# Python与c混合编程

|  |  |  |  |
| --- | --- | --- | --- |
| 修改人 | 时间 | 操作 | 说明 |
| 贾贺杰 | 2014-1-13 | 创建 |  |
|  |  |  |  |
|  |  |  |  |

[Python与c混合编程 1](#_Toc377408059)

[一、目录环境配置 3](#_Toc377408060)

[1、 Python环境安装 3](#_Toc377408061)

[2、Python C API版本查看 3](#_Toc377408062)

[二、C++调用Python 4](#_Toc377408063)

[三、Python调用C++ 4](#_Toc377408064)

[1、Python API封装导出 4](#_Toc377408065)

[2、boost封装导出 4](#_Toc377408066)

[四、多线程与Python – GIL(Global Interpreter Lock) 6](#_Toc377408067)

## 一、目录环境配置

工程选用rhel-server-5.4 32/64位操作系统作为开发平台。

工程选用Python版本2.7.6，Python API版本1013。

编译器使用Redhat5自带g++，版本g++ (GCC) 4.1.2 20080704 (Red Hat 4.1.2-46)。

### 1、Python环境安装

1)Python下载地址<http://www.python.org/ftp/python/2.7.6/Python-2.7.6.tar.xz>；

2)将下载后的安装包想办法放到你的redhat系统里，位置随意；

3)在Python-2.7.6.tar.xz所在目录执行解压命令：tar -zxf Python-2.7.6.tgz，成功后进入下一步；

4)进入解压后的python目录执行一下目录：

./configure --enable-shared -enable-unicode=ucs4 --prefix=/usr/local/python2.7

make

make install

注：这里讲Python安装到了/usr/local目录下，可以将其直接安装到其他地方，别覆盖了旧版本的python就好。

--enable-shared 为了使python编译出libpython2.7.so.1.0这个共享库，之后使用python C API时有用。

-enable-unicode=ucs4让python支持这个字符集，不加这个在编译时会连接错误。参考<https://www.rosettacommons.org/node/1901>。

5) 将Python库复制到默认执行环境cp -r /usr/local/python2.7/lib/ /usr/lib/python2.7；

6) 将Python可执行程序复制到默认执行环境，并替换现有Python可执行程序（只保留一个python2.4给yum用即可）cp /usr/local/python2.7/bin/python\* /usr/bin/；

7) 删除原有的python可执行程序rm /usr/bin/python;

8) 建立软连接ln -s /usr/local/python2.7/bin/python /usr/bin/python;

9) 复制库cp /usr/local/python2.7/lib/libpython2.7.so\* /usr/lib/;

10) 复制头文件cp -r /usr/local/python2.7/include/python2.7/ /usr/include/;

11) 修改yum文件的Python版本为Python2.4，

vim /usr/bin/yum

#!/usr/bin/python 改为 #!/usr/bin/python2.4

保存退出；

执行python –V输出Python 2.7.6安装成功。

### 2、Python C API版本查看

执行命令：vim /usr/include/python2.7/modsupport.h;

第46行会看到以下内容：

46 #define PYTHON\_API\_VERSION 1013

47 #define PYTHON\_API\_STRING "1013"

确认使用的是1013版本的API。

## 二、C++调用Python

本章主要描述如何使用C++调用Python模块，Python模块内函数，如何向Python模块内函数传参，以及处理Python函数返回值。由于Python提供的C API不是线程安全的，因此将会在第四章说明如何在C++多线程环境中使用Python。

## 三、Python调用C++

### 1、Python API封装导出

### 2、boost封装导出

使用Boost封装导出函数的可执行程序会大很多。

Template.h

**namespace** Python{

**extern** "C" **void** **initTemplate**();

}

Template.cpp

**#include** "Template.h"

**#include** <boost/python.hpp>

**namespace** Python{

////定义要导出的函数

**int** **SayHello**(){

**printf**("python call c ...\n");

**return** 0;

}

**int** **SayHelloS**(**char**\* string){

**printf**("python call c %s ...\n",string);

**return** 0;

}

**int** **SayHelloSI**(**char**\* string,**int** d){

**printf**("python call c %s %d ...\n",string,d);

**return** 0;

}

////

BOOST\_PYTHON\_MODULE(Template){

**using** **namespace** boost::python;

def("SayHello",SayHello);////导出函数

def("SayHelloS",SayHelloS);

def("SayHelloSI",SayHelloSI);

}

}

Main.cpp

{

PyThreadStateAutoLock lock;////GIL全局解释器锁

Python::initTemplate();////初始化模块

}

Test.py

#! /usr/bin/env python2.7

# -\*- coding: utf8 -\*-

import sys

import time

import Template

def sayhi(name):

print ' >>>',name

Template.SayHello()

Template.SayHelloS('hello c++')

Template.SayHelloSI('hello c++',101)

return name

#return 110

if \_\_name\_\_=='\_\_main\_\_':

sayhi("main")

导出类



## 四、多线程与Python – GIL(Global Interpreter Lock)