat1,lon2,lat2),
7. "import p1")
8. **print** "Pure python function", t.timeit(num), "sec"

约2.2秒 。它太慢了！

让我们试着快速地用Cython改写它，然后看看是否有差别：

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. **import** math
3. **def** great\_circle(float lon1,float lat1,float lon2,float lat2):
4. cdef float radius = 3956.0
5. cdef float pi = 3.14159265
6. cdef float x = pi/180.0
7. cdef float a,b,theta,c
9. a = (90.0-lat1)\*(x)
10. b = (90.0-lat2)\*(x)
11. theta = (lon2-lon1)\*(x)
12. c = math.acos((math.cos(a)\*math.cos(b)) + (math.sin(a)\*math.sin(b)\*math.cos(theta)))
13. **return** radius\*c

请注意，我们仍然importmath——cython让您在一定程度上混搭Python和C数据类型在。转换是自动的，但并非没有代价。在这个例子中我们所做的就是定义一个Python函数，声明它的输入参数是浮点数类型，并为所有变量声明类型为C浮点数据类型。计算部分它仍然使用了Python的 math 模块。

现在我们需要将其转换为C代码再编译为Python扩展。完成这一部的最好的办法是编写一个名为setup.py发布脚本。但是，现在我们用手工方式 ，以了解其中的巫术：

**[plain]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. <pre name="code" class="plain"># this will create a c1.c file - the C source code to build a python extension
2. cython c1.pyx
4. # Compile the object file
5. gcc -c -fPIC -I/usr/include/python2.5/ c1.c
7. # Link it into a shared library
8. gcc -shared c1.o -o c1.so</pre>

现在你应该有一个c1.so（或.dll）文件，它可以被Python import。现在运行一下：

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. t = timeit.Timer("c1.great\_circle(%f,%f,%f,%f)" % (lon1,lat1,lon2,lat2),
2. "import c1")
3. **print** "Cython function (still using python math)", t.timeit(num), "sec"

约1.8秒 。并没有我们一开始期望的那种大大的性能提升。使用 python 的 math 模块应该是瓶颈。现在让我们使用C标准库替代之：

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. cdef extern **from** "math.h":
2. float cosf(float theta)
3. float sinf(float theta)
4. float acosf(float theta)
6. **def** great\_circle(float lon1,float lat1,float lon2,float lat2):
7. cdef float radius = 3956.0
8. cdef float pi = 3.14159265
9. cdef float x = pi/180.0
10. cdef float a,b,theta,c
12. a = (90.0-lat1)\*(x)
13. b = (90.0-lat2)\*(x)
14. theta = (lon2-lon1)\*(x)
15. c = acosf((cosf(a)\*cosf(b)) + (sinf(a)\*sinf(b)\*cosf(theta)))
16. **return** radius\*c

与 import math 相应，我们使用cdef extern 的方式使用从指定头文件声明函数（在此就是使用C标准库的math.h）。我们替代了代价高昂的的Python函数，然后建立新的共享库，并重新测试：

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. t = timeit.Timer("c2.great\_circle(%f,%f,%f,%f)" % (lon1,lat1,lon2,lat2),
2. "import c2")
3. **print** "Cython function (using trig function from math.h)", t.timeit(num), "sec"

现在有点喜欢它了吧？0.4秒 -比纯Python函数有5倍的速度增长。我们还有什么方法可以再提高速度？c2.great\_circle（）仍是一个Python函数调用，这意味着它产生Python的API的开销（构建参数元组等），如果我们可以写一个纯粹的C函数的话，我们也许能够加快速度。

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. cdef extern **from** "math.h":
2. float cosf(float theta)
3. float sinf(float theta)
4. float acosf(float theta)
6. cdef float \_great\_circle(float lon1,float lat1,float lon2,float lat2):
7. cdef float radius = 3956.0
8. cdef float pi = 3.14159265
9. cdef float x = pi/180.0
10. cdef float a,b,theta,c
12. a = (90.0-lat1)\*(x)
13. b = (90.0-lat2)\*(x)
14. theta = (lon2-lon1)\*(x)
15. c = acosf((cosf(a)\*cosf(b)) + (sinf(a)\*sinf(b)\*cosf(theta)))
16. **return** radius\*c
18. **def** great\_circle(float lon1,float lat1,float lon2,float lat2,int num):
19. cdef int i
20. cdef float x
21. **for** i **from** 0 < = i < num:
22. x = \_great\_circle(lon1,lat1,lon2,lat2)
23. **return** x

请注意，我们仍然有一个Python函数（ def ），它接受一个额外的参数 num。这个函数里的循环使用for i from 0 < = i< num: ，而不是更Pythonic，但慢得多的for i in range(num):。真正的计算工作是在C函数（cdef）中进行的，它返回float类型。这个版本只要0.2秒——比原先的Python函数速度提高10倍。

为了证明我们所做的已经足够优化，可以用纯C写一个小应用，然后测定时间：

**[python]** [view plaincopy](http://blog.csdn.net/gzlaiyonghao/article/details/4561611)

1. #include <math .h>
2. #include <stdio .h>
3. #define NUM 500000
5. float great\_circle(float lon1, float lat1, float lon2, float lat2){
6. float radius = 3956.0;
7. float pi = 3.14159265;
8. float x = pi/180.0;
9. float a,b,theta,c;
11. a = (90.0-lat1)\*(x);
12. b = (90.0-lat2)\*(x);
13. theta = (lon2-lon1)\*(x);
14. c = acos((cos(a)\*cos(b)) + (sin(a)\*sin(b)\*cos(theta)));
15. **return** radius\*c;
16. }
18. int main() {
19. int i;
20. float x;
21. **for** (i=0; i < = NUM; i++)
22. x = great\_circle(-72.345, 34.323, -61.823, 54.826);
23. printf("%f", x);
24. }

用gcc -lm -octest ctest.c编译它，测试用time./ctest ...大约0.2秒 。这使我有信心，我Cython扩展相对于我的C代码也极有效率（这并不是说我的C编程能力很弱）。

能够用 cython 优化多少性能通常取决于有多少循环，数字运算和Python函数调用，这些都会让程序变慢。已经有一些人报告说在某些案例上 100 至 1000 倍的速度提升。至于其他的任务，可能不会那么有用。在疯狂地用Cython重写 Python 代码之前，记住这一点：

"我们应该忘记小的效率，过早的优化是一切罪恶的根源，有 97% 的案例如此。"——DonaldKnuth

换句话说，先用 Python 编写程序，然后看它是否能够满足需要。大多数情况下，它的性能已经足够好了……但有时候真的觉得慢了，那就使用分析器找到瓶颈函数，然后用cython重写，很快就能够得到更高的性能。

外部链接  
WorldMill（http://trac.gispython.org/projects/PCL/wiki/WorldMill）——由Sean Gillies 用 Cython 编写的一个快速的，提供简洁的 python 接口的模块，封装了用以处理矢量地理空间数据的 libgdal 库。

编写更快的 Pyrex 代码（http://www.sagemath.org:9001/WritingFastPyrexCode）——Pyrex，是 Cython 的前身，它们有类似的目标和语法。